基于粒子群优化的无人机协同烟幕干扰策略研究

摘要

在明确敌方导弹轨迹后，如何利用无人机投放烟雾干扰弹，最大化地为保护目标提供视觉遮蔽，是本文的核心问题。本文结合解析几何、运动学和粒子群优化算法 (PSO)，建立了精确的数学模型，针对不同情境设计了无人机飞行与干扰弹投放及引爆的策略，从而最大化烟雾云团对导弹视线的遮蔽时长。

对于问题一，目标为在给定参数的前提下，计算无人机 $FY1$ 投放单枚干扰弹对导弹 $M1$ 的有效遮蔽时长。基于导弹、无人机、干扰弹、真目标的动态空间关系，我们全面讨论了烟雾云团是否有效遮蔽导弹视线的判断方法：首先，通过几何推导严格证明了“圆柱体真目标被完全遮蔽”等价于“其上下表面圆周被完全遮蔽”的命题；随后，在深入讨论最简单直观的距离指标模型的过程中，发现逻辑纰漏，于是创新性地提出了完备、精确的**角度指标模型**。利用该模型，计算出问题一给定参数下，通过**小步长检索**，$FY1$ 投放一枚干扰弹的有效遮蔽时长为 1.40 s。

对于问题二，首先将单架无人机 $FY1$ 干扰导弹 $M1$ 的最大有效时长问题，转化为有效遮蔽时长的优化问题，参数组包含无人机的飞行速度、方向、干扰弹的投放时刻及引爆延迟时长四个参数的优化问题。基于问题特性，论证了**粒子群优化算法** PSO 的适用性。最终得到的 $FY1$ 通过单枚干扰弹对导弹 $M1$ 的最大有效遮蔽时长为 4.51 s。

对于问题三，需要讨论单架无人机 $FY1$ 投放三枚干扰弹对导弹 $M1$ 的最优遮蔽策略。同样地，将该问题转化为目标函数为总有效遮蔽时长的优化问题，参数维数由四维升高到八维。仍然考虑采用 PSO 解决该优化问题，但为了降低八维参数空间带来的计算挑战，提出了将圆柱体目标简化为其中心点的**目标简化模型**。首先，通过真目标与导弹的相对空间关系给出了该简化之合理性的定性描述；其次，通过误差分析，确定该操作相对误差在 5% 以内。因此，我们更新了问题一所建立的计算模型。最终结果表明，$FY1$ 投放三枚干扰弹的总有效遮蔽时长可达 6.40 s。

对于问题四，需要讨论三架无人机 $FY1, FY2, FY3$ 各投放一枚干扰弹协同干扰导弹 $M1$ 的策略。将该问题转化为待优化参数为十二维向量的优化问题。为了降低计算复杂度，我们再次对 PSO 进行优化。在计算遮蔽时长的过程中，采用粗略扫描变号区间与 $Brent$ **求零点**相结合的模式替代原本的微小步长遍历法，大幅提升了算法的效率与精度。最终计算结果表明，最大总有效遮蔽时长为 14.825 s。

对于问题五，该问题是一个待优化参量高达 40 维的高复杂度优化问题。为规避全局优化的巨大计算成本，我们设计了一种**结合贪心算法的分而治之策略**。该策略基于同一无人机后续干扰弹的效用递减这一先验规律，通过分步决策将复杂问题分解。最终，给出了 5 架无人机共 15 枚干扰弹的投放与引爆参数，使得总最大遮蔽时长为 42.260 s。

关键字：　角度指标模型　粒子群优化　$Brent$ 方法　贪心算法

一、 问题重述

1.1 问题背景

当前，烟幕干扰弹作为一种高效的防御手段，被广泛用于对抗精确制导武器。本问题旨在研究无人机投放烟幕干扰弹的最优策略，以保护特定地面目标免受来袭导弹的威胁。烟幕弹起爆后形成特定范围的有效遮蔽云团，并以恒定速度下沉。无人机在接收任务后，需在指定速度范围内以匀速直线方式飞行。我们需要综合考虑无人机飞行路径、烟幕弹投放与起爆时机，为不同数量的无人机和导弹组合，设计能够使总有效遮蔽时间最长的投放策略。

1.2 问题提出

问题 1：在给定的飞行与投放参数下（无人机 $FY1$ 以 $120\,m/s$ 速度飞向假目标，任务开始 $1.5\,s$ 后投放，$3.6\,s$ 后起爆），求解单枚烟幕弹对来袭导弹 $M1$ 所能形成的有效遮蔽时长。这是一个对特定策略效果进行定量计算的问题。

问题 2：针对无人机 $FY1$ 使用单枚烟幕弹干扰导弹 $M1$ 的场景，建立优化模型。需要将 $FY1$ 的飞行方向、飞行速度、烟幕弹的投放点与起爆点作为决策变量，以实现对 $M1$ 的单次有效遮蔽时间最长为优化目标。

问题 3：将问题二的单弹模型扩展为多弹模型。在仅使用无人机 $FY1$ 的条件下，设计一个包含 3 枚烟幕弹的连续或非连续投放策略，以最大化对导弹 $M1$ 的总有效遮蔽时间，并按要求格式输出结果。

问题 4：研究多机协同干扰策略。需要为 $FY1$、$FY2$、$FY3$ 三架无人机进行任务规划，每架无人机各投放一枚烟幕弹，共同对单一导弹 $M1$ 实施干扰。目标是设计最优的协同投放方案，以最大化总遮蔽时长。

问题 5：面对最为复杂的饱和攻击情景，要求为全部 5 架无人机设计综合干扰方案，以应对 $M1$、$M2$、$M3$ 三枚导弹的威胁，其中每架无人机最多可使用 3 枚烟幕弹。需要建立全面的优化模型，实现对真目标总有效遮蔽时间的最大化。

二、 问题分析

2.1 问题一分析

本问题的核心在于对给定参数下的有效遮蔽时长进行精确计算。思路是首先通过运动学原理，建立导弹（飞向原点假目标）、无人机、干扰弹（平抛运动）以及烟雾云团

（匀速下沉）的位置随时间变化的函数。接着，关键一步是将复杂的圆柱体真目标被遮蔽问题，通过**几何论证简化**为判断其上下两个圆周是否均在由导弹位置和球形云团构成的视线盲区锥内。最后，为避免直接计算点线距离可能出现的逻辑谬误，我们构建一个基于**角度比较的判别模型**，使用**小步长遍历**稳健地判断任意时刻目标是否被完全遮蔽。

2.2 问题二分析

本问题本质上是一个**最优化问题**，即在无人机速度、航向等约束条件下，寻找一组最优的行动参数，以使对导弹 $M1$ 的有效遮蔽时间最长。我们将问题一建立的遮蔽时长计算模型作为目标函数。由于无人机航线和投放时机组合多样，目标函数很可能具有非凸、多峰值的特性，常规的梯度优化方法难以找到全局最优解。因此，采用**粒子群优化** (PSO) 这类具有强大全局搜索能力的智能算法，是求解此类问题的有效技术路线。

2.3 问题三分析

本问题在问题二的基础上增加了干扰弹数量，待优化参数维度从四维增至八维，计算复杂度剧增。核心思路是首先进行模型降维与简化，我们通过**误差分析**验证了将圆柱体真目标近似为其中心点的**简化点模型**的可行性，从而大幅降低单次目标函数评估的计算成本。在此基础上，构建一个包含 8 个决策变量的 PSO 优化模型，并需在优化过程中严格满足三枚干扰弹投放时间间隔不小于 $1s$ 的约束。优化目标是三**个遮蔽时间段并集的总长度**。

2.4 问题四分析

本问题是多智能体（三架无人机）的协同优化问题，其决策空间维度进一步扩大至十二维，对算法效率提出了极高要求。我们的分析重点在于优化目标函数的计算效率。传统的小步长时域扫描法计算量巨大且存在精度损失。因此，我们计划采用一种混合搜索策略：先通过大步长扫描快速定位遮蔽状态可能发生变化的变号区间，再利用高效且精确的数值求根算法 **Brent's Method** 来精确定位区间的端点。这一技术路线旨在实现计算速度和求解精度的平衡，以应对高维优化挑战。

2.5 问题五分析

本问题是多智能体、多目标、多任务的饱和攻击场景，其核心难点在于任务分配的组合爆炸与优化参数的超高维度。我们的技术路线是**分而治之**：首先，采用基于拦截效率的**贪心策略**为每架无人机及其携带的干扰弹指派主攻导弹，将问题分解为三个独立的子优化任务。随后，对每个子任务（多无人机协同干扰单导弹），沿用问题四中高效的

高维 PSO 优化模型，并利用简化点模型与 **Brent's Method** 加速求解。最终，将各子任务的优化结果汇总，得到全局最优投放策略。

三、 模型假设

- 假设烟雾弹在脱离无人机后的运动过程中仅受重力影响，忽略空气阻力等其他一切次要因素。
- 假设烟雾弹起爆过程瞬时完成，并即刻形成一个半径为 10 m 的均匀球形云团，该云团在形成后的 20 s 内始终保持有效遮蔽能力，之后则完全失效。
- 假设所有导弹均以恒定速率沿直线飞向假目标（坐标原点），其飞行路径和速度不受烟雾或其他环境因素影响而改变。
- 假设无人机在接收指令后能瞬时调整至目标航向和速度，并开始等高匀速直线飞行。同时，假设所有信息传递（如雷达探测、指令下达）均无延迟。

四、 符号说明

符号	说明	单位
$X_{ij}(t)$	第 i 架无人机投放的第 j 枚烟雾弹在时刻 t 的中心位置	m
$X_{导i}(t)$	第 i 枚导弹在时刻 t 的位置	m
$R_{团}$	烟雾云团的半径	m
$R_{柱}$	目标圆柱体的半径	m
$H_{柱}$	目标圆柱体的高度	m
$T_{团}$	烟雾云团的有效持续时间	s
t_{ij1}	第 i 架无人机投放第 j 枚干扰弹的时刻	s
t_{ij2}	第 i 架无人机投放的第 j 枚干扰弹的引爆时刻	s
$T_{总遮ijn}$	第 i 枚导弹被第 j 架飞机的第 n 枚烟雾弹遮蔽的总时长	s
$V_{飞i}$	第 i 架无人机的飞行速度	m/s
$V_{导}$	导弹的飞行速度	m/s
g	重力加速度	m/s^2
γ_i	第 i 架无人机飞行方向与 x 轴正向的夹角	rad
$\vec{e_i}$	第 i 架无人机的初始飞行方向向量	–
$\Delta_{\max}(t)$	时刻 t 用于判断是否遮蔽的角度判别指标	–
$\|\|\cdot\|\|$	表示对区间的并集求测度	–

五、 问题一模型建立与求解

我们为了高效解决圆柱的形状问题，选择将**圆柱被遮蔽情况等价转化为上下表面圆周被遮盖的等价条件**。

我们首先过分析运动轨迹，计算确定爆炸时刻的时间和爆炸点的位置，接着分类导弹是否进入云团中的情况，证明了两个命题的正确性，接着我们指出导弹连线到云团中

心距离算法的一个在将射线认定为直线所导致的小漏洞，进而在此算法基础上进行微调，提出角度比较模型来确认某时刻圆柱是否被完全遮蔽，最后我们再对于时间进行小步长遍历，获得完全遮蔽的总时长 $T_{\text{总遮 }111}$。

建立衡量某一时刻是否发生遮蔽的判断指标的具体流程如图 1：

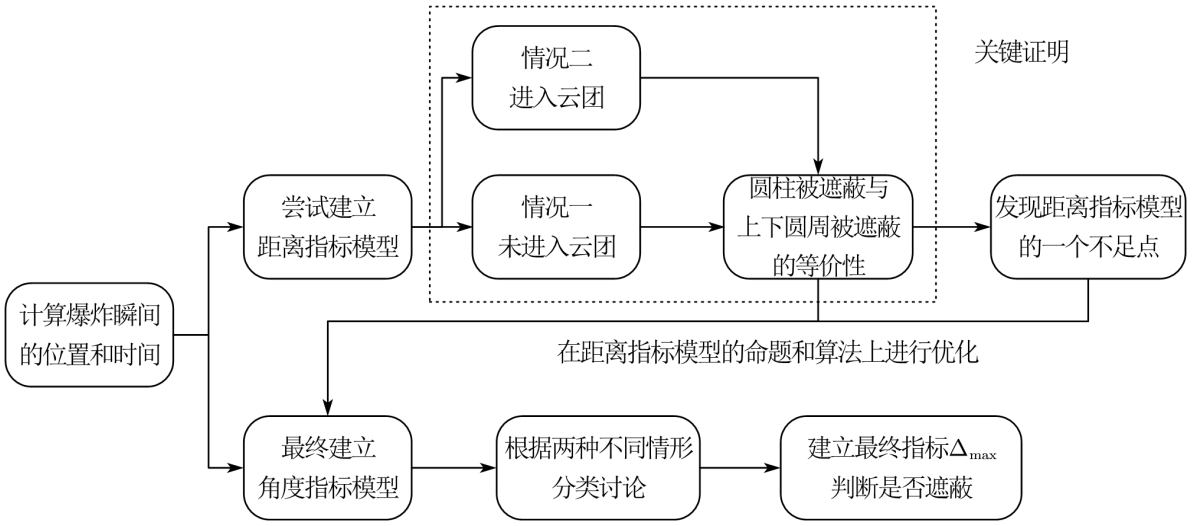


图 1　问题一流程图

5.1 距离指标模型

为了区分符号我们把第 i 架飞机即 FYi 的第 j 枚烟雾弹的第 n 时刻（第零时刻对应飞机启动，第一时刻对应抛下烟雾弹，第二阶段对应烟雾弹爆炸）的时间作为 t_{ijn}（注意到 $t_{ij0}=0\,\text{s}$）。而这一时刻烟雾弹即烟雾中心对应的位置为 X_{ijn}（第 i 架无人机的速度，初始位置和方向为 $V_{\text{飞 i}}$，X_i 和 $\vec{e_i}$），在结束爆炸后，我们的第 i 架飞机即 FYi 的第 j 枚烟雾弹的位置为 X_{ij}。

其中，$i\in\{1,2,\ldots,5\}$，$j\in\{1,2,3\}$，$n\in\{0,1,2\}$。

我们设导弹的速度为 $V_{\text{导}}$，第 i 枚导弹距离假目标的初始位置为 $X_{\text{导 i0}}$。

由导弹会向原点飞行，可以得到第 i 枚导弹位置 $X_{\text{导 i}}$ 与时间变化的函数:

$$X_{\text{导 i}}(t)=\left(1-\frac{V_{\text{导}}\cdot t}{||X_{\text{导 i0}}||}\right)X_{\text{导 i0}} \tag{1}$$

同时我们补充终止时间 $T_{\text{终}}=\frac{||X_{\text{导 i0}}||}{V_{\text{导}}}$ 作为第 i 枚导弹的终止时间。

在 $[t_{110},t_{111}]=[0\,\text{s},1.5\,\text{s}]$ 内，干扰弹随无人机 $FY1$ 做匀速直线运动，起始点坐标为 $X_1=(x_1,y_1,z_1)=(17800,0,1800)$，速度大小为 $V_{\text{飞 1}}=120\,\text{m/s}$，速度方向为水平面内朝向 z 轴，即 $\vec{e_1}=(-1,0,0)$。则我们可以得到抛下烟雾弹的位置 X_{111}:

$$X_{111}=X_1+\vec{e_1}(t_{111}-t_{110})V_{\text{飞 1}}$$

在 $[t_{111}, t_{112}] = [1.5\,\text{s}, 5.1\,\text{s}]$ 内，干扰弹在离开无人机 $FY1$ 后，在重力的作用下做平抛运动，水平速度为无人机速度 $V_{飞1} = 120\,\text{m/s}$，重力加速度 $g = 9.80665\,\text{m/s}^2$。则我们可以得到烟雾弹爆炸的位置 X_{112}:

$$\begin{aligned} X_{112} &= X_{111} + \vec{e_1}(t_{112} - t_{111})V_{飞1} + (0, 0, -\frac{1}{2}g(t_{112} - t_{111})^2) \\ &= X_1 + \vec{e_1}(t_{111} - t_{110})V_{飞1} + (0, 0, -\frac{1}{2}g(t_{112} - t_{111})^2) \end{aligned}$$

设云团下降速度为 $v_{降}$ 随时间的变化，烟雾云团中心点坐标 X_{11} 为:

$$\begin{aligned} X_{11} =& X_{112} + (0, 0, -v_{降}(t - t_{112})) \\ =& X_1 + \vec{e_1}(t_{111} - t_{110})V_{飞1} + (0, 0, -\frac{1}{2}g(t_{112} - t_{111})^2) + (0, 0, -v_{降}(t - t_{112})) \\ =& X_{11}(t) \end{aligned} \tag{2}$$

接下来分两种情况进行论述。

5.1.1 情况一

第一种情况为在烟雾干扰弹形成的云团匀速下降的过程中，在导弹处于云团外时，会形成为开圆锥体的视线盲区，如图 2所示：

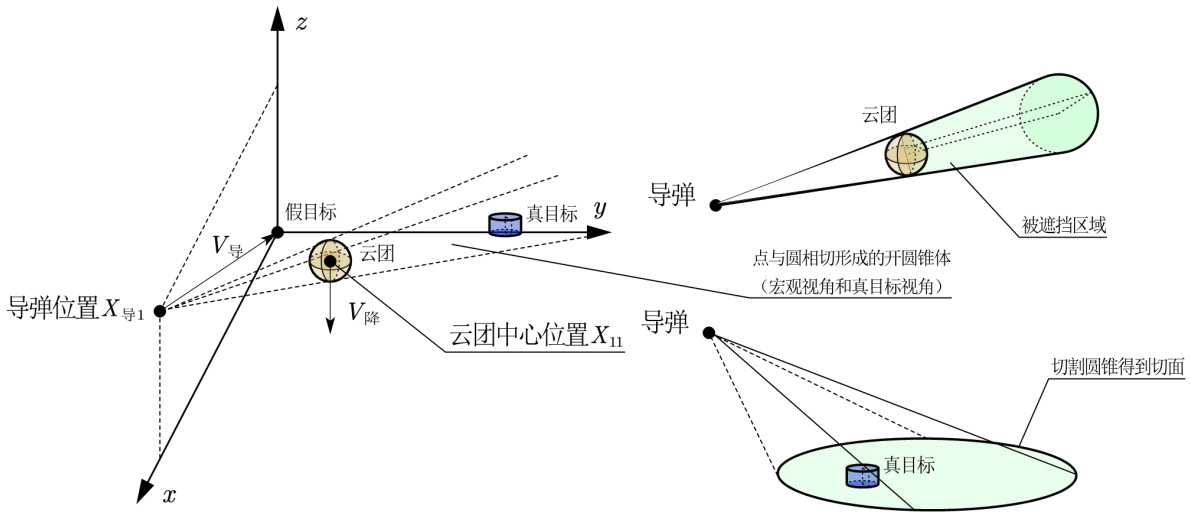


图 2　情况一示意图

当真目标圆柱体完全被开圆锥盲区包含的时候，云团对真目标实现了有效遮蔽，所以计算烟雾干扰弹对导弹的有效遮蔽时长，即计算真目标圆柱体被开圆锥盲区完全包含的时长。

为了有效简化圆柱体模型，我们提出以下命题。

定理 1 圆柱体被开圆锥盲区完全包含的充要条件是圆柱体上下表面被开圆锥盲区包含。

证明：

必要性显然。

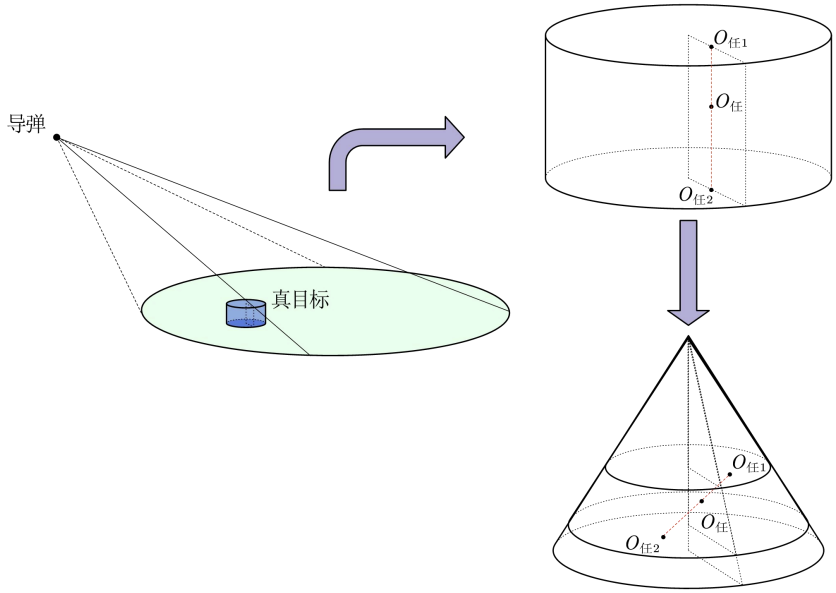


图 3　命题一证明示意图

下证充分性。如图 3所示，考察圆柱体内任意点 $O_{任}$，过 $O_{任}$ 点做平行于圆柱体侧面的铅垂线，分别交圆柱体上下表面于 $O_{任1}$、$O_{任2}$ 两点，由于圆柱体的上下表面均在开圆锥内，所以 $O_{任1}$、$O_{任2}$ 两点均在开圆锥内，所以线段 $O_{任1}O_{任2}$ 在开圆锥内，而 $O_{任}$ 是线段 $O_{任1}O_{任2}$ 上的点，故 $O_{任}$ 在开圆锥内，充分性证毕。

