烟幕干扰弹最佳投放策略建模优化

摘 要

本文针对无人机投放烟幕干扰弹的问题，综合运用运动学分析、几何建模、数值优化等方法，对不同场景下的干扰弹投放策略进行研究。通过建立无人机与导弹的运动模型、烟幕云团的遮蔽模型，结合优化算法求解最优投放参数，以最大化对真目标的有效遮蔽时间。

针对问题一 本文构建了烟雾有效遮蔽的计算模型。首先根据题意要求条件建立了导弹，无人机，干扰弹的运动模型，进而推断每一时刻导弹与烟雾球之间的位置关系。然后，由基于空间尺度下的几何近似，将导弹视角下的真目标简化为长方形截面，然后运用基于关键点的遮蔽效果验证方法，用截面长方形的四个端点代替整个目标验证遮蔽效果。进而根据关键点与导弹的线段方程，得到判断函数并计算有效遮蔽时间。最终本文得到了给定条件下烟雾对导弹的**有效遮蔽时间为$1.401s$**。

针对问题二 本文构建了以有效遮蔽时间为目标函数的优化模型。该模型以问题一的有效遮蔽计算模型为基础，聚焦单无人机单导弹的干扰情形。首先，本文通过采样得出解空间中满足所有物理与任务约束的可行解分布极为稀疏的结论，后通过双重退火算法得到初步解，并在初步最优解邻域内精细调整参数，进一步提升解的精度，最终得到最优投放参数，使烟幕干扰弹对真目标的有效遮蔽时间达到**最大值$4.594s$**。

针对问题三 首先提出“基准时序引导+重叠控制”的优先投放策略：首先通过参数调整，优化三枚干扰弹的投放时间。以第一枚干扰弹有效遮蔽起始时刻最早化为基准，规划后续两枚弹的投放-起爆时序，在优化参数配置得到初步解后，进一步精细调整参数，最终得到最优投放参数，使三次烟幕投射下对真目标的有效遮蔽时间达到最大值$6.64s$。

针对问题四 第一阶段以“单无人机遮蔽时长最大化”为目标，复用问题二的单目标优化模型，优化各无人机的最优投放参数，得到的单架无人机的最大屏蔽时长。第二阶段在第一阶段初步解中，时间重合度不高的特定，通过以初步解为基础调整参数上下界的方法，最终得到了对目标最大的**有效屏蔽总时长$12.13s$**

针对问题五 本文采用了“多目标协同优化”的策略，首先分析了无人机的空间位置关系，得到了初步的投放策略：即无人机F_1单独优化，其他无人机协同优化。其次，建立了单无人机多导弹的优化模型，通过贪心思想与双退火算法，得到每架无人机对三枚导弹的最长遮蔽时间，并发现不同无人机的对同一导弹的遮蔽时间重合度不高。因此基于初步解，采用粒子群算法对所有无人机的投放参数进行联合优化，最终得到最优投放参数，使得 5 架无人机投放的干扰弹对 3 枚导弹的有效干扰总时长达到**最大值$33.93s$**。

关键字： 有效遮蔽　双重退火　精细搜索　单目标优化　多目标协同

一、 问题重述

1.1 问题背景

烟幕干扰是常见的防空干扰手段，凭借成本低、效果显著的优势，在现代防空作战中发挥着关键作用。其核心原理是通过形成烟幕或气溶胶云团，在目标和导弹之间特定位置形成有效遮蔽，干扰敌方导弹对目标的探测与打击。在实际防空场景中，主要利用无人机搭载烟幕干扰弹进行投放。然而，如何设计无人机的有关参数与投放策略，以使得烟幕干扰弹对真目标的有效遮蔽时间尽可能长，

1.2 问题重述

在某次任务中，侦测到 3 枚导弹正以匀速直线运动向一个假目标飞来。为保护真目标，部署了 5 架无人机，其具体位置如图 1。无人机可在指定高度上以一定的飞行速度，飞行方向匀速直线飞行。每架无人机可最多投放三枚烟雾干扰弹，且每枚干扰弹的投放时间间隔不小于 $1s$。现需要建立数学模型，设计无人机的飞行参数与投放策略，使得烟幕干扰弹对真目标的有效遮蔽时间尽可能长。现需要建立数学模型，解决如下问题：

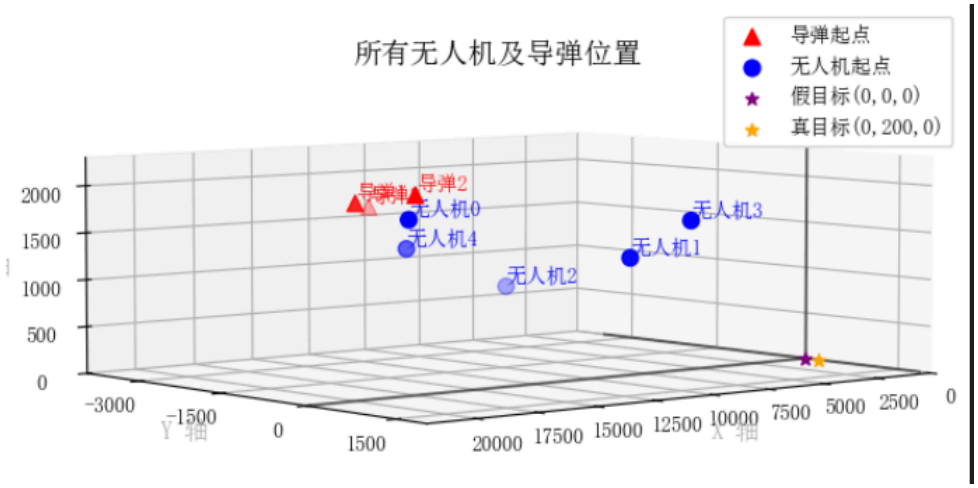


图 1　导弹，无人机，目标位置

问题一：对于一组确定的的导弹位置和无人机位置，并给定无人机的飞行速度，飞行方向，无人机投放烟幕干扰弹的时间和引爆时间，建立数学模型，计算投放干扰弹后，烟幕干扰弹有效遮蔽真目标的时间。

问题二：对于一组确定的导弹位置和无人机位置，设计无人机的飞行速度，飞行方向，无人机一次投放烟幕干扰弹的时间和引爆时间，建立数学模型，求解使得烟幕干扰弹对真目标的有效遮蔽时间最长的投放策略。

问题三：在问题二的基础上，假设无人机投放三枚烟幕干扰弹，设计无人机运动参数和投放策略，建立数学模型，求解使得三枚干扰弹对真目标的有效遮蔽时间最长的投放策略。

问题四： 在问题二的基础上，将无人机总数改为 3，·每架无人机投放一枚烟幕干扰弹，设计无人机运动参数和投放策略，建立数学模型，求解使得三枚干扰弹对真目标的有效遮蔽时间最长的投放策略。

问题五：给定三枚导弹的初始位置和飞行速度，设计 5 架无人机的运动参数和投放策略，建立数学模型，求解使得 5 架无人机投放的干扰弹对真目标的有效遮蔽时间最长的投放策略。

二、 模型假设

1.假设考虑重力作用对无人机和干扰弹的影响，其他外力忽略不计。

2.导弹有制导功能，严格做匀速直线运动。

3.烟幕干扰弹起爆后瞬时形成球状烟幕云团，其运动与遮蔽特性严格遵循题意。

4.无人机受领任务后，可根据需要瞬时调整飞行方向，并在水平上做匀速直线运动。

三、 符号说明

符号	含义	单位
M_i	第i个导弹$(i=1,2,3)$	/
F_i	第i架无人机$(i=1,2,3,4,5)$	/
O	假目标（原点）	/
O_t	真目标	/
R_s	烟雾云团半径	m
D_i	第i个真目标上的参考点$(i=1,2,3,4)$	/
T	从发现导弹到导弹击中假目标的时间序列	s
u_i	第i架无人机的飞行速度	m/s
θ_i	第i架无人机飞行方向与x轴正向的夹角	rad
$t_{L,i,j}$	第i架无人机的投放第j个干扰弹的时间	s
$t_{R,i,j}$	第i架无人机的第j个干扰弹的释放-引爆时间	s
$z_{s,i}(t)$	第i个无人机投放的第j个烟幕云团时刻t时的高度	m

四、 问题分析

4.1 整体分析

根据题中信息建立三维直角坐标系，标定无人机，真假目标及导弹的坐标，结合相关信息建立数学模型，针对不同情形设计烟幕干扰弹的投放策略，包括无人机飞行方向、飞行速度、烟幕干扰弹投放点、烟幕干扰弹起爆点等，调节相关参数，使得烟幕干扰弹对真目标的有效遮蔽时间尽可能长。

