基于离散视锥理论的烟幕遮蔽策略研究

摘　要

本文旨在构建一套无人机协同投放烟幕干扰弹的优化建模框架，通过深入分析空地导弹威胁的动态性与多机协同的复杂性，为非直接物理拦截防御策略的科学规划与高效实施提供理论依据与技术支撑。

对于问题一，我们建立了**基于离散视锥理论的有效遮蔽时长计算模型**，构建导弹、无人机和烟幕弹运动学方程的同时，将导弹指向目标的连续观测射线转化为对目标圆柱体外表面离散点的视线线段集合，并证明了仅采集圆柱体上下表面边缘的等效性。通过将烟幕球心到每一条离散视线的最短距离与有效半径进行比较，即可判断“有效遮蔽”。最终，通过 0.001s 步长的时间离散化与 17640 条视锥离散化，经过 2s 的代码求解，得出**有效遮蔽时长为 1.405 秒**。此外，我们还通过**单因子扰动法**进行了灵敏度分析，验证了模型可对投放时间等变量灵敏响应。

对于问题二，我们建立了**单目标多变量优化模型**，将遮蔽时间最大化作为优化目标，并采用了**粗细网格相结合的并行搜索方法**（时间步 10ms、视锥线 130 条），通过**多线程并行计算**筛选出多个候选方案并细化得到最优解。经过求解，得出了 **4.665 秒**的最大遮蔽时长，方案的具体参数为：**无人机航向角为 178.20°，飞行速度为 105.00m/s，投放时刻为 0.00s，引信延迟为 3.00s**。

对于问题三，我们将其构建为**混合整数二阶锥规划（MISOCP）模型**进行求解。首先将总遮蔽时长作为优化目标，通过时间离散化的预求解转化为二阶锥约束。再通过**分层启发式并行算法**，内层在固定的航向和速度下采用**贪心排程**策略，外层则对此进行**网格化搜索**。通过航向角 180.00°、速度 140m/s 等决策，三枚烟幕弹接力式地实现了总计 **6.712 秒**的有效遮蔽时长，单弹自身遮蔽时长为 **2.952s，2.672s，1.297s**；再引入圆柱外接球近似遮蔽新机制进行求解，得到结果高度相似，证明了此投放策略具有鲁棒性与合理性。

对于问题四，沿用问题三框架进行建模。在求解过程中，我们采取了**多阶段的启发式并行搜索**。该方法首先通过粗细网格搜索为每架无人机筛选出高质量的独立最优解，然后对这些最优解进行三机联合组合搜索。通过求解，我们得出三架无人机协同实现了 **11.630 秒**的总遮蔽时长。该方案通过将每架无人机的独立最优遮蔽时间（FY1: 4.580s，FY2: 3.970s，FY3: 3.110s）进行有效组合，使得总遮蔽时间远超单机能力。

对于问题五，结合问题三、四的已有模型进行求解。首先对每架无人机独立进行航向与速度的**网格搜索**，并在每个网格点上运用**贪心算法**，确定每架无人机的最佳烟幕弹投放策略。其次是对所有无人机的最优方案进行全局汇总与评估，以所有导弹的累计遮蔽时长之和作为最终的评分，选出全局最优解。最终，该协同策略实现了 **26.954 秒**的总遮蔽时长。结果显示，无人机 FY1、FY2、FY3 和 FY5 协同工作，其中 FY1 主要针对 M1，而其他无人机则跨目标进行机动支援，无人机 FY4 因其性价比较低而未被使用。

关键词:　离散视锥理论　混合整数二阶锥规划　网格搜索　贪心排程

一　问题重述

1.1　问题背景

随着现代战争形态的演变，以及武器装备的提升，高科技对抗已经成为了核心。其中，防御体系的构建与快速反应能力至关重要。传统的防御手段在面对当前高速、高精度的空地导弹等武器是，面临着巨大的挑战。为提高生存能力，采用非直接物理拦截的干扰技术，尤其是利用准确投放的烟幕进行隐蔽，已经成为一种高性价比的防御策略。当前，烟幕干扰技术正朝着精确化与智能化方向发展。相较于传统的烟幕投放，通过无人机精准定点投放烟幕干扰弹不仅效率大幅提升，而且覆盖范围广、更能控制引爆时间，形成了具有战略价值的烟幕云团。

本研究旨在解决烟幕干扰弹的投放策略优化问题。尽管无人机投放技术为烟幕布控提供了新的可能，但在实际应用中仍会面临许多的挑战。例如如何根据空地导弹的飞行路径和速度，以及无人机的初始状态与位置来科学地规划该无人机的飞行方向、速度、投放位置等关键参数，并且要对目标进行有效且持久的掩护。为应对一系列的挑战，我们将建立数学模型，针对不同数量的无人机和空地导弹以及他们的初始位置来规划无人机的最优投放方案，同时为未来无人机的多方面运用和防空反导实践提供新的思路。

1.2　问题要求

问题 1：针对无人机 FY1 投放 1 枚烟幕弹干扰导弹 M1，计算有效遮蔽时长。

问题 2：针对无人机 FY1 投放 1 枚烟幕弹干扰导弹 M1, 确定最优的投放策略，使遮蔽时间最长。

问题 3：针对无人机 FY1 投放 3 枚烟幕弹干扰导弹 M1，设计其投放策略，使遮蔽时间最长。

问题 4：针对无人机 FY1、FY2 和 FY3 各投放 1 枚烟幕弹干扰导弹 M1，设计其投放策略，使遮蔽时间最长。

问题 5：针对 FY1-FY5 五架无人机，最多投放 3 枚烟幕弹干扰导弹 M1、M2 和 M3，设计其投放策略，使遮蔽时间最长。

二　模型假设

1. 假设导弹、无人机、烟幕弹的运动为质点运动，即不考虑导弹、无人机、烟幕弹运动路径相交导致全部损毁的情况，不考虑空气阻力的影响。

2. 假设导弹向假目标为匀速直线运动，不考虑其实际应用中可能的弹道设计；并假设已有的有效遮蔽时间的长短，不对之后的飞行造成影响。

3. 假设烟幕弹起爆后，残留弹体的爆炸飞溅无任何实际影响。

三　符号说明

符号	含义说明	值	单位
V_m	导弹飞行速度	300	米每秒
V_f	无人机飞行速度	70-140	米每秒
V_{fog}	烟幕下沉速度	3	米每秒
t_{fog}	烟幕有效时间	20	秒
t_{yb}	烟幕弹投放时刻	-	-
t_{ys}	受领任务到投放烟幕弹的时间间隔	-	秒
t_{yf}	起爆时间间隔	-	秒
r	圆柱半径	7	米
h	圆柱高度	10	米
ψ	航向角	-	度

表 1: 符号说明

四　问题一：基于离散视锥理论有效遮蔽时长计算的数学模型

在问题一中，要求计算利用无人机 FY1 对向假目标直线进发的 M1 导弹进行对真目标有效遮蔽时长计算的数学模型，并且已知 FY1 确定的航向、速度及其单枚烟幕弹投放与起爆时刻与 M1 坐标与速度等具体参数。通过光路可逆性，将目标圆柱体外表面离散为有限个点，从而将导弹的连续视线转化为有限的离散线段。首先，需要利用题设现有的三维直角坐标系，再由运动学方程、视锥[1] 理论及其离散化建立模型，得到遮蔽时长的计算公式，代入已知的参数值得到求解的结果，具体大致建模过程如下。

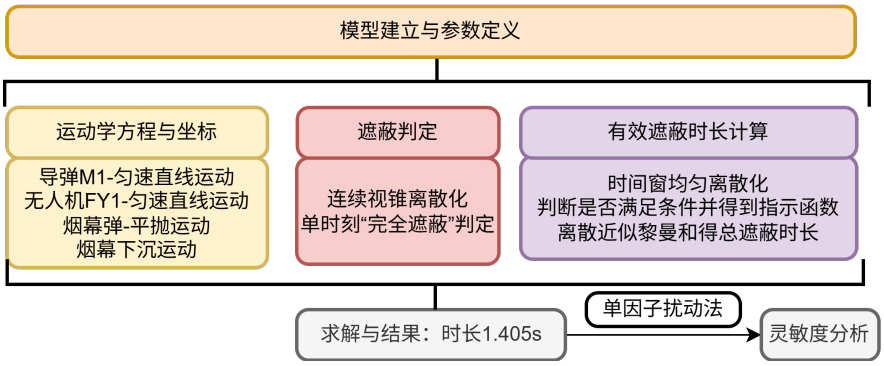

图 1: 问题一建模大致过程

4.1 模型建立

要得出遮蔽时间，首先要刻画出导弹、无人机、烟幕弹（及其烟雾的下沉）在坐标系中是如何随时间进行的演化，得到实时的导弹与烟幕的路径与范围，以此为有效遮蔽提供计算的基础。因此需要在坐标系中，先得出这三个物体的运动学方程及其坐标。

4.1.1 各物体的运动学方程与坐标变化

1. 导弹 M1

已知导弹飞行方向直接指向假目标，设定导弹飞行的终点为坐标 $\boldsymbol{M}_f = (0,0,0)$，初始坐标为 $\boldsymbol{M1}_0 = (20000,0,2000)$，飞行速度为 $V_m = 300m/s$，同时做均速直线运动，并考虑起始飞行时刻 $t=0$ 到击中目标后的结束，其随时间的运动学方程为

$$\boldsymbol{M1}(t) = \boldsymbol{M1}_0 + V_m \frac{\boldsymbol{M}_f - \boldsymbol{M1}_0}{\|\boldsymbol{M}_f - \boldsymbol{M1}_0\|} t, \quad 0 \le t \le \frac{\|\boldsymbol{M}_f - \boldsymbol{M1}_0\|}{V_m}.$$

为方便后续的求解，给出等价坐标形式，时间约束同上。

$$\boldsymbol{M1}(t) = \Big(\boldsymbol{M1}_{0x} + \frac{V_m t(\boldsymbol{M}_{fx} - \boldsymbol{M1}_{0x})}{\|\boldsymbol{M}_f - \boldsymbol{M1}_0\|},\ \boldsymbol{M1}_{0y} + \frac{V_m t(\boldsymbol{M}_{fy} - \boldsymbol{M1}_{0y})}{\|\boldsymbol{M}_f - \boldsymbol{M1}_0\|},\ \boldsymbol{M1}_{0z} + \frac{V_m t(\boldsymbol{M}_{fz} - \boldsymbol{M1}_{0z})}{\|\boldsymbol{M}_f - \boldsymbol{M1}_0\|}\Big) \tag{1}$$

2. 无人机 FY1

无人机也向假目标飞行（即按沿水平 x 负向飞行,，航向角为 $\psi = \pi$），初始坐标为 $\boldsymbol{FY1}_0 = (17620,0,1800)$，飞行速度为 $V_f = 120m/s$，也为匀速直线运动，其随时间的运动学方程与坐标为

$$\boldsymbol{FY1}(t) = \boldsymbol{FY1}_0 + V_f\big(\cos\psi,\ \sin\psi,\ 0\big)\, t = \big(\boldsymbol{FY1}_{0x} - V_f t,\ \boldsymbol{FY1}_{0y},\ \boldsymbol{FY1}_{0z}\big), \quad t \ge 0. \tag{2}$$

3. 烟幕弹与下沉烟幕

干扰弹脱离后仅受重力作用（忽略空气阻力），投放后做平抛运动，若在 $t_{\text{ys}} = 1.5s$ 投放，在起爆时间间隔 $t_{\text{yf}} = 3.6s$，考虑抛出时初始的坐标为 $\boldsymbol{Y}_0$，直接给出等效坐标表示为

$$\boldsymbol{Y}(t) = \Big(\boldsymbol{Y}_{0x} - V_f\,(t - t_{\text{ys}}),\ \boldsymbol{Y}_{0y},\ \boldsymbol{Y}_{0z} - \tfrac{1}{2}g\,(t - t_{\text{ys}})^2\Big), t_{\text{ys}} \ \le\ t\ \le\ t_{\text{ys}} + t_{\text{yf}} \tag{3}$$

其中，$\boldsymbol{Y}_0$ 的坐标可由投放时间 t_{ys} 与无人机的运动轨迹式(3)确定，即 $\boldsymbol{Y}_0 = \boldsymbol{FY1}(t_{\text{ys}})$。由平抛轨迹结束点得起爆点为 $\boldsymbol{F}_0 = \boldsymbol{Y}(t_{\text{yf}})$，考虑烟幕下沉速度为 $V_{\text{fog}} = 3m/s$，烟幕有效时间为 $t_{\text{fog}} = 20s$，烟幕球心等效坐标表示为

$$\boldsymbol{F}(t) = \big(\boldsymbol{F}_{0x},\ \boldsymbol{F}_{0y},\ \boldsymbol{F}_{0z} - V_{\text{fog}}\,(t - (t_{\text{ys}} + t_{\text{yf}}))\big), \quad t_{\text{ys}} + t_{\text{yf}} \le t \le t_{\text{ys}} + t_{\text{yf}} + t_{\text{fog}} \tag{4}$$

4.1.2 导弹离散化锥形视线与烟幕球的遮蔽判定

在已知了烟幕实时范围、导弹的实时路径后，即可进行逐时刻的遮蔽判定。可以将导弹视作一个“视点”，或者电磁波传感器，即从导弹位置发出的，能打到真目标的圆柱 G 的所有视线的集合 E，组成一个“视锥”（其实为从 M 发出的所有的射线的全集中，打到圆柱 G 的子集）。那么烟幕球在有效遮蔽目标圆柱体 G 时，即可等效为视锥集合 E，当其被所有的有效烟幕所截断，即所有射线至中心点 $\boldsymbol{F}$ 的距离，均小于题设的 $R_{\mathrm{l}} = 10m$。

1. 离散视锥理论的建立

可以发现，上述的射线集合 R 实质上还是由方向角连续参数化的不可数集合[5]，是考虑了所有到达 G 的射线，同时导弹与烟幕随时间连续演化，这样的建模及求解过于复杂，因此需要对此射线集合进行离散化近似分析。根据光路（电磁波）的可逆性，对于“视点”的视线离散，可以转换为对于目标物体的离散化，两者产生的视线在物理上完全等效，因此以下是对目标圆柱的离散化操作。

已知真目标圆柱的半径为 $r = 7m$、高为 $h = 10m$，其下底圆心为 $\boldsymbol{B} = (0, 200, 0)$，轴向与 z 轴一致。为近似“完全遮蔽”判定，将圆柱的外表面离散为有限点集，首先为方便离散，将侧面与顶面利用柱坐标进行参数表示。侧面的表示如下：

$$\boldsymbol{P}_{\mathrm{side}}(\varphi, z) = \boldsymbol{B} + (r\cos\varphi,\, r\sin\varphi,\, 0) + (0, 0, z),\ 0 \le \varphi < 2\pi,\ 0 \le z \le h \tag{5}$$

再是顶面的柱坐标表示：

$$\boldsymbol{P}_{\mathrm{disk}}(\rho, \varphi, \bar{z}) = \boldsymbol{B} + (\rho\cos\varphi,\, \rho\sin\varphi,\, h),\ 0 \le \rho \le r,\ 0 \le \varphi < 2\pi \tag{6}$$

在此基础上，为生成圆柱面的离散点集，取整数 N_φ, N_z, N_ρ 分别为角向、轴向、径向的离散密度参数，则各部分点位的参数可以表述为

$$\varphi_i = \tfrac{2\pi i}{N_\varphi},\ i = 0, \ldots, N_\varphi - 1; \qquad z_j = \tfrac{j}{N_z}\,h,\ j = 0, \ldots, N_z; \qquad \rho_m = \tfrac{m}{N_\rho}\,r,\ m = 0, \ldots, N_\rho \tag{7}$$

以此可以将参数代入到坐标参数表示式(5)和式(6)中，得到以下圆柱面离散化点集

$$\mathcal{S} = \{\boldsymbol{P}_{\mathrm{side}}(\varphi_i, z_j)\} \cup \{\boldsymbol{P}_{\mathrm{disk}}(\rho_m, \varphi_i, 0)\} \cup \{\boldsymbol{P}_{\mathrm{disk}}(\rho_m, \varphi_i, h)\} \tag{8}$$

由此也可以算得点集的点数，计算时应当去除圆盘中心点 $\rho = 0$ 共 $N_\varphi - 1$ 个的重合情况

$$|\mathcal{S}| = N_\varphi\,(N_z + 1) \;+\; 2\,(N_\varphi\,N_\rho + 1) = N_\varphi\,(N_z + 1 + 2N_\rho) \;+\; 2$$

可见，网格越密，即 N_φ, N_z, N_ρ 越大，对“完全遮蔽”的判定越逼近真实结论。下图即为离散视锥定义的示意：

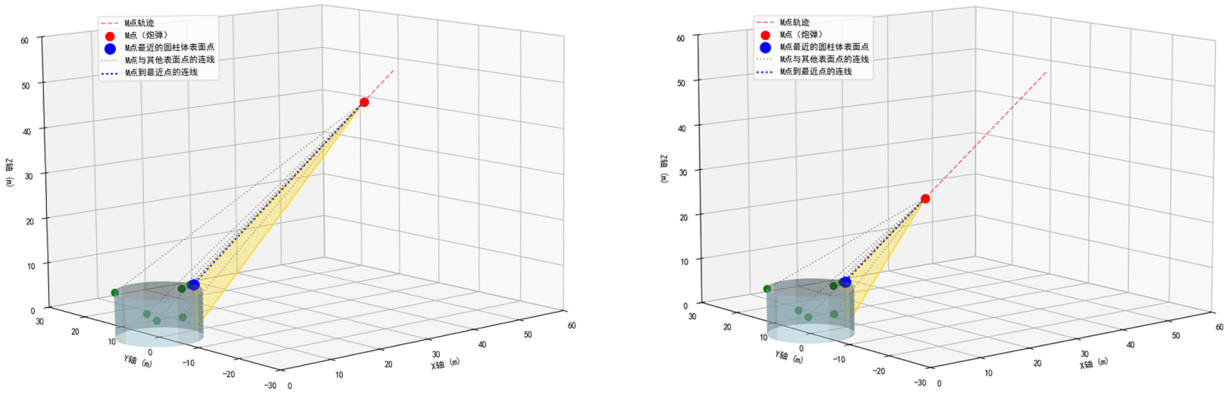

图 2: 导弹离散视锥随时间变化示例

2. 目标圆柱等效简化离散证明

通过以上圆柱离散化，可以对圆柱全体是否被烟幕遮蔽进行有效的近似判定，但是这一办法存在大量的离散点冗余问题，即存在许多离散点对于遮蔽的判断不起决定性作用；实质上，只需要在上下顶面、底面边缘圆周进行离散点采集，即可“代表”侧面、顶面诸点的遮蔽情况。

【命题】：仅沿上下顶面、底面边缘圆周的离散点采集，与顶面采集、侧面采集在全圆柱体的遮蔽判定上完全等效。

【证明】: 设导弹此时在点 M，烟幕球 $B(F,r)$。记所有遮蔽锥内的射线，与烟幕球 $B(F,r)$ 的交不为空，写为下式

$$\mathcal{L} := \{\, M + \lambda v : \lambda \geq 0,\ (M + \mathbb{R}v) \cap B(F,r) \neq \emptyset \,\}.$$

设真目标为半径 R、高 h 的直立圆柱 C，其上下底面边缘圆周分别为 R_0，R_h。则上述的命题可以转化为连续的加强形式，即圆柱闭合凸集 C 包含于 $\mathcal{L}$ 的充要条件为上下底面边缘圆周 R_0，R_h 包含于 $\mathcal{L}$，写为下式

$$C \subset \mathcal{L} \quad \Longleftrightarrow \quad R_0 \cup R_h \subset \mathcal{L}.$$