接着我们进一步细化点集范围，将点的范围从上下圆转为上下圆周。

定理 2 圆柱体的上下表面被开圆锥盲区完全包含的充要条件是圆柱体的上下表面圆周被开圆锥盲区包含。

证明：

必要性显然。

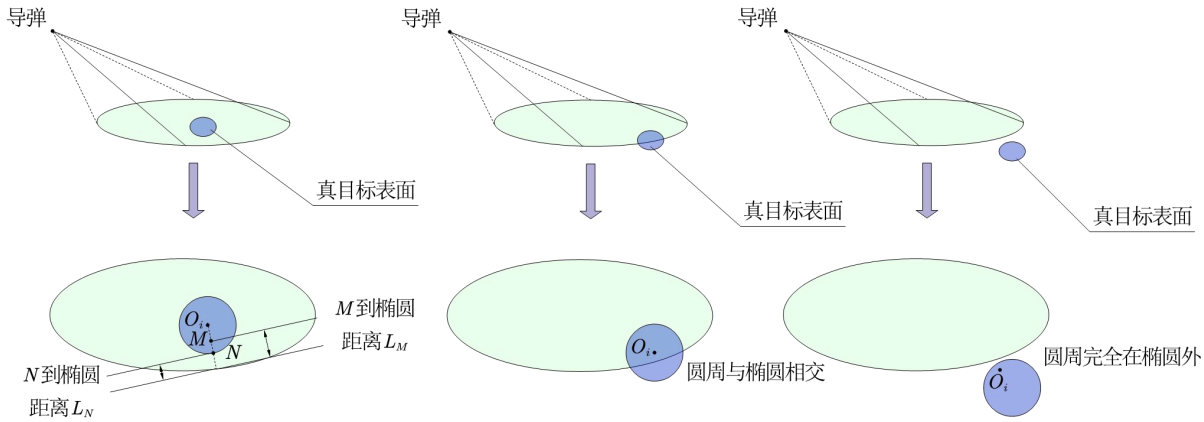


图 4　命题二证明示意图

下证明充分性。如图 4所示，圆柱体的上下表面圆和开圆锥的空间关系有且仅有三种可能情形。对于每一种情形，取表面圆所在的水平面，对开圆锥做截面。由立体几何，圆锥被不垂直于轴线的平面所截，得到的必为圆，椭圆，双曲线或抛物线（图中示例为椭圆，其他条件同理）。通过分类，我们确定圆周在开圆锥内的情况对应图 4中最左图。

而如图 4中最左图所示，若圆周完全包含在椭圆内，则此时由圆上任意一点 M 到椭圆的距离会大于 N 点（M 相应的最近距离连接线对应的线段与圆的交点），因此圆内的点到椭圆的距离会小于等于圆周距离椭圆的距离，即知圆内所有点都会在椭圆内。

至此，我们成功将第一种情况下需要考虑的点集合从圆柱体转化为其**上下表面的圆周**，这对于计算遮蔽时间在计算量上是一个巨大的进步。

5.1.2 情况二

我们现在可以定义所谓的遮蔽发生在爆炸发生时刻 t_{112} 后的云团持续时间 $T_{团} = 20\,\mathrm{s}$ 内，所有满足以下条件的时刻：上下表面的圆周的所有点与导弹连线到云团中心的最大距离小于云团半径 $R_{团} = 10\,\mathrm{m}$。

圆周上单个点 X 与导弹 $X_{导\,1}$ 连线到云团中心 X_{11} 的距离 d 依据点线距离求法：

$$\begin{aligned} d &= \frac{\|(X_{导\,1} - X) \times (X_{11} - X)\|}{\|X_{导\,1} - X\|} \\ &= d(X) \end{aligned}$$

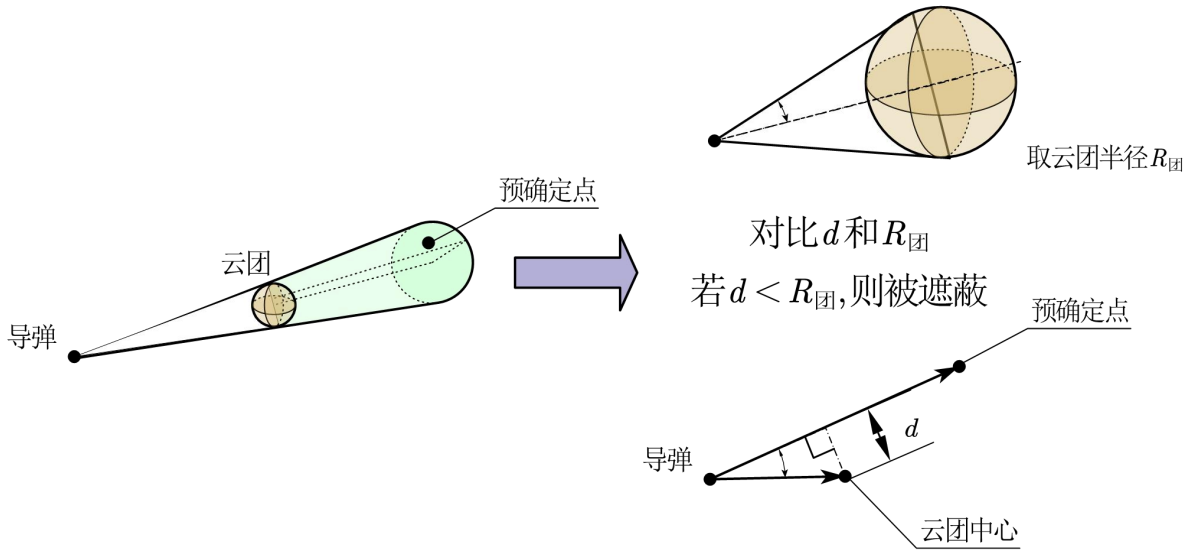


图 5　距离模型示意图

在得到单个点的距离 d 后，我们等步长遍历上下圆周（设分别为 O_1，O_2）的所有点，找到上述 d 的一个最大值 d_{max}：

$$d_{max} = \max_{X \in O_1 \cup O_2} \{d(X)\} \tag{3}$$

我们判断是否遮蔽的标准是：

$$d_{max} < R_{团} = 10\,\mathrm{m}$$

第二种情况是当导弹进入云团，视野被完全遮挡。这一小段时间内，真目标被有效遮蔽。

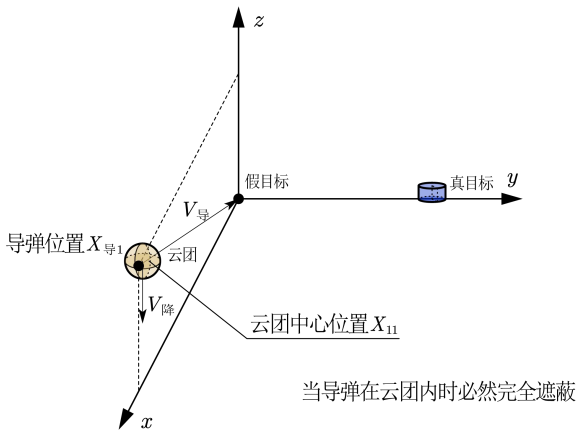


图 6　情况二示意图

注意到在这种情况下我们就无法像在第一种情况一样，做开圆锥体说明圆柱体和上下圆周的等价性，不过非常幸运的是，由于这种情况必然会导致遮蔽成功，我们参照第一种情况的最大距离计算方法也是成立的。

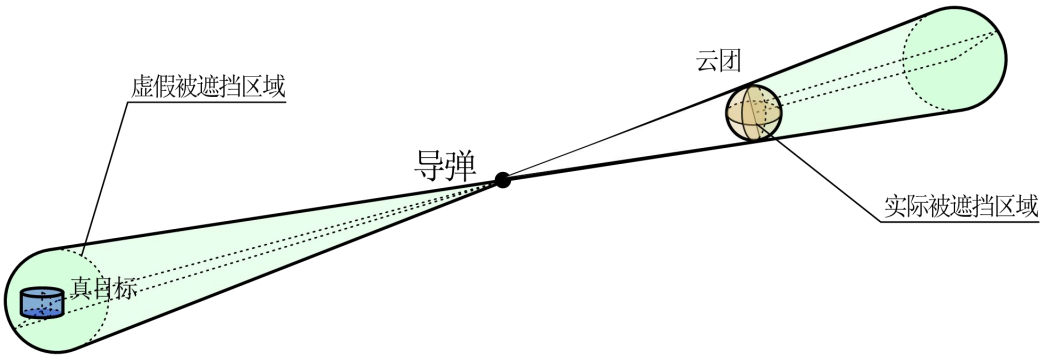
详细说明如下：显然在导弹在云团内时，从上下圆周的任意点出发，做和导弹的连线，此连线与与云团中心的距离必然小于等于导弹到云团中心的距离（点到线的距离必然小于等于点到线上一点的距离），而此时导弹到云团中心的距离由导弹在云团内知小于云团半径 $R_{团} = 10\,\mathrm{m}$，则知我们参照第一种情况的算法，此时 $d_{max} < R_{团} = 10\,\mathrm{m}$，和导弹被遮蔽这个事实相符合，所以说明这种算法在第二种情况是完全适用的。

至此，我们将判断是否遮蔽的标准是：

$$d_{max} < R_{团} = 10\,\mathrm{m}$$

完全推广到了所有情况。

以上算法还有一个细微问题，事实上，我们会漏掉一个关键情况，即我们这个投影可能不落在射线上，而是在反方向上，此时我们对计算点到线上的距离，会出现图中这样满足点到线距离小于云团半径的情况。



虽然我们实际遮蔽不到，但是由于距离模型会计算云团到视线射线反方向上的距离，距离模型会判断能够遮蔽

图 7　反方向上的距离计算

如图 7所示，我们在计算 d 时，会出现概率算的是云团 X_{11} 到视线反方向上的距离，而如果此距离小于云团半径 $R_{团}$，则依据我们算法的逻辑，实际不遮蔽的情况将会被判定为遮蔽。

距离模型的最大价值是引导我们发现这一类由于**视线的单向传播**所导致的问题，进而在这个模型的角度上微调，获得简单高效的新模型。

为了解决视线单向传播的问题，我们有以下两种方案：

- 计算云团中心到导弹和观测点连线的线段的距离
- 利用接下来的角度模型

从计算流程来看，角度指标模型需要额外讨论导弹进入云团的时间段，但这只是简单解一个一元二次函数即可得到。同时，在求解点到线段的模型中，也必然会分析点的投影是否在线段外，因此模型计算复杂度是一致的。

我们最终采用以下角度指标模型的原因正是因为我们可以**额外获得导弹是否进入云团的定性情况**，进而可以和我们的可视化结果相互佐证，为整个运动过程提供更多有价值的信息。

所以我们最后的选择是，在上述方法的基础上进行微调，使用角度进行精确估计。提出以下角度指标模型。

5.2 角度指标模型

首先讨论距离指标模型的第二种情况，从而确定我们导弹在球体中的时间段。通过 $X_{11}(t)$ 和 $X_{导\,1}(t)$ 直接计算满足式 (4):

$$||X_{11} - X_{导\,1}|| < R_{团} \tag{4}$$

两边平方求解式 (4) 得到二次函数，得到解为 $(t_{进\,111}, t_{出\,111})$（$t_{进\,ijn}$ 和 $t_{出\,ijn}$ 表示第 i 枚导弹相对于第 j 架飞机的第 n 个烟雾弹云团的进入和离开时间），此时间段即导弹在云团内的时间。

而在以上所求时间段之外，即 $t \in [0, t_{进\,111}] \cup [t_{出\,111}, T_{终\,1}]$，此时导弹在云团外，我们通过比较角度来确定是否被遮蔽。如图 8所示：

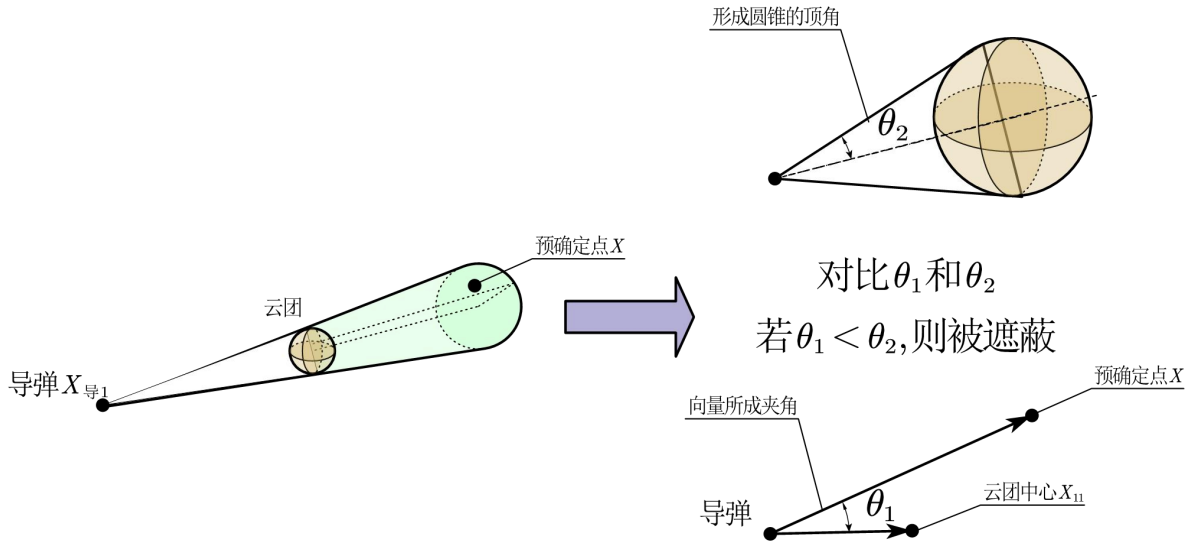


图 8　角度模型示意图

θ_1 为以导弹为起点，预估计点和云团为中心的两个向量所形成的夹角（θ_1 满足 $\theta_1 \in [0, \pi]$）：

$$\theta_1 = \arccos \frac{(X - X_{导\,1})(X_{11} - X_{导\,1})}{||X - X_{导\,1}|| \cdot ||X_{11} - X_{导\,1}||}$$

θ_2 为当导弹在云团外时，与云团相切的圆锥的顶角（θ_2 满足 $\theta_2 \in [0, \frac{\pi}{2}]$）：

$$\theta_2 = \arcsin \frac{R_{团}}{||X_{11} - X_{导\,1}||}$$

当出现类似图 7的情况时，$\theta_1 > \dfrac{\pi}{2}$，因此在出现类似图 7的情况时，$\theta_1 - \theta_2 > 0$ 且不遮蔽。而对于 θ_1 满足 $\theta_1 \in [0, \dfrac{\pi}{2}]$ 时，我们比较大小，知当 $\theta_1 < \theta_2$ 时被遮蔽，当

$\Delta_\theta = \theta_1 \geq \theta_2$ 时没有被遮蔽。以此我们可以将 θ_1 和 θ_2 的大小关系作为判断单点是否被遮蔽的依据。

自此我们引入一个新的衡量指标 Δ_{max}，在某个时刻 t 时，由于我们已经将圆柱模型简化为上下表面圆周的模型，因此遍历采样的难度大大下降，可以通过对**单个参数（圆周上的点所对应的圆周角 α）**，在 $[0, 2\pi]$ 进行采样，来获取对圆周上的不同点的遮蔽情况的角度大小关系，确定圆柱是否被完全遮住（我们可以参照距离指标模型中的证明，保证只需分析圆柱上下表面圆周 O_1，O_2）：

$$\begin{aligned}\Delta_{\max} &= \max_{X \in O_1 \cup O_2} \{\cos\theta_2(X,t) - \cos\theta_1(X,t)\} \\ &= \max_{X \in O_1 \cup O_2} \left\{ \sqrt{1 - \frac{R_{\text{团}}^2}{\|X_{11}(t) - X_{\text{导}1}(t)\|^2}} - \frac{(X - X_{\text{导}1}(t)) \cdot (X_{11}(t) - X_{\text{导}1}(t))}{\|X - X_{\text{导}1}(t)\| \cdot \|X_{11}(t) - X_{\text{导}1}(t)\|} \right\} \\ &= \Delta_{\max}(t) \end{aligned} \tag{5}$$

将式 (1) 和 (2) 中的 $X_{\text{导}1}(t)$ 和 $X_{11}(t)$ 代入从而计算单个时刻 $\Delta_{\max}$ 的值，被遮蔽判别条件为：

$$\Delta_{\max} < 0 \tag{6}$$

通过在 $[0, t_{\text{进}111}] \cup [t_{\text{出}111}, T_{\text{终}1}]$ 内分析 $\Delta_{\max}(t)$ 的正负情况判断是否被遮蔽，设得到时间段结果为 $T_{\text{遮}111}$（$T_{\text{遮}ijn}$ 表示第 i 个导弹被第 j 架飞机的第 n 个烟雾弹所遮时长）。

$$T_{\text{遮}111}$$

最终**总遮蔽时长 $T_{\text{总遮}111}$**：

$$T_{\text{总遮}111} = T_{\text{遮}111} + t_{\text{出}111} - t_{\text{进}111} \tag{7}$$