4.2 子问题分析

问题 1：对于问题一，无人机 F_1 按照既定投放策略投放 1 枚烟幕干扰弹，干扰 M_1 导弹。此问题是在限定投放策略下的效果评估，可通过分析无人机、烟幕云团和导弹的运动轨迹，结合有效遮蔽的判定条件来求解。有效遮蔽就是从导弹的视角观察，真目标完全被烟幕云团遮蔽，基于空间尺度下的几何近似，将烟雾云团遮蔽空间近似为圆柱，再利用观测视角下的目标形态近似，将导弹视角下的立体目标简化为长方形截面，然后运用基于关键点的遮蔽效果验证方法，用截面长方形的四个端点代替整个目标验证遮蔽效果。分别建立四点与导弹的线段方程，利用判断函数方程得出结果。

问题 2：对于问题二，利用无人机F_1投放 1 枚烟幕干扰弹实施对导弹M_1的干扰，需确定F_1的飞行方向、飞行速度、烟幕干扰弹投放点、烟幕干扰弹起爆点，使得遮蔽时间尽可能长。这是一个单无人机单弹的优化问题，需要建立无人机，导弹和干扰弹的球形方程，对无人机的飞行参数和烟幕弹的投放、起爆参数进行优化。

问题 3：利用无人机F_1投放 3 枚烟幕干扰弹实施对导弹M_1的干扰。这个问题是单无人机多弹的优化，要考虑多弹之间的时间和空间协同。在问题二的基础上增加了两枚干扰弹的投放，要考虑投弹间隔有时间限制，遮蔽时间可能有重叠，所以要让第一枚干扰弹尽早开始干扰，以避免干扰弹之间的相互干扰和覆盖，提高干扰的效率，并注意最终的有效遮蔽时间是三次时间的并集。

问题 4：利用F_1、F_2、F_3等 3 架无人机，各投放 1 枚烟幕干扰弹实施对M_1的干扰。属于多无人机单弹的协同优化问题，要考虑多无人机之间的协同配合。已知导弹做斜向下匀速直线运动，从 2000 米的高空极速降落。而三台无人机飞行高度分别为 1800、1400、700 米，且只能平行于地面飞行。干扰弹位置变化相对较慢，难以通过自身移动来调整与导弹的相对位置。为最大程度让干扰弹挡在导弹与目标之间，应按无人机高度从高到低的顺序，让干扰弹依次对导弹进行干扰。

问题 5：利用 5 架无人机，每架无人机至多投放 3 枚烟幕干扰弹，实施对M_1、M_2、M_3等 3 枚来袭导弹的干扰。需要设计多无人机多弹针对多导弹的投放策略，实现有效遮蔽并保存结果。这是最复杂的多无人机多弹多导弹的协同优化问题，需要综合考虑各方面的协同与约束。

五、 模型的建立与求解

5.1 建模前准备

5.1.1 有效遮蔽

考虑到题中所给的导弹坐标，$M_1(20000,0,2000)$等值，可知本题中长度尺度较大，故导弹可直接看做质点。以导弹为观察点，当烟雾球能使导弹看向目标的全部视角范围内，从导弹到目标表面各点的所有视线（线段）都与烟雾球存在交点时，即实现对目标的有效遮蔽。

5.1.2 真目标遮蔽的简化

如果遍历所有真目标上的点，那么计算量将大量增长，因此需要简化模型，选取少量点来代表真目标的形状。

要使选取的少量点能够代表真目标的形状，则需要满足：观测点到选取点的视角要尽量地与原图形接近。 记导弹与真目标的距离为L，真目标的尺寸D为其上最远点的距离，即$D=\sqrt{(2R)^2+H_t^2}$，则当$L \gg R_t$时，导弹视角θ极小，则有$\sin(\theta) \approx \theta$，故其视角可近似表示为：

$$\theta \approx \frac{D}{L} \tag{1}$$

为了简化模型，需要选取真目标O_t上距离尽量远的点。在三维直角坐标系构建的空间模型中，在前中期导弹与目标连线同 xoz 平面的夹角极小，趋近于零度。从导弹位置观察目标时，所看到的目标形态相当于其在 yoz 平面上的投影，该投影近似为一个长方形。

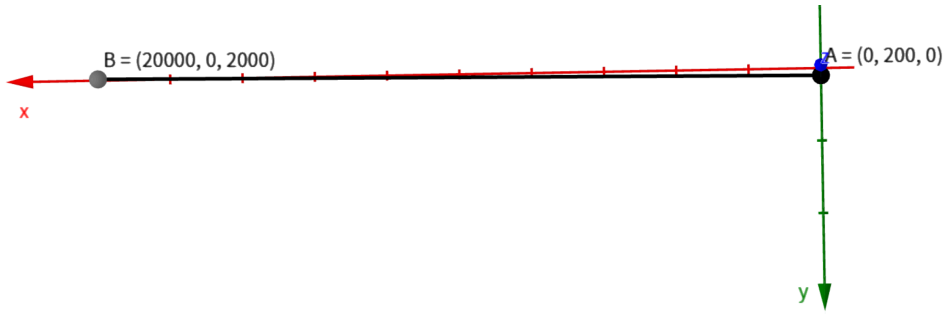


图 2　导弹视角下的目标投影

由此可知，导弹视角下目标的形状可以近似表示为长方形，四个顶点分别为$D_1(0,193,0)$, $D_2(0,193,10)$, $D_3(0,207,10)$, $D_4(0,207,0)$。 因此我们选取了真目标O_t的四个顶点D_1, D_2, D_3, D_4作为代表点，来近似表示真目标的形状。

5.2 问题一模型的建立与求解

根据题中信息建立三维直角坐标系，标定无人机，真假目标及导弹的坐标，：$F_1(17800,0,1800)$，真目标O_t的简化点$D_1(0,193,0)$, $D_2(0,193,10)$, $D_3(0,207,10)$, $D_4(0,207,0)$，假目标$O(0,0,0)$，$M_1(20000,0,2000)$，干扰弹投放策略已明确：

在烟幕干扰弹被投放前，干扰弹与无人机一起运动，对一架无人机进行分析，已知其初始坐标为$P_{FY_i}(0)\left(x_{fi0},y_{fi0},z_{fi0}\right)$,飞行速度为$u_i$，飞行速度与 x 轴正方向的夹角为$\theta$，由于无人机一直在一个水平面飞行，因此无人机的位置坐标函数为：

$$P(FY_i)(t)=\left(x_{fi0}+u_i\cos(\theta_i)t, y_{fi0}+u_i\sin(\theta_i)t, z_{fi0}\right) \tag{2}$$

以无人机收到任务指令为零时刻，设第 i 架无人机投放第 k 枚烟幕干扰弹的时刻为$t_{\text{lau},i,k}$，由于投放前烟幕干扰弹与无人机一起运动，故由无人机位置坐标函数可以得到烟幕干扰弹的位置坐标$P_{\text{lau},i,k}$为:

$$P_{\text{lau},i,k}=P_{FY_i}\left(t_{\text{lau},i,k}\right) \tag{3}$$

由于不考虑空气阻力，烟幕干扰弹从被投放到被引爆这段时间内竖直方向做自由落体运动，水平方向分速度与对应的无人机速度相同，可知在引爆时，烟幕干扰弹引爆点的坐标为：

$$P_{\text{det},i,k}=P_{FY_i}\left(t_{\text{det},i,k}\right)+\left(0,0,-\frac{1}{2}g\left(t_{\text{det},i,k}-t_{\text{lau},i,k}\right)^2\right) \tag{4}$$

依题意得，烟幕干扰弹被引爆后形成的云团以 3m/s 的速度匀速下沉，则可得到云团中心的位置坐标$C_{\text{cloud},i,k}$为:

$$C_{\text{cloud},i,k}(t)=P_{\text{det},i,k}+\left(0,0,-3\left(t-t_{\text{det},i,k}\right)\right),t\geq t_{\text{det},i,k} \tag{5}$$

又依题意得，云团中心 10m 范围内在 20s 的烟雾浓度能够实现有效遮蔽，则可用线段与云团中心的距离是否大于半径，判定是否实现有效遮蔽。

STEP2 建立导弹实时坐标模型：

导弹速度大小v_M为300m/s，方向始终指向假目标，为更好描述导弹的空间状态，建立导弹实时坐标模型

1）导弹飞行方向单位向量 设第 j 枚导弹初始位置为$P_{Mj0}\left(x_{j0},y_{j0},z_{j0}\right)$，假目标坐标为$O(0,0,0)$，则可以得到导弹运动方向的方向向量为$\overrightarrow{P_{Mj0}O}$。

2）时刻 t 导弹位置 设 t 时刻导弹位置坐标为$P_{mj}(t)$，题中导弹速度已知，则可以根据运动学公式得出时刻导弹的位置坐标为:

$$P_{mj}(t)=P_{Mj0}+v_M\cdot t\cdot\frac{\overrightarrow{P_{Mj0}O}}{\|\overrightarrow{P_{Mj0}O}\|} \tag{6}$$