由于 $\mathcal{L}$ 包含的是所有从 $\boldsymbol{M1}(t)$ 出发经过烟幕球的射线（视线），因此 $\mathcal{L}$ 显然为凸集，即在集合内的处处连线段仍然在集合内。每个底面实心圆盘是其边界圆的凸包，而侧面上任一点位于连接 R_0 与 R_h 的某条轴向线段上。因此，圆柱为两条边缘圆周的凸包，即

$$C = \mathrm{conv}(R_0 \cup R_h).$$

由以上分析可以立得等价：

$$R_0 \cup R_h \subset \mathcal{L} \iff \mathrm{conv}(R_0 \cup R_h) \subset \mathcal{L} \iff C \subset \mathcal{L},$$

得证。以此构建新的圆柱上下边缘离散化集合为 $\mathcal{S}'$，定义如下式

$$\begin{aligned}\mathcal{S}' = &\left\{(r\cos\varphi'_i, r\sin\varphi'_i, 0) \mid \varphi'_i = \frac{2\pi i}{N_\varphi}, i = 0, \ldots, N`_\varphi - 1\right\} \\ &\cup \left\{(r\cos\varphi'_i, r\sin\varphi'_i, h) \mid \varphi'_i = \frac{2\pi i}{N_\varphi}, i = 0, \ldots, N_\varphi - 1\right\}\end{aligned} \tag{9}$$

$$\varphi'_i = \frac{2\pi i}{N_\varphi},\ i = 0, \ldots, N_\varphi - 1 \tag{10}$$

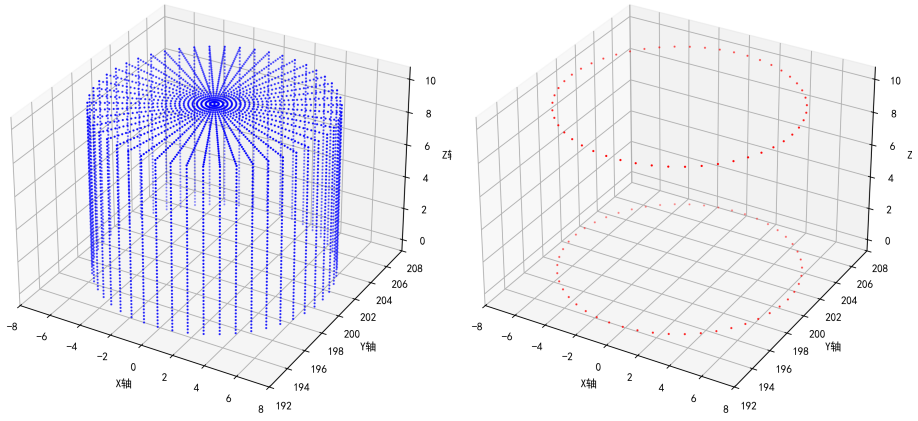

图 3: 圆柱面不同离散点集示意，左图为包括顶面侧面的离散，右图为上下边缘的简化离散

3. 单时刻的“完全遮蔽”判定

在进行了从圆柱体离散化从而视线有效离散化处理后，结合烟幕中心的坐标表示后，可以进行单时刻的“完全遮蔽”的判定。首先需要将离散视线集进行定量表述。给定时刻 t，记导弹位置 $\boldsymbol{M1}(t)$，烟幕球心 $\boldsymbol{F}(t)$，阈值半径 $R_l = 10\text{m}$。再利用圆柱离散点集合 $\mathcal{S}$ 定义“离散视线集合”为

$$E(t) = \left\{ \overline{\boldsymbol{M1}(t)\,\boldsymbol{P}} \ :\ \boldsymbol{P} \in \mathcal{S} \right\} \tag{11}$$

由此可知确定的线段方程与烟幕点中心点坐标，即可计算出烟幕球心到该视线线段的最短距离 δ。利用 clamp 函数控制最短距离线段在 $\boldsymbol{M1}(t)$ 与 $\boldsymbol{P}$ 之间（含端点），不超出线段的范围，具体的最短距离推导在此省略，见附录

$$\delta = \left\| \boldsymbol{F} - \boldsymbol{M} - \text{clamp}\left(\frac{(\boldsymbol{F}-\boldsymbol{M})\cdot(\boldsymbol{P}-\boldsymbol{M})}{\|\boldsymbol{P}-\boldsymbol{M}\|^2},\ 0,\ 1 \right)(\boldsymbol{P}-\boldsymbol{M}) \right\| \tag{12}$$

据此定义二元变量 $X_i(t)$（被遮蔽记 0，未遮蔽记 1），记某条视线段为 $\overline{\boldsymbol{M}\boldsymbol{P}_i}$，其最短距离为 $\delta_i(t)$。给定阈值 $R_l = 10$ m，定义为

$$X_i(t) = \begin{cases} 0, & \delta_i(t) \le R_l \quad \text{（被烟幕遮蔽）}, \\ 1, & \delta_i(t) > R_l \quad \text{（未被遮蔽）}. \end{cases}$$

为了在后续的最优化模型中，避免出现关于 $\delta_i(t) > R_l$ 的控制转移计算，将上式利用取整与反三角函数内取极限的方式，等效转化如下

$$X_i(t) = \left\lfloor \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan(k\delta_i(t) - kR_l)\right)\right\rfloor \quad \text{取 } k \to \infty. \tag{13}$$

算出单个射线的各个遮蔽情况后，对二元量进行求和，所有射线均遮蔽即求和为 0 时，才可推得“完全遮蔽”

$$\sum_{i=1}^{N} X_i(t) = 0 \iff \forall i,\ \delta_i(t) \le R_l.$$

4. 时间离散与有效遮蔽时长

分析出单个时刻的有效遮蔽情况后，即可进行时间的离散，分析各个步长的有效遮蔽时长情况，并最终得到总的有效的遮蔽时长。先将“有效遮蔽”视作一个随时间变化的指示函数，定义该时刻整体有效遮蔽指示

$$\chi(t) = \begin{cases} 1, & \sum_{i=1}^{N} X_i(t) = 0 \\ 0, & \sum_{i=1}^{N} X_i(t) \neq 0 \end{cases}$$

同 $X_i(t)$ 的处理，为避免控制转移，利用求积函数给出 $\chi(t)$ 的如下等效转化

$$\chi(t) = \prod_{i=1}^{N}\big(1 - X_i(t)\big) \tag{14}$$

在起爆有效窗 $[t_b,\, t_b + t_{\text{fog}}]$ 上取均匀时间网格 $t_k = t_b + k\,\Delta t$ $(k = 0, 1, \ldots, K,\ K\,\Delta t \approx t_{\text{fog}})$。对所有的离散化的时刻进行分析，计算 $\chi(t)$ 值，即可得最后的目标总体有效遮蔽时间取离散近似左端点黎曼和为

$$T \approx \sum_{k=0}^{K-1} \chi(t_k)\,\Delta t$$

4.1.3 有效遮蔽时长的目标函数

综上所有分析，可以得出有效遮蔽时长的计算模型为

$$T = \int_{t_b}^{t_b + t_{\text{fog}}} \chi(t)\,\mathrm{d}t \;\approx\; \sum_{k=0}^{K-1} \chi(t_k)\,\Delta t, \quad t_k = t_b + k\,\Delta t\ . \tag{15}$$

对于此遮蔽时长的目标函数，由物理推导而得的约束为

$$
\begin{cases}
\chi(t)=\prod_{i=1}^{N}\big(1-X_i(t)\big) & \text{(单时刻遮蔽判定函数)}\\
X_i(t)=\left\lfloor \frac{1}{2}\left(1+\frac{2}{\pi}\arctan(k\delta_i(t)-kR_l)\right)\right\rfloor \quad \text{取 } k\to\infty. & \text{(单条视线二元变量)}\\
\delta_i(t)=\big\|\boldsymbol{F}(t)-\boldsymbol{M1}(t)-s_i^*(\boldsymbol{P}_i-\boldsymbol{M1}(t))\big\| & \text{(视线最短距离公式)}\\
s_i^*=\operatorname{clamp}\left(\dfrac{(\boldsymbol{F}(t)-\boldsymbol{M1}(t))\cdot(\boldsymbol{P}_i-\boldsymbol{M1}(t))}{\|\boldsymbol{P}_i-\boldsymbol{M1}(t)\|^2},0,1\right) & \text{(截断投影系数)}\\
\boldsymbol{E}(t)=\{\overline{\boldsymbol{M1}(t)\boldsymbol{P}}:\boldsymbol{P}\in\mathcal{S}'\},\quad R_l=10\,\mathrm{m}. & \text{(视线集合)}\\
\mathcal{S}'=\left\{(r\cos\varphi_i',r\sin\varphi_i',0)\mid\varphi_i'=\dfrac{2\pi i}{N_\varphi},i=0,\dots,N`_\varphi-1\right\}\\
\quad\cup\left\{(r\cos\varphi_i',r\sin\varphi_i',h)\mid\varphi_i'=\dfrac{2\pi i}{N_\varphi},i=0,\dots,N`_\varphi-1\right\} & \text{(目标圆柱离散化)}\\
\varphi_i'=\dfrac{2\pi i}{N_\varphi},\ i=0,\dots,N`_\varphi-1 & \text{(离散化参数)}\\
\boldsymbol{F}(t)=\boldsymbol{F}_0+(0,0,-V_{\mathrm{fog}})(t-(t_{\mathrm{ys}}+t_{\mathrm{yf}})),\quad t\in[t_b,t_b+t_{\mathrm{fog}}] & \text{(烟幕球心运动学)}\\
\boldsymbol{FY1}(t)=\boldsymbol{FY1}_0+V_f(\cos\psi,\sin\psi,0)t & \text{(无人机运动学)}\\
\boldsymbol{M1}(t)=\boldsymbol{M1}_0+V_m\dfrac{\boldsymbol{M}_f-\boldsymbol{M1}_0}{\|\boldsymbol{M}_f-\boldsymbol{M1}_0\|}t & \text{(导弹运动学)}\\
\boldsymbol{Y}(t)=\big(\boldsymbol{Y}_{0x}-V_f(t-t_{\mathrm{ys}}),\boldsymbol{Y}_{0y},\boldsymbol{Y}_{0z}-\dfrac{1}{2}g(t-t_{\mathrm{ys}})^2\big),\\
\qquad\qquad t_{\mathrm{ys}}\le t\le t_{\mathrm{ys}}+t_{\mathrm{yf}}. & \text{(烟幕弹运动学)}
\end{cases}
\tag{16}
$$

4.2 模型求解与结果

根据上述建立的计算模型(15)及其物理约束(16)，将问题一的题设诸如飞行速度 $V_f=120m/s$、投放时刻 $t_{\mathrm{ys}}=1.5s$、起爆时间间隔 $t_{\mathrm{yf}}=3.6s$ 等参数代入模型；再利用 Python 编写程序求解（步长设置 0.001 s，此步长导弹位移单步仅 3m，相对 20km 尺度占比 0.015%，其精度完全足够）。对于顶面/底面/侧面都进行真目标圆柱体采集的离散点集合 $\mathcal{S}'$，利用了 17640 根视锥线进行求解，代码运行用时 2s，得到总体的**有效遮蔽时长为 1.405s**；对于仅在上下圆柱边缘采集的离散点集合 $\mathcal{S}'$, 利用了 720 根视锥线进行求解，求解结果也为 1.405s；这也在工程上再次证明了 $\mathcal{S}'$ 与 $\mathcal{S}'$ 的离散点集合，即使 $\mathcal{S}'$ 的采样更少，但是依然为等效采样（同等角参数 φ_i 取值下），以下是详细的求解数据。

表 2: 导弹与烟幕遮蔽过程求解结果表

阶段	时刻	导弹位置	烟幕中心	导弹到烟幕距离
遮蔽开始	$t=8.044\,\mathrm{s}$	[17598.7763, 0, 1759.8776]	[17188, 0, 1727.5992]	412.0 m
遮蔽结束	$t=9.448\,\mathrm{s}$	[17179.6666, 0, 1717.9667]	[17188, 0, 1723.3872]	9.9 m
完全遮蔽总时长：1.405 s				

以下利用 94 根绿色的视锥线进行遮蔽过程的可视化示意，进一步展示实际求解的内容机理。

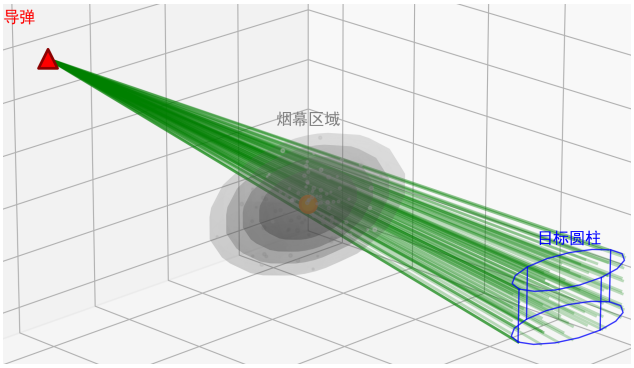

图 4: 完全遮蔽情况示意

4.3 灵敏度分析

为了评估各参数对烟幕遮蔽效果的影响程度，我们采用单因子扰动法进行灵敏度分析。选取 9 个关键参数进行分析，包括真目标几何与位置参数（圆柱半径 R、高度 H、坐标位置 Y）、运动参数（导弹速度、无人机速度、云团下沉速度）、时间参数（投放时间、起爆延时）和效果参数（有效遮蔽半径）。对每个参数在基准值基础上分别进行 10% 的扰动，保持其他参数不变，计算对应的有效遮蔽时长。并使用灵敏度系数公式：$S_i = \frac{\Delta T/T_0}{\Delta p_i/p_{i,0}}$，其中 $T_0 = 1.405s$ 为解出的基准遮蔽时长，$p_{i,0}$ 为参数基准值。观察发现，导弹速度（S = -4.435）影响最显著，速度越快遮蔽时长越短，符合物理直觉；有效遮蔽半径（S = +1.735）烟雾范围越大，遮蔽效果将显著提升；控制其他变量的情况下，的确存在最佳的投放时刻与起爆时刻；无人机速度（S = +1.111）将会影响投放点选择和时机配合。

表 3: 灵敏度分析结果

参数	灵敏度系数	影响程度	相关性
导弹速度 (m/s)	-4.435	高	负相关
有效遮蔽半径 (m)	1.735	高	正相关
投放时间 (s)	-1.189	高	负相关
无人机速度 (m/s)	1.111	高	正相关
云团下沉速度 (m/s)	0.964	高	正相关
坐标位置 (m)	-0.372	中	负相关
目标半径 (m)	-0.014	低	负相关
起爆延时 (s)	0.001	低	正相关
目标高度 (m)	-0.001	低	负相关

通过以上的分析，可以发现导弹和无人机的速度参数对遮蔽效果影响远大于目标几何参数；投放时间的精确控制至关重要，延迟投放将显著降低效果；同时其扰动方向及对应结果基本符合直观认知，这也进一步说明了模型能够有效捕捉物理参量变化。

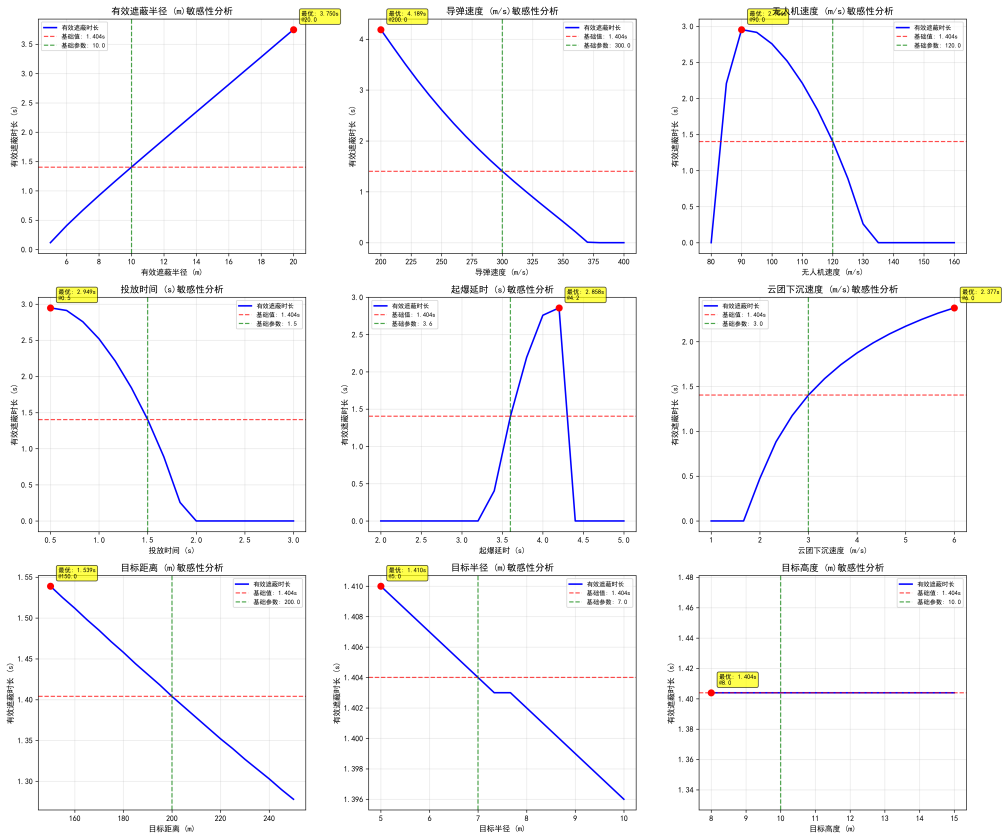

图 5: 真目标、运动、时间、烟幕效果共 9 个参数的灵敏度分析

五　问题二: 单机单弹遮蔽时间最优化模型

在问题二中，要求求解出 FY1 对于 M1 导弹的最优遮蔽时长的方案，包括航向角 ψ，无人机飞行速度 V_f，烟幕弹投放时刻 t_{yb} 与起爆时间间隔 t_{yf} 四个决策变量，结合已有的运动学方程与各物体坐标随时间变化，可得题设要求的烟幕弹的投放点 $\boldsymbol{Y}(0)$ 与起爆点 $\boldsymbol{Y}(t_{\mathrm{yf}})$。由于问题一的有效遮蔽时间的基本计算模型中，已经将上述的决策变量作为参量考虑，因此直接利用问题一的模型进行优化即可，结合网格并行搜索得到最优化的遮蔽时间 T。

5.1　模型建立

5.1.1　优化目标

设起爆时刻 $t_b = t_{\mathrm{ys}} + t_{\mathrm{yf}}$。在起爆有效窗 $[t_b,\ t_b + t_{\mathrm{fog}}]$ 上取均匀网格 $t_k = t_b + k\Delta t$，$k = 0, \ldots, K-1$，$K\Delta t \approx t_{\mathrm{fog}}$。则遮蔽时长离散近似最大化为目标函数

$$\max_{\psi, V_f, t_{\mathrm{ys}}, t_{\mathrm{yf}}} \quad T(\psi, V_f, t_{\mathrm{ys}}, t_{\mathrm{yf}}) = \sum_{k=0}^{K-1} \chi(t_k)\, \Delta t$$

5.1.2 约束条件

1. 航向角、飞行速度、投放时间以及引信延时的决策变量定义域约束

$$\psi \in [0, 2\pi), \quad V_f \in [70, 140], \quad t_{\text{ys}} > 0, \quad t_{\text{yf}} > 0 \tag{17}$$