5.3 模型结果

经过实验，我们为了平衡精确性和复杂度，最后选用结果精度为 0.01 的步长对时间和圆周分别进行采样。

通过我们的角度指标模型和抽样算法，我们计算得到导弹 $M1$ 被第一个烟雾弹完全遮蔽的总时长 $T_{\text{总遮}111}$ 为：

$$T_{\text{总遮}111} = 1.40\,\text{s} \tag{8}$$

为了全面验证该结果的可靠性与模型的合理性，我们从三维轨迹仿真，关键指标时变分析和结果估计三个维度进行交叉验证。

5.3.1 三维轨迹仿真验证

我们在三维空间中对导弹、无人机及烟雾弹的完整运动轨迹进行了仿真，如图 9所示。

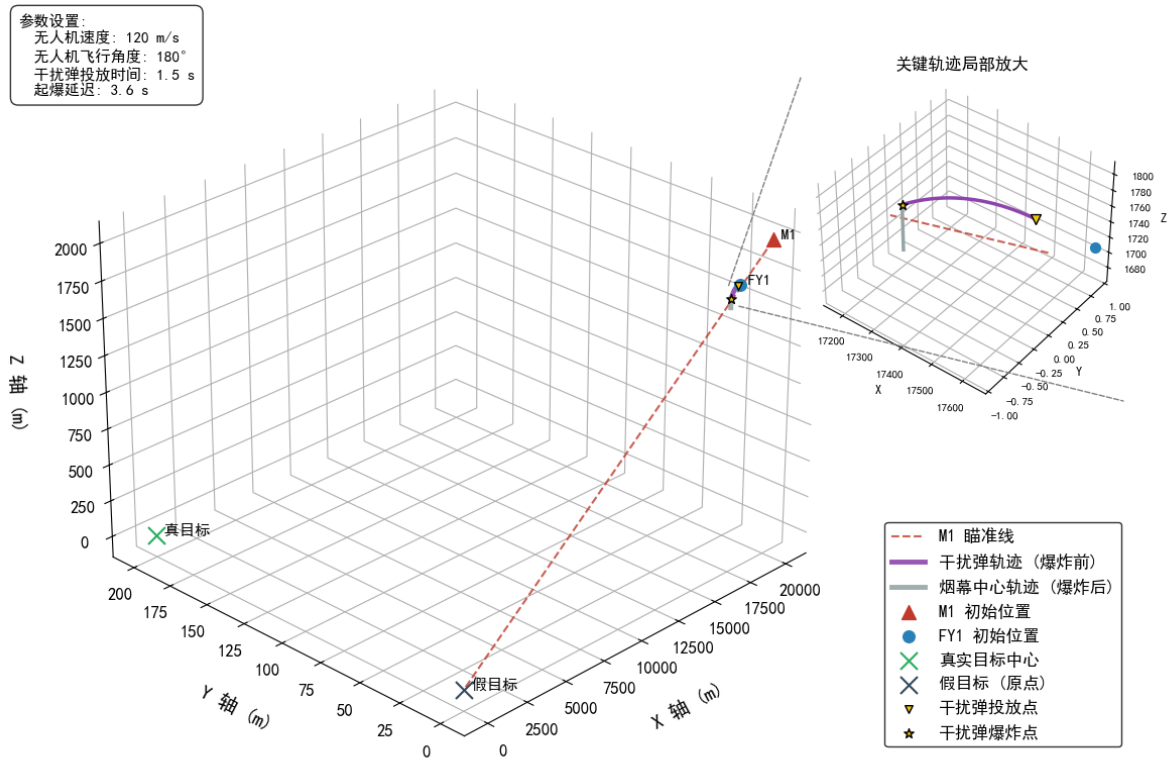


图 9　问题一结果仿真

从**仿真轨迹结果** (图 9) 来看，图中清晰地展示了无人机 $FY1$ 的飞行路径、干扰弹的投放点、爆炸点以及爆炸后烟雾云团的沉降轨迹。至关重要的是，图中导弹 $M1$ 的瞄准线（红色虚线）明确地穿过了烟雾云团的轨迹范围。局部放大图进一步揭示了这一关键细节：导弹的飞行路径确实与烟雾云团发生了空间上的重叠。

这一仿真结果直观地证实了我们模型分析中的一个重要前提——导弹确实会经历进入云团内部的阶段（即情况二）。这与我们通过解算距离不等式得到的 $t_{进\,111}$ 和 $t_{出\,111}$ 存在且 $t_{出\,111} - t_{进\,111} > 0$ 的计算结果完全吻合，从而有力地证明了我们模型对物理过程的描述是准确的。

5.3.2 关键指标时变分析

为了从数学上精确验证遮蔽过程，我们绘制了关键判别指标 Δ_{max} 随时间 t 变化的函数曲线，如图 10所示。

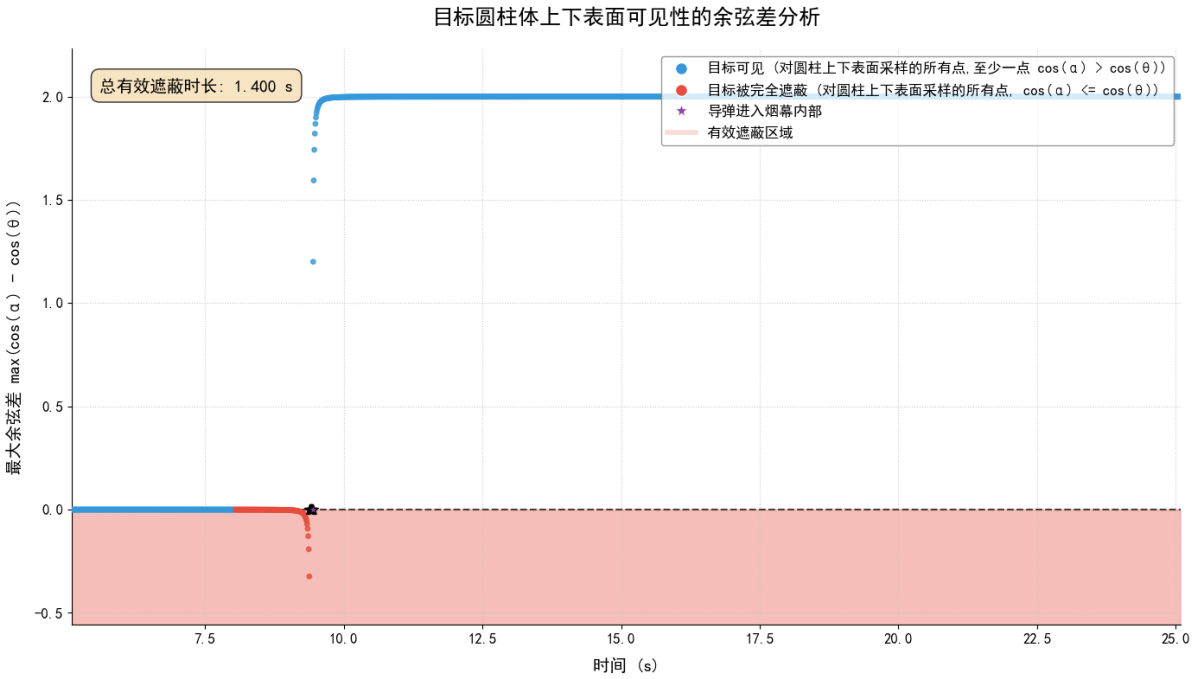


图 10　问题一指标变化曲线图

该图详细揭示了遮蔽状态的动态演化过程，其变化趋势与我们模型的物理过程高度吻合：

- **遮蔽前期:** 曲线位于 0 值上方（蓝色点），表明 $\Delta_{max} > 0$，目标完全可见。
- **锥体遮蔽期:** 曲线首次下穿 0 值（红色点），表示此刻由导弹视线与烟雾云团构成的遮蔽锥体开始完全覆盖目标圆柱，有效遮蔽开始。
- **内部遮蔽期:** 图中数据点出现了一段明显的“真空期”，并由紫色星号标记。这并非数据缺失，而是我们模型逻辑的完美体现。该时间段对应导弹进入云团内部，此时目标被完全遮挡，而我们的判别指标 Δ_{max} 定义于导弹在云团外的情况，因此不适用也无需计算。这一“真空期”的存在，再次验证了模型分情况讨论的正确性。
- **遮蔽后期:** 导弹飞出云团后，Δ_{max} 的值瞬间陡增至一个很大的正数。这精准地对应了我们模型中讨论的“反方向计算”情况：此时导弹已飞过目标，云团位于其后方，视角 $\theta_1 > \dfrac{\pi}{2}$，导致 $\cos\theta_1$ 为负，Δ_{max} 变得很大，正确地判别为不遮蔽。

整个曲线的变化生动地再现了可见 → 锥体遮蔽 → 内部遮蔽 → 飞越后可见的完整过程，总有效遮蔽时长（图中红色区域和紫色星号标记的内部遮蔽期）与我们的计算结果 1.40 s 完全一致，为模型的合理性提供了强有力的定量支持。

这里还观察到一个重要特性，在左端蓝色区间，模型判定此时无法遮蔽，但是此时判别指标 Δ_{max} 极度接近 0，这就意味着我们后续采用变步长搜索确定时间区间是不可行的，因为步长要根据逼近效果变化，而我们的状态始终保持极度接近。这也为我们最后采用 **Brent's Method** 来作为优化后确定时间区间的算法埋下伏笔。

5.3.3 估计值对比验证

为了进一步从侧面验证结果的合理性，我们进行了一个简化估算。我们仅考虑目标圆柱的中心点，计算其被遮蔽的总时长的估计值 $T'_{\text{总遮}\,111}$：

$$T'_{\text{总遮}\,111} = 1.44\,\text{s} \tag{9}$$

我们发现 $T'_{\text{总遮}\,111}$ 略大于 $T_{\text{总遮}\,111}$。这是符合物理直觉的：因为中心点是整个圆柱体的一部分，当整个圆柱体被遮蔽时，中心点必然被遮蔽，但存在某个时刻，可能只有中心点被遮蔽而圆柱边缘并未完全被遮蔽。因此，$T'_{\text{总遮}\,111} \geq T_{\text{总遮}\,111}$ 是合理的。同时，由于圆柱体本身的尺寸（半径 $R_{\text{柱}} = 7\,\text{m}$、高 $H_{\text{柱}} = 10\,\text{m}$）相对于整个运动空间和遮蔽范围而言并不大，两个时长的数值非常接近，这也从另一个角度说明了我们的角度指标模型具有优秀的稳定性和合理性。

六、 问题二模型建立与求解

6.1 多参数优化模型

本问题旨在寻找无人机 $FY1$ 的最佳机动和投放策略，以最大化对真目标的有效遮蔽时间。该策略由四个核心独立参数决定：无人机起始速度大小 $v_{\text{飞}\,1}$，飞行方向与 x 轴正向的夹角 γ，投放干扰弹的时刻 t_{111}，以及引爆干扰弹的时刻 t_{112}。因此，问题二等价为一个四维向量参数组 $\vec{X_2}$ 的优化问题：

$$\vec{X_2} = (v_{\text{飞}\,1}, \gamma, t_{111}, t_{112})$$

目标函数定义为干扰弹的有效遮蔽时间 $T_{\text{总遮}\,111}$，它是上述四个参数的函数：

$$T_{\text{总遮}\,111} = \vec{X_2} = T_{\text{总遮}\,111}(v_{\text{飞}\,1}, \gamma, t_{111}, t_{112}) \tag{10}$$

目标函数的计算方法完全沿用第一问中建立的角度指标模型。即把真目标被遮蔽等价为上下表面圆周被遮蔽，当满足式 (6) 中 $\Delta_{\max} < 0$ 时，则认为目标被完全遮蔽。

该优化问题具有以下两个重要特点：一方面，由于当无人机航向确定后，该路径上几乎总是能得到遮蔽时间的最值，所以优化过程的目标函数有相当多的局部极值；另一方面，由于无人机不同航线的遮蔽效果差异巨大，所以每一条航向上的局部极值之间的差距又会非常大。基于这些特性，我们选择采用粒子群优化（PSO）算法[1] 解决该问题。首先，粒子群算法由于初始粒子的分布随机性，具有良好的全局搜索能力。其次，粒子群算法中，每个粒子的运动变化除了参考自身优化历史的“个体认知权重”，还会参考其余粒子优化结果的“社会认知权重”；在本问题中，不同局部极值的巨大差距会直接影响到粒子群“社会认知权重”的大小，从而引导粒子群更快地朝着全局最优解收敛。

通过式 (2) 求解轨迹，我们通过改变参数 $\vec{X_2}$ 可以获得不同的运动轨迹。

下面讨论参数组 $\vec{X_2} = (v_{飞1}, \gamma, t_{111}, t_{112})$ 中各个参数的取值范围。值得注意的是，由于粒子群优化算法中“群体社会权重”的存在，初始表现不佳的粒子会朝着粒子群中表现好的粒子调整参数的各个分量，在有限次的迭代后，粒子会自发地将参数组的各个分量调整到合理的区间内。因此，我们仅需要对参数组各分量做简单的限制即可。

- **飞行速度** $v_{飞1}$: 根据问题设定，无人机本身有飞行速度限制，取值范围是 $[70\,\text{m/s}, 140\,\text{m/s}]$。
- **飞行角度** γ: 根据无人机和真目标、导弹的空间方位，我们将 γ 的初始搜索范围设定为 $[\frac{1}{4}\pi, \frac{5}{4}\pi]$，则飞行方向单位向量为 $(\cos\gamma, \sin\gamma, 0)$。
- **投放时刻** t_{111}: 投放时刻决定了无人机匀速直线运动的时间。考虑到导弹到达目标的时间约为 $T_{终1} = 66.999\,\text{s}$，我们将 t_{111} 的范围设定在一个较宽泛的区间 $[0\,\text{s}, 20\,\text{s}]$ 内。
- **引爆时刻** t_{112}: 引爆时刻 t_{112} 必须在投放时刻 t_{111} 之后。$t_{112} - t_{111}$ 的差值决定了平抛运动的时间。为了充分发挥干扰作用且避免干扰弹落地，结合第一问的经验，我们将 t_{112} 的范围设定为 $[t_{111}, t_{111} + 10\,\text{s}]$。

最终的参数约束设定如下：

$$\begin{cases} v_{飞1} \in [70\,\text{m/s}, 140\,\text{m/s}] \\ \gamma \in [\frac{1}{4}\pi, \frac{5}{4}\pi] \\ t_{111} \in [0\,\text{s}, 20\,\text{s}] \\ t_{112} \in [t_{111}, t_{111} + 10\,\text{s}] \end{cases} \tag{11}$$

在该问题中，基于粒子群算法的通常经验，将初始参数设为如下值：粒子数量 (100)，迭代次数 (500)，惯性权重 (0.9)，个体认知权重 (0.5)，群体社会权重 (0.3)。我们将参数约束范围式 (11) 代入，**目标函数设为有效遮蔽时间** $T_{总遮111}$，其最大值记为 $M_{总遮111}$ ($M_{总遮ijn}$ 表示在题设条件计算得到的最优遮挡情况下，第 i 个导弹被第 j 架飞机的第 n 个烟雾弹所遮时长)。粒子群算法详细流程见以下流程图。

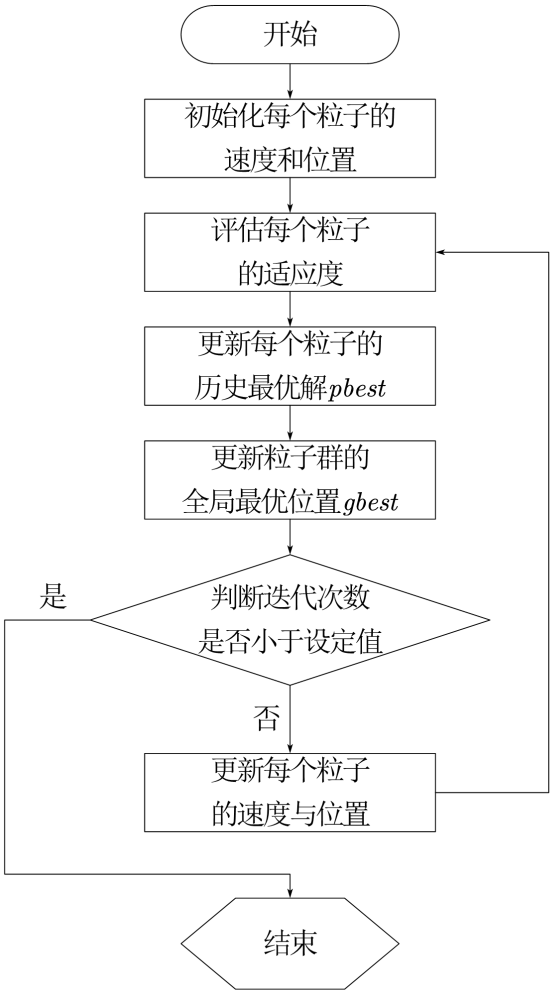


图 11　粒子群算法流程图

step1： 初始化粒子群。在解空间内随机生成 100 个粒子的初始位置和速度。计算每个粒子的适应度值，将当前位置设为该粒子的“个体历史最优位置（$pbest$）”，并从所有 $pbest$ 中选出最优者作为“全局最优位置（$gbest$）”。

step2： 进入迭代循环。设置当前迭代次数为 1，开始执行总共 500 次的迭代。

step3： 更新粒子状态。对于每一个粒子，依据其自身的 $pbest$ 和群体的 $gbest$，更新其速度和位置。

step4： 更新个体最优。计算每个粒子在新位置的适应度值。如果新位置的适应度优于其 $pbest$，则用新位置更新该粒子的 $pbest$。

step5： 更新全局最优。在所有粒子更新完各自的 $pbest$ 后，比较所有 $pbest$。如果发现某个 $pbest$ 优于当前的 $gbest$，则用这个 $pbest$ 更新 $gbest$。

step6： 判断终止条件。检查迭代次数是否达到 500 次。若未达到，则迭代次数加 1 并返回 **step3**；若已达到，则结束循环。

step7： 输出结果。算法终止，输出最终的全局最优位置 $gbest$ 作为问题的最优解。

6.2 模型结果

通过粒子群算法优化求解[2]，得到最大有效遮挡时间 $M_{\text{总遮}111}$ 为：

$$M_{\text{总遮}111} = 4.509\,\text{s} \tag{12}$$

取得最大遮蔽时间时对应的最优参数组及关键轨迹点坐标如表 1所示。

表 1　最优策略下的参数与关键节点坐标

最优飞行参数		关键事件时刻	
飞行速度	107.268 m/s	投出时刻	0.229 s
飞行角度	178.651°	引爆时刻	3.349 s
关键节点三维坐标 (X, Y, Z)			
投放点坐标	$(1.7775 \times 10^4,$	0.58,	$1.8000 \times 10^3)$
引爆点坐标	$(1.7441 \times 10^4,$	8.46,	$1.7523 \times 10^3)$