3）导弹飞行到假目标时间为 $\frac{\|\overrightarrow{P_{Mj0}O}\|}{v_M}$，若导弹击中假目标，则针对该导弹的有效遮蔽时间不再考虑击中后根据判断函数求得的时间段。

STEP3 建立判断是否实现有效遮蔽模型

对于图 3 所示的位置简图， 由 **STEP2** 可求:

导弹的位置坐标P_{mj}，在t时刻针对 4 个顶点$D_{n(n=1,2,3,4)}$，有线段向量$\overrightarrow{P_{mj}(t)D_n}$;

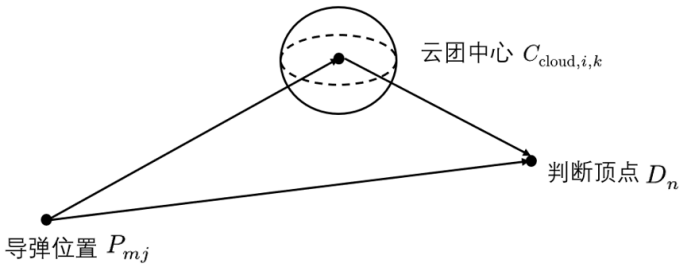


图 3　向量计算示意图

由 **STEP1** 求得:

有云团中心的坐标为=$C_{\text{cloud},ik}$，t时刻导弹到云团中心位置的向量为$\overrightarrow{P_{mj}C_{\text{cloud},i,k}}$，对每条线段$P_{m,j}D_{n(n=1,2,3,4)}$，记云团中心到线段$P_{m,j}D_n$的距离为$d_{\text{seg},n}$，云团中心到直线$P_{m,j}D_n$的距离为$d_{\text{line},n}$，根据点到直线距离的计算公式，求得中心到直线的距离$d_{\text{line},n}$为：

$$d_{\text{line},n}=\|\frac{\overrightarrow{P_{m,j}D_n}\times\overrightarrow{P_{m,j}C_{\text{cloud},i,k}}\|}{(\|\left(D_nP_{m,j}\right\|)} \tag{7}$$

再根据点积判断线段向量$D_nP_{m,j}$投影的位置，根据点积公式得到 $P_{m,j}D_n\cdot P_{m,j}C_{\text{cloud},i,k}$=$p_xb_{x,n}+p_yb_{y,n}+p_zb_z,n$;$D_nP_{m,j}\cdot D_nC_{\text{cloud},i,k}=\left(-b_{x,n}\right)\cdot\left(p_x-b_{x,n}\right)+\left(-b_{y,n}\right)\cdot\left(p_y-b_{y,n}\right)+\left(-b_{z,n}\right)\cdot\left(p_z-b_{z,n}\right)$ 因此我们得到云团中心到线段$P_{m,j}D_n$的距离$d_{\text{seg},n}$为： $d_{\text{seg},n}$=$\|P_{m,j}C_{\text{cloud},i,k}\|$

将得到的云团中心坐标代入 **STEP1** 中得到的有效遮蔽模型并使用遍历搜索进行求解，求得的答案为

起始遮蔽时间	结束遮蔽时间	有效遮蔽时间
$8.048s$	$9.448s$	$1.401s$

5.3 问题二模型的建立与求解

5.3.1 问题二模型的建立

此问题为单目标规划问题，题目需要确定无人机 FY1 的飞行方向及速度，烟幕干扰弹投放点及起爆点使得最大化遮蔽时间， 结合题意有以下约束条件：

1.无人机方向约束，0-2π

2.无人机速度约束，70-140 m/s

5.时间窗口约束，导弹击中目标后不再考虑干扰

首先明确导弹，无人机投放干扰弹引爆产生干扰云团中心和真目标的位置；通过问题一建立的模型可以得到导弹坐标(公式)，干扰弹投放点坐标(公式)，干扰弹引爆点坐标(公式)，爆炸后云团中心坐标(公式)，然后将相关参数代入问题一中建立的判断是否实现有效遮蔽模型联立参数方程解出。 因此可建立以下单目标优化模型：

决策变量：

$$\left(u_1, \theta_1, t, t_{\text{det},1,1}\right) \tag{8}$$

目标函数：

$$\sum_{0}^{t_{\max}} (\text{is\_effective}(t)\Delta t, t = 0..t_{\max}) \tag{9}$$

其中，$t_{\max}$为最大时间，is_effective(t)为指示函数，表明该时间点是否实现有效遮蔽，Δt为离散时间步。

约束条件：

$$\begin{cases} 70 \leq u_1 \leq 140 \\ 0 < t < t_{\max} \\ 0 \leq \theta_1 < 2\pi \end{cases} \tag{10}$$

5.3.2 问题二的求解

从图 4 中可以看出，由于问题二的决策变量（无人机飞行方向、速度，干扰弹投放点、起爆点等）涉及多维度参数且存在强耦合关系，导致高维解空间中满足所有约束的可行解分布极为稀疏，可行解的数量非常有限。

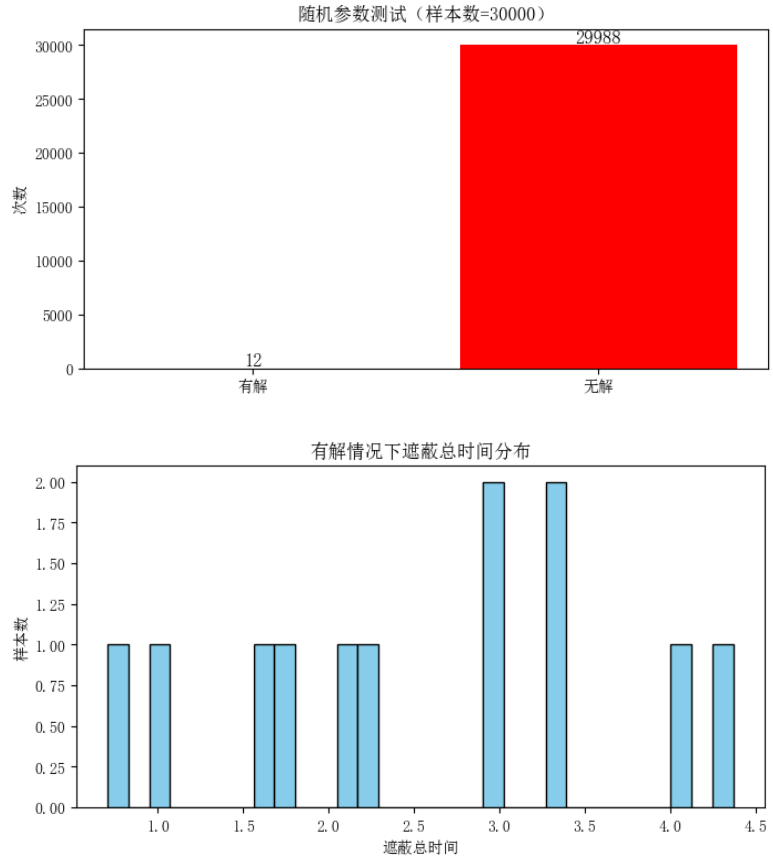


图 4　问题二解空间可行域示意

这种情况下，若直接采用“遍历搜索”，一方面解空间维度高、可行解占比极低，会造成极大的算力浪费；另一方面，目标函数的强非线性也使得普通优化算法容易陷入局部最优。因此，需要选择更具全局搜索能力的智能优化算法来应对这一挑战。

因此本文使用改进后的双重模拟退火算法实现这一目标。其具体流程如图 5 所示。该算法是传统 SA 基础上的启发式优化算法，核心为“外层参数优化 + 内层解空间搜索”的双层迭代框架：内层以传统 SA 为核心，外层将内层 SA 的关键控制参数作为自身优化变量，以“内层输出的最优解质量”为外层目标函数，通过退火机制优化内层参数，直至外层目标收敛，以此增强突破局部最优能力，更高效求全局最优解。

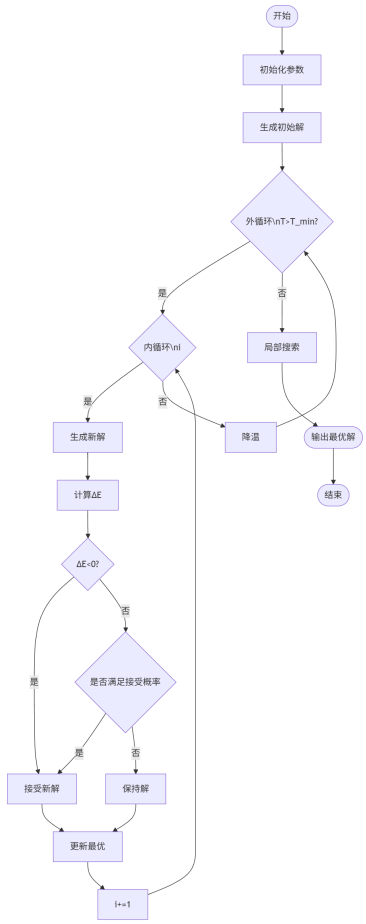


图 5　双重模拟退火算法流程图

确定初始温度 20000，迭代 8000 轮，得到的结果为：

u_1	θ_1	$t_{r,1,1}$	$t_{\det,1,1}$
81.45	3.907	0.002967	2.6504

此时有效遮蔽时间为 4.88s，进一步精细化搜索有：

其可视化结果如图 6 所示，最终求得的最优解为：

u_1	θ_1	$t_{r,1,1}$	$t_{\det,1,1}$
111			

其中各时段为:

起始遮蔽时间	结束遮蔽时间	有效遮蔽时间
2.78	$7.32s$	$4.54s$

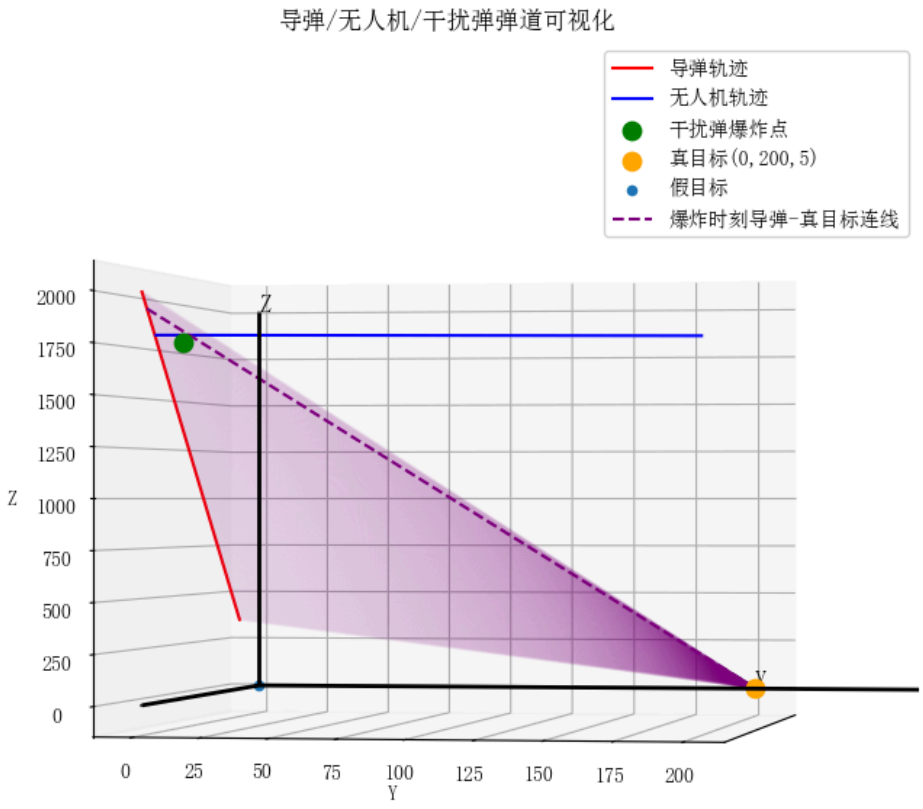


图 6　问题二结果

5.4 问题三模型的建立与求解

问题三：为单目标优化问题，但在问题二的基础上增加了两枚干扰弹的投放，投弹间隔有时间限制，遮蔽时间可能有重叠，所以最终的有效遮蔽时间是三次时间的并集 结合题意有以下约束条件： 1 无人机方向约束，0-2π
2 无人机速度约束，70-140m/s
3 投弹间隔约束，每两枚干扰弹投放时间间隔至少 1s
4 起爆时间约束，起爆在投放后
5 云团高度约束，云团不能沉入地底
6 时间窗口约束，导弹击中目标后不再考虑干扰
因此可建立以下单目标优化模型：
决策变量:

$$\left(u_1, \theta_1, t_{\text{lau},1,1}, t_{\text{det},1,1}, t_{\text{lau},1,2}, t_{\text{det},1,2}, t_{\text{lau},1,3}, t_{\text{det},1,3}\right) \tag{11}$$

目标函数:

$$\sum_{0}^{t_{\max}} (\text{is\_effective}(t)\Delta t, t = 0..t_{\max}) \tag{12}$$

约束条件：

$$\begin{cases} 70 \le u_1 \le 140 \\ 0 < t_{\text{lau},1,1} < t_{\max} \\ 0 \le \theta_1 < 2\pi \\ t_{\text{lau},1,2} - t_{\text{lau},1,1} \ge 1 \\ t_{\text{lau},1,3} - t_{\text{lau},1,2} \ge 1 \\ 0 \le t_{\text{det}} < 18.7 \end{cases} \tag{13}$$

考虑到问题三与问题二相似，有效遮蔽时长的计算涉多物理过程耦合，目标函数呈强非线性多局部最优特征，解空间维度较高，因此仍先采用双重模拟退火算法，后精细搜索目标区间，以获得全局最优解。

放一张问题 3 的图

5.5 问题四模型的建立与求解

5.5.1 模型的建立

仍为单目标优化问题，约束条件同问题二，在问题二的基础上，增设两台无人机，三台无人机与导弹和目标的距离都不同且差距较大，运动平面也不同，考虑到无人机的速度明显低于导弹速度，为更高效的干扰导弹，需提前规划无人机飞行路径，采取时序分段拦截的策略进行优化，因为导弹 M1 的坐标变化规律是 x 坐标随时间均匀递减，且 FY1,FY2,FY3 的 x 坐标相差较大，所以以导弹与无人机 x 坐标的差值作为时序分段的依据，安排 FY1,FY2,FY3 分别负责前期、中期、后期三个时段对导弹 M1 的干扰，使得三架无人机在干扰弹干扰时间段尽量不重叠的情况下最大化每枚干扰弹的有效遮蔽时间，同时也为设置更好的初始解创造条件。 综上，本题的优化模型如下：

决策变量

$$\left(u_i, \theta_i, t_{\text{lau},i,1}, t_{\text{det},i,1}\right), i = 1, 2, 3 \tag{14}$$

优化目标

$$\sum_{i=1}^{3} \left(\sum_{0}^{t_{\max}} (\text{is\_effective}(t_i) \Delta t) \right), t = 0..t_{\max} \tag{15}$$

其中,t_i表示第i架无人机的干扰弹的有效遮蔽时间。

约束条件

$$\begin{cases} 70 \le u_i \le 140 \\ 0 < t_{\text{lau},i,1} < t_{\max} \\ 0 \le \theta_i < 2\pi \\ 0 \le t_{\text{det},i,1} < 18.7 \end{cases} \tag{16}$$

其中，$t_{\text{det},i,1}$的上界由 求解步骤如下：

参数：

通过 python 代码求解得：

5.6 问题五模型的建立与求解

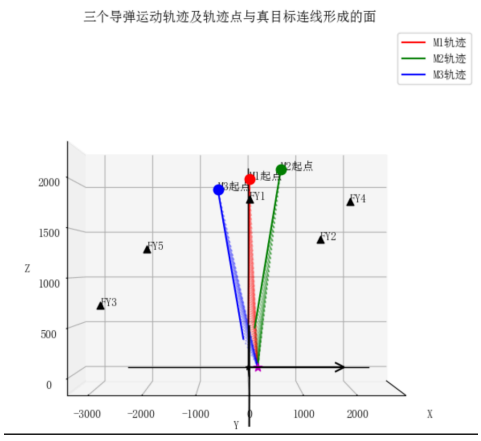
5.6.1 模型的建立

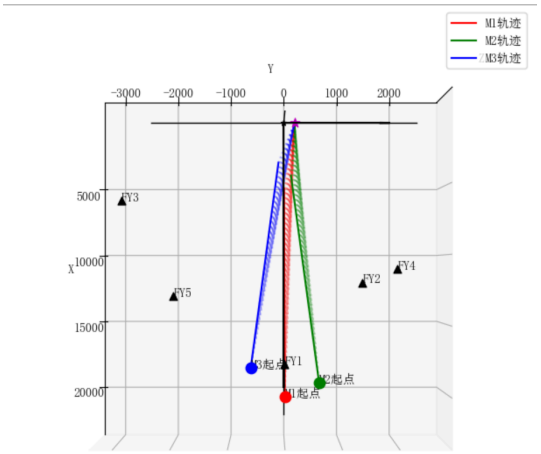
5.6.1.1 无人机调度策略的分析

问题三针对单无人机单导弹进行拦截，问题四针对多无人机单导弹进行拦截，问题五为多无人机多导弹的拦截问题，问题五的复杂度更高，解空间更大，若直接使用双重模拟退火算法求解，计算量将非常大，且不易收敛，因此需要对问题进行简化处理。