2. 投放时间、起爆时间的有效性约束

首先，要起爆要在导弹到达假目标前的视锥存在时间内（总飞行时长），即为下式

$$t_{\text{ys}} + t_{\text{yf}} < t_{\text{hit}} = \frac{\|\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0\|}{V_m} = \frac{\sqrt{(20000)^2 + (2000)^2}}{300} \approx 67.0 \text{ s} \tag{18}$$

还要考虑下抛段的时间不能过长，不能使烟幕弹都落到地面上（XOY 平面），即为下式，代入题设参数算得 $t_{yf} < 19.17s$

$$\boldsymbol{F}_{0z} = \boldsymbol{FY}\mathbf{1}_{0z} - \frac{1}{2}gt_{yf}^2 > 0 \quad \Rightarrow \quad 0 < t_{yf} < \sqrt{\frac{2FY\mathbf{1}_{0z}}{g}} \tag{19}$$

同样地，考虑烟幕的下沉效应，起爆高度必须足够高，其题设的有效时长 $t_{\text{fog}} = 20s$，真正的烟幕有效时长应截断在触地前，因此需要避免烟幕触地导致无效，即为下式，代入题设算得 $t_{yf} < 18.85s$

$$F_{0z} - V_{\text{fog}}\, t_{\text{fog}} > 0 \implies t_{\text{yf}} < \sqrt{\frac{2\big(\boldsymbol{FY}\mathbf{1}_{0z} - V_{\text{fog}}\, t_{\text{fog}}\big)}{g}} \tag{20}$$

最关键的是，不能在无效的时间段进行投放与起爆的操作，这样会直接使得遮蔽时间为 0；即必须存在一个时刻，使得在有效窗内至少存在一个时刻完全遮蔽

$$\min_{t \in [t_{\text{ys}} + t_{\text{yf}},\, t_{\text{ys}} + t_{\text{yf}} + t_{\text{fog}}]} g(t) \;\le\; 0 \quad \Longleftrightarrow \quad \exists\, t^\star \in [t_{\text{ys}} + t_{\text{yf}},\, t_{\text{ys}} + t_{\text{yf}} + t_{\text{fog}}] \text{ s.t. } \chi(t^\star) = 1 \tag{21}$$

3. 运动学方程约束

由问题一的运动学推导得四个物体（导弹、无人机、烟幕弹、烟幕中心）物理运动的物理约束，即为下式

$$\begin{cases} \boldsymbol{M}\mathbf{1}(t) = \boldsymbol{M}\mathbf{1}_0 + V_m \dfrac{\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0}{\|\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0\|} t \\ \boldsymbol{FY}\mathbf{1}(t) = \boldsymbol{FY}\mathbf{1}_0 + V_f(\cos\psi, \sin\psi, 0)t \\ \boldsymbol{F}(t) = \boldsymbol{F}_0 + (0, 0, -V_{\text{fog}})\left(t - (t_{\text{ys}} + t_{\text{yf}})\right), \quad t \in [t_b,\, t_b + t_{\text{fog}}] \\ \boldsymbol{Y}(t) = \left(\boldsymbol{Y}_{0x} - V_f\,(t - t_{\text{ys}}),\;\; \boldsymbol{Y}_{0y},\;\; \boldsymbol{Y}_{0z} - \frac{1}{2}g\,(t - t_{\text{ys}})^2\right), t_{\text{ys}} \;\le\; t \;\le\; t_{\text{ys}} + t_{\text{yf}} \end{cases} \tag{22}$$

4. 常量约束

对于题设的常量，包括导弹匀速直线运动之速度 300m/s，重力常量 g，烟幕有效半径 10m，烟幕有效时间 20s，下沉速度 3m/s 以及导弹 M1 与无人机 FY1 的初始坐标，上述常量对最优化求解的约束作用也要考虑，约束如下式：

$$\begin{aligned} &V_m = 300\,\mathrm{m/s}, g = 9.8\,\mathrm{m/s^2}, t_{\mathrm{fog}} = 20\,\mathrm{s}, R_l = 10m \\ &V_{\mathrm{fog}} = 3\,\mathrm{m/s}, \boldsymbol{FY1}(0) = (17800, 0, 1800), \boldsymbol{M1}(0) = (20000, 0, 2000) \end{aligned} \tag{23}$$

5.1.3 完整模型

根据问题二，建立以下优化模型，以最大化单枚烟幕弹的总体遮蔽时间 $T(\psi, V_f, t_{\mathrm{yb}}, t_{\mathrm{yf}})$ 为目标，决策变量为航向角 ψ、飞行速度 V_f、投放时间 t_{yb} 和起爆时间 t_{yf}，约束包括上文以及问题一的视锥视线的离散化与遮蔽判定。

$$\max_{\psi, V_f, t_{\mathrm{ys}}, t_{\mathrm{yf}}} T(\psi, V_f, t_{\mathrm{ys}}, t_{\mathrm{yf}}) = \sum_{k=0}^{K-1} \chi(t_k)\,\Delta t$$

$s.t.$

$$\begin{cases} \chi(t) = \prod_{i=1}^{N} \big(1 - X_i(t)\big) & \text{(单时刻遮蔽)} \\ X_i(t) = \left\lfloor \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan(k\delta_i(t) - kR_l)\right)\right\rfloor \quad \text{取 } k \to \infty & \text{(单视线遮蔽)} \\ V_m = 300\,\mathrm{m/s},\ g = 9.8\,\mathrm{m/s^2},\ t_{\mathrm{fog}} = 20\,\mathrm{s}, R_l = 10\,\mathrm{m}, \\ V_{\mathrm{fog}} = 3\,\mathrm{m/s},\ \boldsymbol{FY1}(0) = (17800, 0, 1800),\ \boldsymbol{M1}(0) = (20000, 0, 2000) & \text{(常量设定约束)} \end{cases}$$

$$\begin{cases} E(t) = \{\overline{\boldsymbol{M1}(t)\,\boldsymbol{P}} :\ \boldsymbol{P} \in \mathcal{S}'\} & \text{(视线线段集合)} \\ \delta_i(t) = \big\|\boldsymbol{F}(t) - \boldsymbol{M1}(t) - s_i^{\star}\big(\boldsymbol{P}_i - \boldsymbol{M1}(t)\big)\big\| & \text{(烟幕球心到线段最短距离)} \\ s_i^{\star} = \mathrm{clamp}\Big(\frac{(\boldsymbol{F} - \boldsymbol{M1})\cdot(\boldsymbol{P}_i - \boldsymbol{M1})}{\|\boldsymbol{P}_i - \boldsymbol{M1}\|^2},\ 0,\ 1\Big) & \text{(线段投影参数截断在 } [0,1] \text{ 内)} \end{cases} \tag{24}$$

$$\begin{cases} \varphi_i = \dfrac{2\pi i}{N_\varphi}, \quad i = 0, \ldots, N_\varphi - 1 \quad \text{(角向离散)} \\ z_j = \dfrac{j}{N_z} h, \quad j = 0, \ldots, N_z \quad \text{(高度方向离散)} \\ \rho_m = \dfrac{m}{N_\rho} r, \quad m = 0, \ldots, N_\rho \quad \text{(径向离散)} \end{cases} \tag{25}$$

$$
\begin{cases}
\boldsymbol{P}_{\text{side}}(\varphi_i, z_j) = \boldsymbol{B} + (r\cos\varphi_i,\ r\sin\varphi_i, 0) + (0,0,z_j) & \text{(圆柱侧面离散点)} \\
\boldsymbol{P}_{\text{disk}}(\rho_m, \varphi_i, \zeta) = \boldsymbol{B} + (\rho_m\cos\varphi_i,\ \rho_m\sin\varphi_i,\ \zeta),\quad \zeta \in \{0, h\} & \text{(圆柱顶/底面离散点)} \\
\mathcal{S}' = \{\boldsymbol{P}_{\text{side}}(\varphi_i, z_j)\} \cup \{\boldsymbol{P}_{\text{disk}}(\rho_m, \varphi_i, 0)\} \cup \{\boldsymbol{P}_{\text{disk}}(\rho_m, \varphi_i, h)\} & \text{(圆柱点离散集合)}
\end{cases}
$$

$$
\begin{cases}
\psi \in [0, 2\pi), \quad V_f \in [70, 140] & \text{(航向角与无人机速度约束)} \\
t_{\text{yb}} + t_{\text{yf}} < \dfrac{\sqrt{(20000)^2 + (2000)^2}}{300} & \text{(导弹到目标飞行时间上限} \approx 67.0 \text{ s)} \\
0 < t_{\text{yf}} < \sqrt{\dfrac{2\,FY1_{0z}}{g}} & \text{(下落时长上限} \approx 19.17 \text{ s)} \\
t_{\text{yf}} < \sqrt{\dfrac{2\,(FY1_{0z} - V_{\text{fog}} t_{\text{fog}})}{g}} & \text{(起爆高度时长约束} \approx 18.85 \text{ s)} \\
\exists t^{\star} \in [t_{\text{yb}} + t_{\text{yf}},\ t_{\text{yb}} + t_{\text{yf}} + t_{\text{fog}}],\ \chi(t^{\star}) = 1 & \text{(至少存在一次完全遮蔽)}
\end{cases}
$$

$$
\begin{cases}
\boldsymbol{M1}(t) = \boldsymbol{M1}_0 + V_m \dfrac{\boldsymbol{M}_f - \boldsymbol{M1}_0}{\|\boldsymbol{M}_f - \boldsymbol{M1}_0\|}\, t & \text{(导弹直线匀速飞行)} \\
\boldsymbol{FY1}(t) = \boldsymbol{FY1}_0 + V_f(\cos\psi,\ \sin\psi,\ 0)\, t & \text{(无人机水平匀速飞行)} \\
\boldsymbol{F}(t) = \boldsymbol{F}_0 + (0, 0, -V_{\text{fog}})\left(t - (t_{\text{yb}} + t_{\text{yf}})\right) & \text{(烟幕球心随时间下沉)} \\
\boldsymbol{Y}(t) = \left(\boldsymbol{Y}_{0x} - V_f(t - t_{\text{yb}}),\ \boldsymbol{Y}_{0y},\ \boldsymbol{Y}_{0z} - \frac{1}{2}g(t - t_{\text{yb}})^2\right) & \text{(烟幕弹自由落体轨迹)}
\end{cases}
$$

5.2 模型求解与结果

对于上述过程建立的遮蔽时间的单目标优化模型，我们基于仿真简化判断、粗搜以及细搜的网格并行搜索的思想进行逐步优化求解。

视锥全遮蔽判定与并行搜索算法的步骤

Step1 场景仿真构建

构建导弹、假目标、无人机以及烟幕弹的运动学仿真参数和代码，以此判断逐时刻的烟幕遮蔽情况。

Step2 假目标及无人机参量的离散化

根据前面几何模型的建模，我们在假目标圆柱体的顶面、底面圆周，每个圆周上均匀采样 36 个点；视线方位相对基准航向的偏离限定在 $\pm 10^{\circ}$ 范围内，等距取 21 个角度；无人机飞行速度在 $[70, 140]$ 范围内等距划分为 71 级；投放时刻在 $[0, 50]$ 范围内划分为 51 级；引信延迟在 $[0, 12]$ 范围内划分为 13 级。由此形成“圆周 × 方位 × 速度 × 时刻 × 延迟”的离散参数网格。

Step3 多层级烟幕遮蔽判定策略

视锥全遮蔽判定与并行搜索算法的步骤（续）

根据题目要求，我们基于视锥思想设计了遮蔽判定策略。当导弹到假目标圆柱体上所有可见采样点的视线均被烟幕球体遮挡时，认为导弹对假目标完全“失去视野”，判定为“有效遮蔽”。

Step4 多进程并行参数空间搜索

面对巨大的参数组合空间，我们实现了基于进程池的并行计算框架。搜索过程分为粗搜索和细化两个阶段：首先在全局参数空间进行粗搜索，使用小顶堆动态维护得分最高的 Top-K 解集；然后对这些解进行局部细化，在其附近参数空间进行更精细的网格搜索以获得局部最优解。

最后通过求解简化遮蔽判断的代码，得到最优的遮蔽时间区间为 $[3.000, 7.665]$，**遮蔽时长约为 4.665 s**。在我们设定的精度要求下，代码运行用时用时约 3 分钟。经过人工验算起爆点与遮蔽时间，这个遮蔽时间的结果在误差范围内正确。具体投放策略与可视化示意如下

表 5: 高精度（圆柱离散化）无人机投放策略求解结果

航向角	飞行速度	投放时刻	引信延时	起爆点坐标	遮蔽时间区间
178.20°	105.00 m/s	0.00 s	3.00 s	(17485.16, 9.89, 1755.86)	[2.760, 7.360] s
遮蔽时长			**4.665** s		

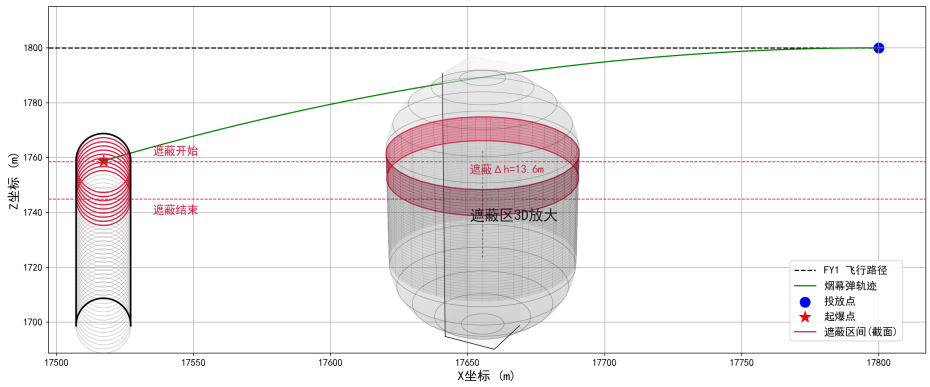

图 6: 问题二投放情景的可视化

六　问题三：单机三弹的遮蔽时长最优化模型

在问题三中，我们设计无人机 FY1 发射 3 颗烟幕弹，以遮蔽导弹 M1 对真实目标的视锥，并使遮蔽时间最长[6]。本节在问题一的离散视锥与时间离散判定框架、问题二的最优化模型的建立基础上，将三枚烟幕的时空布置纳入统一优化，由于问题的复杂度显著提高，同时体现建模求解的多样性，将整数变量（单时刻遮蔽判定 χ_k 的 0-1 二元变量等）与遮

蔽距离的二阶锥约束（SOCC）结合，据此引入了混合整数二阶锥规划（MISOCP）模型[3]，并在问题四、五中都得以应用。[2]

6.1 模型建立

以下对模型增加的各个参量进行新定义，主要是对 3 枚烟雾弹的投放问题进行模型的适配与待决策量及其派生的定义与约束。

6.1.1 模型参量定义

为了预计算 $s^*_{i,k,n}$ 在烟幕中心与视线计算二阶锥约束中的 clamp 形式的非线性量，需将时间离散化，时间网格索引设为 $k = 0, \ldots, K-1$（$t_k = k\Delta t$，$K\Delta t < \frac{\|\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0\|}{V_m}$）。

1. 决策变量

无人机常量航向角 ψ；无人机匀速飞行速度 $V_f \in [70, 140]$；

三枚烟幕弹的投放时刻 $t_{\mathrm{ys},n} \geq 0,\ n = 1, 2, 3$；

三枚烟幕弹各自的引信延时（起爆时间间隔）$t_{\mathrm{yf},n},\ n = 1, 2, 3$；

2. 中间变量

$t_{b,n} = t_{\mathrm{ys},n} + t_{\mathrm{yf},n}$：对各个烟幕弹的起爆时刻定义；

$X_{i,k,n} \in \{0, 1\}$：若第 n 枚在 t_k 对离散点 $\boldsymbol{P}_i$ 遮蔽，则 $X_{i,k,n} = 1$，否则为 0；

$u_{i,k} \in \{0, 1\}$：t_k 时刻点 $\boldsymbol{P}_i$ 是否被三弹“至少其一”遮蔽；

$\chi_k \in \{0, 1\}$：t_k 时刻“完全遮蔽”指示（全部圆柱点位均被遮蔽时 $\chi_k = 1$）。

6.1.2 优化目标

目标为仍然为离散近似的最大化总体有效干扰时长，设决策向量为

$$\boldsymbol{x} = \big(\psi, V_f, t_{\mathrm{ys},1}, t_{\mathrm{ys},2}, t_{\mathrm{ys},3}, t_{\mathrm{yf},1}, t_{\mathrm{yf},2}, t_{\mathrm{yf},3}\big),$$

则优化目标写为

$$\max_{\boldsymbol{x}} \quad T(\boldsymbol{x}) = \sum_{k=0}^{K} \chi_k(t_k)\, \Delta t. \tag{26}$$

6.1.3 约束条件

以下是针对问题三的三弹投放问题，进行的针对性的新增的约束条件。

1. 烟幕有效期与遮蔽逻辑二阶锥（SOCP）约束

为了将“仅在有效期内烟幕才能遮蔽目标”的逻辑用 MISOCP 的形式表达，也便于后续的求解，对每个时刻 t_k、每枚烟幕 n 引入当前时刻烟幕有效指示变量 $z_{k,n} \in \{0, 1\}$，当且仅当 $t_k \in [t_{b,n},\, t_{b,n} + t_{\mathrm{fog}}]$ 时 $z_{k,n} = 1$。首先，要确保当 $z_{k,n} = 1$ 时强制 t_k 落在有效窗区

间内，此指示才有意义；其中 M 是足够大的常数 (取 10^6)，约束如下

$$\begin{aligned} t_k - t_{b,n} \;&\geq\; -M\,(1 - z_{k,n}), \quad \forall k, n, \\ t_k - (t_{b,n} + t_{\mathrm{fog}}) \;&\leq\; M\,(1 - z_{k,n}), \qquad \forall k, n, \end{aligned}$$

再考虑遮蔽单时刻遮蔽判断的 SOCP 改写与等效性，当 $X_{i,k,n} = 1$，则强制 $\delta_{i,k,n} \leq R_l$；若 $X_{i,k,n} = 0$，该约束自动放宽。其中 $\delta_{i,k,n}$ 表示点 $\boldsymbol{P}_i$ 与第 n 枚烟幕在 t_k 时刻的最小距离，R_l 为烟幕半径，定义与计算同问题一，约束如下

$$\delta_{i,k,n} \;\leq\; R_l + M(1 - X_{i,k,n}), \qquad \forall i, k, n,$$

还要保证只有在有效期 ($z_{k,n} = 1$) 内，$X_{i,k,n}$ 才能取 1；若 $z_{k,n} = 0$，则强制 $X_{i,k,n} = 0$。

$$X_{i,k,n} \;\leq\; z_{k,n}, \qquad \forall i, k, n,$$

同时，要对三弹“至少其一”遮蔽派生量 $u_{i,k}$ 进行约束，不可以存在没有一弹遮蔽的的情况下将 $u_{i,k}$ 置 1

$$u_{i,k} \;\geq\; X_{i,k,n}, \qquad \forall i, k, n,$$

另外要控制 $\chi_k = 1$ 的条件，当所有点都被遮蔽时才实现“完全遮蔽”。

$$\chi_k \;\leq\; u_{i,k}, \qquad \forall i, k,$$

最后再规定上述变量类型均为二元变量

$$X_{i,k,n},\; z_{k,n},\; u_{i,k},\; \chi_k \in \{0, 1\}.$$

综上，$z_{k,n}$ 控制时间窗，$X_{i,k,n}$ 刻画点位是否在烟幕内，$u_{i,k}$ 聚合单点的多弹遮蔽结果，最后 χ_k 再聚合成“目标整体是否完全遮蔽”。其中 $\delta_{i,k,n}$ 的最小距离计算及其判定即为二阶锥约束（关于 SOCC 应用的解释与时间离散化对于 SOCC 的具体处理，详见附录[3][4]），与上述整数逻辑结合后，整体即为 MISOCP 模型的一部分约束条件。