为了全面验证该优化结果的可靠性与模型的合理性，我们从三维轨迹仿真，参数敏感性分析和估计值对比验证三个维度进行深入验证。

6.2.1 三维仿真

我们将优化得到的最优参数组代入物理模型，进行三维可视化仿真，结果如图 12所示。

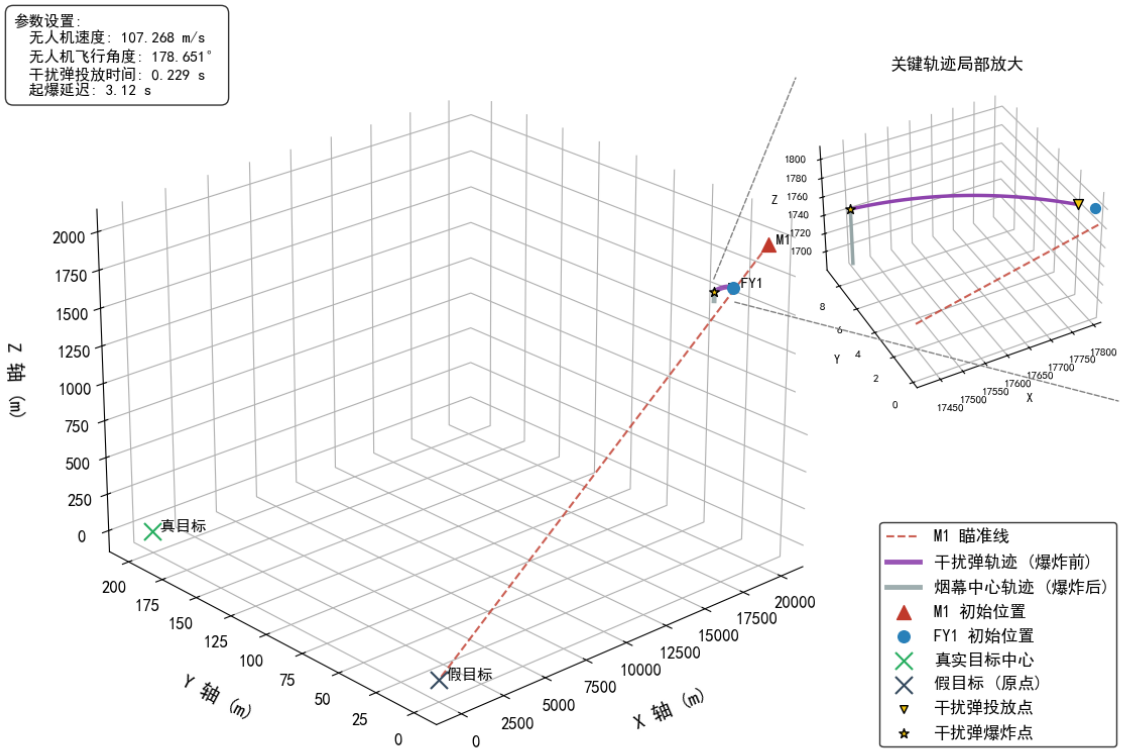


图 12　问题二结果仿真

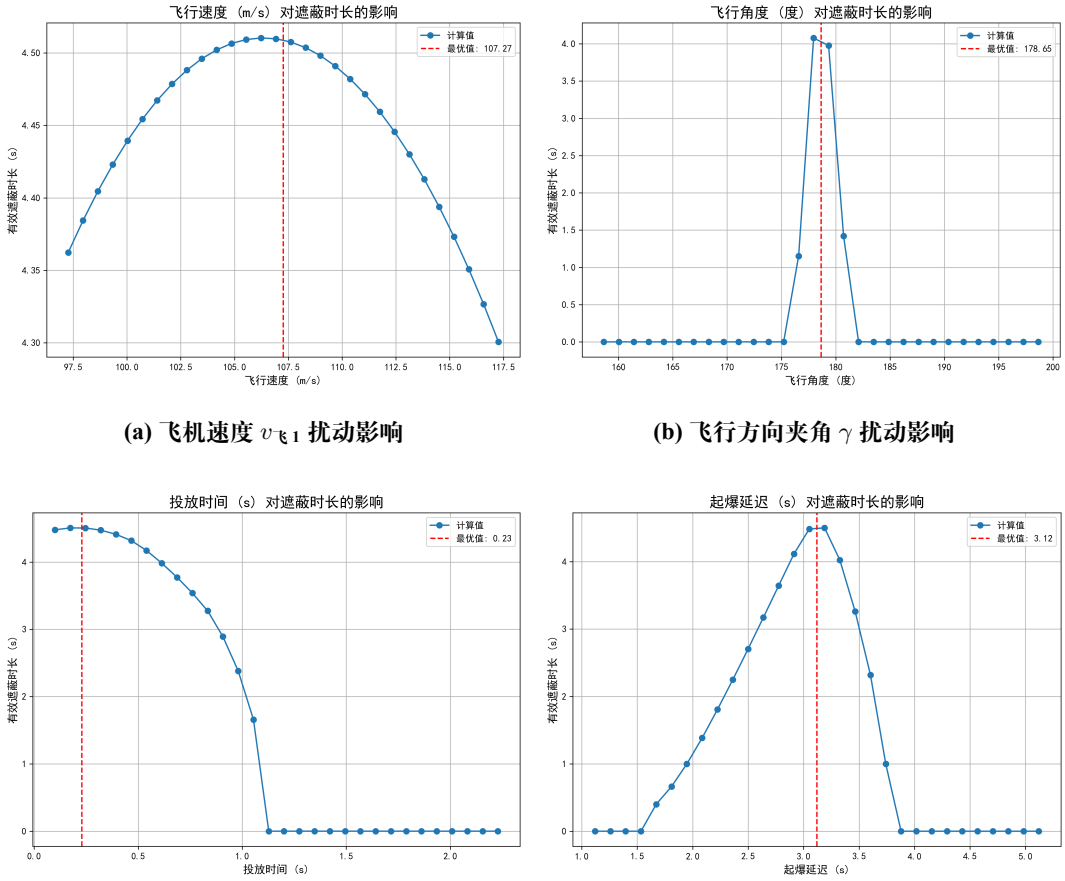
图 12直观地展示了粒子群算法找到的最优策略：

- **策略选择：**无人机 $FY1$ 并未选择复杂的机动，而是采取了近乎沿 x 轴负方向飞行的简洁航线（最优角度 178.651°），在飞行了极短的时间（0.229 s）后便果断投放了干扰弹。
- **精准拦截：**干扰弹经过 3.12 s 的平抛运动后在预定空域引爆，其后形成的烟雾云团（紫色轨迹）的沉降路径，精准地拦截在了导弹 $M1$ 飞向真目标的瞄准线（红色虚线）上。
- **关键细节：**右侧的“关键轨迹局部放大图”为我们提供了决定性的证据。它清晰地显示，导弹的飞行路径在某一时刻与烟雾云团的轨迹发生了空间上的重叠。这表明，在最优策略下，遮蔽是确实发生的物理事件，而非数值上的巧合。

该仿真结果不仅在几何上确认了遮蔽的可行性，更将抽象的四维优化参数转化为了一套清晰、可执行的运动轨迹，有力地证明了我们优化结果的现实意义和有效性。

6.2.2 参数敏感性分析

为了验证所求最优解的稳定性和局部最优性，我们以最优参数组为基准，对四个核心参数分别进行单变量扰动分析，结果如图 13所示。



(a) 飞机速度 $v_{飞1}$ 扰动影响　　**(b) 飞行方向夹角 γ 扰动影响**

(c) 投出时刻 t_{111} 扰动影响　　**(d) 引爆时刻 t_{112} 扰动影响**

图 13　参数扰动影响分析图

这组图像揭示了每个参数对遮蔽效果的影响程度：

- **全局最优性：**四幅子图（图 (13a) ~ (13d)）呈现出高度一致的规律：有效遮蔽时长均在粒子群算法给出的最优值（红色虚线所示）处达到峰值。任何对最优参数的单向偏离，都会导致遮蔽效果的下降。这强有力地证明了我们求得的解是一个高质量的局部最优解。
- **参数敏感度对比：**
  - **飞行角度 γ（图 13b）**的敏感度最高。其影响曲线呈现出一个极其尖锐的窄峰，角度偏离最优值仅 $1 \sim 2$ 度，有效遮蔽时间便会骤降为零。这说明从哪个方向接近是整个策略中最关键、最不容有失的环节。
  - **飞行速度 $v_{飞1}$（图 13a）**和**投放时刻 t_{111}（图 13c）**的影响曲线则相对平缓，呈现出光滑的类抛物线形状，表明在最优值附近存在一定的容错空间，但过大的偏离同样会导致效果显著下降。
  - **引爆时刻 t_{112}（图 13d）**的影响也十分显著，其决定了烟雾云团形成的高度和时机，同样需要精确控制。

6.2.3 估计值对比验证

为了从另一角度验证我们几何模型的合理性，我们再次采用中心点估算法。即放宽遮蔽条件，仅要求目标圆柱的中心点被遮蔽即可。使用调整后的目标函数重新进行粒子群优化，得到估算的最佳遮蔽时长 $M'_{总遮\,111}$ 为：

$$M'_{总遮\,111} = 4.690\,\mathrm{s} \tag{13}$$

将两种参考点选取方式得到的结果对比如表 2所示。

表 2　不同参考点下的优化结果对比

参数	模型（上下圆周）	估算模型（中心点）
有效遮蔽时间	**4.509** s	**4.690** s
飞行速度	107.268 m/s	93.537 m/s
飞行角度	178.651°	178.428°
投出时刻	0.229 s	0.364 s
引爆时刻	3.349 s	3.365 s
投放点 X 坐标	1.7775×10^4	1.7766×10^4
投放点 Y 坐标	0.58	0.93
投放点 Z 坐标	1.8000×10^3	1.8000×10^3
引爆点 X 坐标	1.7441×10^4	1.7485×10^4
引爆点 Y 坐标	8.46	8.63
引爆点 Z 坐标	1.7523×10^3	1.7559×10^3

对比结果显示，估算模型的遮蔽时间 $4.690\,s$ 略大于我们精确模型的 $4.509\,s$。这完全符合逻辑预期：因为要求整个圆柱体被遮蔽（我们的模型）的条件，远比只要求中心点被遮蔽（估算模型）要苛刻，因此时长更短是合理的。同时，两种模型得到的最优参数组（尤其是最敏感的飞行角度）非常接近，这从另一个角度强有力地佐证了我们所构建的上下圆周等价模型的稳定性和可行性。

七、 问题三模型建立与求解

7.1 简化点模型

不同于问题一、问题二，从本问题开始，单个无人机搭载的干扰弹数目、无人机数目、导弹数目将会逐步增加，使计算过程逐渐复杂化。为了减轻计算压力，同时保证结果的准确性，我们首先建立一个合理的简化模型。

在所有问题中，目标函数均涉及单个干扰弹对某个导弹的有效遮蔽时间的计算。在问题一中，我们从几何角度证明了“圆柱体在导弹的视线盲区内等价于圆柱体的上下表面圆周在导弹的视线盲区内”，随后在问题一和问题二结合算法的具体数字模拟实验中，我们进一步佐证了该结论的合理性。在这个过程中，我们观察到，仅仅以“真目标圆柱体”的中心点作为参考点进行计算时，得到的结果与完备考量整个真目标圆柱体得到的精确结果十分接近。我们将前两问的对比结果列在下表，其中相对误差定义为 $\eta = \frac{|T_{\text{中心点}} - T_{\text{上下圆周}}|}{T_{\text{上下圆周}}}$（参考上下表面圆周得到 $T_{\text{上下圆周}}$，参考圆柱体中心点得到 $T_{\text{中心点}}$）。

- **情形一（问题一固定参数）**：$FY1$ 速度大小为 120 m/s，方向为水平面内朝向 z 轴，投放干扰弹的时刻 $t_{111}=1.5\,\mathrm{s}$，引爆前间隔时间为 $t_{112}-t_{111}=3.6\,\mathrm{s}$。
- **情形二（问题二优化结果）**：$FY1$ 采用粒子群优化后的最优参数，计算最佳遮蔽时长。

表 3　简化计算方法与精确方法对比

情形	参考上下表面圆周模型 $T_{\text{上下圆周}}$	参考圆柱体中心点模型 $T_{\text{中心点}}$	相对误差 η
情形一	1.40 s	1.44 s	2.86%
情形二	4.509 s	4.690 s	4.01%

这启示我们，可以将判断过程简化为仅将真目标视作一个点来处理，这会大大降低目标函数的计算复杂度。下面我们将从定性、定量两个角度来验证这种简化处理的合理性。

首先，我们进行定性分析。在烟雾云团下落的有效 20 s 内，导弹视线盲区在目标所在水平面上的截面通常是一个面积较大的椭圆。相对于这个椭圆，目标圆柱的上下表面（半径 7 m）所占面积很小，可以近似为一个点。此外，目标的竖直高度仅为 10 m，而导弹所处高度在 2000 m 左右，巨大的高差使得导弹观察目标上下表面的视角差极小。因此，无论从水平截面还是竖直高度来看，将真目标近似为一个点是合理的。

为了进一步说明这种近似不会造成不可接受的误差，我们进行了多组模拟实验，定量说明两种计算方法的相对误差普遍很小。

表 4　多组随机参数下两种模型的误差分析

参数组 $(v_{\text{飞}1},\gamma,t_{111},t_{112}-t_{111})$	圆柱体模型 $T_{\text{上下圆周}}$	简化点模型 $T_{\text{中心点}}$	相对误差 η
$(107.27, 178.65, 0.23, 3.12)$	4.509 s	4.675 s	3.68%
$(109.90, 178.95, 1.05, 3.59)$	4.266 s	4.272 s	1.41%
$(103.89, 179.13, 1.48, 3.62)$	3.990 s	4.061 s	1.78%
$(110.96, 179.70, 1.54, 3.99)$	3.789 s	3.915 s	3.33%
$(94.97, 179.35, 0.89, 3.51)$	3.667 s	3.689 s	6.00%

从上表可见，在多种不同的初始参数下，两种模型得到的遮蔽时间非常接近，相对误差绝大多数在 5% 以内，最大也仅为 6%，属于可接受范围。因此，在后续更复杂的

问题中，我们将采用简化点模型进行计算。

7.2 多参数优化模型

与第二问思路类似，本题可看作一个优化问题。问题三中，待优化参数向量 $\vec{X_3}$ 为一个八维向量，由无人机的飞行速度、飞行角度以及三枚干扰弹各自的投放与引爆时刻共 8 个独立标量构成：

$$\vec{X_3} = (v_{飞1}, \gamma, t_{111}, t_{112}, t_{121}, t_{122}, t_{131}, t_{132}) \tag{14}$$

执行粒子群优化算法时，需满足三枚干扰弹两两之间的投放时间间隔大于 1 s 的条件：

$$\begin{cases} t_{121} - t_{111} \geq 1\,\text{s} \\ t_{131} - t_{121} \geq 1\,\text{s} \end{cases}$$

结合问题二的分析，我们将完整的约束条件设定如下：

$$\begin{cases} v_{飞1} \in [70\,\text{m/s}, 140\,\text{m/s}] \\ \gamma \in [\frac{1}{4}\pi, \frac{5}{4}\pi] \\ t_{111}, t_{121}, t_{131} \in [0\,\text{s}, 20\,\text{s}] \\ t_{1k2} \in [t_{1k1}, t_{1k1} + 10\,\text{s}], \quad k = 1, 2, 3 \\ t_{121} - t_{111} \geq 1\,\text{s} \\ t_{131} - t_{121} \geq 1\,\text{s} \end{cases} \tag{15}$$

目标函数设为三枚干扰弹对真目标点总的有效遮蔽时间。设三枚烟雾弹对导弹的遮蔽时间段分别为区间 D_1, D_2, D_3，则优化目标是使下述函数取最大值（$||\cdot||$ 表示对时间区间的并集求测度）：

$$f(\vec{X_3}) = \|D_1 \cup D_2 \cup D_3\| \tag{16}$$

设定算法参数如下：粒子数量 (150)，迭代次数 (500)，惯性权重 (0.9)，个体认知权重 (0.5)，群体社会权重 (0.3)。

7.3 模型结果

代码运行后，得到的最优策略参数如表 5所示。

表 5　问题三最优策略参数

$v_{飞1}$	γ	t_{111}	t_{112}	t_{121}	t_{122}	t_{131}	t_{132}	**总遮蔽时间**
126.0 m/s	179.5°	0.630 s	4.432 s	3.982 s	9.104 s	8.826 s	19.735 s	**6.40** s

7.3.1 三维仿真

为直观理解该策略，我们进行三维仿真，得到图 14。

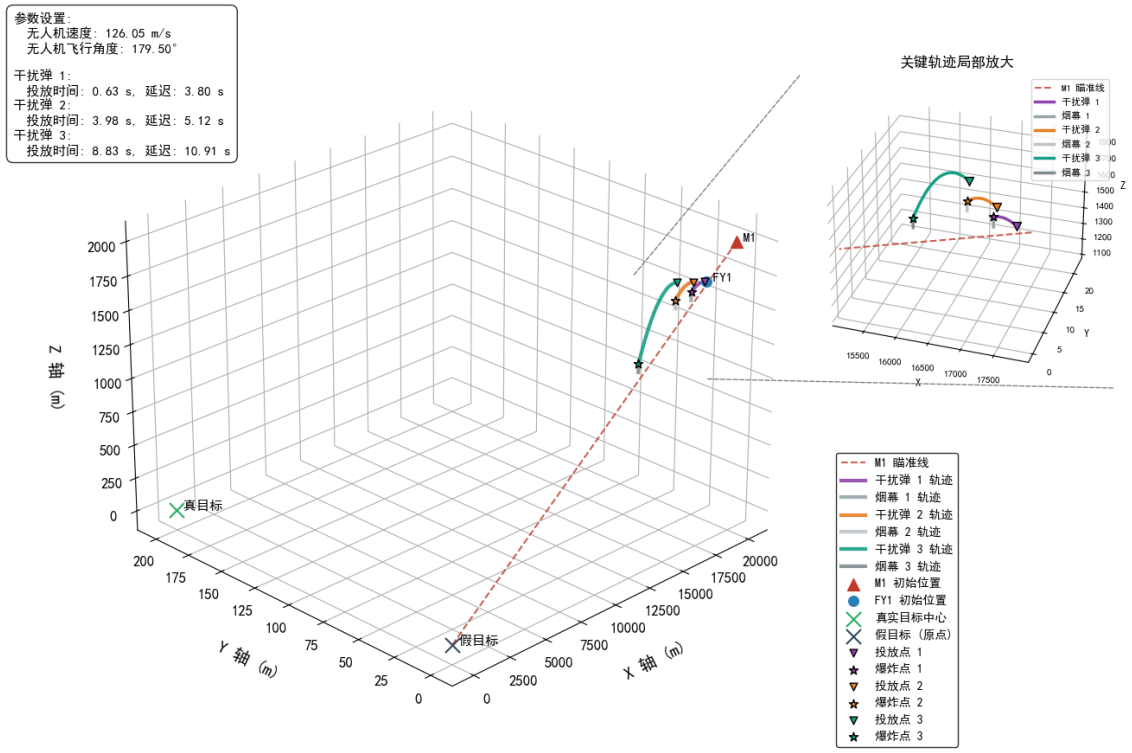


图 14　问题三结果仿真

为直观地理解算法所给出的最优策略，我们对其进行了三维可视化仿真，其结果如图 14所示。该图深刻地揭示了粒子群算法在复杂约束下寻得的精妙解法：

- **接力遮蔽与策略性放弃：** 图中右侧的关键轨迹局部放大图是整个策略的精髓所在。我们可以清晰地看到，前两枚干扰弹形成的烟雾轨迹（紫色与橙色实线）被算法完美地部署在了导弹的瞄准线（红色虚线）上，形成了一段无缝衔接的**“接力式”遮蔽**。然而，第三枚干扰弹的烟雾轨迹（青色实线）却远远偏离了这一关键拦截空域。这并非算法的失误，而是一种**策略性的放弃**。
- **时空协同：** 左上角的参数列表揭示了这一策略的数值基础。算法为前两枚弹精心计算了投放与引爆时刻，使得第一枚弹的烟雾云团即将消散时，第二枚弹的云团恰好“接力”跟上，从而最大化了连续遮蔽时间。
- **反直觉的全局最优：** 这一仿真结果直观地印证了我们之前的数值分析：为了追求总遮蔽时间的最大化，最优解并非让三枚弹“雨露均沾”，而是集中所有资源优化前两枚弹的效能。强行使用第三枚弹，反而会破坏前两枚弹已达成的精妙时空协同，导致总效果下降。