针对问题五的复杂度，本文提出了**“贪心算法+优先调度”**的两步求解策略：

首先通过单无人机多导弹的贪心算法，得到每架无人机对各枚导弹的初步干扰效果评估，筛选出每架无人机对各枚导弹的最佳干扰方案，作为后续多无人机多导弹调度的候选方案。





其次，通过优先调度策略，对候选方案进行排序，选择优先级最高的方案作为最终的调度方案。

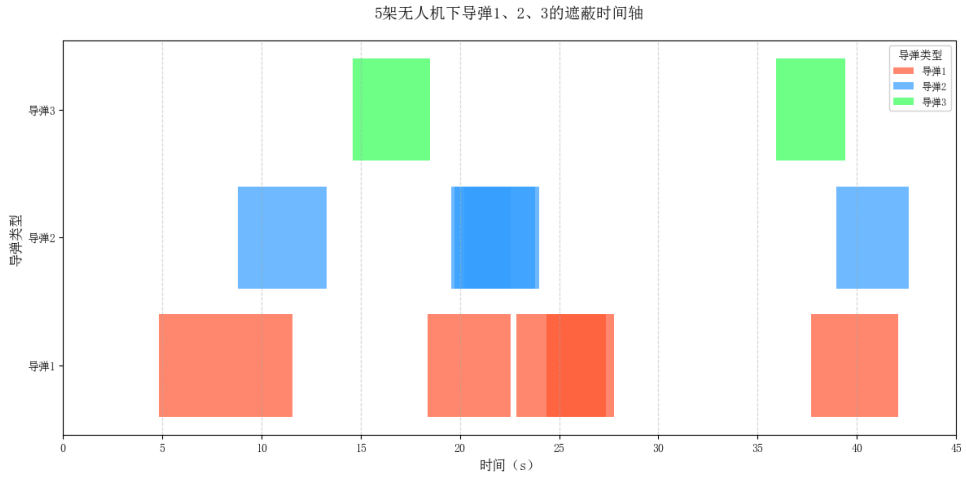
通过图 7 可知,F_2, F_3, F_4, F_5四架无人机分布在导弹两侧，且F_1无人机位于导弹的中心，且F_1无人机的飞行路径与其他无人机有较大差异，因此可以将F_1无人机单独调度，优先考虑F_1无人机对导弹的干扰效果，再考虑其他四架无人机的干扰效果。

5.6.1.2 通过贪心获取初步解

针对每架无人机，计算其对每枚导弹的最大有效遮蔽时间，由此得到初步答案，获得的初步参数为:

无人机	u_i	θ_i	$t_{\text{lau},i,1}$	$t_{\text{lau},i,2}$	$t_{\text{lau},i,3}$	$t_{\text{det},i,1}$	$t_{\text{det},i,2}$	$t_{\text{det},i,3}$
F_1	140.0	3.134	0.006567	3.649	15.45	3.609	5.318	7.988
F_2	114.6	4.987	6.016	7.219	8.694	2.789	5.274	0.3911
F_3	102.2	2.204	28.26	29.62	31.26	7.660	8.080	7.676
F_4	125.3	4.648	3.600	28.76	30.44	11.98	18.07	1.182
F_5	139.5	2.140	11.41	12.62	17.82	2.363	4.483	2.397

其产生的遮蔽时间，可以形成如图 8 所示遮蔽时间图，。



5.6.2 模型求解

六、 模型评价

6.1 结果检验与分析

6.1.1 真目标简化的分析

6.2 优缺点分析

模型优点：

1.将真目标简化为四点，简化计算。从导弹观察角度出发，结合实际数据和场景，将圆柱体简化为平面图形，再进一步用四个点代替整个图形，这一近似处理在保证一定准确性的前提下，显著简化了后续遮蔽效果判断的复杂度，提高了评估效率与可操作性，大大降低了计算量。

2.根据具体数据分析，设计优先级，有序投放各枚干扰弹，简化计算

模型缺点：

1.

七、 参考文献与引用

参考文献对于一篇正式的论文来说是必不可的，在建模中重要的参考文献当然应该列出。Typst 支持使用 BibTeX 来管理参考文献。在 `refs.bib` 文件中添加参考文献的信息，然后在正文中使用 `#cite(<引用的文献的 key>)` 来引用文献。例如：[1]。最后通过 `#bib(bibliography("refs.bib"))` 来生成参考文献列表。

参考文献

[1] ASTLEY R, MORRIS L. At-scale impact of the Net Wok: A culinarically holistic investigation of distributed dumplings[J]Armenian Journal of Proceedings, 2020, 61: 192-219

附录 1：

第一问求解

```
# 参数设置
from calc_cover_time import *
import numpy as np
speed = 120
direction_angle = np.pi
throw_time = 1.5
burst_delay = 3.6
from calc_cover_time import *

smoke_center = get_smoke_center(f1, direction_angle, speed, throw_time, burst_delay, t_list)
missile_traj = get_missile_traj(M1, fake_target, v_M1, t_list)
total_cover_time = get_missile_cover_time_corners(smoke_center, missile_traj,
true_target_corners, smoke_R, t_list,debug=True)
```

附录 2：

问题一

```
from calc_cover_time import *
import numpy as np
speed = 120
direction_angle = np.pi
throw_time = 1.5
burst_delay = 3.6
from calc_cover_time import *
# 烟雾球轨迹
smoke_center = get_smoke_center(f1, direction_angle, speed, throw_time, burst_delay, t_list)
# 导弹轨迹
missile_traj = get_missile_traj(M1, fake_target, v_M1, t_list)
# 遮蔽总时间
total_cover_time = get_missile_cover_time_corners(smoke_center, missile_traj,
true_target_corners, smoke_R, t_list,debug=True)
```

附录 3：

问题 2 求解

```
import matplotlib.pyplot as plt
from scipy.optimize import dual_annealing
from calc_cover_time import *
import numpy as np
# 优化目标函数
# x: [speed, angle, throw_time, burst_delay]
cover_time_hist = []
missile_traj = get_missile_traj(M1, fake_target, v_M1, t_list)
def sa_objective(x):
    global missile_traj
    position = f1  # 无人机初始点
    speed = x[0]
    direction_angle = x[1]
    throw_time = x[2]
    burst_delay = x[3]
   smoke_center = get_smoke_center(position, direction_angle, speed, throw_time, burst_delay,
t_list)
     cover_time = get_missile_cover_time(smoke_center, missile_traj, true_target, smoke_R,
t_list)
    cover_time_hist.append(cover_time)
    # 退火算法是最小化，这里返回负值
```

```
    return -cover_time

# 参数边界：[速度， 角度， 投弹时间， 爆炸间隔]
bounds = [(70, 140), (np.pi/2, np.pi * 3 / 2), (0, 60), (0, 20)]
# 定义回调函数，每次迭代后更新进度条
result = dual_annealing(
    func=sa_objective,
    bounds=bounds,
    maxiter=8000,
    seed=123,
    initial_temp=1000,

)

print('最优参数:', result.x)
print('最大遮挡总时间:', -result.fun)

plt.xlabel('迭代步数')
plt.ylabel('遮挡总时间')
plt.title('退火优化过程中遮挡总时间变化曲线')
plt.show()
```

附录 4：

问题 3 求解

```
from calc_cover_time import *
from scipy.optimize import dual_annealing
cover_time_hist = []
missile_traj = get_missile_traj(M1, fake_target, v_M1, t_list)
def sa_objective(x):
    try:
        speed = x[0]
        direction_angle = x[1]
        throw_times = x[2:5]
        burst_delays = x[5:8]
           smoke_centers = get_smoke_center_multi(f1, direction_angle, speed, throw_times,
burst_delays, t_list)
            cover_time = get_missile_cover_time_multi_corners(smoke_centers, missile_traj,
true_target_corners, smoke_R, t_list)
        cover_time_hist.append(cover_time)
        return -cover_time
    except Exception:
        return 1e6  # 参数非法时返回极小值
# 参数边界：[速度， 角度， 投弹 1， 投弹 2， 投弹 3， 爆炸 1， 爆炸 2， 爆炸 3]
bounds = [
    (result.x[0] * 0.8, result.x[0] * 1.2),           # speed
    (result.x[1] - 0.2, result.x[1] + 0.2),        # angle
    (result.x[2] * 0.8, result.x[2] * 1.2), (result.x[3] * 0.8, result.x[4] * 1.2), (0, 10),
# throw_times
    (0, 10), (0, 10), (0, 10),  # burst_delays
]
result_corners = dual_annealing(sa_objective,
                          bounds,
                          maxiter=3000,
                          seed=42,
                          initial_temp=10000,
                          initial_state=result.x
                         )
print("最优参数:", result_corners.x)
```

```
print("最大遮挡总时间:", -result_corners.fun)
pbar.close()
```

附录 5:

问题 4 求解

```
# 分别对三架无人机进行优化，最后的结果不重合，直接相加即可得出答案
from scipy.optimize import dual_annealing
import matplotlib.pyplot as plt

# 优化目标函数
# x: [speed, angle, throw_time, burst_delay]
cover_time_hist = []
missile_traj = get_missile_traj(M1, fake_target, v_M1, t_list)
def sa_objective(x):
    global missile_traj
    position = f1_position  # 无人机初始点
    speed = x[0]
    direction_angle = x[1]
    throw_time = x[2]
    burst_delay = x[3]
   smoke_center = get_smoke_center(position, direction_angle, speed, throw_time, burst_delay,
t_list)
     cover_time = get_missile_cover_time(smoke_center, missile_traj, true_target, smoke_R,
t_list)
    cover_time_hist.append(cover_time)
    # 退火算法是最小化，这里返回负值
    return -cover_time

# 参数边界：[速度，角度，投弹时间，爆炸间隔]
bounds = [(70, 140), (0, np.pi *2), (0, 60), (0.1, 20)]


# 定义回调函数，每次迭代后更新进度条
def progress_callback(x, f, context):
    pbar.update(1)  # 每次迭代更新进度条
result = dual_annealing(
    func=sa_objective,
    bounds=bounds,
    maxiter=max_iter,
    seed=123,
    initial_temp=1000,

)

print('最优参数:', result.x)
print('最大遮挡总时间:', -result.fun)

from scipy.optimize import minimize

# 局部精细优化目标函数
def local_objective(x):
    try:
        speed = x[0]
        direction_angle = x[1]
        t0 = np.clip(x[2], 0, 60)
        t1 = np.clip(max(x[3], t0+1), 0, 60)
        t2 = np.clip(max(x[4], t1+1), 0, 60)
        throw_times = np.array([t0, t1, t2])
```

```
        burst_delays = x[5:8]
           smoke_centers = get_smoke_center_multi(f1, direction_angle, speed, throw_times,
burst_delays, t_list)
                cover_time = get_missile_cover_time_corners(smoke_centers, missile_traj,
true_target_corners, smoke_R, t_list)
        return -cover_time
    except Exception:
        return 1e6

# 局部优化边界（以 result.x 为中心，±10% 范围）
bounds_local = [
    (max(70, result.x[0]*0.9), min(140, result.x[0]*1.1)),
    (max(0, result.x[1]-0.1), min(np.pi*2, result.x[1]+0.1)),
    (max(0, result.x[2]-3), min(60, result.x[2]+3)),
    (max(0, result.x[3]-0.5), min(20, result.x[3]+0.5)),
]

# 使用 L-BFGS-B 进行局部优化
result_local = minimize(local_objective, result.x, bounds=bounds_local, method='L-BFGS-B')

print("局部优化后参数:", result_local.x)
print("最大遮挡总时间:", -result_local.fun)

smoke_center = get_smoke_center(f1_position, result_local.x[1], result_local.x[0],
result_local.x[2], result_local.x[3], t_list)
get_missile_cover_time_corners(smoke_center, missile_traj, true_target_corners, smoke_R,
t_list, debug=True)
```

附录 6：

问题 5 求解

```
import numpy as np
from scipy.optimize import minimize
from scipy.optimize import dual_annealing
from calc_cover_time import *

params = np.array([
       1.39947949e+02, 3.13366040e+00, 6.56731053e-03, 3.64855823e+00, 1.54525393e+01,
3.60924375e+00, 5.31779428e+00, 7.98774517e+00,
    114.62860037,   4.98699612,   6.01566818,   7.21914132,   8.69434875,   2.78934729,
5.27433416,   0.39108412,
    102.20517464,   2.20408204,  28.25700456,  29.62434123,  31.26438227,   7.66011547,
8.07963526,   7.67639008,
    125.33515088,   4.64755589,   3.60006083,  28.75823868,  30.44272478,  11.98307911,
18.06998011,   1.181712,
     139.51705493,   2.13999875,     11.40811482,   12.61961772, 17.82003334, 2.36261189,
4.48262589, 2.396845741
])


def optimize_drones_params(params, drone_positions, missile_trajs, true_target_corners,
smoke_radius, t_list):
    """
    params: shape (40,), 初始参数
    drone_positions: dict, 5 个无人机位置
    missile_trajs: list of 3 np.ndarray, 每个为(3, len(t_list))
    true_target_corners: list of 4 np.array([x, y, z])
    smoke_radius: float
```

```
    t_list: np.array
    返回：优化后的参数、最大遮蔽总时间
    """
    # 参数 reshape 为(5,8)
    params = np.array(params)
    bounds = []
    for i, p in enumerate(params):
        lower = p * 0.9
        upper = p * 1.1
        bounds.append((lower, upper))

    def objective(x):
        try:
            x = np.array(x).reshape(5, 8)
            return -get_all_drones_missile_cover_times_sum(
                    drone_positions, x, missile_trajs, true_target_corners, smoke_radius,
t_list, debug=False
            )
        except AssertionError:
            return 1e6  # 返回极差值，惩罚不合法参数
    result = dual_annealing(
        func=objective,
        bounds=bounds,
        maxiter=100,
        initial_temp=1000,
    )

    best_x = result.x.reshape(5, 8)
    print("优化结束，最优参数如下：")
    print(best_x)
    print("\n详细遮蔽信息：")
    total_cover_time = get_all_drones_missile_cover_times_sum(
        drone_positions, best_x, missile_trajs, true_target_corners, smoke_radius, t_list,
debug=True
    )
    print(f"\n最大遮蔽总时间：{total_cover_time:.6f}s")
    return best_x, total_cover_time


best_x,  total_cover_time  =  optimize_drones_params(params,  drone_pos,  missile_trajs,
true_target_corners, smoke_R, t_list)
```