2. 最小 1 秒起爆间隔约束

$$t_{b,2} - t_{b,1} \geq 1, \qquad t_{b,3} - t_{b,2} \geq 1. \tag{27}$$

3. 多枚烟幕弹的时间与高度有效性约束

将问题二的单弹的起爆不触地、烟幕下沉不触地以及在导弹 M1 到达假目标前的约束，拓展为每枚的时间—高度可行域：

$$0 < t_{\mathrm{yf},n} < 18.85, \quad t_{\mathrm{ys},n} + t_{\mathrm{yf},n} < 67.0, \quad n = 1, 2, 3. \tag{28}$$

6.1.4 完整模型

通过以上分析，得到单机三弹的遮蔽时间的 MISOCP 最优化模型（为节省篇幅，与问题二重复的航向角、飞行速度、投放时间以及引信延时的决策变量定义域约束（参见式(17)）、无人机/导弹/烟幕弹/烟幕运动学约束（参见式(22)）、常量约束（参见式(23)）、圆柱离散化约束（包括离散化坐标与视线与中心距离，参见式(25)和(24)），以上约束有实际意义；以下展现遮蔽判定与新增约束）

$$\max_{\boldsymbol{x}} \quad T(\boldsymbol{x}) = \sum_{k=0}^{K} \chi_k(t_k)\,\Delta t, \boldsymbol{x} = \left(\psi, V_f, t_{\mathrm{ys},1}, t_{\mathrm{ys},2}, t_{\mathrm{ys},3}, t_{\mathrm{yf},1}, t_{\mathrm{yf},2}, t_{\mathrm{yf},3}\right) \tag{29}$$

$$s.t. \begin{cases} \chi(t) = \prod_{i=1}^{N}\left(1 - X_i(t)\right) & \text{(单时刻遮蔽)} \\ X_i(t) = \left\lfloor \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan(k\delta_i(t) - kR_l)\right)\right\rfloor, \quad k \to \infty & \text{(单视线遮蔽)} \\ t_k - t_{b,n} \;\geq\; -M\,(1 - z_{k,n}), \quad \forall k, n & \text{(时间窗下界)} \\ t_k - (t_{b,n} + t_{\mathrm{fog}}) \;\leq\; M\,(1 - z_{k,n}), \quad \forall k, n & \text{(时间窗上界)} \\ \delta_{i,k,n} \;\leq\; R_l + M\,(1 - X_{i,k,n}), \quad \forall i, k, n & \text{(遮蔽约束)} \\ X_{i,k,n} \;\leq\; z_{k,n}, \quad \forall i, k, n & \text{(有效期联动)} \\ u_{i,k} \;\geq\; X_{i,k,n}, \quad \forall i, k, n & \text{(单点多弹聚合)} \\ \chi_k \;\leq\; u_{i,k}, \quad \forall i, k & \text{(全目标完全遮蔽)} \\ X_{i,k,n},\; z_{k,n},\; u_{i,k},\; \chi_k \in \{0, 1\} & \text{(二元约束)} \\ t_{b,2} - t_{b,1} \;\geq\; 1, \qquad t_{b,3} - t_{b,2} \;\geq\; 1. & \text{(最小起爆间隔)} \\ 0 < t_{\mathrm{yf},n} < 18.85, \qquad t_{\mathrm{ys},n} + t_{\mathrm{yf},n} < 67.0, \quad n = 1, 2, 3 & \text{(时间—高度可行域)} \end{cases} \tag{30}$$

6.2 模型求解与结果

针对“单机三弹”问题，我们借用了问题 2 算法中的遮蔽判定机制与多进程并行搜索，对每一组航向与速度并行地执行三弹贪心排程，收集各自的联合遮蔽时长与详细方案。最后选取最优解并记录其航向角、速度、联合遮蔽区间以及三枚烟幕弹的具体参数与评估信息。

三弹联合遮蔽贪心排程算法步骤

Step1 场景仿真构建

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step1。

Step2 假目标及无人机参量的离散化

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step2。

三弹联合遮蔽贪心排程算法步骤（续）

Step3 多层级烟幕遮蔽判定策略

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step3。

Step4 固定航向与速度下的三弹贪心排程

在给定的航向角与飞行速度下，进行贪心搜索：第一枚以自身遮蔽时长最大为目标；第二与第三枚在满足投放至少间隔一秒的约束下，以最大化对当前联合遮蔽的增量贡献为目标；每次确定一枚后，更新联合区间并保存该弹的自身时长、增量贡献、投放时间与引信延时参数、对应的遮蔽区间以及位置信息；最终得到该对航向与速度的联合区间与总时长。

Step5 早停机制与多进程并行搜索

首先快速判定输入的航向角、飞行速度、投放时刻与引信延时是否存在越界异常；再计算投放点与起爆点，并在稀疏采样上进行早停预判。多进程搜索实现参照问题 2 视锥全遮蔽判定与并行搜索算法 Step4。

经过以上求解步骤，利用原始模型的圆柱离散遮蔽判定，视线段条数为 35 条，时间离散的粗略步长为 0.02s，精细步长为 0.001s，求解所得**总遮蔽时长约为 6.712 s**。虽然单枚烟幕弹自身遮蔽时长分别达到 2.952s，2.672s，1.297s，由于区间重叠导致有 0.206s 损耗，但有效遮蔽区间连续，实现了烟雾覆盖的连续性。同时投放时刻间隔大于 1s，满足题目要求，

图 7: 投放策略结果展示

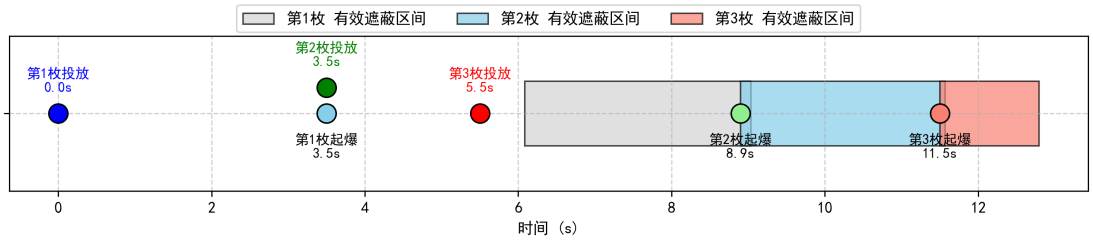

表 7: 圆柱离散求解方式

参数	最优联合遮蔽总时长	航向角	飞行速度	三枚烟幕弹遮蔽区间并集
数值	6.712 s	180.00°	140.00 m/s	[6.084, 12.796]

参数	**第 1 枚**	**第 2 枚**	**第 3 枚**
自身遮蔽时长	2.952 s	2.672 s	1.297 s
对总体新增贡献	2.951 s	2.536 s	1.225 s
投放时刻	0.00 s	3.50 s	5.50 s
引信延时	3.50 s	5.40 s	6.00 s
起爆时刻	3.50 s	8.90 s	11.50 s
投放点	[17800, 0, 1800]	[17310, 0, 1800]	[17030, 0, 1800]
起爆点	[17310, 0, 1739.914]	[16554, 0, 1656.97]	[16190, 0, 1623.42]
遮蔽区间	[6.084, 9.035]	[8.900, 11.571]	[11.500, 12.796]

同时利用真目标的圆柱外接球近似解析遮蔽判定求解，不需要对圆柱进行离散化，通过立体几何的解析关系判定遮蔽（详见附录），得到的遮蔽总时长为 6.656 s，相差仅为 0.83%，两者在决策向量上差别不大（航向角、速度、投放时刻一致，引信延时略微不同）。通过不同遮蔽建模方法的对比，验证了所得到的最优解在几何近似变化下仍然稳定，说明最优化结果具有鲁棒性与合理性。

七　问题四：多机单弹的遮蔽时长最优化模型

在问题四中，给定 3 架无人机 FY1, FY2, FY3（各投放 1 枚烟幕干扰弹）对导弹 M1 实施干扰，目标是在导弹飞向假目标的过程中，使真目标被 “完全遮蔽” 的总时长最大。该情形可视作问题二（单机单弹）的并行扩展，并继续沿用问题三的 MISOCP 框架[3]。多架无人机的航向、速度及各自烟幕的投放与起爆时刻共同决定 3 个烟幕云团的时空分布；任一时刻若真目标圆柱离散点集 $\mathcal{S}$ 的每一个离散点都至少被其中一枚当前仍有效的烟幕遮挡，则记为 “完全遮蔽”。[4]

7.1　模型建立

7.1.1　模型参量定义

设 3 架无人机索引 $m = 1, 2, 3$；离散视线（目标点）索引 $i = 1, \ldots, N$；再仿照问题三的时间离散化，时间网格索引仍然设为 $k = 0, \ldots, K-1$（$t_k = k\Delta t$，$K\Delta t < \frac{\|\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0\|}{V_m}$）。

1. 决策变量

第 m 架无人机匀速水平航向角、匀速飞行速度、烟幕弹投放时刻、引信延时分别为 $\psi_m \in [0, 2\pi)$、$V_{f,m} \in [70, 140]$ 、$t_{\mathrm{ys},m}, t_{\mathrm{yf},m} > 0$。

2. 派生量

起爆时刻 $t_{b,m} = t_{\mathrm{ys},m} + t_{\mathrm{yf},m}$；烟幕有效窗 $[t_{b,m},\ t_{b,m} + t_{\mathrm{fog}}]$。以及对于 M1 导弹的单时刻遮蔽指示量 $\chi_k(t)$、各无人机 m 对离散点 $\boldsymbol{P}_i$ 的单视线遮蔽指示量 $X_{i,k,m}(t)$，以及三弹是否存在任一有效遮蔽指示量 $z_{i,k,m}(t)$。

7.1.2　优化目标

定义问题四的决策向量为

$$\boldsymbol{y} = (\psi_1, \psi_2, \psi_3,\ V_{f,1}, V_{f,2}, V_{f,3},\ t_{\mathrm{ys},1}, t_{\mathrm{ys},2}, t_{\mathrm{ys},3},\ t_{\mathrm{yf},1}, t_{\mathrm{yf},2}, t_{\mathrm{yf},3})$$

优化目标仍然为最大化总体完全遮蔽时长：

$$\max_{\boldsymbol{y}}\ T(\boldsymbol{y}) = \sum_{k=0}^{K-1} \chi_k\, \Delta t \tag{31}$$

7.1.3 约束条件

1. 多无人机的烟幕有效期与遮蔽逻辑二阶锥（SOCP）约束

对于每一架无人机，仿照问题三的约束方式，进行关于多无人机的遮蔽判定的约束改写；仍然设 R_l 为遮蔽半径；二元变量：$z_{k,m}$（有效期）、$X_{i,k,m}$（遮蔽点）、$u_{i,k}$（点被至少一枚遮蔽）、χ_k（完全遮蔽）。仍使用 Big-M 法进行约束，约束条件如下

$$t_k - t_{b,m} \geq -M(1 - z_{k,m}), t_k - (t_{b,m} + t_{\text{fog}}) \leq M(1 - z_{k,m}), (\text{有效窗约束}) \tag{32}$$

$$\delta_{i,k,m} \leq R_l + M(1 - X_{i,k,m}), X_{i,k,m} \leq z_{k,m}, (\text{SOCC 最短距离约束与有效期约束}) \tag{33}$$

$$u_{i,k} \geq X_{i,k,m},\ u_{i,k} \leq \sum_{m=1}^{3} X_{i,k,m}, (\text{单视线三弹其一遮蔽合理性约束}) \tag{34}$$

$$\chi_k \leq u_{i,k},\ \chi_k \geq \sum_{i=1}^{N} u_{i,k} - (N-1), (\text{单时刻三弹其一遮蔽合理性约束}) \tag{35}$$

$$\sum_{k=0}^{K-1} \chi_k \geq 1, (\text{全时刻步进至少一次完全遮蔽约束}) \tag{36}$$

$$X_{i,k,m}, z_{k,m}, u_{i,k}, \chi_k \in \{0,1\}, (\text{二元变量定义域约束}) \tag{37}$$

2. 起爆前导弹仍在飞行约束

$$t_{b,m} = t_{\text{ys},m} + t_{\text{yf},m} < \frac{\|\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0\|}{V_m} \approx 67.0\ \text{s}, \qquad m = 1,2,3.$$

3. 烟幕弹下落不触地与烟幕下沉不触地约束

与问题二同理，对于不同无人机的每枚烟幕，要控制起爆时间，确保起爆及其烟幕的有效性：

$$0 < t_{\text{yf},m} < \min\left(\sqrt{\frac{2\,FY_{m0z}}{g}},\ \sqrt{\frac{2(FY_{m0z} - V_{\text{fog}}t_{\text{fog}})}{g}}\right), \qquad m = 1,2,3. \tag{38}$$

4. 无人机坐标常量约束

考虑多无人机时，要引入另外两个无人机的题设坐标作为常量的约束，作为式(23)已有常量约束的补充，约束如下式

$$\boldsymbol{FY2}(0) = (12000, 1400, 1400), \boldsymbol{FY3}(0) = (6000, -3000, 700) \tag{39}$$

7.1.4 完整模型

单目标最优化目标仍然为遮蔽时长，完整模型如下（为节省篇幅，与问题一、二、三重复约束不再展示，包括了航向角、飞行速度、投放时间以及引信延时的决策变量定义域

约束（参见式(17)）、无人机/导弹/烟幕弹/烟幕运动学约束（参见式(22)）、常量约束（参见式(23)）、圆柱离散化约束（包括离散化坐标与视线与中心距离，参见式(25)和(24)），最小 1 秒起爆间隔约束（参见式(27)），以上约束有实际意义；以下展现遮蔽判定与新增约束）

$$\max_{\boldsymbol{y}} T(\boldsymbol{y}) = \sum_{k=0}^{K-1} \chi_k \, \Delta t, \boldsymbol{y} = (\psi_1, \psi_2, \psi_3,\ V_{f,1}, V_{f,2}, V_{f,3},\ t_{\mathrm{ys},1}, t_{\mathrm{ys},2}, t_{\mathrm{ys},3},\ t_{\mathrm{yf},1}, t_{\mathrm{yf},2}, t_{\mathrm{yf},3}) \quad (40)$$

$$s.t. \begin{cases} \chi(t) = \prod_{i=1}^{N} (1 - X_i(t)) & \text{(单时刻遮蔽)} \\ X_i(t) = \left\lfloor \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan(k\delta_i(t) - kR_l)\right) \right\rfloor,\ k \to \infty & \text{(单视线遮蔽)} \\ -M(1 - z_{k,m}) \le t_k - t_{b,m} \le M(1 - z_{k,m}) + t_{\mathrm{fog}}, \forall k, m & \text{(有效窗约束)} \\ \delta_{i,k,m} \le R_l + M(1 - X_{i,k,m}), \quad X_{i,k,m} \le z_{k,m}, \forall i, k, m & \text{(SOCC 最短距离约束)} \\ X_{i,k,m} \le z_{k,m}, \forall i, k, m & \text{(有效期约束)} \\ X_{i,k,m} \le u_{i,k} \le \sum_{m=1}^{3} X_{i,k,m}, \forall i, k & \text{(单点聚合约束)} \\ \sum_{i=1}^{N} u_{i,k} - (N-1) \le \chi_k \le u_{i,k}, \forall i, k & \text{(完全遮蔽逻辑)} \\ \sum_{k=0}^{K-1} \chi_k \ge 1 & \text{(至少一次完全遮蔽)} \\ X_{i,k,m},\ z_{k,m},\ u_{i,k},\ \chi_k \in \{0, 1\} & \text{(二元变量定义域)} \\ t_{b,m} = t_{\mathrm{ys,m}} + t_{\mathrm{yf,m}} < t_{\mathrm{hit}} \approx 67.0s, m = 1, 2, 3 & \text{(起爆早于命中)} \\ 0 < t_{\mathrm{yf,m}} < \min\left(\sqrt{\frac{2FY_{m0z}}{g}},\ \sqrt{\frac{2(FY_{m0z} - V_{\mathrm{fog}} t_{\mathrm{fog}})}{g}} \right), & \text{(下落与下沉不触地)} \\ \boldsymbol{FY2}(0) = (12000, 1400, 1400),\ \boldsymbol{FY3}(0) = (6000, -3000, 700) & \text{(无人机坐标补充)} \end{cases}$$

7.2 模型的求解与结果

针对“三机一弹”问题，我们同样借用了问题 2 算法中的遮蔽判定机制与多进程并行搜索，收集各自的联合遮蔽时长与详细方案，储存在小顶堆中。再将三个无人机的 Top-K 方案进行三重笛卡尔组合，以“并集总时长”为最终目标再进行搜索。

三机联合遮蔽算法步骤

Step1 场景仿真构建

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step1。

Step2 假目标及无人机参量的离散化

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step2。

Step3 多层级烟幕遮蔽判定策略

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step3。

Step4 三机联合多进程并行搜索

从三架无人机各自小顶堆中取前若干条结果，进行三重笛卡尔组合。对每一组组合，计算其遮蔽区间的并集长度与并集区间。以“并集总时长”为最轴目标进行全表搜索。其余实现参照问题 3 三弹联合遮蔽贪心排程算法 Step5。

通过以上求解，解得 11.630 秒的总遮蔽时长，其中 3 枚烟幕弹的重叠时间为 0.03s，具体结果如下表

表 9: FY1–FY3 单弹结果汇总表

指标	FY1	FY2	FY3
单独有效遮蔽时长 (s)	4.580	3.970	3.110
飞行角度 (deg / rad)	5.00 / 0.0873	-63.00 / -1.0996	74.00 / 1.2915
飞行速度 (m/s)	79.67	138.67	126.33
投放时刻 (s)	0.80	6.00	24.40
引信延时 (s)	0.60	5.00	1.00
投放点	[17863.49, 5.55, 1800.00]	[12377.72, 658.68, 1400.00]	[6849.66, -36.88, 700.00]
起爆时刻 (s)	1.400	11.000	25.400
起爆点	[17911.11, 9.72, 1798.23]	[12692.49, 40.92, 1277.38]	[6884.48, 84.56, 695.10]
遮蔽区间	[1.560, 6.130]	[11.500, 15.460]	[25.440, 28.540]
遮蔽区间时长 (s)	4.570	3.960	3.100

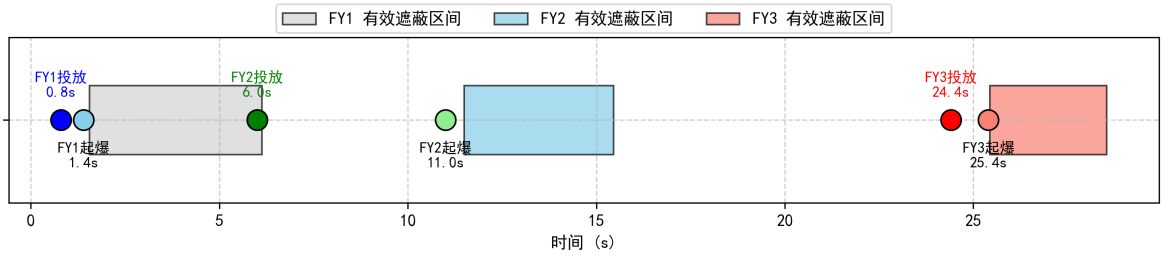

图 8: 投放策略结果展示

八　问题五: 多机多弹的遮蔽时长最优化模型

针对问题五，在前四问建模框架基础上，同时考虑 5 架无人机（至多各投放 3 枚）与 3 枚来袭导弹 $M1, M2, M3$ 的协同遮蔽调度。目标是在导弹飞向假目标的整个飞行时间窗内，通过合理规划各无人机的**航向、速度、每枚干扰弹的投放时刻与引信延时（起爆时刻）**，使针对各导弹的真目标“完全遮蔽”时间指标达到既可追求总遮蔽时长最大化[4]。