综上，图 14不仅从几何上验证了我们优化结果的正确性，更将抽象的数值解转化

为了一套清晰、可执行的运动轨迹，生动地展示了在复杂多体协同问题中，真正的全局最优解往往是深刻且反直觉的。

7.3.2 跳出局部优化的对比实验

从数值结果看，粒子群优化算法给出的最长遮蔽时间中，干扰弹 3 的遮蔽时长为 0 s。为验证这并非算法陷入局部最优，而是真实的全局最优策略，我们设计了对比实验。

出现此结果仅有两种可能：一是算法在权衡后，发现牺牲干扰弹 3 能为前两枚弹换取更优的参数空间，从而获得更长的总遮蔽时间；二是算法因随机初始化等因素陷入局部最优。

为了验证具体原因，我们微调了粒子群算法，在目标函数中加入奖励项，强制要求干扰弹 3 的有效遮蔽时间必须大于 0。在该约束下重新寻优，得到的结果如下：

表 6　强制第三枚干扰弹生效的优化结果

$v'_{飞1}$	γ'	t'_{111}	t'_{112}	t'_{121}	t'_{122}	t'_{131}	t'_{132}	**总遮蔽时间**
88.4 m/s	179.1°	0.574 s	3.844 s	1.989 s	5.518 s	3.824 s	8.002 s	**5.88** s

从表 6数据可以看出，当强行要求干扰弹 3 生效后，得到的最优解变为 5.88 s，相较于原算法的 6.40 s 减少了约 8.13%，这是相当大的性能损失。这说明，原算法确实考虑过让干扰弹 3 生效的策略，但这些策略在迭代过程中因性能不佳而被淘汰。最终收敛于放弃第三枚干扰弹的策略，是真正的全局最优选择。

这种现象本质上揭示了这样一个事实：如果强行让干扰弹 3 发挥作用，会严重干扰前两枚弹的最优时空布局；而干扰弹 3 带来的微小收益，远不足以弥补前两枚弹因错过最佳窗口期而造成的巨大损失。这一点从两组参数的巨大差异中也能看出，强制方案中所有参数都发生了剧烈变化，表明系统为了迁就第三枚弹，不得不放弃了原有的高效策略，也为我们验证粒子群算法没有陷入局部最优。

八、 问题四模型建立与求解

与前述问题类似，本问可看作一个多维优化问题，且粒子群优化算法的适用性依然成立。问题四的复杂度显著提升，涉及三架无人机协同作用，每架携带一枚干扰弹。因此，待优化参数为一个十二维向量 $\vec{X_4}$。

8.1 Brent's Method 算法

我们将优化向量 $\vec{X_4}$ 定义为由三架无人机各自的飞行速度、飞行角度、干扰弹投放时刻与引爆时刻共 12 个独立标量构成：

$$\vec{X_4} = (v_{飞1}, \gamma_1, t_{111}, t_{112}, v_{飞2}, \gamma_2, t_{211}, t_{212}, v_{飞3}, \gamma_3, t_{311}, t_{312}) \tag{17}$$

其中，$v_{飞i}$、γ_i、t_{i11}、t_{i12} 分别代表无人机 FYi 的速度、角度、投弹时刻和引爆时刻。

由于参数维度增加，计算压力剧增。为此，我们对目标函数中计算遮蔽时长的核心算法进行了优化。原算法采用固定小步长（如 $\delta t = 0.01\,\mathrm{s}$）遍历时间窗口，判断每个时间点是否被遮蔽。我们观察到，判别函数 $\Delta_{max}(t)$（判断是否遮蔽）随时间变化平稳，变号次数有限，函数出现先递减再递增的趋势。

受此启发，我们引入了更高效的求根策略：首先用较大步长进行粗略扫描，快速锁定 $\Delta_{max}(t)$ 函数值变号的区间；然后，在这些区间内，采用结合了二分法、割线法和逆二次插值思想的 **Brent's Method** 来快速、精确地定位函数的零点（即遮蔽开始和结束的精确时刻）。得到精确的零点后，即可计算出总遮蔽时长 $T_{总遮\,ijn}$。

此算法优化带来了双重优势：

- **效率提升：**大幅减少了需要计算 $\Delta_{max}(t)$ 的次数。尽管增加了 **Brent's Method** 的计算，但其对数级的时间复杂度 $O(\ln \frac{b-a}{\epsilon})$ 远低于遍历法，显著提升了整体运算速率。
- **精度提高：**原算法存在与步长同量级（$0.01\,\mathrm{s}$）的误差，而 **Brent's Method** 可将零点定位的精度提升数个数量级，从而得到更精确的遮蔽时长。

8.2 多参数优化模型

根据问题设定，我们将各优化参数的约束条件设定如下：

$$\begin{cases} v_{飞i} \in [70\,\mathrm{m/s}, 140\,\mathrm{m/s}] \\ \gamma_i \in [0, 2\pi] \\ t_{i11} \in [0\,\mathrm{s}, 20\,\mathrm{s}] \\ t_{i12} \in [t_{i11}, t_{i11} + 10\,\mathrm{s}] \end{cases} \qquad i = 1, 2, 3 \tag{18}$$

优化目标为最大化三枚干扰弹产生的总遮蔽时间，即使得以下值 $f(\vec{X_4})$ 最大化（D_i' 为第 i 枚干扰弹对导弹的遮蔽时间区间）：

$$f(\vec{X_4}) = ||D_1' \cup D_2' \cup D_3'|| \tag{19}$$

8.3 模型结果

经过粒子群算法优化求解，我们得到了最优策略下的各无人机任务执行情况。各机贡献的有效遮蔽时长分别为：$T_{\text{总遮}\,111} = 4.616\,\text{s}$，$T_{\text{总遮}\,121} = 4.798\,\text{s}$，$T_{\text{总遮}\,131} = 5.411\,\text{s}$。

由于 $FY1$ 与 $FY2$ 的遮蔽时间段被算法优化为完全不重合的接力模式，因此总有效干扰时长为两者之和，式 (19) 最大值为：

$$\max f(\vec{X_4}) = 14.825\,\text{s}$$

8.3.1 三维仿真

为直观地理解算法给出的最优策略，我们对其进行了三维可视化仿真，其结果如图 15所示。该图深刻地揭示了算法在更高维度的参数空间中，为实现总遮蔽时间最大化而寻得的精妙协同策略：

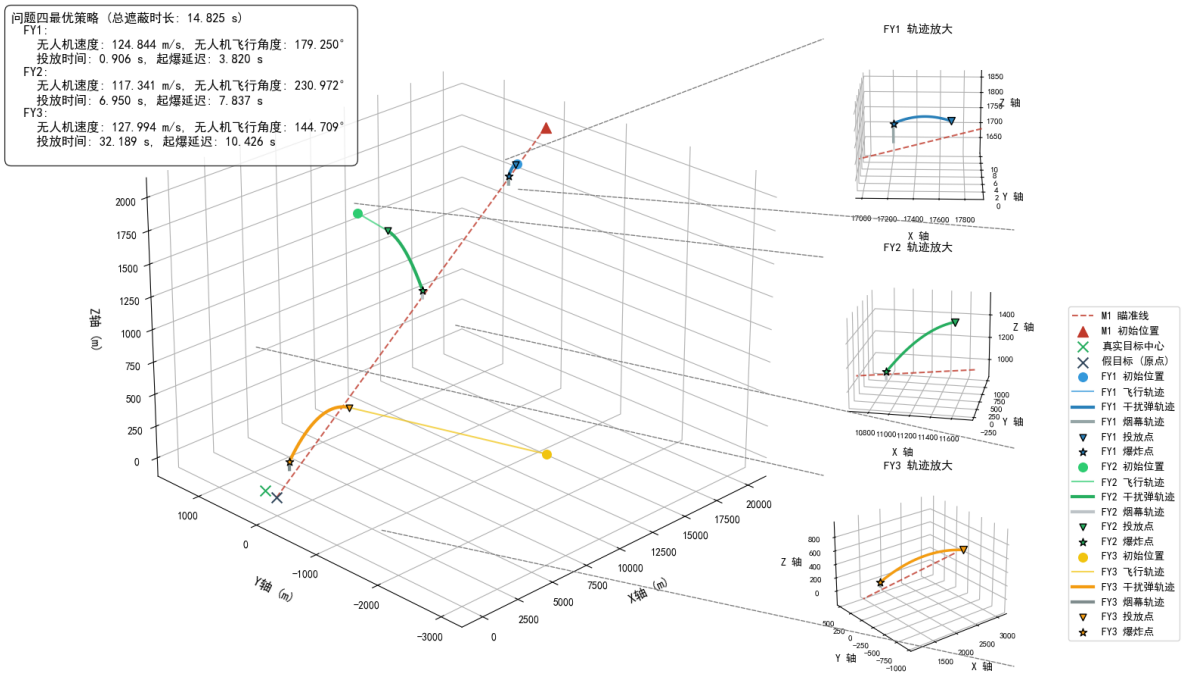


图 15　问题四结果仿真

核心策略：三机时空接力，分段精准遮蔽。与问题三放弃一枚干扰弹的策略不同，在问题四中，三架无人机（$FY1, FY2, FY3$）均发挥了关键作用。它们并非在同一区域进行重叠干扰，而是执行了一套精密的时空接力策略，在导弹飞行的不同阶段，于不同空间位置上依次形成烟幕屏障。

任务分工明确：

- $FY1$（**蓝色轨迹**）与 $FY2$（**绿色轨迹**）负责早期遮蔽。根据左上角数据，它们的投放时间分别为 0.906 s 和 6.950 s。从轨迹图上看，它们在相对靠近原点的位置形成了烟幕，用于干扰导弹飞行的初始阶段。

- $FY3$（**橙色轨迹**）则负责后期遮蔽。其投放时间为 32.189 s，远晚于前两者。从三维图和 $FY3$ 轨迹放大图可以看出，它的飞行路径和爆炸点（星形标记）位于 X 轴更远的位置，这正是为了在导弹飞行后期，当其瞄准线（红色虚线）移动到新的空间位置时，进行精准拦截。

遮蔽效果的完美叠加：文中指出总遮蔽时长为 **14.825** s。结合模型求解数据（各机贡献的有效遮蔽时长分别为 $T_{\text{总遮111}} = 4.616\,\text{s}$，$T_{\text{总遮121}} = 4.798\,\text{s}$，$T_{\text{总遮131}} = 5.411\,\text{s}$），我们可以发现三者之和恰好等于总时长：

$$4.616\,\text{s} + 4.798\,\text{s} + 5.411\,\text{s} = 14.825\,\text{s}$$

这完美地印证了三枚干扰弹的遮蔽时间段被算法优化为**完全不重合的无缝衔接**，从而使总时长等于各分段时长之和，实现了效率的最大化。

综上所述，图 15生动地展示了一套高度复杂的协同方案。算法通过精确计算三架无人机各自的速度、角度、投放及引爆时间，使得三片烟幕在时间和空间上完美接力，对导弹的整个瞄准路径进行了分段式、连续性的有效遮蔽，最终达成了总遮蔽时间的最大化。

九、 问题五模型建立与求解

9.1 分而治之的贪心算法

本问题中，存在五架无人机，每架无人机有三枚干扰弹，需要考虑这些干扰弹对三枚不同位置的导弹的联合干扰效果。如果仍然采用原先的粒子群优化算法试图找到全局最优解，那么待优化参数组为以下五个八维向量，分别对应五架飞机，共 40 个参数。

$$\begin{cases} \vec{Y_1} = (v_{飞1}, \vec{\gamma_1}, t_{111}, (t_{112} - t_{111}), t_{121}, (t_{122} - t_{121}), t_{131}, (t_{132} - t_{131})) \\ \vec{Y_2} = (v_{飞2}, \vec{\gamma_2}, t_{211}, (t_{212} - t_{211}), t_{221}, (t_{222} - t_{221}), t_{231}, (t_{232} - t_{231})) \\ \vec{Y_3} = (v_{飞3}, \vec{\gamma_3}, t_{311}, (t_{312} - t_{311}), t_{321}, (t_{322} - t_{321}), t_{331}, (t_{332} - t_{331})) \\ \vec{Y_4} = (v_{飞4}, \vec{\gamma_4}, t_{411}, (t_{412} - t_{411}), t_{421}, (t_{422} - t_{421}), t_{431}, (t_{432} - t_{431})) \\ \vec{Y_5} = (v_{飞5}, \vec{\gamma_5}, t_{511}, (t_{512} - t_{511}), t_{521}, (t_{522} - t_{521}), t_{531}, (t_{532} - t_{531})) \end{cases} \tag{20}$$

同时无人机 FY_i 的第 j 枚干扰弹对第 n 颗导弹的最佳遮蔽时间段为 $g_{nj}(\vec{Y_i})$，需要分别计算 $i = 1, \ldots, 5$，$j = 1, 2, 3$，$n = 1, 2, 3$，即共有 45 种情况：

$$5 \cdot 3 \cdot 3 = 45\text{种}$$

得到以上 45 个时间段后，进行复杂的并、交计算，最终返回最优参数组合。

$$\Psi = (\vec{Y_1}, \vec{Y_2}, \vec{Y_3}, \vec{Y_4}, \vec{Y_5}) \tag{21}$$

此外，还需要添加各个参数的独立约束条件与参数之间的联合约束条件。可见，试图通过传统的粒子群优化算法来得到该问题的全局最优解几乎是不可能的。

本文采取的办法是，基于前四问的先验经验，拟定分配策略的原则，再结合贪心算法的思路，分阶段、分目标对如此庞大的问题分而治之；这样既能将计算量控制在可操作范围内，又能确保所给出策略将多架无人机、多干扰弹对多导弹的协同遮蔽时长最大化。

9.2 模型结果

Step 1：分析每一架无人机的第一枚干扰弹对各导弹的遮蔽效果。

根据前面四问，我们得到一个普适性的原则：同一架无人机的三枚干扰弹对同一枚导弹的有效遮蔽时长是递减的。换言之，第一枚干扰弹的有效作用总是最显著的。

类似问题二，我们分别考量每一架无人机的第一枚干扰弹针对不同导弹的有效遮蔽时间段与遮蔽时长，结果如表 7所示：

表 7　各无人机首枚干扰弹对各导弹的潜在最大遮蔽效果

	导弹 M_1		导弹 M_2		导弹 M_3	
无人机	时间段 (s)	时长 (s)	时间段 (s)	时长 (s)	时间段 (s)	时长 (s)
FY_1	$[3.711, 8.376]$	4.665	—	0	—	0
FY_2	$[13.387, 18.224]$	4.836	$[8.806, 13.636]$	4.830	$[16.813, 20.681]$	3.868
FY_3	$[27.922, 33.410]$	5.488	$[26.834, 31.908]$	5.074	$[21.880, 26.717]$	4.837
FY_4	$[17.494, 22.245]$	4.751	$[18.023, 22.544]$	4.520	$[17.425, 21.245]$	3.821
FY_5	$[16.609, 20.977]$	4.369	$[22.162, 25.435]$	3.273	$[15.438, 19.266]$	3.828

可以看出，FY_1 的第一枚干扰弹无法对导弹 M_2、M_3 形成有效遮挡，则显然，其第二枚、第三枚干扰弹也无法对导弹 M_2 或 M_3 产生显著影响。因此我们将 FY_1 的作用特化为遮挡 M_1，即 FY_1 的三枚干扰弹全程只针对导弹 M_1 进行有效遮挡。于是，结合问题三的已知结果，可以立刻获得 FY_1 全程的投放策略。

将已知信息可视化表示如图 16：

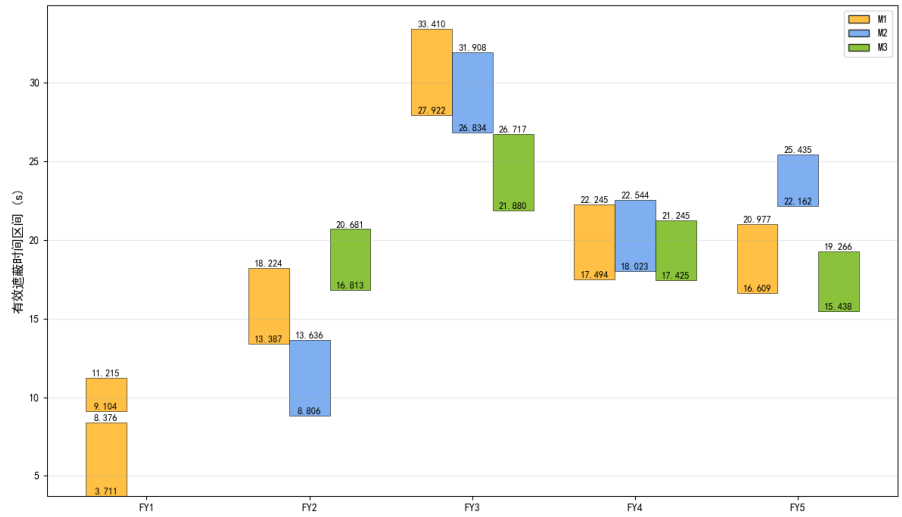


图 16　初步规划

对于剩余导弹的策略安排，是一个简单的规划问题。值得注意的是，我们在规划过程中加入了约束条件：每个导弹的被遮蔽时长必须大于 0，该规划问题可以通过下式描述，其中 T_{total} 为规划目标，希望求得其最大值。

$$T_{\text{total}} = T_{M_1} + T_{M_2} + T_{M_3} \tag{22}$$

$$\text{其中} T_{M_n} = \left|\left| \bigcup_{k \in \mathcal{K}_n} D_{k,n} \right|\right|, \quad \forall n \in \{1,2,3\}$$

（其中，T_{M_n} 表示对导弹 M_n 的总有效遮蔽时长；$\mathcal{K}_n$ 是分配给导弹 M_n 的干扰弹集合；$D_{k,n}$ 是干扰弹 k 为导弹 M_n 提供的有效遮蔽时间段（区间））

最终得到的结果如表 8所示：

表 8　第一枚干扰弹投掷策略

	初始速度	初始角度	投出时刻	引爆时刻	干扰目标
FY_1	89.891	178.667	0.644	3.711	M_1
FY_2	111.279	274.042	5.340	8.769	M_2
FY_3	128.141	119.249	20.584	27.916	M_1
FY_4	125.164	243.714	4.647	17.311	M_1
FY_5	128.527	120.238	12.588	14.495	M_3

目前，已经规划好 FY_1 三枚导弹和 FY_2、FY_3、FY_4、FY_5 第一枚导弹的投放、引爆方式。将目前的结果进行可视化表示如图 17所示，其中不同导弹用不同的颜色表示，纵轴为时间轴，色块表示对应导弹被有效遮蔽时间段。