附录 7：

各种常量及工具函数

```
import math
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = ["SimSun"]
plt.rcParams['axes.unicode_minus'] = False
import numpy as np
M1 = np.array([20000, 0, 2000])
f1 = np.array([17800, 0, 1800])
v_f1 = 120
v_M1 = 300
v_smoke = -3
g = 9.80655
fake_target = np.array([0,0,0])
true_target = np.array([0, 200, 5])  # 此为质心，高为 10，半径为 7
true_target_R = 7
```

```
true_target_H = 10
smoke_R = 10
t_list = np.arange(0, 50, 0.001)
true_target_corner_1 = true_target - np.array([0, true_target_R, true_target_H / 2])
true_target_corner_2 = true_target - np.array([0, true_target_R, -true_target_H / 2])
true_target_corner_3 = true_target - np.array([0, -true_target_R, true_target_H / 2])
true_target_corner_4 = true_target - np.array([0, -true_target_R, -true_target_H / 2])
smoke_duration = 20  # 烟雾存续时间
true_target_corners  =  [true_target_corner_1,  true_target_corner_2,  true_target_corner_3,
true_target_corner_4]

bounds_range = [
     [(130,140),   (np.pi/2,np.pi*3/2),   (0,13.87), (1,13.87), (2, 13.87),       (0,18.97),
(0,18.97),(0,18.97),   ],# 第一架飞机的参数限制
    [(70,140),    (np.pi, np.pi*2),      (0, 50),    (1,50),     (2,50),          (0,16.74),(0,
16.74),(0, 16.74), ],  # 第二架飞机的参数限制
    [(70,140),    (0,np.pi),             (0,60),     (1,60),     (2,60),          (0, 11.9),(0,
11.9),(0, 11.9),   ],# 第三架飞机的参数限制
    [(70,140),    (np.pi, np.pi*2),      (0, 56),    (0, 56),    (0, 56),         (0, 18.97),(0,
18.97),(0, 18.97),],   # 第四架飞机的参数限制
     [(70,140),    (0, np.pi),             (0, 43.7), (1, 43.7), (2, 43.7),       (0,16.12),
(0,16.12),(0,16.12),   ],# 第五架飞机的参数限制
]

missile_starts = [
    np.array([20000, 0, 2000]),
    np.array([19000, 600, 2100]),
    np.array([18000, -600, 1900])
]

drone_pos = {
    0: np.array([17800,0,1800]),
    1: np.array([12000,1400,1400]),
    2: np.array([6000,-3000,700]),
    3: np.array([11000,2000,1800]),
    4: np.array([13000,-2000,1300])
}

def get_missile_traj(start, target, v, t_list):
    """
    start: np.array([x0, y0, z0]) 导弹初始点
    target: np.array([x1, y1, z1]) 目标点 //为原点的假目标
    v: float 导弹速度 (m/s)
    t_list: np.array 时间序列
    返回: (3, T) 导弹轨迹
    """
    direction = (target - start) / np.linalg.norm(target - start)
    traj = start[:, np.newaxis] + v * t_list * direction[:, np.newaxis]
    return traj

missile_trajs = np.array([get_missile_traj(missile_starts[i], fake_target, v_M1, t_list) for
i in range(3)])


def get_smoke_center(position, direction_angle, speed, throw_time, burst_delay, t_list):
    """
    position: np.array([x, y, z]) 无人机初始坐标
    direction_angle: float, 水平飞行方向 (0~2pi, 弧度)
    speed: float, 无人机飞行速度
    throw_time: float, 投弹时间
```

```
    burst_delay: float, 投弹后干扰弹再爆炸的时间
    t_list: np.array, 时间序列
    返回: (3, len(t_list)) 烟雾球心轨迹, 未爆炸/超时高度为 1e6
    """
    global g
    global v_smoke
    global smoke_duration
    global smoke_R

    direction = np.array([np.cos(direction_angle), np.sin(direction_angle), 0.0])
    t_burst = throw_time + burst_delay
    smoke_center = np.ones((3, len(t_list))) * 1e6  # 默认高度为 1e6
    valid_idx = (t_list >= t_burst) & (t_list <= t_burst + smoke_duration)

    # 把抛出后的水平运动也算进无人机位移里
     smoke_start = position + (direction * speed * t_burst) - 0.5 * np.array([0, 0, g]) *
burst_delay ** 2
    t_valid = t_list[valid_idx] - t_burst
    smoke_center[:, valid_idx] = smoke_start.reshape(3, 1)
    smoke_center[2, valid_idx] += v_smoke * t_valid  # 仅竖直速度, 无重力

    return smoke_center


def  get_smoke_center_multi(position,  direction_angle,  speed,  throw_times,  burst_delays,
t_list):
    """
    position: np.array([x, y, z]) 无人机初始坐标
    direction_angle: float, 水平飞行方向 (0~2pi, 弧度)
    speed: float, 无人机飞行速度
    throw_times: 长度为 3 的递增数组, 三次投弹时间, 且相邻间隔>=1
    burst_delays: 长度为 3 的数组, 对应每次投弹后的爆炸延迟
    t_list: np.array, 时间序列
    返回: (3, len(t_list), 3) 三个干扰弹的球心轨迹
    """
    global g
    global v_smoke
    global smoke_duration

    throw_times = np.array(throw_times)
    burst_delays = np.array(burst_delays)
    assert len(throw_times) == 3 and len(burst_delays) == 3
    assert np.all(np.diff(throw_times) >= 1), "投弹时间需递增且间隔>=1"

    direction = np.array([np.cos(direction_angle), np.sin(direction_angle), 0.0])
    smoke_centers = np.ones((3, len(t_list), 3)) * 1e6  # (x,y,z), 时间, 弹序号

    for i in range(3):
        t_burst = throw_times[i] + burst_delays[i]
        valid_idx = (t_list >= t_burst) & (t_list <= t_burst + smoke_duration)
         smoke_start = position + (direction * speed * t_burst) - 0.5 * np.array([0, 0, g])
* burst_delays[i]**2
        t_valid = t_list[valid_idx] - t_burst
        smoke_centers[:, valid_idx, i] = smoke_start.reshape(3, 1)
        smoke_centers[2, valid_idx, i] += v_smoke * t_valid

    return smoke_centers

def    get_smoke_throw_and_burst_point(position,    direction_angle,    speed,    throw_time,
burst_delay):
```

```
    """
    输入：
        position: np.array([x, y, z]) 无人机初始坐标
        direction_angle: float, 水平飞行方向 (0~2pi, 弧度)
        speed: float, 无人机飞行速度
        throw_time: float, 投弹时间
        burst_delay: float, 投弹后干扰弹再爆炸的时间
    返回：
        throw_point: np.array([x, y, z]) 投弹点
        burst_point: np.array([x, y, z]) 爆炸点
    """
    g = 9.80655  # 重力加速度
    direction = np.array([np.cos(direction_angle), np.sin(direction_angle), 0.0])
    # 投弹点：无人机在投弹时刻的位置
    throw_point = position + direction * speed * throw_time
    # 爆炸点：投弹点基础上，干扰弹在空中飞行 burst_delay 后的位置（只考虑重力影响）
    burst_point = throw_point - 0.5 * np.array([0, 0, g]) * burst_delay ** 2
    return throw_point, burst_point



def get_cover_intervals(covered, t_list):
    """
    covered: bool 数组，表示每一时刻是否被遮挡
    t_list: 时间序列
    返回：intervals 列表，每个元素为(起始时刻，结束时刻)
    """
    covered = np.asarray(covered)
    # 找到遮挡区间的起止点
    changes = np.diff(covered.astype(int))
    starts = np.where(changes == 1)[0] + 1
    ends = np.where(changes == -1)[0] + 1

    # 如果开头就是遮挡
    if covered[0]:
        starts = np.insert(starts, 0, 0)
    # 如果结尾是遮挡
    if covered[-1]:
        ends = np.append(ends, len(covered))

    intervals = [(t_list[s], t_list[e-1]) for s, e in zip(starts, ends)]
    return intervals



def  get_missile_cover_time(smoke_center,  missile_traj,  true_target,  smoke_radius,  t_list,
debug=False):

    """
    smoke_center: (3, T) 烟雾球心轨迹
    missile_traj: (3, T) 导弹轨迹
    true_target: np.array([x, y, z]) 目标点
    smoke_radius: float, 烟雾半径
    t_list: np.array, 时间序列
    返回：遮挡总时间 (float)，并打印所有遮挡区间的起止时刻
    """
    P = smoke_center.T  # shape (T, 3)
    A = missile_traj.T
    B = np.broadcast_to(true_target, A.shape)
```

```
    AB = B - A
    AP = P - A
    cross = np.cross(AP, AB)
    dist = np.linalg.norm(cross, axis=1) / np.linalg.norm(AB, axis=1)
    t_proj = np.sum(AP * AB, axis=1) / np.sum(AB * AB, axis=1)
    on_segment = (t_proj >= 0) & (t_proj <= 1)
    covered = (dist < smoke_radius) & on_segment

    # 统计遮挡区间
    total_cover_time = np.sum(covered) * (t_list[1] - t_list[0])
    # 打印所有遮挡区间
    if debug:
        intervals = get_cover_intervals(covered, t_list)
        for start, end in intervals:
            print(f'遮挡区间: {start:.2f}s ~ {end:.2f}s')

        print(f'总遮挡时间: {total_cover_time:.2f}s')

    return total_cover_time



def    get_missile_cover_time_corners(smoke_center,    missile_traj,    true_target_corners,
smoke_radius, t_list, debug=False):
    """
    smoke_center: (3, T) 烟雾球心轨迹
    missile_traj: (3, T) 导弹轨迹
    true_target_corners: list of 4 np.array([x, y, z]) 目标四个角点
    smoke_radius: float, 烟雾半径
    t_list: np.array, 时间序列
    返回: 遮挡总时间 (float), 并打印所有遮挡区间的起止时刻
    """
    covered_all = np.ones(len(t_list), dtype=bool)
    for corner in true_target_corners:
        P = smoke_center.T  # shape (T, 3)
        A = missile_traj.T
        B = np.broadcast_to(corner, A.shape)
        AB = B - A
        AP = P - A
        cross = np.cross(AP, AB)
        t_proj = np.sum(AP * AB, axis=1) / np.sum(AB * AB, axis=1)
        on_segment = (t_proj >= 0) & (t_proj <= 1)
        dist = np.linalg.norm(cross, axis=1) / np.linalg.norm(AB, axis=1)
        covered = (dist < smoke_radius) & on_segment
        covered = dist < smoke_radius
        covered_all &= covered  # 只有所有角都被遮挡才算遮挡


    total_cover_time = np.sum(covered_all) * (t_list[1] - t_list[0])
    # 打印所有遮挡区间
    if debug:
        intervals = get_cover_intervals(covered_all, t_list)
        for start, end in intervals:
            print(f'遮挡区间: {start:.2f}s ~ {end:.2f}s')
        print(f'总遮挡时间: {total_cover_time:.2f}s')
    return total_cover_time


def  get_missile_cover_time_multi(smoke_centers,  missile_traj,  true_target,  smoke_radius,
t_list, debug=False):
```

```
    """
    smoke_centers: (3, len(t_list), N) 多个烟雾球心轨迹 (N为弹数)
    missile_traj: (3, len(t_list)) 导弹轨迹
    true_target: np.array([x, y, z]) 目标点
    smoke_radius: float, 烟雾半径
    t_list: np.array, 时间序列
    返回: 遮挡总时间 (float), 并打印所有遮挡区间的起止时刻
    """
    T = len(t_list)
    N = smoke_centers.shape[2]
    covered_each = np.zeros((T, N), dtype=bool)
    A = missile_traj.T
    B = np.broadcast_to(true_target, A.shape)
    AB = B - A

    for i in range(N):
        P = smoke_centers[:, :, i].T  # shape (T, 3)
        AP = P - A
        cross = np.cross(AP, AB)
        dist = np.linalg.norm(cross, axis=1) / np.linalg.norm(AB, axis=1)
        t_proj = np.sum(AP * AB, axis=1) / np.sum(AB * AB, axis=1)
        on_segment = (t_proj >= 0) & (t_proj <= 1)
        covered_each[:, i] = (dist < smoke_radius) & on_segment

    # 只要任意一个烟雾球遮挡就算遮挡
    covered = np.any(covered_each, axis=1)

    total_cover_time = np.sum(covered) * (t_list[1] - t_list[0])

    if debug:
        print("每个烟雾球的遮挡时间及区间：")
        for i in range(N):
            single_cover_time = np.sum(covered_each[:, i]) * (t_list[1] - t_list[0])
            print(f"第{i+1}个烟雾球遮挡时间：{single_cover_time:.2f}s", end="下面是各区间：")
            intervals_i = get_cover_intervals(covered_each[:, i], t_list)
            for start, end in intervals_i:
                print(f"    区间：{start:.2f}s ~ {end:.2f}s")
        print("")
        intervals = get_cover_intervals(covered, t_list)
        print("总遮挡区间：")
        for start, end in intervals:
            print(f'    {start:.2f}s ~ {end:.2f}s')
        print(f'总遮挡时间：{total_cover_time:.2f}s')
    return total_cover_time



def get_missile_cover_time_multi_corners(smoke_centers, missile_traj, true_target_corners,
smoke_radius, t_list, debug=False):
    """
    smoke_centers: (3, len(t_list), N) 多个烟雾球心轨迹 (N为弹数)
    missile_traj: (3, len(t_list)) 导弹轨迹
    true_target_corners: list of 4 np.array([x, y, z]) 目标四个角点
    smoke_radius: float, 烟雾半径
    t_list: np.array, 时间序列
    返回: 遮挡总时间 (float), 并打印所有遮挡区间的起止时刻
    """
    T = len(t_list)
    N = smoke_centers.shape[2]
    covered_each = np.ones((T, N), dtype=bool)
```

```
    A = missile_traj.T

    for i in range(N):
        P = smoke_centers[:, :, i].T  # shape (T, 3)
        covered_corners = np.ones(T, dtype=bool)
        for corner in true_target_corners:
            B = np.broadcast_to(corner, A.shape)
            AB = B - A
            AP = P - A
            cross = np.cross(AP, AB)
            t_proj = np.sum(AP * AB, axis=1) / np.sum(AB * AB, axis=1)
            on_segment = (t_proj >= 0) & (t_proj <= 1)
            dist = np.linalg.norm(cross, axis=1) / np.linalg.norm(AB, axis=1)
            covered = (dist < smoke_radius) & on_segment
            covered_corners &= covered  # 所有角都被遮挡才算
        covered_each[:, i] = covered_corners

    # 只要任意一个烟雾球遮挡所有角就算遮挡
    covered = np.any(covered_each, axis=1)
    total_cover_time = np.sum(covered) * (t_list[1] - t_list[0])

    if debug:
        print("每个烟雾球的遮挡时间及区间（所有角都被遮挡）：")
        for i in range(N):
            single_cover_time = np.sum(covered_each[:, i]) * (t_list[1] - t_list[0])
            print(f"第{i+1}个烟雾球遮挡时间：{single_cover_time:.2f}s", end=" 下面是各区间：")
            intervals_i = get_cover_intervals(covered_each[:, i], t_list)
            for start, end in intervals_i:
                print(f"    区间：{start:.2f}s ~ {end:.2f}s")
        print("")
        intervals = get_cover_intervals(covered, t_list)
        print("总遮挡区间：")
        for start, end in intervals:
            print(f'    {start:.2f}s ~ {end:.2f}s')
        print(f'总遮挡时间：{total_cover_time:.2f}s')
    return total_cover_time



def get_missile_cover_time_multi_bomb_and_missile(smoke_centers, missile_trajs, true_target,
smoke_radius, t_list, debug=False):
    """
    smoke_centers: (3, len(t_list), N) 多个烟雾球心轨迹（N为弹数）
    missile_trajs: list of 3 np.ndarray，每个为(3, len(t_list))，三个导弹轨迹
    true_target: np.array([x, y, z]) 目标点
    smoke_radius: float，烟雾半径
    t_list: np.array，时间序列
    返回：所有导弹被遮蔽时间的总和（float），debug时输出每次投弹对每个导弹的遮蔽区间
    """
    T = len(t_list)
    N = smoke_centers.shape[2]
    M = len(missile_trajs)
    # 每个导弹的遮蔽情况
    covered_all_missiles = np.zeros((M, T), dtype=bool)
    covered_each = np.zeros((M, T, N), dtype=bool)

    for m in range(M):
        A = missile_trajs[m].T
        B = np.broadcast_to(true_target, A.shape)
```

```
        AB = B - A
        for i in range(N):
            P = smoke_centers[:, :, i].T  # shape (T, 3)
            AP = P - A
            cross = np.cross(AP, AB)
            dist = np.linalg.norm(cross, axis=1) / np.linalg.norm(AB, axis=1)
            t_proj = np.sum(AP * AB, axis=1) / np.sum(AB * AB, axis=1)
            on_segment = (t_proj >= 0) & (t_proj <= 1)
            covered_each[m, :, i] = (dist < smoke_radius) & on_segment
        # 只要任意一个烟雾球遮挡就算遮挡
        covered_all_missiles[m] = np.any(covered_each[m], axis=1)

    # 计算每个导弹被遮蔽的时间
    missile_cover_times = []
    for m in range(M):
        cover_time = np.sum(covered_all_missiles[m]) * (t_list[1] - t_list[0])
        missile_cover_times.append(cover_time)

    total_cover_time = sum(missile_cover_times)

    if debug:
        for m in range(M):
            print(f"导弹{m+1}遮蔽情况：")
            for i in range(N):
                single_cover_time = np.sum(covered_each[m, :, i]) * (t_list[1] - t_list[0])
                print(f"  第{i+1}个烟雾球遮蔽时间：{single_cover_time:.2f}s", end=" 区间：")
                intervals_i = get_cover_intervals(covered_each[m, :, i], t_list)
                for start, end in intervals_i:
                    print(f"    {start:.2f}s ~ {end:.2f}s")
            intervals_m = get_cover_intervals(covered_all_missiles[m], t_list)
            print(f"导弹{m+1}总遮蔽区间：")
            for start, end in intervals_m:
                print(f"    {start:.2f}s ~ {end:.2f}s")
            print(f"导弹{m+1}总遮蔽时间：{missile_cover_times[m]:.2f}s")
        print(f'所有导弹被遮蔽时间总和：{total_cover_time:.2f}s')
    return total_cover_time



def get_missile_cover_bool_multi_corners(smoke_centers, missile_trajs, true_target_corners,
smoke_radius, t_list, debug=False):
    """
    smoke_centers: (3, len(t_list), 3)  # 一个无人机放出的三个烟雾球
    missile_trajs: list of 3 np.ndarray, 每个为(3, len(t_list)), 三个导弹轨迹
    true_target_corners: list of 4 np.array([x, y, z]) 目标四个角点
    smoke_radius: float
    t_list: np.array
    返回: covered_all_missiles (3, len(t_list)), 每个导弹每个时刻是否被遮蔽
    """
    T = len(t_list)
    N = smoke_centers.shape[2]
    M = len(missile_trajs)
    covered_all_missiles = np.zeros((M, T), dtype=bool)
    covered_each = np.ones((M, T, N), dtype=bool)

    for m in range(M):
        A = missile_trajs[m].T  # (T, 3)
        for i in range(N):
            P = smoke_centers[:, :, i].T  # (T, 3)
            covered_corners = np.ones(T, dtype=bool)
```

```
            for corner in true_target_corners:
                B = np.broadcast_to(corner, A.shape)
                AB = B - A
                AP = P - A
                cross = np.cross(AP, AB)
                t_proj = np.sum(AP * AB, axis=1) / np.sum(AB * AB, axis=1)
                on_segment = (t_proj >= 0) & (t_proj <= 1)
                dist = np.linalg.norm(cross, axis=1) / np.linalg.norm(AB, axis=1)
                covered = (dist < smoke_radius) & on_segment
                covered_corners &= covered  # 所有角都被遮挡才算
            covered_each[m, :, i] = covered_corners
        # 只要任意一个烟雾球遮挡所有角就算遮挡
        covered_all_missiles[m] = np.any(covered_each[m], axis=1)

    if debug:
        for m in range(M):
            print(f"导弹{m+1}遮蔽区间：")
            intervals = get_cover_intervals(covered_all_missiles[m], t_list)
            for start, end in intervals:
                print(f"    {start:.2f}s ~ {end:.2f}s")
                 print(f"导弹{m+1}总遮蔽时间：{np.sum(covered_all_missiles[m]) * (t_list[1] -
t_list[0]):.2f}s")
    return covered_all_missiles  # shape (3, len(t_list))
```