8.1　模型建立

8.1.1　模型参量定义

定义无人机索引 $\mathcal{U} = \{1, 2, 3, 4, 5\}$；导弹索引 $\mathcal{M} = \{1, 2, 3\}$；第 u 架无人机的干扰弹序号 $\mathcal{B}_u = \{1, 2, 3\}$。

1. 决策变量

各无人机的航向角：$\psi_u \in [0, 2\pi)$，$u \in \mathcal{U}$、飞行速度：$V_{f,u} \in [70, 140]$，$u \in \mathcal{U}$ 以及实际使用弹数：$n_u \in \{0, 1, 2, 3\}$。各无人机的各烟幕弹投放时刻：$t_{\mathrm{ys,u,b}} \geq 0$，$u \in \mathcal{U}$，$b \in \mathcal{B}_u$ 与引信延时：$t_{\mathrm{yf,u,b}} > 0$，$u \in \mathcal{U}$，$b \in \mathcal{B}_u$

2. 派生量

起爆时刻 $t_{u,b} = t_{\mathrm{ys,u,b}} + t_{\mathrm{yf,u,b}}$；起爆点/投放点/烟团中心轨迹坐标由运动学方程确定；各导弹遮蔽累计时长 $T_m = \sum_k \chi_{k,m} \Delta t$ 为结果量。

8.1.2　优化目标

先将所有的决策变量，构建为一个决策向量 $\boldsymbol{z}$，定义为

$$z = (\underbrace{\psi_1, \ldots, \psi_5}_{\text{各机航向角}}, \underbrace{V_{f,1}, \ldots, V_{f,5}}_{\text{各机速度}}, \underbrace{n_1, \ldots, n_5}_{\text{各机使用弹数}}, \underbrace{t_{\mathrm{ys},1,1}, \ldots, t_{\mathrm{ys},5,3}}_{\text{各机各枚烟幕弹投放时刻}}, \underbrace{t_{\mathrm{yf},1,1}, \ldots, t_{\mathrm{yf},5,3}}_{\text{各机各枚引信延时}})$$

优化目标仍然为最大化总体完全遮蔽时长：

$$\max_{\boldsymbol{z}} \; T(\boldsymbol{y}) = \sum_{k=0}^{K-1} \chi_k \, \Delta t \tag{41}$$

8.1.3　约束条件

1. 导弹到达前烟幕弹起爆约束[7]

所有烟幕弹的起爆时，至少要有一个导弹还没有飞到假目标，在此情况下理论上还存在完全遮蔽的可能，经过对三个导弹的飞行时间的计算，分别为 M1:67.0s、M2:63.75s、M3:60.37s，因此取 M1 的 67.0s 作为起爆时间的上限，约束如下式

$$t_{\mathrm{ys},u,b} + t_{\mathrm{yf},u,b} < \frac{\|\boldsymbol{M}_f - \boldsymbol{M}\mathbf{1}_0\|}{V_m} \approx 67.0 \text{ s}$$

2. 导弹与无人机坐标常量约束

对于 5 架无人机干扰 3 枚导弹，已有约束式(23)与式(39)对部分无人机与导弹的初始坐标进行了约束，现补充其他初始坐标

$$\begin{aligned}\boldsymbol{FY4}(0)=(11000,2000,1800),\boldsymbol{FY5}(0)=(13000,-2000,1300),\\ \boldsymbol{M2}(0)=(19000,600,2100),\boldsymbol{M3}(0)=(18000,-600,1900)\end{aligned}\tag{42}$$

8.1.4 完整模型

单目标最优化目标仍然为遮蔽时长，完整模型如下（为节省篇幅，仿照问题三、四的模型给出方式，与上述问题重复的行方向与飞行速度定义域约束（参见式(17)）、无人机/导弹/烟幕弹/烟幕运动学约束（参见式(22)）、常量约束（参见式(23)与(39)）、圆柱离散化约束（包括离散化坐标与视线与中心距离，参见式(25)和(24)）、最小 1 秒起爆间隔约束（参见式(27)）、烟幕弹下落不触地与烟幕下沉不触地约束（参见式(38)）、多无人机的烟幕有效期与遮蔽逻辑二阶锥（SOCP）约束（参见式(32)至式(37)），以上约束有实际意义；以下展现遮蔽判定与新增约束）

$$\max_{\boldsymbol{z}}\ T(\boldsymbol{z})=\sum_{k=0}^{K-1}\chi_k\,\Delta t,\boldsymbol{z}=(\underbrace{\psi_1,\ldots,\psi_5}_{\text{各机航向角}},\underbrace{V_{f,1},\ldots,V_{f,5}}_{\text{各机速度}},\underbrace{n_1,\ldots,n_5}_{\text{各机使用弹数}},\underbrace{t_{\text{ys},1,1},\ldots,t_{\text{ys},5,3}}_{\text{各机各枚烟幕弹投放时刻}},\underbrace{t_{\text{yf},1,1},\ldots,t_{\text{yf},5,3}}_{\text{各机各枚引信延时}})\tag{43}$$

$$\text{s.t.}\begin{cases}\chi(t)=\prod_{i=1}^{N}(1-X_i(t)) & \text{(单时刻遮蔽)}\\ X_i(t)=\left\lfloor\frac{1}{2}\left(1+\frac{2}{\pi}\arctan(k\delta_i(t)-kR_i)\right)\right\rfloor,\ k\to\infty & \text{(单视线遮蔽)}\\ t_{\text{ys},u,b}-t_{\text{ys},u,b-1}\geq 1,\quad u\in\mathcal{U},\,b=2,3 & \text{(投弹间隔时间约束)}\\ t_{\text{ys},u,b}+t_{\text{yf},u,b}<\dfrac{\|\boldsymbol{M}_f-\boldsymbol{M}\mathbf{1}_0\|}{V_m}\approx 67.0\,\text{s} & \text{(起爆时间约束)}\\ X_{i,k,m}\leq z_{k,m},\quad\forall i,k,m & \text{(有效期约束)}\\ \psi_i\in[0,2\pi],\quad V_{f,i}\in[70,140], & \\ n_i\in\{0,1,2,3\},\quad t_{\text{ys},i,j}>0,\quad t_{\text{yf},i,j}>0,\quad\forall i\in\mathcal{U},\forall j\in\mathcal{M} & \text{(决策变量定义域约束)}\end{cases}$$

8.2 模型求解与结果

在模型求解中，我们借鉴了模型 1 中的视锥仿真器、模型 3 中的一机三弹贪心实现和模型 4 中的多机一弹的联合优化，再加上多个导弹的视锥，使用粗细两层网格搜索，求解出了目前最优的结果。

联合遮蔽优化求解步骤

Step1 场景仿真构建

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step1。

Step2 假目标及无人机参量的离散化

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step2。

Step3 烟幕遮蔽判定策略

参照问题 2 视锥全遮蔽判定与并行搜索算法 Step3。

Step4 固定航向与速度下的三弹贪心排程

参照问题 3 三弹联合遮蔽贪心排程算法 Step4。

Step5 五机联合多进程并行搜索

将 3 架无人机改为 5 架，其余实现参照问题 4 三机联合遮蔽算法 Step5。

表 11: 五机三弹的具体投放策略

无人机编号	航向	速度	烟幕弹编号	投放时刻/引信延迟/起爆时刻	释放坐标	爆炸坐标	遮蔽情况
FY1	180.00	138.00	1	0.00 / 3.50 / 3.50	[17800, 0, 1800]	[17317, 0, 1739.914]	M1: 2.961s
			2	4.00 / 5.50 / 9.50	[17248, 0, 1800]	[16489, 0, 1651.624]	M1: 2.293s
			3	6.00 / 6.00 / 12.00	[16972, 0, 1800]	[16144, 0, 1623.42]	M1: 0.941s
FY2	-136.35	132.00	1	5.00 / 6.00 / 11.00	[11522.479, 944.397, 1400]	[10949.454, 397.674, 1223.42]	M2: 3.723s
			2	7.00 / 8.00 / 15.00	[11331.471, 762.156, 1400]	[10567.437, 33.192, 1086.08]	M1: 3.560s
			3	11.00 / 8.00 / 19.00	[10949.454, 397.674, 1400]	[10185.421, -331.29, 1086.08]	M3: 2.631s
FY3	88.43	131.00	1	19.00 / 3.00 / 22.00	[6067.979, -511.928, 700]	[6078.713, -119.075, 655.855]	M3: 2.964s
			2	20.00 / 3.50 / 23.50	[6071.557, -380.977, 700]	[6084.08, 77.352, 639.914]	M1: 2.943s
			3	24.00 / 1.00 / 25.00	[6085.869, 142.827, 700]	[6089.446, 273.778, 695.095]	M2: 2.074s
FY4	-349.70	82.00	未投放	–	–	–	–
FY5	122.25	136.00	1	12.00 / 2.00 / 14.00	[12129.049, -619.831, 1300]	[11983.89, -389.802, 1280.38]	M3: 3.476s
			2	13.00 / 4.50 / 17.50	[12056.469, -504.816, 1300]	[11729.863, 12.747, 1200.674]	M1: 3.217s
						有效遮蔽总时长	**26.954s**

我们的结果体现了多机协同与目标侧重。在满足题目要求的约束下，FY1 专注遮蔽 M1，FY2、FY3 和 FY5 跨目标机动遮蔽，FY4 因性价比低而没有被使用。多次起爆通过时序错峰把烟雾“拼接”起来，使得三枚导弹的累计遮蔽分别约 13.493s/5.797s/7.664s，**总体有效遮蔽时间约 26.954s**。我们的结果显示无人机并没有投掷所有烟雾弹，但以较优的时空分配获得较优的效果，体现了资源受限下的调度效率与对关键目标的优先保障。

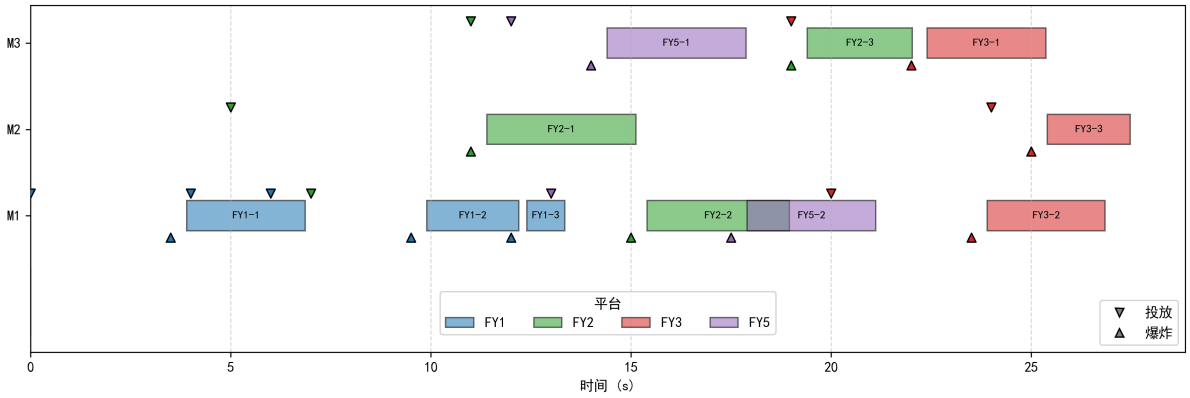

图 9: 投放策略结果展示

九　模型的评价

9.1　模型的优点

1. 物理运动与几何判定的融合

模型在问题一中，将复杂的导弹、无人机、烟幕弹的物理运动学方程与几何遮蔽判定机制深度融合，并将连续的“视锥”问题巧妙地转化为离散化点集的线段集合，实现了从连续物理现象到离散计算模型的精确映射。这种建模方法不仅确保了遮蔽判定在数学上的严谨性，也为后续所有优化问题提供了坚实、可计算的基础，使其结果具有显著的科学性和可解释性。

2. 系统的多维度优化与求解效率

模型在问题二、三、四中，创新性地构建了一个多维度的优化框架，并针对性地采用了粗细网格并行搜索和分层启发式算法。通过将无人机的航向、速度、投放与起爆时间等多个决策变量整合到统一的优化目标中，模型能够自动寻找最优解。同时，外接圆锥简化判断和多线程并行计算等策略的引入，极大地提升了求解效率，使得在面对高维、非凸的复杂优化问题时，仍能迅速获得高质量的近似最优解，兼顾了模型的精确性与实用性。

3. 复杂逻辑与数学规划的统一转化

模型在问题三、四、五中，成功地将“至少一枚烟幕弹有效”、“所有视线均被遮蔽”等多重逻辑判断和复杂的几何距离约束，转化为混合整数二阶锥规划（MISOCP）框架。该框架将离散的二元变量与连续的几何距离计算有机结合，使得原本难以求解的协同遮蔽问题，能够被纳入到成熟的数学规划工具中进行求解。这种转化不仅为解决大规模、多维度的协同调度问题提供了新的思路，也展现了数学规划在处理复杂物理逻辑方面的强大能力。

9.2　模型的缺点

1. 网格搜索算法对全局最优性的局限性

在问题二中，我们对不同分辨率与覆盖范围的网格结果进行对比，发现了目标函数存在多模态特性，即存在多个局部最优解。尽管分层网格搜索算法在计算效率上表现出色，但它容易在分辨率不高的情况下采样不到优质盆地，产生早期选择偏置，无法从理论上保证找到绝对的全局最优解。因此，在目标函数不一定严格凸性、且网格步长不趋于 0 的情况下，该方法难以给出全局最优性的理论保证。

2. 缺少对外部环境的动态适应性

尽管模型考虑了多机协同，但其核心仍基于固定的初始条件和匀速直线运动的假设进行推演。在实际应用中，来袭导弹的轨迹可能并非严格直线，无人机也可能因气流等因素发生偏离。模型缺乏对这些外部动态不确定性的实时感知与快速自适应调整机制。当实际情况与模型假设存在显著偏离时，原定的最优投放策略可能不再有效，需要人工干预或重新计算。

参考文献

[1] 尹周平, 丁汉, & 熊有伦. (2003). 基于可视锥的可接近性分析方法及其应用. 中国科学: *E* 辑, **33**(11), 979-988.

[2] 逯宏亮, & 李伟仁. (2003). 单机多目标攻击战术决策的算法研究. 弹箭与制导学报, **23**,(S6), 141-143.

[3] 别朝红, 刘凡, 王旭. 基于混合整数二阶锥规划的电-气耦合系统日前经济调度方法 [P]. *CN108846507B*, 2018-11-20.

[4] Lorenzetti J, Chen M, Landry B, et al. Reach-Avoid Games Via Mixed-Integer Second-Order Cone Programming [C]. 见: IEEE. 2018 IEEE Conference on Decision and Control (CDC). 美国: *IEEE*, 2018.

[5] DeepSeek, DeepSeek-R1-0528, 深度求索（DeepSeek）, 2025-09-05.

[6] 通义千问, qwen3-coder-plus-2025-07-22, 阿里云, 2025-09-05.

[7] Kimi, kimi-k2-0905-preview, 月之暗面（Moonshot）, 2025-09-05.

附录 A　其他求解结果与推导补充

A.1 问题三的目标圆柱外接球简化遮蔽判断的求解结果

此求解采用了更为保守但解析性更强的遮蔽判定方法，与原圆柱离散化的判定不同，该方法将半径 $r = 7$ m、高 $h = 10$ m 的圆柱形真目标用其最小外接球替代，球半径为 $R_t = \sqrt{r^2 + (h/2)^2} \approx 8.6$ m，从而将三维遮蔽问题转化为纯角度覆盖判据。具体而言，从导弹视点观察，目标外接球张开的角半径为 $\alpha_t = \arctan(R_t/L)$，其中 L 为导弹到目标中心的距离；类似地，半径为 $R = 10$ m 的烟幕球相对导弹的角半径为 $\alpha_s = \arcsin(R/d_{sm})$，$d_{sm}$ 为导弹到烟球中心距离，而烟球中心偏离视轴的角度为 $\phi = \arctan(r_\perp/u)$，其中 u 为球心沿视轴的投影距离，$r_\perp$ 为垂直于视轴的径向距离。

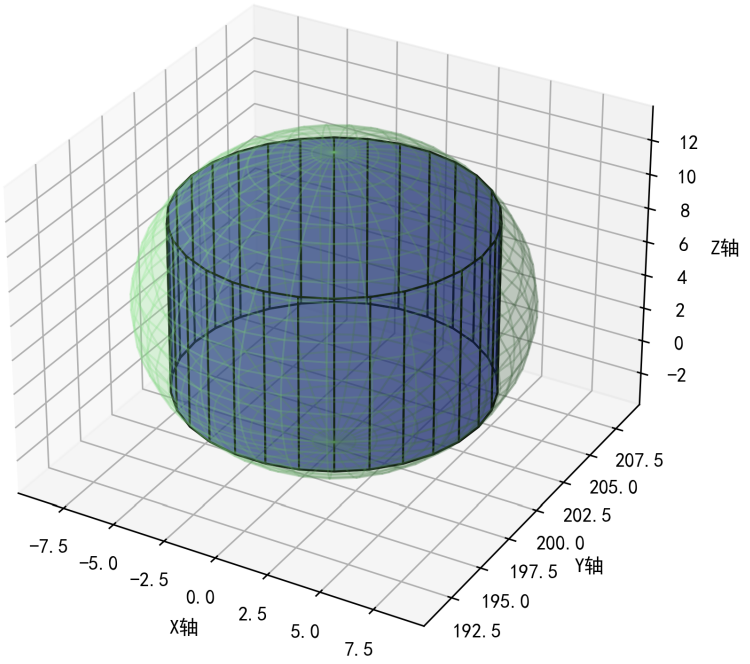

图 10: 圆柱体包络球示意

遮蔽成立的充要条件变为 $\alpha_s \geq \phi + \alpha_t$，即烟球的角域必须完全包含目标角域加上偏移角，这保证了从导弹视角看，目标的整个投影圆盘被烟幕完全覆盖，通过 arcsin、arctan 等初等函数即可完成遮蔽状态的计算。此求解结果与问题三的圆柱离散化求解基本一致，求解结果如下

表 13: 外接球近似求解方式

参数	最优联合遮蔽总时长	航向角	飞行速度	三枚烟幕弹遮蔽区间并集
数值	6.656 s	180.00°	140.00 m/s	[6.104, 12.760]

参数	第 1 枚	第 2 枚	第 3 枚
自身遮蔽时长	2.912 s	2.556 s	1.264 s
对总体新增贡献	2.908 s	2.540 s	1.208 s
投放时刻	0.00 s	3.50 s	5.50 s
引信延时	3.50 s	5.50 s	6.00 s
起爆时刻	3.50 s	9.00 s	11.50 s
投放点	[17800. 0. 1800.]	[17310. 0. 1800.]	[17030. 0. 1800.]
起爆点	[17310. 0. 1739.914]	[16540. 0. 1651.624]	[16190. 0. 1623.42]
遮蔽区间	[6.104, 9.012]	[9.000, 11.552]	[11.500, 12.760]