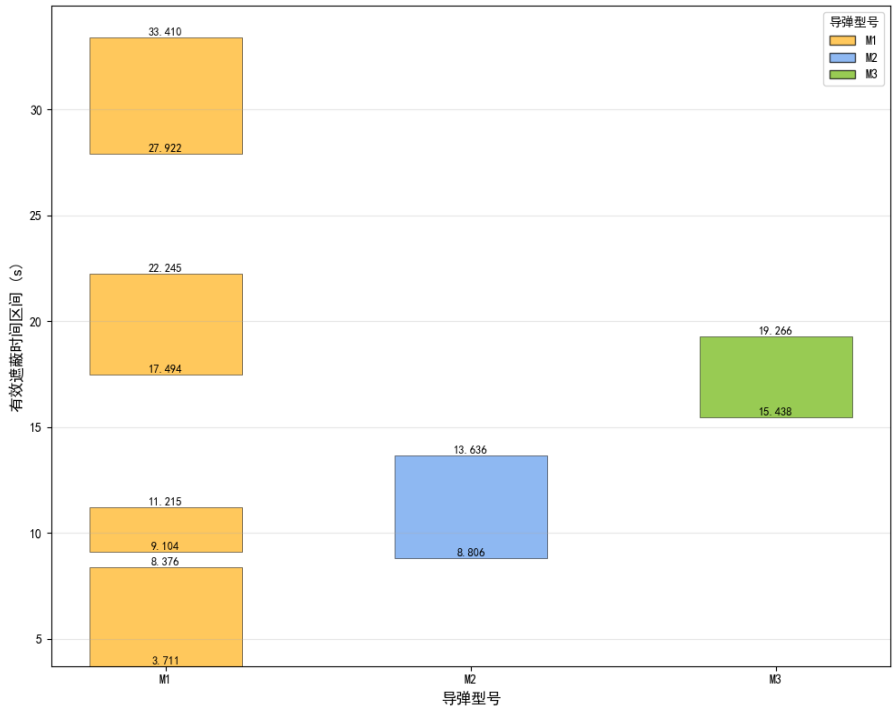


图 17　规划后示意图

Step 2：分析每一架无人机的第二枚干扰弹对各导弹的遮蔽效果。

在 $Step1$ 中，我们指出任何无人机的干扰弹都是第一枚的作用最佳。因此在我们的分配策略中，所有无人机的第一枚干扰弹投放后，对应的速度大小和方向就已经确定了，且投放时间也如前述表格所示完成了记录。在此基础上，我们进一步用粒子群优化算法分别对 FY_2、FY_3、FY_4、FY_5 的第二枚导弹做分析（投放间隔大于 1 s 放在参数的约束条件中），求得它们针对特定导弹的遮蔽时间段与遮蔽时长，如表 9所示：

表 9　各无人机第二枚干扰弹对各导弹的潜在最大遮蔽效果

	导弹 M_1		导弹 M_2		导弹 M_3	
无人机	时间段 (s)	时长 (s)	时间段 (s)	时长 (s)	时间段 (s)	时长 (s)
FY_2	$[15.601, 20.087]$	4.486	$[17.277, 21.257]$	3.980	$[16.170, 19.707]$	3.537
FY_3	$[67.865, 67.875]$	0.010	$[29.019, 34.034]$	5.014	$[61.735, 61.753]$	0.019
FY_4	—	0	—	0	$[20.524, 24.229]$	3.704
FY_5	$[19.961, 24.044]$	4.083	$[21.377, 24.790]$	3.413	$[19.275, 19.910]$	0.635

随后的规划策略与 $Step1$ 相同，计算后得到的结果如表 10所示：

表 10　第二枚干扰弹分配结果

无人机	$FY2$	$FY3$	$FY4$	$FY5$
干扰目标	$M2$	$M2$	$M3$	$M2$

Step 3：分析每一架无人机的第三枚干扰弹对各导弹的遮蔽效果。

类似于 $Step2$ 的计算过程，统计粒子群优化算法的结果并进行简单的规划后，得到的结果如表 11所示：

表 11　第三枚干扰弹分配结果

无人机	$FY2$	$FY3$	$FY4$	$FY5$
干扰目标	$M3$	$M3$	$M1$	$M3$

值得注意的是，尽管算法显示 $FY3$、$FY4$、$FY5$ 的第三枚干扰弹对相应的导弹具有一定的遮蔽时长，但是它们的取值均在 0.01 s 数量级。这主要由两方面的原因造成，一是随着时间越来越长，导弹越来越接近地面，投出干扰弹的引爆点几乎无法在竖直方向上追赶上导弹；二是在规划算法中，为了使总遮盖时长最大，第三枚干扰弹的分配会尽可能避免前面所有干扰弹已经成功遮蔽的时间段，这使得优化空间非常小。

最后得到对于三枚导弹的最大遮蔽时长为：

$$\max T_{\text{total}} = 42.260\,\text{s} \tag{23}$$

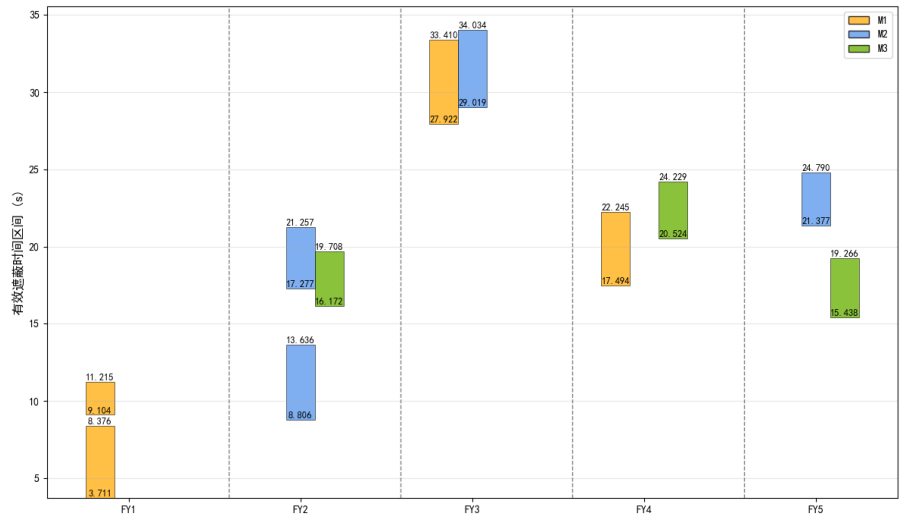


图 18　规划后示意图

将最终所有干扰弹对所有导弹的遮蔽时间段与遮蔽时长可视化表示如图 18，关于投放点、引爆点的数据保存在 $result3$ 中。

其中横坐标表示五架无人机，纵坐标为时间轴。不同色块分别表示干扰弹的遮蔽的不同导弹；色块底部的纵坐标表示引爆时刻，即烟雾云团生成的时刻。色块的纵向长度表示该干扰弹对相应导弹的持续遮蔽时间长度。

可以看出，除了 $FY2$ 的三枚干扰弹均发挥了作用以外，其余无人机搭载的干扰弹均只有两枚能发挥有效的遮蔽作用；且所有无人机全程都只能对一枚或两枚导弹产生有效的遮蔽。这恰恰是需要多架无人机、多枚干扰弹协同发挥作用的原因。

这种现象符合物理直觉的：无人机飞行轨迹、导弹飞行轨迹这些位置关系、它们各自不同的运动模式的存在，使得同一架无人机在空间上对多枚导弹产生遮蔽效果，或者在时间上无人机总是能恰好对准导弹相对容易被遮蔽的时间窗口这种情况几乎是不可能发生的。这也侧面说明了我们计算结果的合理性。

十、 模型的评价

10.1 模型的优点

- 从点到线段模型改为角度指标模型不改变复杂度，同时额外获得了导弹是否进入云团的定性情况。
- 使用奖励机制尝试让模型跳出局部最优，弥补粒子群算法的本质弱点。
- 在问题五这样参数庞大的问题中结合贪心算法有效降低问题复杂度。

10.2 模型的缺点

- 角度指标模型需要额外讨论导弹是否进入云团的情况，讨论过程略显繁琐。
- 简化模型由于算力原因将圆柱简化为单个点而非多个点。

参考文献

[1] 高哲, 杜柯. 基于粒子群优化算法的改进研究 [J]. 价值工程,2025,44(23):101-103.

[2] DeepSeek, DeepSeek-chat-v3.1, 深度求索 (DeepSeek), 2025-09-06.

附录 A　文件列表

表 12　文件列表

文件名	功能描述
judge_angle.py	判断角度相关功能的函数
solve_q1.py	问题一求解代码
solve_q2.py	问题二求解代码
solve_q3.py	问题三求解代码
solve_q4.py	问题四求解代码
solve_q5.py	问题五求解代码

附录 B　代码

judge_angle.py

```python
import numpy as np

# --- 真实目标的参数定义 ---
TARGET_REAL_BOTTOM_CENTER = np.array([0, 200, 0])
TARGET_HEIGHT = 10
TARGET_RADIUS = 7
SPHERE_RADIUS = 10

def judge_point_visible(
    vertex: np.ndarray,
    sphere_center: np.ndarray,
    point_to_check: np.ndarray
) -> bool:
    # if vertex[2] <= sphere_center[2] - SPHERE_RADIUS:
    #     return True

    # 1. 计算从顶点到球心的向量，作为圆锥的轴线
    axis_vector = sphere_center - vertex

    # 2. 计算顶点到球心的距离 (d)
```

```
    #    np.linalg.norm 用于计算向量的模（长度）
    dist_vertex_to_center = np.linalg.norm(axis_vector)

    # 检查顶点是否在球的外部，否则无法形成有效的切向圆锥
    if dist_vertex_to_center <= SPHERE_RADIUS:
        return False

    # 3. 计算圆锥半顶角 (α) 的余弦值
    #    cos(α) = sqrt(d^2 - r^2) / d
    cos_half_angle = np.sqrt(dist_vertex_to_center**2 -
SPHERE_RADIUS**2) / dist_vertex_to_center

    # 1. 创建从顶点到待测点的向量
    vector_to_point = point_to_check - vertex

    # 2. 计算轴向量与 vector_to_point 之间夹角 (θ) 的余弦值
    #    cos(θ) = (A · P_vec) / (|A| * |P_vec|)
    #    使用 np.dot 进行点积运算
    dot_product = np.dot(axis_vector, vector_to_point)
    cos_theta = dot_product / (np.linalg.norm(axis_vector) *
np.linalg.norm(vector_to_point))

    # 3. 比较角度的余弦值
    #    如果 cos(θ) < cos(α)，说明 θ > α，点在圆锥外，是可见
的
    return cos_theta < cos_half_angle



def generate_circle_points(center, radius, num_points=36):
    """
    在 3D 空间中，在一个 xy 平面上生成一个圆周上的点。

    参数:
    center (np.ndarray): 圆心坐标 [x, y, z]
```

```
    radius (float): 圆的半径
    num_points (int): 要生成的样本点数量

    返回:
    list[np.ndarray]: 包含圆周上所有点坐标的列表
    """
    points = []
    # 遍历 0 到 2*pi 的所有角度
    for i in range(num_points):
        angle = 2 * np.pi * i / num_points
        # 圆的参数方程
        dx = radius * np.cos(angle)
        dy = radius * np.sin(angle)
        # 点的坐标是圆心坐标加上偏移量
        point = center + np.array([dx, dy, 0])
        points.append(point)
    return points

def judge_target_visible(missile_pos, smoke_center,
   num_samples=36):
    """
    判断目标圆柱体是否在给定时刻被烟幕完全遮蔽。
    遮蔽标准：上、下两个底面圆周上的所有采样点都必须不可见。

    参数:
    missile_pos (np.ndarray): 导弹的位置 (即 judge_visible 中
   的 vertex)
    smoke_center (np.ndarray): 烟幕球心的位置
    num_samples (int): 在每个圆周上采样的点的数量

    返回:
    bool: True 如果完全遮蔽, False 如果有任何部分可见
    """
    # 1. 计算上表面圆心位置
    top_circle_center = TARGET_REAL_BOTTOM_CENTER + np.array
```

```
    ([0, 0, TARGET_HEIGHT])
86
87      # 2. 生成上表面圆周的样本点
88      top_points = generate_circle_points(top_circle_center,
    TARGET_RADIUS, num_samples)
89
90      # 3. 检查上表面圆周是否有任何一个点可见
91      for point in top_points:
92          if judge_point_visible(vertex=missile_pos,
    sphere_center=smoke_center, point_to_check=point):
93              # 只要找到一个可见点，就意味着没有完全遮蔽，立即返
    回 False
94              return True
95
96      # 4. 生成下表面圆周的样本点
97      bottom_points = generate_circle_points(
    TARGET_REAL_BOTTOM_CENTER, TARGET_RADIUS, num_samples)
98
99      # 5. 检查下表面圆周是否有任何一个点可见
100     for point in bottom_points:
101         if judge_point_visible(vertex=missile_pos,
    sphere_center=smoke_center, point_to_check=point):
102             # 同样，只要找到一个可见点，就立即返回 False
103             return True
104
105     # 6. 如果执行到这里，说明上下两个圆周的所有样本点都不可见
106     #    因此，我们判定圆柱体被完全遮蔽
107     return False
```

solve_q1.py

```
1  import numpy as np
2  import utils  # 导入工具函数模块
3  import judge_simple  # 导入判断逻辑模块
4  import judge_angle  # 导入复杂判断逻辑模块
5
```

```python
def solve_question_1():
    """
    根据问题1的给定参数，计算烟幕对M1的有效遮蔽时长。
    """
    print("--- 开始求解问题1 ---")

    # 1. 根据题目设定参数
    uav_name = "FY1"
    missile_name = "M1"
    v_uav_norm = 120.0  # m/s
    t_launch = 1.5      # s, 投放时间

    t_delay_detonation = 3.6 # s, 投放后到起爆的延迟

    # 确定无人机飞行方向：朝向假目标 (0,0,0)
    # FY1 初始位置: (17800, 0, 1800)
    # 飞行方向在 xy 平面内是沿 x 轴负方向，因此角度为 180 度。
    uav_direction_vec = -utils.initial_positions[uav_name][:2]
     # 取 x, y 分量
    angle_uav_deg = utils.get_angle_from_vector(
   uav_direction_vec)

    # 2. 计算关键状态
    # 计算无人机在投放时刻的位置和速度
    p_launch = utils.get_uav_position(uav_name, t_launch,
   v_uav_norm, angle_uav_deg)
    v_launch = utils.get_v_uav(v_uav_norm, angle_uav_deg)
    print(f"干扰弹投放时刻 (t={t_launch}s):")
    print(f" - 无人机位置: {np.round(p_launch, 2)}")
    print(f" - 无人机速度: {np.round(v_launch, 2)}")

    # 计算干扰弹起爆的时刻和位置
    t_detonation = t_launch + t_delay_detonation
    p_detonation = utils.get_grenade_position(p_launch,
   v_launch, t_launch, t_detonation)
```

```
    print(f"\n干扰弹起爆时刻 (t={t_detonation}s):")
    print(f" - 起爆位置: {np.round(p_detonation, 2)}")

    # 3. 建立仿真循环
    t_start_effective = t_detonation
    t_end_effective = t_detonation + utils.T_SMOKE_EFFECTIVE
    dt = 0.01  # 时间步长 (s)，值越小结果越精确

    total_effective_time = 0.0

    print(f"\n开始在时间窗口 [{t_start_effective:.1f}s, {t_end_effective:.1f}s] 内进行遮蔽判断...")

    for t in np.arange(t_start_effective, t_end_effective, dt):
        # 4. 在每个时间点 t，计算导弹和烟幕的位置
        p_missile = utils.get_missile_position(missile_name, t)
        p_smoke_center = utils.get_smoke_center_position(p_detonation, t_detonation, t)

        # 调用判断函数
        # judge_visible 返回 True 表示"可见"，返回 False 表示"被遮蔽"
        # is_visible = judge_simple.judge_target_visible(missile_pos=p_missile, smoke_center=p_smoke_center, num_samples=100)
        # is_visible = judge_simple.judge_point_visible(vertex=p_missile, sphere_center=p_smoke_center, point_to_check=utils.P_TARGET_REAL_CENTER)

        is_visible = judge_angle.judge_target_visible(missile_pos=p_missile, smoke_center=p_smoke_center, num_samples=100)
        # is_visible = judge_angle.judge_point_visible(vertex=
```

```
    p_missile, sphere_center=p_smoke_center, point_to_check=
    utils.P_TARGET_REAL_CENTER)
61
62          # 5. 如果被遮蔽 (不可见)，则累计时长
63          if not is_visible:
64              total_effective_time += dt
65
66      # 6. 输出最终结果
67      print("\n--- 求解完成 ---")
68      print(f"烟幕干扰弹对 M1 的总有效遮蔽时长为: {
    total_effective_time:.2f} 秒")
69
70  if __name__ == "__main__":
71      solve_question_1()
```

solve_q2.py

```
1   # solve_q2_angle.py
2
3   import numpy as np
4   import pyswarms as ps
5   from scipy.optimize import brentq
6   import utils
7   import judge_angle  # 导入新的判断模块
8
9   # --- 全局常量 ---
10  UAV_NAME = "FY1"
11  MISSILE_NAME = "M1"
12  NUM_SAMPLES = 36 # 采样点数量，可以根据需要调整
13
14  # 预先生成目标采样点，避免重复计算
15  TOP_CIRCLE_CENTER = judge_angle.TARGET_REAL_BOTTOM_CENTER + np
    .array([0, 0, judge_angle.TARGET_HEIGHT])
16  TOP_POINTS = judge_angle.generate_circle_points(
    TOP_CIRCLE_CENTER, judge_angle.TARGET_RADIUS, NUM_SAMPLES)
17  BOTTOM_POINTS = judge_angle.generate_circle_points(judge_angle
```

```
    .TARGET_REAL_BOTTOM_CENTER, judge_angle.TARGET_RADIUS,
    NUM_SAMPLES)
ALL_TARGET_POINTS = TOP_POINTS + BOTTOM_POINTS

def boundary_function_angle(t, p_detonation, t_detonation):
    """
    基于角度法的连续边界函数。
    返回 min(cos(theta_i) - cos(alpha)) over all sample points
    i.
    返回 >= 0 表示遮蔽成功。
    返回 < 0 表示遮蔽失败 (可见)。
    """
    # 1. 计算当前时刻的导弹和烟幕位置
    missile_pos = utils.get_missile_position(MISSILE_NAME, t)
    smoke_center = utils.get_smoke_center_position(
    p_detonation, t_detonation, t)

    # 2. 计算圆锥半顶角的余弦 cos(alpha)
    axis_vector = smoke_center - missile_pos
    dist_vertex_to_center = np.linalg.norm(axis_vector)

    if dist_vertex_to_center <= judge_angle.SPHERE_RADIUS:
        return 1.0 # 导弹在烟幕内，视为不可见

    cos_half_angle = np.sqrt(dist_vertex_to_center**2 -
    judge_angle.SPHERE_RADIUS**2) / dist_vertex_to_center

    # 3. 遍历所有采样点，计算 cos(theta) - cos(alpha)，并找到
    最小值
    min_value = float('inf')

    for point in ALL_TARGET_POINTS:
        vector_to_point = point - missile_pos
        norm_vector_to_point = np.linalg.norm(vector_to_point)