A.2 关于烟幕球心到视线的最短距离公式的推导

有一条线段：$\overline{\boldsymbol{MP}}$， 起点$\boldsymbol{M}$，终点$\boldsymbol{P}$，还有一个点（烟幕球心）$\boldsymbol{F}$ 要计算点 $\boldsymbol{F}$ 到线段 $\overline{\boldsymbol{MP}}$ 的最短距离。先算投影参数：

$$s = \frac{(\boldsymbol{F}-\boldsymbol{M})\cdot(\boldsymbol{P}-\boldsymbol{M})}{\|\boldsymbol{P}-\boldsymbol{M}\|^2}$$

即考虑无限延长的直线，点 $\boldsymbol{F}$ 在这条线上的投影位置用多少比例 s 来表示。当 $s=0$ 对应点 $\boldsymbol{M}$，$s=1$ 对应点 $\boldsymbol{P}$。如果投影参数 $s<0$，说明投影落在 $\boldsymbol{M}$ 外面，最短点就是 $\boldsymbol{M}$。如果 $s>1$，说明投影落在 $\boldsymbol{P}$ 外面，最短点就是 $\boldsymbol{P}$。如果 $0\le s\le 1$，说明投影正好落在线段内部，就用这个投影点。所以可以用 Clamp 截断

$$s^\star = \mathrm{clamp}(s, 0, 1)$$

Clamp 的作用就是保证投影比例不会跑到 [0,1] 之外，这样，投影点始终在线段上，而不是跑到线段外面的延长线上。把上面的步骤合起来，就得到最短距离公式：

$$\delta = \left\| \boldsymbol{F}-\boldsymbol{M}-\mathrm{clamp}\Big(\tfrac{(\boldsymbol{F}-\boldsymbol{M})\cdot(\boldsymbol{P}-\boldsymbol{M})}{\|\boldsymbol{P}-\boldsymbol{M}\|^2},\,0,\,1\Big)(\boldsymbol{P}-\boldsymbol{M}) \right\|$$

A.3 问题三、四、五的二阶锥约束（SOCC）应用解释

二阶锥规划的核心在于通过二阶锥约束描述几何或距离相关的非线性约束[3, 4]。在问题三、四、五应用的 MISOCP 模型中，二阶锥约束主要体现在遮蔽判定的距离约束上，对于每个离散点 $\boldsymbol{P}i$、时刻 t_k 和第 n 枚烟幕弹，定义 $\delta i,k,n$ 为点 $\boldsymbol{P}i$ 与第 n 枚烟幕弹在时刻 t_k 的最小距离。遮蔽条件要求 $\delta i,k,n \le R_l$（烟幕弹的等效半径）。这一约束被改写为：

$$\delta_{i,k,n} \le R_l + M(1-X_{i,k,n}), \quad \forall i,k,n$$

$\delta_{i,k,n}$ 原始计算式不妨重提一下，即为下式，其中的 $\boldsymbol{F}$ 为烟幕中心坐标，$\boldsymbol{M}$ 为导弹坐标，$\boldsymbol{P}$ 圆柱上待遮蔽离散点

$$\delta = \left\| \boldsymbol{F} - \boldsymbol{M} - \mathrm{clamp}\Big(\tfrac{(\boldsymbol{F}-\boldsymbol{M})\cdot(\boldsymbol{P}-\boldsymbol{M})}{\|\boldsymbol{P}-\boldsymbol{M}\|^2},\, 0,\, 1 \Big)(\boldsymbol{P}-\boldsymbol{M}) \right\| \tag{44}$$

可以将此改写为更加明显的 SOCC 形式[3]

$$\|\boldsymbol{v}_{i,k,n}\|_2 \le R_l, \quad \text{其中} \quad \boldsymbol{v}_{i,k,n} = \boldsymbol{F}_n(t_k) - \boldsymbol{M}\mathbf{1}(t_k) - s^*_{i,k,n}(\boldsymbol{P}_i - \boldsymbol{M}\mathbf{1}(t_k))$$

由上式即可看出，如果 $s^*_{i,k,n}$ 是已知或固定的常数，那么上述约束是一个标准的二阶锥约束，因为它直接约束了一个三维向量的欧几里得范数，但是我们回顾 $s^*_{i,k,n}$ 的定义

$$s^*_{i,k,n} = \mathrm{clamp}\left(\frac{(\boldsymbol{F}_n(t_k) - \boldsymbol{M}\mathbf{1}(t_k)) \cdot (\boldsymbol{P}_i - \boldsymbol{M}\mathbf{1}(t_k))}{\|\boldsymbol{P}_i - \boldsymbol{M}\mathbf{1}(t_k)\|^2}, 0, 1 \right).$$

然而，$s^*_{i,k,n}$ 的定义引入了非线性，其中的 clamp 函数 $\max(0, \min(1, x))$ 是一个分段函数。即上述的原始定义，并不是一个标准的 SOCC[4]。

但是我们在问题三的建模开始，就将时间离散化 t_k ，以及决策变量（ψ、V_f、$t_{\mathrm{ys},n}$、$t_{\mathrm{yf},n}$）都是网格化的求解（并非解析连续求解），这些处理可以将 $s^*_{i,k,n}$ 预计算为常数，使得 $\delta_{i,k,n}$ 的计算可控，$\delta_{i,k,n} \le R_l + M(1 - X_{i,k,n}), \quad \forall i,k,n$ 成为标准的二阶锥约束。

A.4 结果有效性检验

A.4.1 变量与场景约定

真目标（竖直圆柱）底面圆心 $\boldsymbol{c} = (0, 200, 0)^\top$，半径 $7m$，高度 $10m$。圆柱侧面上任一点可参数化为

$$\boldsymbol{p}(\theta, h) = \begin{bmatrix} 7\cos\theta \\ 200 + 7\sin\theta \\ h \end{bmatrix}, \qquad \theta \in [0, 2\pi),\ h \in [0, 10]. \tag{45}$$

导弹初始位置 $\boldsymbol{m}_0 \in \mathbb{R}^3$，匀速直线朝原点运动，速度大小 $v_m > 0$。令 $\hat{\boldsymbol{d}}_m = -\frac{\boldsymbol{m}_0}{\|\boldsymbol{m}_0\|}$ 为指向原点的单位向量, 则导弹在时刻 $t \ge 0$ 的位置

$$\boldsymbol{m}(t) = \boldsymbol{m}_0 + v_m\, t\, \hat{\boldsymbol{d}}_m, \qquad t \in \big[0,\, T_{\mathrm{end}}\big], \quad T_{\mathrm{end}} = \frac{\|\boldsymbol{m}_0\|}{v_m}. \tag{46}$$

无人机初始位置 $\boldsymbol{u}_0 \in \mathbb{R}^3$，水平定高直线飞行，航向角（方位角）为 ψ（0° 沿 $+x$，逆

时针为正），速度大小 $v_u > 0$。其水平单位航向

$$\hat{\boldsymbol{b}}_u = \begin{bmatrix} \cos\psi \\ \sin\psi \\ 0 \end{bmatrix}, \qquad \boldsymbol{u}(t) = \boldsymbol{u}_0 + v_u\, t\, \hat{\boldsymbol{b}}_u. \tag{47}$$

烟幕弹与云团投掷时刻 T_d、引信延时 T_f，起爆时刻 $T_{boom} = T_{drop} + T_{delay}$，重力加速度 $g = 9.81 m/s$，云团半径 $R_s = 10m$，起爆后云团中心以速度 $v_s = 3m/s$ 沿竖直方向匀速下沉，有效遮蔽持续 $T_s = 20s$。

A.4.2 起爆位置的检验

投掷瞬间 $t = T_{drop}$，无人机位置在 $\boldsymbol{u}(T_d) = \boldsymbol{u}_0 + v_u\, T_{drop}\, \hat{\boldsymbol{b}}_u$。
投掷到起爆期间的抛体（仅竖直方向受重力，初始竖直速度为 0）：

$$\boldsymbol{b}(t) = \boldsymbol{u}(T_{drop}) + v_u\,(t - T_{drop})\,\hat{\boldsymbol{b}}_u - \frac{1}{2}\, g\,(t - T_{drop})^2\, \hat{\boldsymbol{e}}_z, \qquad t \in [T_{drop},\, T_{boom}], \tag{48}$$

其中 $\hat{\boldsymbol{e}}_z = (0, 0, 1)^\top$。于是起爆位，也就是云团初始中心，在

$$\boldsymbol{s}_0 = \boldsymbol{b}(T_{boom}) = \boldsymbol{u}(T_d) + v_u\, T_f\, \hat{\boldsymbol{b}}_u - \frac{1}{2} g\, T_{delay}^2\, \hat{\boldsymbol{e}}_z. \tag{49}$$

起爆后、在有效期内云团中心坐标随时间变化为

$$\boldsymbol{s}(t) = \boldsymbol{s}_0 - v_s\,(t - T_{boom})\,\hat{\boldsymbol{e}}_z, \qquad t \in [T_{boom},\, T_{boom} + T_s]. \tag{50}$$

A.4.3 遮蔽的几何检验

在给定时刻 t，对圆柱侧面采样点集合 $\{\boldsymbol{p}_k\}_{k=1}^{N}$，对每个采样点考察线段

$$\overline{\boldsymbol{m}(t)\boldsymbol{p}_k} = \{\boldsymbol{\ell}(\lambda) = \boldsymbol{m}(t) + \lambda\,[\boldsymbol{p}_k - \boldsymbol{m}(t)],\ \lambda \in [0, 1]\}.$$

云团为球体 $\mathcal{S}'(t) = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{s}(t)\| \le R_s\}$。线段与球体是否相交可等价为：

1. 先计算线段到球心的最近点参数

$$\lambda^\star = \mathrm{clip}_{[0,1]}\left(\frac{(\boldsymbol{s}(t) - \boldsymbol{m}(t)) \cdot (\boldsymbol{p}_k - \boldsymbol{m}(t))}{\|\boldsymbol{p}_k - \boldsymbol{m}(t)\|^2} \right), \tag{51}$$

   其中 $\mathrm{clip}_{[0,1]}(x) = \min(1, \max(0, x))$；

2. 最近点与球心间距

$$d_k(t) = \big\|\boldsymbol{m}(t) + \lambda^\star\,[\boldsymbol{p}_k - \boldsymbol{m}(t)] - \boldsymbol{s}(t)\big\|. \tag{52}$$

3. 若 $d_k(t) \le R_s$，则判为该采样点视线被遮蔽。

所以可以求得有效遮蔽的条件为

$$\chi(t) = \prod_{k=1}^{N} \mathbf{1}\{d_k(t) > R_s\}. \tag{53}$$

A.4.4 时间有效性检验

云团仅在 $t \in [T_{boom},\, T_{boom} + T_s]$ 且导弹仍在飞向原点的时间窗 $t \in [0,\, T_{\text{end}}]$，$T_{\text{end}} = \frac{\|\boldsymbol{m}_0\|}{v_m}$ 内才可能产生遮蔽。所以只需要检查输出结果的时间是否包含在区间

$$\mathcal{T} = \left[\ \max(0, T_{boom}),\ \min(T_{boom} + T_s,\, T_{\text{end}})\ \right]$$

中即可。

附录 B　论文主要代码

Listing 1: 问题一的 Python 代码

```python
import numpy as np
from tqdm import tqdm

R = 7.0
H = 10.0
CY_CX = 0.0
CY_CY = 200.0
g = 9.81
v_missile = 300.0
v_uav = 120.0
t_drop = 1.5
t_burst_after_drop = 3.6
t_burst = t_drop + t_burst_after_drop
sink_v = 3.0
r_eff = 10.0
dt = 0.001
T_effect = 20.0

VIEW_ELEV_DEG = 5
VIEW_AZIM_DEG = -80

M1_0 = np.array([20000.0, 0.0, 2000.0], dtype=float)
FY1_0 = np.array([17800.0, 0.0, 1800.0], dtype=float)

u_m = -M1_0 / np.linalg.norm(M1_0)

u_uav_xy = -np.array([FY1_0[0], FY1_0[1]], dtype=float)
u_uav_xy = u_uav_xy / np.linalg.norm(u_uav_xy)
u_uav = np.array([u_uav_xy[0], u_uav_xy[1], 0.0], dtype=float)

FY1_pos_drop = FY1_0 + u_uav * v_uav * t_drop
proj_pos0 = FY1_pos_drop.copy()
proj_vel0 = u_uav * v_uav

t_fly = t_burst_after_drop
cloud0 = proj_pos0 + proj_vel0 * t_fly + np.array([0.0, 0.0, -0.5 * g * t_fly * t_fly], dtype=float
    )

deg2rad = np.pi / 180.0

thetas_side = np.arange(0.0, 360.0, 5.0) * deg2rad
zs_side = np.arange(0.0, H + 1e-9, 1.0)
TT, ZZ = np.meshgrid(thetas_side, zs_side, indexing='xy')
X_side = np.stack([
```

```
    CY_CX + R * np.cos(TT).ravel(),
    CY_CY + R * np.sin(TT).ravel(),
    ZZ.ravel()
], axis=1)

thetas_disk = np.arange(0.0, 360.0, 2.0) * deg2rad
rhos_disk = np.arange(0.0, R + 1e-9, 1.0)
RR, TTd = np.meshgrid(rhos_disk, thetas_disk, indexing='xy')
disk_xy = np.stack([
    CY_CX + RR.ravel() * np.cos(TTd).ravel(),
    CY_CY + RR.ravel() * np.sin(TTd).ravel()
], axis=1)

X_top = np.concatenate([disk_xy, np.full((disk_xy.shape[0], 1), H)], axis=1)
X_bot = np.concatenate([disk_xy, np.zeros((disk_xy.shape[0], 1))], axis=1)

def distances_point_to_segments(S, P, Xs):
    V = Xs - P[None, :]
    SP = S[None, :] - P[None, :]
    denom = np.einsum('ij,ij->i', V, V) + 1e-12
    t = np.einsum('ij,ij->i', SP, V) / denom
    t = np.clip(t, 0.0, 1.0)
    closest = P[None, :] + t[:, None] * V
    diff = closest - S[None, :]
    d2 = np.einsum('ij,ij->i', diff, diff)
    return d2

t0 = t_burst
t1 = t_burst + T_effect
t_grid = np.arange(t0, t1 + 1e-9, dt)

mask_occluded = []

for t in tqdm(t_grid):
    M1_t = M1_0 + u_m * v_missile * t
    cloud_t = cloud0 + np.array([0.0, 0.0, -sink_v * (t - t_burst)], dtype=float)
    thr2 = r_eff * r_eff
    all_blocked = True
    d2s = distances_point_to_segments(cloud_t, M1_t, X_side)
    if not np.all(d2s <= thr2):
        all_blocked = False
    else:
        d2s = distances_point_to_segments(cloud_t, M1_t, X_top)
        if not np.all(d2s <= thr2):
            all_blocked = False
        else:
            d2s = distances_point_to_segments(cloud_t, M1_t, X_bot)
            if not np.all(d2s <= thr2):
```

```
            all_blocked = False
    mask_occluded.append(all_blocked)

total_time = np.sum(np.array(mask_occluded, dtype=float)) * dt

intervals = []
in_cov = False
start_t = None
for i, oc in enumerate(mask_occluded):
    if oc and not in_cov:
        in_cov = True
        start_t = t_grid[i]
    elif (not oc) and in_cov:
        intervals.append((start_t, t_grid[i]))
        in_cov = False
if in_cov:
    intervals.append((start_t, t_grid[-1] + dt))

print(f"导弹初始位置 {M1_0}")
print(f"无人机初始位置 {FY1_0}")
print(f"投放时刻 T_drop = {t_drop:.3f} s, 起爆时刻 T_boom = {t_burst:.3f} s")
print(f"投放点 {FY1_pos_drop}")
print(f"起爆点 {cloud0}")
print(f"云团有效时段: [{t0:.3f}, {t1:.3f}] s, 步长 dt = {dt:.3f} s")

if intervals:
    print("遮蔽时间区间: ")
    for a, b in intervals:
        print(f"[{a:.3f}, {b:.3f}] s, 时长 {b-a:.3f} s")
print(f"有效遮蔽总时长: {total_time:.3f} s")
```

Listing 2: 问题二的 Python 代码

```
"""
CPU: i5-12600KF 10核16线程
RAM: 32GB
系统: Debian12.8 amd64
Python: 3.11.13
运行时间: 约3分钟
代码是多线程的, 所以会占用较多CPU资源。
"""

import os
import heapq
import itertools
import numpy as np
import multiprocessing
from tqdm import tqdm
from numba import njit
```

```python
from concurrent.futures import ProcessPoolExecutor


@njit(fastmath=True)
def _segment_sphere_distance(m, p, c):
    d = p - m
    denom = d[0]*d[0] + d[1]*d[1] + d[2]*d[2]
    if denom < 1e-12:
        # 退化为点
        dx = m[0]-c[0]; dy = m[1]-c[1]; dz = m[2]-c[2]
        return np.sqrt(dx*dx + dy*dy + dz*dz)
    # t = ((c - m) · d)/(d · d)
    t = ((c[0]-m[0])*d[0] + (c[1]-m[1])*d[1] + (c[2]-m[2])*d[2]) / denom
    if t < 0.0:
        cx = m[0]; cy = m[1]; cz = m[2]
    elif t > 1.0:
        cx = p[0]; cy = p[1]; cz = p[2]
    else:
        cx = m[0] + t*d[0]
        cy = m[1] + t*d[1]
        cz = m[2] + t*d[2]
    dx = cx - c[0]; dy = cy - c[1]; dz = cz - c[2]
    return np.sqrt(dx*dx + dy*dy + dz*dz)

def _make_cylinder_samples(center, radius, half_h, n_theta_caps=36, include_centers=True):
    cx, cy, cz = center
    pts = []
    nms = []
    # 只采样顶/底面圆周
    thetas_cap = np.linspace(0.0, 2*np.pi, n_theta_caps, endpoint=False)
    z_top = cz + half_h
    z_bot = cz - half_h
    for th in thetas_cap:
        x = cx + radius*np.cos(th)
        y = cy + radius*np.sin(th)
        pts.append([x,y,z_top]); nms.append([0.0,0.0, 1.0])
        pts.append([x,y,z_bot]); nms.append([0.0,0.0,-1.0])
    if include_centers:
        pts.append([cx,cy,z_top]); nms.append([0.0,0.0, 1.0])
        pts.append([cx,cy,z_bot]); nms.append([0.0,0.0,-1.0])
    return np.array(pts, dtype=np.float64), np.array(nms, dtype=np.float64)

# 圆柱几何
CYL_CENTER = np.array([0.0, 200.0, 5.0], dtype=np.float64)
CYL_RADIUS = 10.0/2.0
CYL_HALF_H = 7.0/2.0
CYL_PTS, CYL_NMS = _make_cylinder_samples(
    CYL_CENTER, CYL_RADIUS, CYL_HALF_H,
```

```
)

@njit(fastmath=True)
def angle_to_unit(theta):
    return np.array([np.cos(theta), np.sin(theta)], dtype=np.float64)

@njit(fastmath=True)
def occlusion_mask_and_margin_sampled(ts,
                                      m0, m_dir, v_m,
                                      smoke_xy_x, smoke_xy_y, smoke_z0, sink, t_burst, R,
                                      cyl_pts, cyl_nms):
    n = len(ts)
    mask = np.zeros(n, dtype=np.bool_)
    margin = np.empty(n, dtype=np.float64)

    n_pts = cyl_pts.shape[0]

    for i in range(n):
        t = ts[i]
        mx = m0[0] + m_dir[0]*v_m*t
        my = m0[1] + m_dir[1]*v_m*t
        mz = m0[2] + m_dir[2]*v_m*t
        sz = smoke_z0 - sink*(t - t_burst)
        sx = smoke_xy_x
        sy = smoke_xy_y

        any_visible = False
        fully_blocked = True
        local_min_margin = 1e9

        for k in range(n_pts):
            px = cyl_pts[k,0]; py = cyl_pts[k,1]; pz = cyl_pts[k,2]
            nx = cyl_nms[k,0]; ny = cyl_nms[k,1]; nz = cyl_nms[k,2]

            vmx = mx - px
            vmy = my - py
            vmz = mz - pz
            dotnp = nx*vmx + ny*vmy + nz*vmz
            if dotnp <= 0.0:
                continue

            any_visible = True

            m_vec = np.array([mx, my, mz], dtype=np.float64)
            p_vec = np.array([px, py, pz], dtype=np.float64)
            c_vec = np.array([sx, sy, sz], dtype=np.float64)
            d = _segment_sphere_distance(m_vec, p_vec, c_vec)
            mrg = R - d
```

```
            if mrg < 0.0:
                fully_blocked = False
            if mrg < local_min_margin:
                local_min_margin = mrg
            if (not fully_blocked) and (local_min_margin < -R):
                break

        if not any_visible:
            mask[i] = False
            margin[i] = -0.05
            continue

        mask[i] = fully_blocked
        if local_min_margin > 1e8:
            local_min_margin = 0.0
        margin[i] = local_min_margin

    return mask, margin

def _extract_intervals_from_mask(ts, mask):
    idx = np.flatnonzero(mask)
    if idx.size == 0:
        return []
    splits = np.where(np.diff(idx) > 1)[0]
    starts = np.r_[0, splits + 1]
    ends = np.r_[splits, idx.size - 1]
    intervals = []
    for s, e in zip(starts, ends):
        i0 = idx[s]
        i1 = idx[e]
        t0 = ts[i0]
        t1 = ts[i1]
        intervals.append((t0, t1))
    return intervals


def simulate_occlusion(theta, v_uav, t_release, t_fuse, dt_coarse=0.10):
    g = 9.81
    v_m = 300.0
    smoke_sink = 3.0
    smoke_R = 10.0
    smoke_effect = 20.0

    # M1 导弹
    m0 = np.array([20000.0, 0.0, 2000.0], dtype=np.float64)
    O = np.array([0.0,0.0,0.0], dtype=np.float64)
    L0 = np.linalg.norm(m0 - O)
    if L0 < 1e-9:
```

```
        return 0.0, [], None, None, {}
    m_dir = (O - m0) / L0
    T_impact = L0 / v_m

    # FY1
    fy0 = np.array([17800.0, 0.0, 1800.0], dtype=np.float64)
    if (v_uav < 70.0) or (v_uav > 140.0) or (t_release < 0.0) or (t_fuse < 0.0):
        return 0.0, [], None, None, {}

    u_dir = angle_to_unit(theta)
    uav_vel = np.array([u_dir[0]*v_uav, u_dir[1]*v_uav, 0.0], dtype=np.float64)

    pos_release = fy0 + uav_vel * t_release
    t_burst = t_release + t_fuse
    if t_burst >= T_impact:
        return 0.0, [], pos_release, None, {}

    burst_pos = pos_release + uav_vel * t_fuse + np.array([0.0, 0.0, -0.5*g*t_fuse**2], dtype=np.
        float64)
    smoke_xy = burst_pos[:2].copy()
    smoke_z0 = burst_pos[2]

    t0 = t_burst
    t1 = min(t_burst + smoke_effect, T_impact)
    if t1 <= t0:
        return 0.0, [], pos_release, burst_pos, {}

    dt_fine = 0.001

    # 超粗筛
    triage_K = 5
    triage_ts = np.linspace(t0, t1, triage_K)
    mask_tri, margin_tri = occlusion_mask_and_margin_sampled(
        triage_ts, m0, m_dir, v_m,
        smoke_xy[0], smoke_xy[1], smoke_z0, smoke_sink, t_burst, smoke_R,
        CYL_PTS, CYL_NMS
    )
    if (not mask_tri.any()) and (margin_tri.max() < -0.1):
        extra = {"t_burst": t_burst, "t_end": t1, "dt": dt_coarse,
                "dt_fine": dt_fine, "early_pruned": True}
        return 0.0, [], pos_release, burst_pos, extra

    # 粗步长
    ts_coarse = np.arange(t0, t1 + 1e-12, dt_coarse, dtype=np.float64)
    mask_c, _ = occlusion_mask_and_margin_sampled(
        ts_coarse, m0, m_dir, v_m,
        smoke_xy[0], smoke_xy[1], smoke_z0, smoke_sink, t_burst, smoke_R,
        CYL_PTS, CYL_NMS
```

```
    )
    if not mask_c.any():
        extra = {"t_burst": t_burst, "t_end": t1, "dt": dt_coarse,
                 "dt_fine": dt_fine, "early_pruned": False}
        return 0.0, [], pos_release, burst_pos, extra

    coarse_intervals = _extract_intervals_from_mask(ts_coarse, mask_c)

    total_time = 0.0
    fine_intervals_global = []

    for (ca, cb) in coarse_intervals:
        fa = max(t0, ca - dt_coarse)
        fb = min(t1, cb + dt_coarse)
        ts_fine = np.arange(fa, fb + 1e-12, dt_fine, dtype=np.float64)

        mask_f, _ = occlusion_mask_and_margin_sampled(
            ts_fine, m0, m_dir, v_m,
            smoke_xy[0], smoke_xy[1], smoke_z0, smoke_sink, t_burst, smoke_R,
            CYL_PTS, CYL_NMS
        )
        if mask_f.any():
            fine_intervals = _extract_intervals_from_mask(ts_fine, mask_f)
            fine_intervals_global.extend(fine_intervals)
            total_time += mask_f.sum() * dt_fine

    if fine_intervals_global:
        fine_intervals_global.sort(key=lambda x: x[0])
        merged = []
        eps = 2*dt_fine
        cur_s, cur_e = fine_intervals_global[0]
        for (s,e) in fine_intervals_global[1:]:
            if s - cur_e <= eps:
                cur_e = max(cur_e, e)
            else:
                merged.append((cur_s, cur_e))
                cur_s, cur_e = s, e
        merged.append((cur_s, cur_e))
        fine_intervals_global = merged

    extra = {"t_burst": t_burst, "t_end": t1, "dt": dt_coarse,
             "dt_fine": dt_fine, "early_pruned": False}
    return total_time, fine_intervals_global, pos_release, burst_pos, extra

def _eval_one(args):
    theta, v, tr, tf, dt = args
    score, intervals, pos_rel, pos_bur, extra = simulate_occlusion(theta, v, tr, tf, dt)
    return (score, theta, v, tr, tf, intervals, pos_rel, pos_bur, extra)
```

```python

if __name__ == '__main__':
    multiprocessing.freeze_support() # Windows 多线程支持

    fy0_xy = np.array([17800.0, 0.0])
    base_vec = (np.array([0.0, 0.0]) - fy0_xy)
    base_theta = np.arctan2(base_vec[1], base_vec[0])

    np.random.seed(0)
    coarse_theta_grid = base_theta + np.deg2rad(np.linspace(-10, 10, 21))
    coarse_speed_grid = np.linspace(70.0, 140.0, 71)
    coarse_t_release = np.linspace(0, 50, 51)
    coarse_t_fuse = np.linspace(0, 12, 13)
    dt_coarse = 0.10

    WORKERS = os.cpu_count()
    print(f"检测到CPU核心数: {WORKERS}, 将使用 {WORKERS} 个进程并行。")
    TOPK = 5

    print("进入粗搜阶段...")
    coarse_total = len(coarse_theta_grid)*len(coarse_speed_grid)*len(coarse_t_release)*len(
        coarse_t_fuse)
    coarse_iter = (
        (theta, v, tr, tf, dt_coarse)
        for theta, v, tr, tf in itertools.product(coarse_theta_grid, coarse_speed_grid,
            coarse_t_release, coarse_t_fuse)
    )

    best_heap = []
    counter = 0
    with ProcessPoolExecutor(max_workers=WORKERS) as ex:
        for rec in tqdm(ex.map(_eval_one, coarse_iter), total=coarse_total, desc="粗搜并行",
            dynamic_ncols=True):
            counter += 1
            score = rec[0]
            if len(best_heap) < TOPK:
                heapq.heappush(best_heap, (score, counter, rec))
            else:
                if score > best_heap[0][0]:
                    heapq.heapreplace(best_heap, (score, counter, rec))

    best_list = [h[2] for h in sorted(best_heap, key=lambda x: x[0], reverse=True)]

    dt_fine = 0.01
    print("进入细化阶段, 需要细化的解的数量:", len(best_list), ", 细化精度 dt = ", dt_fine)
    refined_best = None
    topN = len(best_list)
    for k in tqdm(range(topN), desc="细化 Top-K", dynamic_ncols=True):
```

```
        _, b_theta, b_v, b_tr, b_tf, _, _, _, _ = best_list[k]
        theta_grid = b_theta + np.deg2rad(np.linspace(-2, 2, 21))
        speed_grid = np.clip(np.linspace(b_v-5, b_v+5, 11), 70, 140)
        tr_grid = np.clip(np.linspace(b_tr-2, b_tr+2, 11), 0, 60)
        tf_grid = np.clip(np.linspace(b_tf-2, b_tf+2, 11), 0, 8)

        fine_total = len(theta_grid)*len(speed_grid)*len(tr_grid)*len(tf_grid)
        fine_iter = (
            (theta, v, tr, tf, dt_fine)
            for theta, v, tr, tf in itertools.product(theta_grid, speed_grid, tr_grid, tf_grid)
        )

        with ProcessPoolExecutor(max_workers=WORKERS) as ex:
            for rec in tqdm(ex.map(_eval_one, fine_iter), total=fine_total,
                            desc=f"细化 k={k+1}/{topN}", leave=False, dynamic_ncols=True):
                if (refined_best is None) or (rec[0] > refined_best[0]):
                    refined_best = rec

    final = refined_best if refined_best is not None else best_list[0]
    best_score, best_theta, best_v, best_tr, best_tf, best_intervals, best_rel, best_bur, best_extra
        = final

    deg_heading = np.rad2deg(best_theta)
    deg_base = np.rad2deg(base_theta)
    deg_offset = np.rad2deg(((best_theta - base_theta + np.pi) % (2*np.pi)) - np.pi)

    print(f"时间步长 dt={best_extra['dt']:.3f} s")
    print(f"最优遮蔽总时长: {best_score:.3f} s")
    print(f"航向角(相对x轴): {deg_heading:.2f}°")
    print(f"飞行速度: {best_v:.2f} m/s")
    print(f"投放时间: {best_tr:.2f} s")
    print(f"投放位置: {best_rel}")
    print(f"引信延时: {best_tf:.2f} s")
    print(f"起爆时间: {best_tr} + {best_tf:.2f} s")
    print(f"起爆位置: {best_bur}")
    print()
    print("遮蔽时间区间")
    if best_intervals:
        for (a,b) in best_intervals:
            print(f"[{a:.3f}, {b:.3f}]  时长 {(b-a):.3f} s")
```