```

```
        # 避免除以零
        if norm_vector_to_point < 1e-6:
            continue

        dot_product = np.dot(axis_vector, vector_to_point)
        cos_theta = dot_product / (dist_vertex_to_center * norm_vector_to_point)

        value = cos_theta - cos_half_angle
        if value < min_value:
            min_value = value

    return min_value


def calculate_obscuration_time_precise_angle(params):
    """
    使用求根法和新的边界函数，精确计算有效遮蔽时长。
    """
    v_uav_norm, angle_uav_deg, t_launch, t_delay_detonation = params

    # 计算关键位置 (与之前版本相同)
    try:
        p_launch = utils.get_uav_position(UAV_NAME, t_launch, v_uav_norm, angle_uav_deg)
        v_launch = utils.get_v_uav(v_uav_norm, angle_uav_deg)
        t_detonation = t_launch + t_delay_detonation
        p_detonation = utils.get_grenade_position(p_launch, v_launch, t_launch, t_detonation)
        if p_detonation[2] < 0: return 0.0
    except (ValueError, ZeroDivisionError):
        return 0.0

    t_start = t_detonation
```

```
    t_end = t_detonation + utils.T_SMOKE_EFFECTIVE

    # 1. 粗略扫描，寻找包含根的区间
    coarse_step = 1.0  # 大步长
    scan_points = np.arange(t_start, t_end + coarse_step, coarse_step)

    # 使用 lambda 函数传递额外参数
    f_to_solve = lambda t: boundary_function_angle(t, p_detonation, t_detonation)

    scan_values = [f_to_solve(t) for t in scan_points]

    roots = []
    for i in range(len(scan_values) - 1):
        if np.sign(scan_values[i]) != np.sign(scan_values[i+1]):
            try:
                # 使用 brentq 精确求根
                root = brentq(f_to_solve, scan_points[i], scan_points[i+1])
                roots.append(root)
            except (ValueError, RuntimeError):
                continue

    # 2. 根据根的位置，计算总的有效时长
    event_points = sorted(list(set([t_start] + roots + [t_end])))
    total_effective_time = 0.0

    for i in range(len(event_points) - 1):
        t_current = event_points[i]
        t_next = event_points[i+1]

        mid_point = (t_current + t_next) / 2.0
```

```
108
109         # 在区间中点检查状态
110         if f_to_solve(mid_point) >= 0:
111             total_effective_time += (t_next - t_current)
112
113     return total_effective_time
114
115 def objective_function_for_pso(x):
116     """ PSO 目标函数 """
117     n_particles = x.shape[0]
118     fitness_scores = np.array([-
    calculate_obscuration_time_precise_angle(x[i]) for i in
    range(n_particles)])
119     return fitness_scores
120
121 def solve_question_2_angle():
122     """ 使用新的角度判断法和变步长算法求解问题2 """
123     print("--- 开始使用 PSO (角度法+精确求根) 求解问题2 ---")
124
125     # PSO 参数设置 (与之前相同)
126     options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
127     n_particles = 100
128     dimensions = 4
129     iters = 500
130
131     min_bounds = [70.0, 0.0, 0.1, 0.1]
132     max_bounds = [140.0, 360.0, 20.0, 15.0]
133     bounds = (np.array(min_bounds), np.array(max_bounds))
134
135     optimizer = ps.single.GlobalBestPSO(n_particles=
    n_particles,
136                                         dimensions=dimensions,
137                                         options=options,
138                                         bounds=bounds)
139
```

```
    print(f"开始优化... (共 {iters} 次迭代, {n_particles} 个粒子)")
    best_cost, best_pos = optimizer.optimize(objective_function_for_pso, iters=iters)
    print("优化完成！")

    # 结果展示 (与之前相同)
    max_obscuration_time = -best_cost
    best_v, best_angle, best_t_launch, best_t_delay = best_pos

    print("\n--- 优化结果 ---")
    print(f"找到的最大有效遮蔽时长: {max_obscuration_time:.3f} 秒")
    print("\n实现该效果的最佳策略参数为:")
    print(f" - 无人机飞行速度: {best_v:.3f} m/s")
    print(f" - 无人机飞行角度: {best_angle:.3f} 度")
    print(f" - 干扰弹投放时间: {best_t_launch:.3f} s")
    print(f" - 投放后起爆延迟: {best_t_delay:.3f} s")

    p_launch = utils.get_uav_position(UAV_NAME, best_t_launch, best_v, best_angle)
    v_launch = utils.get_v_uav(best_v, best_angle)
    t_detonation = best_t_launch + best_t_delay
    p_detonation = utils.get_grenade_position(p_launch, v_launch, best_t_launch, t_detonation)

    print("\n在最佳策略下:")
    print(f" - 干扰弹投放位置: {np.round(p_launch, 2)}")
    print(f" - 干扰弹起爆时刻: {t_detonation:.3f} s")
    print(f" - 干扰弹起爆位置: {np.round(p_detonation, 2)}")


if __name__ == "__main__":
    solve_question_2_angle()
```

solve_q3.py

```python
import numpy as np
import pandas as pd
import pyswarms as ps
from scipy.optimize import brentq
import utils
import judge_angle

# --- 全局常量 ---
UAV_NAME = "FY1"
MISSILE_NAME = "M1"
NUM_GRENADES = 3

def calculate_individual_obscuration_time(grenade_info):
    """
    计算单枚干扰弹的有效遮蔽时长。

    参数:
    grenade_info (dict): 包含单枚弹信息的字典
                         {'t_det', 'p_det', 't_start', 't_end'}

    返回:
    float: 该枚干扰弹的独立有效遮蔽时长
    """
    if not grenade_info:
        return 0.0

    all_roots = []
    f_to_solve = lambda t: boundary_function_angle(t, grenade_info['p_det'], grenade_info['t_det'])

    coarse_step = 1.0
    scan_points = np.arange(grenade_info['t_start'], grenade_info['t_end'] + coarse_step, coarse_step)
```

```
    scan_values = [f_to_solve(t) for t in scan_points]

    for i in range(len(scan_values) - 1):
        if np.sign(scan_values[i]) != np.sign(scan_values[i+1]):
            try:
                root = brentq(f_to_solve, scan_points[i], scan_points[i+1])
                all_roots.append(root)
            except (ValueError, RuntimeError):
                continue

    event_points = sorted(list(set([grenade_info['t_start']] + all_roots + [grenade_info['t_end']])))
    individual_time = 0.0

    for i in range(len(event_points) - 1):
        t_current = event_points[i]
        t_next = event_points[i+1]
        mid_point = (t_current + t_next) / 2.0

        if f_to_solve(mid_point) >= 0:
            individual_time += (t_next - t_current)

    return individual_time

def boundary_function_angle(t, p_detonation, t_detonation):
    """
    基于您提供的 solve_q2_1.py 中的简化版边界函数。
    只检查目标中心点是否被遮蔽。
    返回 >= 0 表示遮蔽成功。
    """
    missile_pos = utils.get_missile_position(MISSILE_NAME, t)
    smoke_center = utils.get_smoke_center_position(p_detonation, t_detonation, t)
```

```
63
64      axis_vector = smoke_center - missile_pos
65      dist_vertex_to_center = np.linalg.norm(axis_vector)
66
67      if dist_vertex_to_center <= judge_angle.SPHERE_RADIUS:
68          return 1.0  # 导弹在烟幕内，视为完全遮蔽
69
70      cos_half_angle = np.sqrt(dist_vertex_to_center**2 -
    judge_angle.SPHERE_RADIUS**2) / dist_vertex_to_center
71
72      vector_to_point = utils.P_TARGET_REAL_CENTER - missile_pos
73      norm_vector_to_point = np.linalg.norm(vector_to_point)
74
75      if norm_vector_to_point < 1e-6:
76          return -1.0
77
78      dot_product = np.dot(axis_vector, vector_to_point)
79      cos_theta = dot_product / (dist_vertex_to_center *
    norm_vector_to_point)
80
81      return cos_theta - cos_half_angle
82
83  def calculate_total_obscuration_time(params):
84      """
85      计算3枚干扰弹策略下的总有效遮蔽时长。
86      采用区间合并算法，避免重复计算。
87      """
88      v_uav, angle_uav, t1, d1, dt2, d2, dt3, d3 = params
89      launch_times = [t1, t1 + dt2, t1 + dt2 + dt3]
90      delays = [d1, d2, d3]
91
92      grenade_info = []
93      try:
94          v_launch_vec = utils.get_v_uav(v_uav, angle_uav)
95          for i in range(NUM_GRENADES):
```

```
            p_launch = utils.get_uav_position(UAV_NAME, launch_times[i], v_uav, angle_uav)
            t_detonation = launch_times[i] + delays[i]
            p_detonation = utils.get_grenade_position(p_launch, v_launch_vec, launch_times[i], t_detonation)
            if p_detonation[2] > 0:
                grenade_info.append({
                    't_det': t_detonation, 'p_det': p_detonation,
                    't_start': t_detonation, 't_end': t_detonation + utils.T_SMOKE_EFFECTIVE
                })
    except (ValueError, ZeroDivisionError):
        return 0.0

    if not grenade_info:
        return 0.0

    # --- 修正逻辑开始 ---
    # 1. 为每个烟幕确定其独立的有效遮蔽时间段
    effective_intervals = []
    for info in grenade_info:
        f_to_solve = lambda t: boundary_function_angle(t, info['p_det'], info['t_det'])

        roots = []
        coarse_step = 1.0
        scan_points = np.arange(info['t_start'], info['t_end'] + coarse_step, coarse_step)
        scan_values = [f_to_solve(t) for t in scan_points]
        for i in range(len(scan_values) - 1):
            if np.sign(scan_values[i]) != np.sign(scan_values[i+1]):
                try:
                    root = brentq(f_to_solve, scan_points[i],
```

```
    scan_points[i+1])
                    roots.append(root)
                except (ValueError, RuntimeError):
                    continue

        event_points = sorted(list(set([info['t_start']] +
    roots + [info['t_end']])))
        for i in range(len(event_points) - 1):
            t_current = event_points[i]
            t_next = event_points[i+1]
            mid_point = (t_current + t_next) / 2.0
            if f_to_solve(mid_point) >= 0:
                effective_intervals.append((t_current, t_next)
    )

     if not effective_intervals:
         return 0.0

     # 2. 合并所有重叠的区间并计算总长度
     effective_intervals.sort(key=lambda x: x[0])

     merged_intervals = []
     if not effective_intervals:
         return 0.0

     current_start, current_end = effective_intervals[0]

     for i in range(1, len(effective_intervals)):
         next_start, next_end = effective_intervals[i]
         if next_start < current_end:
             current_end = max(current_end, next_end)
         else:
             merged_intervals.append((current_start,
    current_end))
             current_start, current_end = next_start, next_end
```

```
155
156     merged_intervals.append((current_start, current_end))
157
158     # 3. 计算合并后区间的总长度
159     total_effective_time = sum(end - start for start, end in
    merged_intervals)
160
161     return total_effective_time
162
163
164 def objective_function_for_pso(x):
165     # (此函数与上一版本完全相同)
166     n_particles = x.shape[0]
167     return np.array([-calculate_total_obscuration_time(x[i])
    for i in range(n_particles)])
168
169 #
    ============================================================================

170 # --- 新增：保存结果到 Excel 的函数 ---
171 #
    ============================================================================

172 import openpyxl
173 from openpyxl.styles import Alignment
174
175 def save_results_to_excel(best_pos, total_best_time, filename=
    "result1.xlsx"):
176
177     v, angle, t1, d1, dt2, d2, dt3, d3 = best_pos
178     launch_times = [t1, t1 + dt2, t1 + dt2 + dt3]
179     delays = [d1, d2, d3]
180
181     workbook = openpyxl.Workbook()
182     worksheet = workbook.active
```

```
183     worksheet.title = '策略'
184 
185     # 修改表头，增加"总有效时长"列
186     headers = [
187         '无人机运动方向(度)', '无人机运动速度(m/s)', '烟幕干扰弹编号',
188         '投放点x(m)', '投放点y(m)', '投放点z(m)',
189         '起爆点x(m)', '起爆点y(m)', '起爆点z(m)',
190         '有效干扰时长(s)' # , '【总】有效干扰时长(s)'
191     ]
192     worksheet.append(headers)
193 
194     v_launch_vec = utils.get_v_uav(v, angle)
195     for i in range(NUM_GRENADES):
196         p_launch = utils.get_uav_position(UAV_NAME, launch_times[i], v, angle)
197         t_detonation = launch_times[i] + delays[i]
198         p_detonation = utils.get_grenade_position(p_launch, v_launch_vec, launch_times[i], t_detonation)
199 
200         # 为当前这枚弹单独计算时长
201         single_grenade_info = {
202             't_det': t_detonation,
203             'p_det': p_detonation,
204             't_start': t_detonation,
205             't_end': t_detonation + utils.T_SMOKE_EFFECTIVE
206         }
207         individual_time = calculate_individual_obscuration_time(single_grenade_info)
208 
209         row_data = [
210             f"{angle:.3f}" if i == 0 else "",
211             f"{v:.3f}" if i == 0 else "",
212             i + 1,
213             f"{p_launch[0]:.2f}",
```

```
            f"{p_launch[1]:.2f}",
            f"{p_launch[2]:.2f}",
            f"{p_detonation[0]:.2f}",
            f"{p_detonation[1]:.2f}",
            f"{p_detonation[2]:.2f}",
            f"{individual_time:.3f}",  # J列: 写入单枚弹的时长
            # f"{total_best_time:.3f}" if i == 0 else "" # K列: 写入总时长
        ]
        worksheet.append(row_data)

    # 合并单元格
    if NUM_GRENADES > 1:
        worksheet.merge_cells(start_row=2, start_column=1, end_row=1 + NUM_GRENADES, end_column=1)
        worksheet.merge_cells(start_row=2, start_column=2, end_row=1 + NUM_GRENADES, end_column=2)
        # 合并新的"总时长"列
        # worksheet.merge_cells(start_row=2, start_column=11, end_row=1 + NUM_GRENADES, end_column=11)

    # 添加备注和格式化 (这部分逻辑不变)
    worksheet.cell(row=3 + NUM_GRENADES, column=1, value='注：无人机运动方向以x轴正向为0度，逆时针为正，取值0~360（度）。')
    center_align = Alignment(horizontal='center', vertical='center')
    for col in worksheet.columns:
        max_length = 0
        column_letter = col[0].column_letter
        for cell in col:
            cell.alignment = center_align
            try:
                if len(str(cell.value)) > max_length:
                    max_length = len(str(cell.value))
```

```
            except:
                pass
        adjusted_width = (max_length + 2) * 1.2
        worksheet.column_dimensions[column_letter].width = adjusted_width

    workbook.save(filename)
    print(f"\n结果已成功保存到文件: {filename}")


def solve_question_3():
    """ 使用PSO求解问题3: FY1投放3枚干扰弹 """
    print("--- 开始使用 PSO (精确求根法) 求解问题3 ---")

    options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
    n_particles = 150
    dimensions = 8
    iters = 500

    min_bounds = [70.0, 0.0, 0.1, 0.1, 1.0, 0.1, 1.0, 0.1]
    max_bounds = [140.0, 360.0, 15.0, 20.0, 10.0, 20.0, 10.0, 20.0]
    bounds = (np.array(min_bounds), np.array(max_bounds))

    optimizer = ps.single.GlobalBestPSO(n_particles=n_particles,
                                        dimensions=dimensions,
                                        options=options,
                                        bounds=bounds)

    print(f"开始优化... (共 {iters} 次迭代, {n_particles} 个粒子, {dimensions}个维度)")
    best_cost, best_pos = optimizer.optimize(objective_function_for_pso, iters=iters, verbose=True)
    print("优化完成! ")
```

```
272
273     # --- 结果展示部分 ---
274     max_obscuration_time = -best_cost
275
276     # 打印到控制台
277     # (此部分与上一版本相同，为简洁省略)
278     v, angle, t1, d1, dt2, d2, dt3, d3 = best_pos
279     t2 = t1 + dt2
280     t3 = t2 + dt3
281     print("\n--- 优化结果 ---")
282     print(f"找到的最大有效遮蔽时长: {max_obscuration_time:.3f}
    秒")
283     print("\n实现该效果的最佳策略参数为:")
284     print(f" - 无人机飞行速度: {v:.3f} m/s")
285     print(f" - 无人机飞行角度: {angle:.3f} 度")
286     print("\n--- 干扰弹 1 ---")
287     print(f" - 投放时间 (t1): {t1:.3f} s")
288     print(f" - 引爆延迟 (d1): {d1:.3f} s")
289     print("--- 干扰弹 2 ---")
290     print(f" - 投放时间 (t2): {t2:.3f} s (间隔 dt2={dt2:.3f} s
    )")
291     print(f" - 引爆延迟 (d2): {d2:.3f} s")
292     print("--- 干扰弹 3 ---")
293     print(f" - 投放时间 (t3): {t3:.3f} s (间隔 dt3={dt3:.3f} s
    )")
294     print(f" - 引爆延迟 (d3): {d3:.3f} s")
295
296     # --- 调用新函数，保存结果到Excel ---
297     save_results_to_excel(best_pos, max_obscuration_time, '
    result1_tt.xlsx')
298
299 if __name__ == "__main__":
300     solve_question_3()
```

solve_q4.py

```python
import numpy as np
import pandas as pd
import pyswarms as ps
from scipy.optimize import brentq
import utils
import judge_angle
import openpyxl
from openpyxl.styles import Alignment

# --- 全局常量 ---
UAV_NAMES = ["FY1", "FY2", "FY3"]
MISSILE_NAME = "M1"
NUM_UAVS = 3

# ==========================================================================

# --- The following two functions are identical to the previous version, ---
# --- as their logic is generic and still applies. ---
# ==========================================================================


def boundary_function_angle(t, p_detonation, t_detonation):
    missile_pos = utils.get_missile_position(MISSILE_NAME, t)
    smoke_center = utils.get_smoke_center_position(p_detonation, t_detonation, t)
    axis_vector = smoke_center - missile_pos
    dist_vertex_to_center = np.linalg.norm(axis_vector)
    if dist_vertex_to_center <= judge_angle.SPHERE_RADIUS:
        return 1.0
    cos_half_angle = np.sqrt(dist_vertex_to_center**2 - judge_angle.SPHERE_RADIUS**2) / dist_vertex_to_center
    vector_to_point = utils.P_TARGET_REAL_CENTER - missile_pos
```

```
    norm_vector_to_point = np.linalg.norm(vector_to_point)
    if norm_vector_to_point < 1e-6:
        return -1.0
    dot_product = np.dot(axis_vector, vector_to_point)
    cos_theta = dot_product / (dist_vertex_to_center * norm_vector_to_point)
    return cos_theta - cos_half_angle

def calculate_individual_obscuration_time(grenade_info):
    if not grenade_info:
        return 0.0
    all_roots = []
    f_to_solve = lambda t: boundary_function_angle(t, grenade_info['p_det'], grenade_info['t_det'])
    coarse_step = 1.0
    scan_points = np.arange(grenade_info['t_start'], grenade_info['t_end'] + coarse_step, coarse_step)
    scan_values = [f_to_solve(t) for t in scan_points]
    for i in range(len(scan_values) - 1):
        if np.sign(scan_values[i]) != np.sign(scan_values[i+1]):
            try:
                root = brentq(f_to_solve, scan_points[i], scan_points[i+1])
                all_roots.append(root)
            except (ValueError, RuntimeError):
                continue
    event_points = sorted(list(set([grenade_info['t_start']] + all_roots + [grenade_info['t_end']])))
    individual_time = 0.0
    for i in range(len(event_points) - 1):
        t_current = event_points[i]
        t_next = event_points[i+1]
        mid_point = (t_current + t_next) / 2.0
        if f_to_solve(mid_point) >= 0:
```

```
            individual_time += (t_next - t_current)
    return individual_time

# ==========================================================================

# --- The core fitness function is adapted for 3 independent UAVs ---
# ==========================================================================


def calculate_total_obscuration_time(params):
    # 1. Decode the 12-dimensional parameter vector
    strategies = [params[i*4:(i+1)*4] for i in range(NUM_UAVS)]

    grenade_info = []
    try:
        for i in range(NUM_UAVS):
            v_uav, angle_uav, t_launch, delay = strategies[i]
            uav_name = UAV_NAMES[i]

            v_launch_vec = utils.get_v_uav(v_uav, angle_uav)
            p_launch = utils.get_uav_position(uav_name, t_launch, v_uav, angle_uav)
            t_detonation = t_launch + delay
            p_detonation = utils.get_grenade_position(p_launch, v_launch_vec, t_launch, t_detonation)

            if p_detonation[2] > 0:
                grenade_info.append({
                    't_det': t_detonation, 'p_det': p_detonation,
                    't_start': t_detonation, 't_end':
```

```
    t_detonation + utils.T_SMOKE_EFFECTIVE
                })
     except (ValueError, ZeroDivisionError):
         return 0.0

     if not grenade_info:
         return 0.0

     # --- 修正逻辑开始 ---
     # 2. 为每个烟幕确定其独立的有效遮蔽时间段
     effective_intervals = []
     for info in grenade_info:
         f_to_solve = lambda t: boundary_function_angle(t, info['p_det'], info['t_det'])

         # 寻找根
         roots = []
         coarse_step = 1.0
         scan_points = np.arange(info['t_start'], info['t_end'] + coarse_step, coarse_step)
         scan_values = [f_to_solve(t) for t in scan_points]
         for i in range(len(scan_values) - 1):
             if np.sign(scan_values[i]) != np.sign(scan_values[i+1]):
                 try:
                     root = brentq(f_to_solve, scan_points[i], scan_points[i+1])
                     roots.append(root)
                 except (ValueError, RuntimeError):
                     continue

         # 确定有效区间
         event_points = sorted(list(set([info['t_start']] + roots + [info['t_end']])))
         for i in range(len(event_points) - 1):
```

```
            t_current = event_points[i]
            t_next = event_points[i+1]
            mid_point = (t_current + t_next) / 2.0
            if f_to_solve(mid_point) >= 0:
                effective_intervals.append((t_current, t_next))

    if not effective_intervals:
        return 0.0

    # 3. 合并所有重叠的区间并计算总长度
    # 按起始时间对区间进行排序
    effective_intervals.sort(key=lambda x: x[0])

    merged_intervals = []
    current_start, current_end = effective_intervals[0]

    for i in range(1, len(effective_intervals)):
        next_start, next_end = effective_intervals[i]
        if next_start < current_end:
            # 区间有重叠，合并它们
            current_end = max(current_end, next_end)
        else:
            # 区间不重叠，保存当前合并的区间，并开始一个新的
            merged_intervals.append((current_start, current_end))
            current_start, current_end = next_start, next_end

    # 添加最后一个合并的区间
    merged_intervals.append((current_start, current_end))

    # 4. 计算合并后区间的总长度
    total_effective_time = sum(end - start for start, end in merged_intervals)