Listing 3: 问题三的 Python 代码

```
import os
import itertools
import numpy as np
import pandas as pd
from tqdm import tqdm
from numba import njit
```

```
from concurrent.futures import ProcessPoolExecutor, as_completed
import multiprocessing

@njit(fastmath=True)
def _segment_sphere_distance(m, p, c):
    d = p - m
    denom = d[0]*d[0] + d[1]*d[1] + d[2]*d[2]
    if denom < 1e-12:
        dx = m[0]-c[0]; dy = m[1]-c[1]; dz = m[2]-c[2]
        return np.sqrt(dx*dx + dy*dy + dz*dz)
    t = ((c[0]-m[0])*d[0] + (c[1]-m[1])*d[1] + (c[2]-m[2])*d[2]) / denom
    if t < 0.0:
        cx, cy, cz = m[0], m[1], m[2]
    elif t > 1.0:
        cx, cy, cz = p[0], p[1], p[2]
    else:
        cx = m[0] + t*d[0]
        cy = m[1] + t*d[1]
        cz = m[2] + t*d[2]
    dx = cx - c[0]; dy = cy - c[1]; dz = cz - c[2]
    return np.sqrt(dx*dx + dy*dy + dz*dz)

def _make_cylinder_samples(center, radius, half_h,
                        n_theta_lateral=16, n_z=6,
                        n_theta_caps=16, include_centers=True):
    cx, cy, cz = center
    z_levels = np.linspace(cz - half_h, cz + half_h, n_z)
    thetas = np.linspace(0.0, 2*np.pi, n_theta_lateral, endpoint=False)
    pts = []
    nms = []
    for z in z_levels:
        for th in thetas:
            x = cx + radius*np.cos(th)
            y = cy + radius*np.sin(th)
            pts.append([x,y,z])
            nms.append([np.cos(th), np.sin(th), 0.0])
    thetas_cap = np.linspace(0.0, 2*np.pi, n_theta_caps, endpoint=False)
    z_top = cz + half_h
    z_bot = cz - half_h
    for th in thetas_cap:
        x = cx + radius*np.cos(th)
        y = cy + radius*np.sin(th)
        pts.append([x,y,z_top]); nms.append([0.0,0.0, 1.0])
        pts.append([x,y,z_bot]); nms.append([0.0,0.0,-1.0])
    if include_centers:
        pts.append([cx,cy,z_top]); nms.append([0.0,0.0, 1.0])
        pts.append([cx,cy,z_bot]); nms.append([0.0,0.0,-1.0])
    return np.array(pts, dtype=np.float64), np.array(nms, dtype=np.float64)
```

```
55
56  CYL_CENTER = np.array([0.0, 200.0, 5.0], dtype=np.float64)
57  CYL_RADIUS = 7.0
58  CYL_HALF_H = 5.0
59  CYL_PTS, CYL_NMS = _make_cylinder_samples(
60      CYL_CENTER, CYL_RADIUS, CYL_HALF_H,
61      n_theta_lateral=20, n_z=7, n_theta_caps=20, include_centers=True
62  )
63
64  @njit(fastmath=True)
65  def angle_to_unit(theta):
66      return np.array([np.cos(theta), np.sin(theta)], dtype=np.float64)
67
68  @njit(fastmath=True)
69  def occlusion_mask_and_margin_sampled(ts,
70                                        m0, m_dir, v_m,
71                                        smoke_xy_x, smoke_xy_y, smoke_z0, sink, t_burst, R,
72                                        cyl_pts, cyl_nms):
73      n = len(ts)
74      mask = np.zeros(n, dtype=np.bool_)
75      margin = np.empty(n, dtype=np.float64)
76      n_pts = cyl_pts.shape[0]
77
78      for i in range(n):
79          t = ts[i]
80          mx = m0[0] + m_dir[0]*v_m*t
81          my = m0[1] + m_dir[1]*v_m*t
82          mz = m0[2] + m_dir[2]*v_m*t
83          sz = smoke_z0 - sink*(t - t_burst)
84          sx = smoke_xy_x
85          sy = smoke_xy_y
86
87          any_visible = False
88          fully_blocked = True
89          local_min_margin = 1e9
90
91          m_vec = np.array([mx, my, mz], dtype=np.float64)
92          c_vec = np.array([sx, sy, sz], dtype=np.float64)
93
94          for k in range(n_pts):
95              px = cyl_pts[k,0]; py = cyl_pts[k,1]; pz = cyl_pts[k,2]
96              nx = cyl_nms[k,0]; ny = cyl_nms[k,1]; nz = cyl_nms[k,2]
97
98              vmx = mx - px
99              vmy = my - py
100             vmz = mz - pz
101             if nx*vmx + ny*vmy + nz*vmz <= 0.0:
102                 continue
```

```
            any_visible = True

            p_vec = np.array([px, py, pz], dtype=np.float64)
            d = _segment_sphere_distance(m_vec, p_vec, c_vec)
            mrg = R - d
            if mrg < 0.0:
                fully_blocked = False
            if mrg < local_min_margin:
                local_min_margin = mrg
            if (not fully_blocked) and (local_min_margin < -R):
                break

        if not any_visible:
            mask[i] = False
            margin[i] = -0.05
            continue

        mask[i] = fully_blocked
        if local_min_margin > 1e8:
            local_min_margin = 0.0
        margin[i] = local_min_margin

    return mask, margin

@njit(fastmath=True)
def angle_to_unit(theta):
    return np.array([np.cos(theta), np.sin(theta)], dtype=np.float64)

@njit(fastmath=True)
def occlusion_mask_and_margin(ts, m0, m_dir, v_m, target_center,
                              rt, sx, sy, z0, sink, t_burst, R):
    n = len(ts)
    mask = np.zeros(n, dtype=np.bool_)
    margin = np.empty(n, dtype=np.float64)

    for i in range(n):
        t = ts[i]
        mx = m0[0] + m_dir[0]*v_m*t
        my = m0[1] + m_dir[1]*v_m*t
        mz = m0[2] + m_dir[2]*v_m*t

        ax = target_center[0] - mx
        ay = target_center[1] - my
        az = target_center[2] - mz
        L = np.sqrt(ax*ax + ay*ay + az*az)
        if L < 1e-12:
            mask[i] = False
            margin[i] = 1e9
```

```
            continue
        ahx, ahy, ahz = ax/L, ay/L, az/L

        sz = z0 - sink*(t - t_burst)
        dx = sx - mx
        dy = sy - my
        dz = sz - mz

        u = dx*ahx + dy*ahy + dz*ahz
        px = dx - u*ahx
        py = dy - u*ahy
        pz = dz - u*ahz
        r_perp = np.sqrt(px*px + py*py + pz*pz)
        d_sm = np.sqrt(u*u + r_perp*r_perp)

        if (u <= 0.0) or (u >= L) or (d_sm <= 1e-12):
            mask[i] = False
            margin[i] = 1e9
            continue

        alpha_s = np.arcsin(min(1.0, R / d_sm))
        phi    = np.arctan2(r_perp, u)
        alpha_t = np.arctan(rt / L)

        marg = alpha_s - (phi + alpha_t)
        margin[i] = marg
        mask[i] = (marg >= 0.0)

    return mask, margin

def _extract_intervals_from_mask(ts, mask):
    idx = np.flatnonzero(mask)
    if idx.size == 0:
        return []
    splits = np.where(np.diff(idx) > 1)[0]
    starts = np.r_[0, splits + 1]
    ends = np.r_[splits, idx.size - 1]
    intervals = []
    for s, e in zip(starts, ends):
        i0 = idx[s]
        i1 = idx[e]
        intervals.append((float(ts[i0]), float(ts[i1])))
    return intervals

def simulate_occlusion(theta, v_uav, t_release, t_fuse, dt_base):
    g = 9.81
    v_m = 300.0
    smoke_sink = 3.0
```

```
    smoke_R = 10.0
    smoke_effect = 20.0

    O = np.array([0.0, 0.0, 0.0], dtype=np.float64)
    m0 = np.array([20000.0, 0.0, 2000.0], dtype=np.float64)
    L0 = np.linalg.norm(m0 - O)
    if L0 < 1e-9:
        return 0.0, [], None, None, {}
    m_dir = (O - m0)/L0
    T_impact = L0 / v_m

    fy0 = np.array([17800.0, 0.0, 1800.0], dtype=np.float64)
    if (v_uav < 70.0) or (v_uav > 140.0) or (t_release < 0.0) or (t_fuse < 0.0):
        return 0.0, [], None, None, {}

    u_dir = angle_to_unit(theta)
    uav_vel = np.array([u_dir[0]*v_uav, u_dir[1]*v_uav, 0.0], dtype=np.float64)

    pos_release = fy0 + uav_vel * t_release
    t_burst = t_release + t_fuse
    if t_burst >= T_impact:
        return 0.0, [], pos_release, None, {}

    pos_burst = pos_release + uav_vel * t_fuse + np.array([0,0,-0.5*g*t_fuse**2], dtype=np.float64)
    smoke_xy = pos_burst[:2].copy()
    smoke_z0 = pos_burst[2]

    t0 = t_burst
    t1 = min(t_burst + smoke_effect, T_impact)
    if t1 <= t0:
        return 0.0, [], pos_release, pos_burst, {}

    dt_coarse = max(0.02, float(dt_base))
    dt_fine = 0.001

    triage_ts = np.linspace(t0, t1, 5)
    mask_tri, margin_tri = occlusion_mask_and_margin_sampled(
        triage_ts, m0, m_dir, v_m,
        smoke_xy[0], smoke_xy[1], smoke_z0, smoke_sink, t_burst, smoke_R,
        CYL_PTS, CYL_NMS
    )

    if (not mask_tri.any()) and (margin_tri.max() < -0.1):
        return 0.0, [], pos_release, pos_burst, {
            "t_burst": t_burst, "dt": dt_coarse, "dt_fine": dt_fine, "early_pruned": True
        }

    ts_coarse = np.arange(t0, t1 + 1e-12, dt_coarse, dtype=np.float64)
```

```
    mask_c, _ = occlusion_mask_and_margin_sampled(
        ts_coarse, m0, m_dir, v_m,
        smoke_xy[0], smoke_xy[1], smoke_z0, smoke_sink, t_burst, smoke_R,
        CYL_PTS, CYL_NMS
    )
    if not mask_c.any():
        return 0.0, [], pos_release, pos_burst, {
            "t_burst": t_burst, "dt": dt_coarse, "dt_fine": dt_fine, "early_pruned": False
        }

    coarse_intervals = _extract_intervals_from_mask(ts_coarse, mask_c)

    total_time = 0.0
    fine_intervals_global = []
    for (ca, cb) in coarse_intervals:
        fa = max(t0, ca - dt_coarse)
        fb = min(t1, cb + dt_coarse)
        ts_fine = np.arange(fa, fb + 1e-12, dt_fine, dtype=np.float64)
        mask_f, _ = occlusion_mask_and_margin_sampled(
            ts_fine, m0, m_dir, v_m,
            smoke_xy[0], smoke_xy[1], smoke_z0, smoke_sink, t_burst, smoke_R,
            CYL_PTS, CYL_NMS
        )
        if mask_f.any():
            fine_intervals = _extract_intervals_from_mask(ts_fine, mask_f)
            fine_intervals_global.extend(fine_intervals)
            total_time += mask_f.sum() * dt_fine

    if fine_intervals_global:
        fine_intervals_global.sort(key=lambda x: x[0])
        merged = []
        eps = 2*dt_fine
        cs, ce = fine_intervals_global[0]
        for s,e in fine_intervals_global[1:]:
            if s - ce <= eps:
                ce = max(ce, e)
            else:
                merged.append((cs, ce))
                cs, ce = s, e
        merged.append((cs, ce))
        fine_intervals_global = merged

    return total_time, fine_intervals_global, pos_release, pos_burst, {
        "t_burst": t_burst, "dt": dt_coarse, "dt_fine": dt_fine, "early_pruned": False
    }


def merge_intervals(intervals, eps=1e-6):
```

```
    if not intervals:
        return []
    intervals = sorted(intervals, key=lambda x: x[0])
    merged, cs, ce = [], intervals[0][0], intervals[0][1]
    for s, e in intervals[1:]:
        if s <= ce + eps:
            ce = max(ce, e)
        else:
            merged.append((cs, ce))
            cs, ce = s, e
    merged.append((cs, ce))
    return merged

def intervals_length(intervals):
    return sum(max(0.0, e - s) for (s, e) in intervals)

def union_intervals(a, b):
    return merge_intervals(list(a) + list(b))

def incremental_gain(candidate_intv, current_union):
    before = intervals_length(current_union)
    after = intervals_length(union_intervals(current_union, candidate_intv))
    return max(0.0, after - before)

def greedy_three_for_theta_v(theta_fix, v_fix):
    tr_grid = np.linspace(0, 60, 241)
    tf_grid = np.linspace(0, 6, 61)
    dt_eval = 0.001
    min_gap = 1.0

    def _search_next(kth, t_release_min, union_so_far):
        best = None
        best_gain = -1.0
        iterator = itertools.product(tr_grid, tf_grid)
        total = len(tr_grid)*len(tf_grid)
        for tr, tf in tqdm(iterator, total=total, dynamic_ncols=True, leave=False,
                           desc=f"( ,v)内第{kth}枚"):
            if tr < t_release_min - 1e-9:
                continue
            score, intervals, rel, bur, extra = simulate_occlusion(theta_fix, v_fix, tr, tf, dt_eval)
            if score <= 0.0 or not intervals:
                continue
            gain = incremental_gain(intervals, union_so_far)
            if (gain > best_gain + 1e-9) or (abs(gain - best_gain) <= 1e-9 and (best is None or score
                > best[0])):
                best_gain = gain
                best = (score, gain, tr, tf, intervals, rel, bur, extra)
        if best is None:
```

```
            return (0.0, 0.0, np.nan, np.nan, [], None, None, {})
        return best

    items = []
    union_intv = []

    # 第1枚
    s1, g1, tr1, tf1, intv1, rel1, bur1, ex1 = _search_next(1, 0.0, union_intv)
    union_intv = union_intervals(union_intv, intv1)
    items.append({"score_self": s1, "gain": g1, "tr": tr1, "tf": tf1,
                  "intervals": intv1, "rel": rel1, "bur": bur1})

    # 第2枚
    s2, g2, tr2, tf2, intv2, rel2, bur2, ex2 = _search_next(2, tr1 + min_gap if np.isfinite(tr1)
        else 0.0, union_intv)
    union_intv = union_intervals(union_intv, intv2)
    items.append({"score_self": s2, "gain": g2, "tr": tr2, "tf": tf2,
                  "intervals": intv2, "rel": rel2, "bur": bur2})

    # 第3枚
    start3 = tr2 + min_gap if np.isfinite(tr2) else (tr1 + min_gap if np.isfinite(tr1) else 0.0)
    s3, g3, tr3, tf3, intv3, rel3, bur3, ex3 = _search_next(3, start3, union_intv)
    union_intv = union_intervals(union_intv, intv3)
    items.append({"score_self": s3, "gain": g3, "tr": tr3, "tf": tf3,
                  "intervals": intv3, "rel": rel3, "bur": bur3})

    score_union = intervals_length(union_intv)
    return {"score_union": score_union, "items": items, "theta": theta_fix, "v": v_fix, "union":
        union_intv}


if __name__ == '__main__':
    multiprocessing.freeze_support()

    fy0_xy = np.array([17800, 0])
    base_vec = (np.array([0, 0]) - fy0_xy)
    base_theta = np.arctan2(base_vec[1], base_vec[0])

    theta_grid = base_theta + np.deg2rad(np.linspace(-10, 10, 21))
    speed_grid = np.linspace(70, 140, 71)

    pairs = list(itertools.product(theta_grid, speed_grid))
    results = []

    WORKERS = os.cpu_count()
    print(f"检测到CPU核心数: {WORKERS}, 将使用␣{WORKERS}␣个进程并行。")
    with ProcessPoolExecutor(max_workers=WORKERS) as ex:
        futures = {ex.submit(greedy_three_for_theta_v, th, vv): (th, vv) for th, vv in pairs}
```

```
        for fut in tqdm(as_completed(futures), total=len(futures), desc="( ,v)␣外层评估",
            dynamic_ncols=True):
            res = fut.result()
            results.append(res)

    results.sort(key=lambda x: x["score_union"], reverse=True)
    best = results[0] if results else {"score_union": 0.0, "items": [], "theta": base_theta, "v":
        120.0, "union": []}

    theta_star = best["theta"]
    v_star = best["v"]
    union_best = best["union"]
    deg_heading = np.rad2deg(theta_star)
    deg_base = np.rad2deg(base_theta)
    deg_offset = np.rad2deg(((theta_star - base_theta + np.pi) % (2*np.pi)) - np.pi)


    print(f"最优联合遮蔽总时长:␣{best['score_union']:.3f}␣s")
    print(f"航向角(相对+x轴):␣{deg_heading:.2f}°")
    print(f"飞行速度:␣{v_star:.2f}␣m/s")
    print("联合遮蔽区间并集: ")
    for (a,b) in union_best:
        print(f"␣␣[{a:.3f},␣{b:.3f}]␣␣时长 {(b-a):.3f}␣s")

    for i, it in enumerate(best["items"], 1):
        print(f"\n——␣第{i}枚␣——")
        print(f"自身遮蔽时长:␣{it['score_self']:.3f}␣s")
        print(f"对总体新增贡献:␣{it['gain']:.3f}␣s")
        print(f"投放时刻:␣{it['tr']:.2f}␣s")
        print(f"引信延时:␣{it['tf']:.2f}␣s")
        print(f"起爆时刻:␣{it['tr']+it['tf']␣if␣np.isfinite(it['tr'])␣else␣np.nan:.2f}␣s")
        if it["rel"] is not None:
            print(f"投放点:␣{np.round(it['rel'],3)}")
        if it["bur"] is not None:
            print(f"起爆点:␣{np.round(it['bur'],3)}")
        if it["intervals"]:
            for (a,b) in it["intervals"]:
                print(f"␣␣区间␣[{a:.3f},␣{b:.3f}]␣␣时长 {(b-a):.3f}␣s")
        else:
            print("该弹无遮蔽。")
```

Listing 4: 合理性检验代码

```
import math
from typing import List, Tuple

# 真目标: 竖直圆柱体
CYL_BOTTOM_CENTER = (0.0, 200.0, 0.0) # 底面圆心 (x,y,z)
CYL_RADIUS = 7.0                  # 半径 (m)
```

```
CYL_HEIGHT = 10.0                # 高度（m）

# 导弹初始信息
MISSILE_INIT_POS = (20000.0, 0.0, 2000.0)
MISSILE_SPEED = 300.0            # m/s，沿指向原点方向匀速直线

UAV_INIT_POS = (17800.0, 0.0, 1800.0) # FY1
UAV_SPEED = 138.67                  # m/s
UAV_YAW_DEG = -63.00 # 在水平面内的方位角，0度沿+x，逆时针为+（朝+y）

# 烟幕弹时序
DROP_TIME = 6.00     # s，投放时刻（接到任务后）
FUSE_DELAY = 5.00    # s，引信延时（自投放起计）
# 起爆时刻
DETONATE_TIME = DROP_TIME + FUSE_DELAY

# 烟幕云团参数
SMOKE_RADIUS = 10.0  # m，云团有效半径（浓度阈值内）
SMOKE_SINK_V = 3.0   # m/s，起爆后云团中心匀速下沉
SMOKE_EFFECT_T = 20.0 # s，有效遮蔽持续时长（自起爆起计）

# 重力加速度（仅用于投放后的抛体运动阶段）
G = 9.81             # m/s^2，向下

# 遮蔽判定参数
OCCLUSION_RATIO_THRESHOLD = 1 # 采样点被遮蔽比例达到该阈值，则认为“有效遮蔽”

# 仿真时间
DT = 0.001            # s，时间步长
# 仿真终止：到达“原点”的打击时刻（或到原点前的足够长时间）
MISSILE_TO_ORIGIN_T = math.dist(MISSILE_INIT_POS, (0.0, 0.0, 0.0)) / MISSILE_SPEED
T_END = MISSILE_TO_ORIGIN_T # 可按需增大

# 采样圆柱体
CYL_SAMPLE_THETA = 72 # 侧面周向采样数
CYL_SAMPLE_HEIGHT = 5 # 高度层数

Vector = Tuple[float, float, float]

def v_add(a: Vector, b: Vector) -> Vector:
    return (a[0]+b[0], a[1]+b[1], a[2]+b[2])

def v_sub(a: Vector, b: Vector) -> Vector:
    return (a[0]-b[0], a[1]-b[1], a[2]-b[2])

def v_mul(a: Vector, s: float) -> Vector:
    return (a[0]*s, a[1]*s, a[2]*s)

```

```
55 def v_dot(a: Vector, b: Vector) -> float:
56     return a[0]*b[0] + a[1]*b[1] + a[2]*b[2]
57 
58 def v_norm(a: Vector) -> float:
59     return math.sqrt(v_dot(a, a))
60 
61 def v_unit(a: Vector) -> Vector:
62     n = v_norm(a)
63     if n == 0.0:
64         return (0.0, 0.0, 0.0)
65     return (a[0]/n, a[1]/n, a[2]/n)
66 
67 def clamp01(x: float) -> float:
68     if x < 0.0: return 0.0
69     if x > 1.0: return 1.0
70     return x
71 
72 def segment_sphere_intersect(seg_a: Vector, seg_b: Vector, sph_center: Vector, sph_r: float) ->
       bool:
73     ab = v_sub(seg_b, seg_a)
74     ac = v_sub(sph_center, seg_a)
75     ab_len2 = v_dot(ab, ab)
76     if ab_len2 == 0.0:
77         return v_norm(v_sub(sph_center, seg_a)) <= sph_r
78     t = v_dot(ac, ab) / ab_len2
79     t = clamp01(t)
80     closest = v_add(seg_a, v_mul(ab, t))
81     return v_norm(v_sub(closest, sph_center)) <= sph_r
82 
83 CYL_POINTS: List[Vector] = []
84 _cx, _cy, _cz = CYL_BOTTOM_CENTER
85 for k in range(CYL_SAMPLE_HEIGHT + 1):
86     z = _cz + CYL_HEIGHT * (k / CYL_SAMPLE_HEIGHT)
87     for i in range(CYL_SAMPLE_THETA):
88         theta = 2.0 * math.pi * (i / CYL_SAMPLE_THETA)
89         x = _cx + CYL_RADIUS * math.cos(theta)
90         y = _cy + CYL_RADIUS * math.sin(theta)
91         CYL_POINTS.append((x, y, z))
92 
93 
94 yaw = math.radians(UAV_YAW_DEG)
95 uav_bore = (math.cos(yaw), math.sin(yaw), 0.0)
96 
97 UAV_VEL = v_mul(v_unit((uav_bore[0], uav_bore[1], 0.0)), UAV_SPEED)
98 UAV_DROP_POS = v_add(UAV_INIT_POS, v_mul(UAV_VEL, DROP_TIME))
99 vx, vy, vz0 = UAV_VEL[0], UAV_VEL[1], 0.0
100 SMOKE_DETONATE_POS = (
101     UAV_DROP_POS[0] + vx * FUSE_DELAY,
```

```
    UAV_DROP_POS[1] + vy * FUSE_DELAY,
    UAV_DROP_POS[2] + vz0 * FUSE_DELAY - 0.5 * G * (FUSE_DELAY ** 2)
)

cover_flags: List[bool] = []
cover_times: List[float] = []

last_visible_cnt = 0
last_covered_cnt = 0

num_steps = int(T_END / DT) + 1
for step in range(num_steps):
    t = step * DT
    cur_to_origin = v_unit(v_sub((0.0, 0.0, 0.0), MISSILE_INIT_POS))
    miss_pos = v_add(MISSILE_INIT_POS, v_mul(cur_to_origin, MISSILE_SPEED * t))

    if v_dot(v_sub((0.0,0.0,0.0), MISSILE_INIT_POS), v_sub(miss_pos, MISSILE_INIT_POS)) < 0:
        break

    miss_bore = v_unit(v_sub((0.0, 0.0, 0.0), miss_pos))

    uav_pos = v_add(UAV_INIT_POS, v_mul(UAV_VEL, t))

    smoke_active = False
    smoke_center = None

    if t < DROP_TIME:
        smoke_active = False
    elif DROP_TIME <= t < DETONATE_TIME:
        tau = t - DROP_TIME
        bomb_pos = (
            UAV_DROP_POS[0] + vx * tau,
            UAV_DROP_POS[1] + vy * tau,
            UAV_DROP_POS[2] + vz0 * tau - 0.5 * G * (tau ** 2)
        )
        smoke_active = False
    elif DETONATE_TIME <= t <= (DETONATE_TIME + SMOKE_EFFECT_T):
        tau2 = t - DETONATE_TIME
        smoke_center = (
            SMOKE_DETONATE_POS[0],
            SMOKE_DETONATE_POS[1],
            SMOKE_DETONATE_POS[2] - SMOKE_SINK_V * tau2
        )
        smoke_active = True
    else:
        smoke_active = False

    is_covered = False
```

```
    visible_cnt = 0
    covered_cnt = 0

    if smoke_active:
        for P in CYL_POINTS:
            visible_cnt += 1
            if segment_sphere_intersect(miss_pos, P, smoke_center, SMOKE_RADIUS):
                covered_cnt += 1
        if visible_cnt > 0:
            ratio = covered_cnt / visible_cnt
            is_covered = (ratio >= OCCLUSION_RATIO_THRESHOLD)
        else:
            is_covered = False
    else:
        is_covered = False

    cover_flags.append(is_covered)
    cover_times.append(t)
    last_visible_cnt = visible_cnt
    last_covered_cnt = covered_cnt

intervals: List[Tuple[float, float]] = []
if cover_flags:
    in_cov = False
    start_t = 0.0
    for i in range(len(cover_flags)):
        flag = cover_flags[i]
        tt = cover_times[i]
        if flag and not in_cov:
            in_cov = True
            start_t = tt
        elif (not flag) and in_cov:
            in_cov = False
            end_t = tt
            intervals.append((start_t, end_t))
    if in_cov:
        intervals.append((start_t, cover_times[-1]))

total_cover = sum((b - a) for (a, b) in intervals)


print("——␣参数概览␣——")
print(f"导弹初始位置:␣{MISSILE_INIT_POS},␣速度:␣{MISSILE_SPEED}␣m/s")
print(f"真目标:␣圆柱底心={CYL_BOTTOM_CENTER},␣R={CYL_RADIUS}␣m,␣H={CYL_HEIGHT}␣m,␣采样点数={len(
    CYL_POINTS)}")
print(f"无人机初始位置:␣{UAV_INIT_POS},␣速度:␣{UAV_SPEED}␣m/s,␣方位角␣{UAV_YAW_DEG:.2f}°)")
print(f"投放时刻:␣{DROP_TIME}␣s,␣引信延时:␣{FUSE_DELAY}␣s,␣起爆时刻:␣{DETONATE_TIME}␣s")
print(f"起爆位置(云团初始中心):␣({SMOKE_DETONATE_POS[0]:.3f},␣{SMOKE_DETONATE_POS[1]:.3f},␣{
```

```
    SMOKE_DETONATE_POS[2]:.3f}),␣云团半径:␣{SMOKE_RADIUS}␣m,␣下沉:␣{SMOKE_SINK_V}␣m/s,␣有效:␣{
    SMOKE_EFFECT_T}␣s")

print("——␣有效遮蔽时间段␣——")
for (a, b) in intervals:
    print(f"[{a:.4f}s,␣{b:.4f}s], 时长␣{b-a:.4f}s")
print(f"总有效遮蔽时长:␣{total_cover:.4f}␣s")
```

问题四完整 python 代码见附件

问题五完整 python 代码见附件