```

```
    return total_effective_time

def objective_function_for_pso(x):
    n_particles = x.shape[0]
    return np.array([-calculate_total_obscuration_time(x[i]) for i in range(n_particles)])


def save_results_to_excel_q4(best_pos, total_best_time, filename="result2_tem.xlsx"):
    strategies = [best_pos[i*4:(i+1)*4] for i in range(NUM_UAVS)]

    workbook = openpyxl.Workbook()
    worksheet = workbook.active
    worksheet.title = '策略'

    headers = [
        '无人机编号', '运动方向(度)', '运动速度(m/s)',
        '投放点x(m)', '投放点y(m)', '投放点z(m)',
        '起爆点x(m)', '起爆点y(m)', '起爆点z(m)',
        '【单枚】有效干扰时长(s)', '【总】有效干扰时长(s)'
    ]
    worksheet.append(headers)

    for i in range(NUM_UAVS):
        v, angle, t_launch, delay = strategies[i]
        uav_name = UAV_NAMES[i]

        v_launch_vec = utils.get_v_uav(v, angle)
        p_launch = utils.get_uav_position(uav_name, t_launch, v, angle)
        t_detonation = t_launch + delay
        p_detonation = utils.get_grenade_position(p_launch, v_launch_vec, t_launch, t_detonation)
```

```
175
176         single_grenade_info = {
177             't_det': t_detonation, 'p_det': p_detonation,
178             't_start': t_detonation, 't_end': t_detonation +
    utils.T_SMOKE_EFFECTIVE
179         }
180         individual_time =
    calculate_individual_obscuration_time(single_grenade_info)
181
182         row_data = [
183             uav_name,
184             f"{angle:.3f}",
185             f"{v:.3f}",
186             f"{p_launch[0]:.2f}", f"{p_launch[1]:.2f}", f"{
    p_launch[2]:.2f}",
187             f"{p_detonation[0]:.2f}", f"{p_detonation[1]:.2f}"
    , f"{p_detonation[2]:.2f}",
188             f"{individual_time:.3f}",
189             f"{total_best_time:.3f}" if i == 0 else ""
190         ]
191         worksheet.append(row_data)
192
193     if NUM_UAVS > 1:
194         worksheet.merge_cells(start_row=2, start_column=11,
    end_row=1 + NUM_UAVS, end_column=11)
195
196     worksheet.cell(row=3 + NUM_UAVS, column=1, value='注：无人
    机运动方向以x轴正向为0度，逆时針为正，取值0~360（度）。')
197     center_align = Alignment(horizontal='center', vertical='
    center')
198     for col in worksheet.columns:
199         max_length = 0
200         column_letter = col[0].column_letter
201         for cell in col:
202             cell.alignment = center_align
```

```
            try:
                if len(str(cell.value)) > max_length:
                    max_length = len(str(cell.value))
            except: pass
        adjusted_width = (max_length + 2) * 1.2
        worksheet.column_dimensions[column_letter].width = adjusted_width

    workbook.save(filename)
    print(f"\n结果已成功保存到文件: {filename}")

# ============================================================================

# --- Main solver function for Question 4 ---
# ============================================================================

def solve_question_4():
    print("--- 开始使用 PSO (精确求根法) 求解问题4 ---")

    options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
    # 12D is a much harder problem, so we increase particles and iterations
    n_particles = 200
    dimensions = 12
    iters = 500

    # Bounds for one UAV: [v, angle, t_launch, delay]
    min_b = [70.0, 0.0, 0, 0.1]
    max_b = [140.0, 360.0, 20.0, 20.0]

    # Repeat bounds for all 3 UAVs
    min_bounds = np.array(min_b * NUM_UAVS)
```

```
    max_bounds = np.array([140.0, 360.0, 20.0, 15.0] + [140.0, 360.0, 20.0, 15.0]  + [140.0, 360.0, 50.0, 12.0])
    bounds = (min_bounds, max_bounds)

    optimizer = ps.single.GlobalBestPSO(n_particles=n_particles,
                                        dimensions=dimensions,
                                        options=options,
                                        bounds=bounds)

    print(f"开始优化... (共 {iters} 次迭代, {n_particles} 个粒子, {dimensions}个维度)")
    best_cost, best_pos = optimizer.optimize(objective_function_for_pso, iters=iters, verbose=True)
    print("优化完成！")

    max_obscuration_time = -best_cost

    # --- Print results to console ---
    strategies = [best_pos[i*4:(i+1)*4] for i in range(NUM_UAVS)]
    print("\n--- 优化结果 ---")
    print(f"找到的最大有效遮蔽时长: {max_obscuration_time:.3f} 秒")
    for i in range(NUM_UAVS):
        v, angle, t_launch, delay = strategies[i]
        print(f"\n--- 无人机 {UAV_NAMES[i]} 策略 ---")
        print(f" - 飞行速度: {v:.3f} m/s")
        print(f" - 飞行角度: {angle:.3f} 度")
        print(f" - 投放时间: {t_launch:.3f} s")
        print(f" - 引爆延迟: {delay:.3f} s")

    # --- Save results to the specified Excel file ---
    save_results_to_excel_q4(best_pos, max_obscuration_time, filename="result2_t.xlsx")
```

```

if __name__ == "__main__":
    solve_question_4()
```

solve_q5.py

```
import numpy as np
import pyswarms as ps
from scipy.optimize import brentq
import utils
import judge_angle  # 导入新的判断模块

#
   ============================================================================

# --- 新增：无人机固定飞行参数 ---
# 无人机飞行速度和角度现在是常量，根据无人机名称获取。
#
   ============================================================================

UAV_FLIGHT_CONSTANTS = {
    "FY1": {"speed": 89.891, "angle": 178.667},
    "FY2": {"speed": 111.279, "angle": 274.042},
    "FY3": {"speed": 128.141, "angle": 119.249},
    "FY4": {"speed": 125.164, "angle": 243.714},
    "FY5": {"speed": 128.527, "angle": 120.238},
}

#
   ============================================================================

# --- 修改：根据无人机和导弹确定优化参数边界 ---
# 格式：'无人机名称': {'导弹名称': {'min_bounds': [...], '
   max_bounds': [...]}}
# 参数顺序：[投放时间，引爆延迟]
#
```

```
    ================================================================================


# 第二轮导弹优化参数
# UAV_PARAMETER_BOUNDS = {
#     "FY1": {
#         "M1": {"min_bounds": [0.0, 0.1], "max_bounds": [40.0, 20.0]},
#         "M2": {"min_bounds": [0.0, 0.1], "max_bounds": [40.0, 20.0]},
#         "M3": {"min_bounds": [0.0, 0.1], "max_bounds": [40.0, 20.0]},
#     },
#     "FY2": {
#         "M1": {"min_bounds": [6.340, 0], "max_bounds": [40.0, 17.0]},
#         "M2": {"min_bounds": [6.340, 0], "max_bounds": [40.0, 17.0]},
#         "M3": {"min_bounds": [6.340, 0], "max_bounds": [40.0, 17.0]},
#     },
#     "FY3": {
#         "M1": {"min_bounds": [21.584, 0], "max_bounds": [50.0, 12.0]},
#         "M2": {"min_bounds": [21.584, 0], "max_bounds": [50.0, 12.0]},
#         "M3": {"min_bounds": [21.584, 0], "max_bounds": [50.0, 12.0]},
#     },
#     "FY4": {
#         "M1": {"min_bounds": [5.647, 0], "max_bounds": [50.0, 20.0]},
#         "M2": {"min_bounds": [5.647, 0], "max_bounds": [50.0, 20.0]},
#         "M3": {"min_bounds": [5.647, 0], "max_bounds":
```

```
   [50.0, 20.0]},
#     },
#     "FY5": {
#         "M1": {"min_bounds": [13.588, 0], "max_bounds":
   [40.0, 17.0]},
#         "M2": {"min_bounds": [13.588, 0], "max_bounds":
   [40.0, 17.0]},
#         "M3": {"min_bounds": [13.588, 0], "max_bounds":
   [40.0, 17.0]},
#     },
# }


UAV_PARAMETER_BOUNDS = {
    "FY1": {
        "M1": {"min_bounds": [1.644, 0.1], "max_bounds":
   [40.0, 20.0]},
        "M2": {"min_bounds": [1.644, 0.1], "max_bounds":
   [40.0, 20.0]},
        "M3": {"min_bounds": [1.644, 0.1], "max_bounds":
   [40.0, 20.0]},
    },
    "FY2": {
        "M1": {"min_bounds": [7.344, 0], "max_bounds": [40.0,
   17.0]},
        "M2": {"min_bounds": [7.344, 0], "max_bounds": [40.0,
   17.0]},
        "M3": {"min_bounds": [7.344, 0], "max_bounds": [40.0,
   17.0]},
    },
    "FY3": {
        "M1": {"min_bounds": [23.211, 0], "max_bounds": [50.0,
    12.0]},
        "M2": {"min_bounds": [23.211, 0], "max_bounds": [50.0,
    12.0]},
```

```
        "M3": {"min_bounds": [23.211, 0], "max_bounds": [50.0, 12.0]},
    },
    "FY4": {
        "M1": {"min_bounds": [9.134, 0], "max_bounds": [50.0, 20.0]},
        "M2": {"min_bounds": [9.134, 0], "max_bounds": [50.0, 20.0]},
        "M3": {"min_bounds": [9.134, 0], "max_bounds": [50.0, 20.0]},
    },
    "FY5": {
        "M1": {"min_bounds": [20.832, 0], "max_bounds": [40.0, 17.0]},
        "M2": {"min_bounds": [20.832, 0], "max_bounds": [40.0, 17.0]},
        "M3": {"min_bounds": [20.832, 0], "max_bounds": [40.0, 17.0]},
    },
}


def boundary_function_angle(t, p_detonation, t_detonation, missile_name):
    """
    基于角度法的连续边界函数。
    返回 >= 0 表示遮蔽成功。
    返回 < 0 表示遮蔽失败 (可见)。
    """
    missile_pos = utils.get_missile_position(missile_name, t)
    smoke_center = utils.get_smoke_center_position(p_detonation, t_detonation, t)
    axis_vector = smoke_center - missile_pos
    dist_vertex_to_center = np.linalg.norm(axis_vector)

```

```
95      if dist_vertex_to_center <= judge_angle.SPHERE_RADIUS:
96          return 1.0
97
98      cos_half_angle = np.sqrt(dist_vertex_to_center**2 -
    judge_angle.SPHERE_RADIUS**2) / dist_vertex_to_center
99      vector_to_point = utils.P_TARGET_REAL_CENTER - missile_pos
100     norm_vector_to_point = np.linalg.norm(vector_to_point)
101
102     if norm_vector_to_point < 1e-6:
103         return -1.0
104
105     dot_product = np.dot(axis_vector, vector_to_point)
106     cos_theta = dot_product / (dist_vertex_to_center *
    norm_vector_to_point)
107
108     return cos_theta - cos_half_angle
109
110 def calculate_obscuration_time_precise_angle(params,
    v_uav_norm, angle_uav_deg, uav_name, missile_name):
111     """
112     使用求根法精确计算有效遮蔽时长和区间。
113     params: [投放时间, 引爆延迟]
114     """
115     t_launch, t_delay_detonation = params
116
117     try:
118         p_launch = utils.get_uav_position(uav_name, t_launch,
    v_uav_norm, angle_uav_deg)
119         v_launch = utils.get_v_uav(v_uav_norm, angle_uav_deg)
120         t_detonation = t_launch + t_delay_detonation
121         p_detonation = utils.get_grenade_position(p_launch,
    v_launch, t_launch, t_detonation)
122         if p_detonation[2] < 0: return 0.0, []
123     except (ValueError, ZeroDivisionError):
124         return 0.0, []
```

```
125
126     t_start = t_detonation
127     t_end = t_detonation + utils.T_SMOKE_EFFECTIVE
128     coarse_step = 0.5
129     scan_points = np.arange(t_start, t_end + coarse_step,
    coarse_step)
130
131     f_to_solve = lambda t: boundary_function_angle(t,
    p_detonation, t_detonation, missile_name)
132     scan_values = [f_to_solve(t) for t in scan_points]
133
134     roots = []
135     for i in range(len(scan_values) - 1):
136         if np.sign(scan_values[i]) != np.sign(scan_values[i
    +1]):
137             try:
138                 root = brentq(f_to_solve, scan_points[i],
    scan_points[i+1])
139                 roots.append(root)
140             except (ValueError, RuntimeError):
141                 continue
142
143     event_points = sorted(list(set([t_start] + roots + [t_end
    ])))
144     total_effective_time = 0.0
145     obscuration_intervals = []
146
147     for i in range(len(event_points) - 1):
148         t_current, t_next = event_points[i], event_points[i+1]
149         mid_point = (t_current + t_next) / 2.0
150
151         if f_to_solve(mid_point) >= 0:
152             total_effective_time += (t_next - t_current)
153             obscuration_intervals.append((t_current, t_next))
154
```

```
    return total_effective_time, obscuration_intervals

def solve_single_uav_mission(uav_name, missile_name):
    """
    使用PSO为单个无人机对抗单个导弹的场景求解最优策略。
    """
    print(f"--- 开始求解: 无人机 {uav_name} vs 导弹 {missile_name} ---")

    # --- 获取固定飞行参数 ---
    if uav_name not in UAV_FLIGHT_CONSTANTS:
        print(f"错误: 未在 UAV_FLIGHT_CONSTANTS 中找到无人机 '{uav_name}' 的飞行参数。")
        return
    flight_consts = UAV_FLIGHT_CONSTANTS[uav_name]
    v_uav_norm = flight_consts["speed"]
    angle_uav_deg = flight_consts["angle"]
    print(f"使用固定飞行参数: 速度={v_uav_norm} m/s, 角度={angle_uav_deg} 度")

    # --- 获取优化参数边界 ---
    if uav_name not in UAV_PARAMETER_BOUNDS or missile_name not in UAV_PARAMETER_BOUNDS[uav_name]:
        print(f"错误: 未找到无人机 '{uav_name}' 对抗导弹 '{missile_name}' 的参数边界配置。")
        return
    bounds_config = UAV_PARAMETER_BOUNDS[uav_name][missile_name]
    bounds = (np.array(bounds_config["min_bounds"]), np.array(bounds_config["max_bounds"]))

    # --- PSO 目标函数 ---
    def objective_function_for_pso(x):
        n_particles = x.shape[0]
        fitness_scores = np.array([-
```

```
    calculate_obscuration_time_precise_angle(x[i], v_uav_norm,
    angle_uav_deg, uav_name, missile_name)[0] for i in range(
    n_particles)])
        return fitness_scores

    # --- PSO 参数设置 ---
    options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
    n_particles = 100
    dimensions = 2  # 维度减少为2 (投放时间, 引爆延迟)
    iters = 500

    optimizer = ps.single.GlobalBestPSO(n_particles=
    n_particles,
                                        dimensions=dimensions,
                                        options=options,
                                        bounds=bounds)

    print(f"开始优化... (共 {iters} 次迭代, {n_particles} 个粒
    子)")
    best_cost, best_pos = optimizer.optimize(
    objective_function_for_pso, iters=iters)
    print("优化完成! ")

    # --- 结果展示 ---
    max_obscuration_time, intervals =
    calculate_obscuration_time_precise_angle(best_pos,
    v_uav_norm, angle_uav_deg, uav_name, missile_name)
    best_t_launch, best_t_delay = best_pos

    print("\n--- 优化结果 ---")
    print(f"找到的最大有效遮蔽时长: {max_obscuration_time:.3f}
    秒")

    if intervals:
        print("对应的遮蔽时间区间为:")
```

```
209         for start, end in intervals:
210             print(f" - 从 {start:.3f} s 到 {end:.3f} s (时长: {end-start:.3f} s)")
211             print(f'[{start:.3f}, {end:.3f}], {end-start:.3f}')
212     else:
213         print("未找到有效的遮蔽时间区间。")
214 
215     print(f'[{v_uav_norm:.3f}, {angle_uav_deg:.3f}, {best_t_launch:.3f}, {best_t_delay:.3f}]')
216 
217     print("\n固定飞行参数为:")
218     print(f" - 无人机飞行速度: {v_uav_norm:.3f} m/s")
219     print(f" - 无人机飞行角度: {angle_uav_deg:.3f} 度")
220 
221     print("\n优化得到的最佳策略参数为:")
222     print(f" - 干扰弹投放时间: {best_t_launch:.3f} s")
223     print(f" - 投放后起爆延迟: {best_t_delay:.3f} s")
224 
225     p_launch = utils.get_uav_position(uav_name, best_t_launch, v_uav_norm, angle_uav_deg)
226     v_launch = utils.get_v_uav(v_uav_norm, angle_uav_deg)
227     t_detonation = best_t_launch + best_t_delay
228     p_detonation = utils.get_grenade_position(p_launch, v_launch, best_t_launch, t_detonation)
229 
230     print("\n在最佳策略下:")
231     print(f" - 干扰弹投放位置: {np.round(p_launch, 2)}")
232     print(f" - 干扰弹起爆时刻: {t_detonation:.3f} s")
233     print(f" - 干扰弹起爆位置: {np.round(p_detonation, 2)}")
234 
235 
236 if __name__ == "__main__":
237     # --- 在此指定要分析的无人机和导弹 ---
238     TARGET_UAV = "FY1"
```

```
    TARGET_MISSILE = "M1"

    solve_single_uav_mission(uav_name=TARGET_UAV, missile_name
=TARGET_MISSILE)
```