赛区评阅编号（由赛区组委会填写）：

2025 高教社杯全国大学生数学建模竞赛

承　诺　书

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》（以下简称“竞赛章程和参赛规则”，可从 http://www.mcm.edu.cn 下载）。

我们完全清楚，在竞赛开始后参赛队员不能以任何方式，包括电话、电子邮件、“贴吧”、QQ 群、微信群等，与队外的任何人（包括指导教师）交流、讨论与赛题有关的问题；无论主动参与讨论还是被动接收讨论信息都是严重违反竞赛纪律的行为。

我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。

我们授权全国大学生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号（从 A/B/C/D/E 中选择一项填写）：A

我们的报名参赛队号（12 位数字全国统一编号）：202508013001

参赛学校（完整的学校全称，不含院系名）：哈尔滨商业大学

参赛队员 (打印并签名) ：1. 邹文杰

2. 李慧

3. 盛炫皓

指导教师或指导教师组负责人 (打印并签名)：郭丽华

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期: 2025 年 9 月 4 日

（请勿改动此页内容和格式。此承诺书打印签名后作为纸质论文的封面，注意电子版论文中不得出现此页。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）

赛区评阅编号：
（由赛区填写）_____

全国评阅编号：
（全国组委会填写）_____

2025 高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅记录（可供赛区评阅时使用）：

评阅人						
备注						

送全国评阅统一编号:
（赛区组委会填写）

（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）

摘要

针对问题一，

针对问题二，首先；然后；最后。

针对问题三，首先；然后；最后。

关键字：关键词 1　　关键词 2

一、 问题重述

1.1 问题背景

在当今国际形势下，国防安全至关重要。为应对现代战争中敌方导弹对我方重要目标的攻击，我方会投放烟幕干扰弹影响来袭导弹发现重要目标，从而提升我国的国防实力，对应对外部威胁和维护国家安全具有重要意义。烟幕干扰弹主要通过在重要目标前方特地区域形成对敌方导弹视线的遮挡从而进行干扰，具有成本低、效费比高等优点。如何通过数学建模通过对无人机的各项参数以及烟雾干扰弹的各项参数进行优化设计，从而实现更为有效的烟雾干扰效果成为了我们有待解决的问题。

1.2 问题一

在题目给定的直角坐标系下，将假目标放置在坐标系原点。根据题目给定的无人机 $FY1$ 的位置，飞行速度（120m/s），飞行方向, 敌方导弹 $M1$ 的位置, 受领任务后投放烟幕干扰弹间隔时间（1.5s）以及干扰弹起爆间隔时间 (3.6s), 无人机 FY1 投放 1 枚烟幕干扰弹实施对 M1 的干扰，计算烟幕干扰弹对 $M1$ 的有效遮挡时间。

1.3 问题二

已知在题目给定的直角坐标系下，无人机 $FY1$ 的位置以及敌方导弹 $M1$ 的位置已知，无人机 $FY1$ 投放 1 枚烟幕干扰弹进行干扰。给出无人机 $FY1$ 的飞行方向，无人机的飞行速度，烟幕干扰弹 $M1$ 的投放点, 烟幕干扰弹起爆点，使得烟幕干扰弹对来袭导弹的遮蔽时间尽可能长。

1.4 问题四

本问多加入 2 架无人机 $FY2$ 和 $FY3$，分别投放 1 枚烟幕干扰弹，实施对敌方导弹的干扰，因此需要重新设计烟幕干扰弹的投放策略。在题目给定的直角坐标系下，3 架无人机的位置以及敌方导弹 $M1$ 的位置已知, 重新给出 3 架无人机的飞行方向以及飞行速度、烟幕干扰弹投放点以及起爆点，从而延长对来袭导弹的遮蔽时间。

1.5 问题五

已知在题目给定的直角坐标系下,5 架无人机的位置以及 3 枚敌方导弹的位置, 每架无人机至多投放 3 枚烟幕干扰弹。• 重新给出 5 架无人机各自的飞行方向以及飞行速度、

3 枚烟幕干扰弹各自的投放点以及起爆点，从而找到对来袭导弹的遮蔽时间的最优解。

二、 问题的分析

2.1 问题一的分析

问题一要我们求解在给定条件下，烟雾干扰弹对导弹 $M1$ 的有效遮挡时长。因此我们需要知道烟雾干扰弹形成的云团球体在某时刻下是否对导弹进行遮挡，即导弹与假目标上任一点的连线是否与云团球体相交。首先，对于某时刻下导弹与真目标上任一点的连线方程，由导弹在某时刻下直角坐标系下的位置坐标与真目标所在圆柱侧面上任一点确定。我们已知导弹 $M1$ 和假目标在题目给定坐标系下的初始位置，以及导弹直指假目标的飞行方向和飞行速度，因此得到导弹 $M1$ 在某时刻下的坐标，通过题目给定条件，可以得到真目标所在的圆柱侧面各点的位置，由此可知某时刻下导弹与真目标上任一点的连线方程；其次，对于云团球体所在的球面方程，由云团中心和半径确定，因此需要知晓在某时刻下云团中心的位置坐标，我们已知无人机 $FY1$ 的初始位置，无人机匀速直线运动至假目标的速度，进而得到烟幕干扰弹投放时的位置，已知烟幕干扰弹在重力作用下运动，从而得到烟幕干扰弹爆炸时的位置，也就是云团中心的初始位置，已知云团匀速下沉的速度，可以得到在某时刻云团球心的位置，进而得到云团的球面方程；最后，将某时刻导弹 $M1$ 与真目标上任一点的连线方程与云团球体在某时刻下的球面方程进行联立，可以判断在某时刻下云团是否对导弹进行遮挡，从而得到烟幕干扰弹对导弹的有效遮挡时长。

2.2 问题二的分析

问题 2 要求我们对遮蔽时长进行优化，因此我们以 FY1 的飞行方向，飞行速度，烟幕干扰弹投放点，烟幕干扰弹起爆点为参数，无人机 FY1 释放的烟幕干扰弹对导弹 M1 的遮蔽时长最大为目标函数，基于上一问模型计算相关数值，再根据题意确立多个约束条件，建立单目标优化模型。

2.3 问题三的分析

对于问题三，与问题二相比，无人机 FY1 从释放一枚烟幕干扰弹到释放三枚烟幕干扰弹, 同样对遮蔽时长进行优化。因此我们以 FY1 的飞行方向，飞行速度，3 枚烟幕干扰弹投放点，3 枚烟幕干扰弹起爆点为参数，无人机 FY1 释放的 3 枚烟幕干扰弹对导弹 M1 的遮蔽时长最大为目标函数，基于第一一问模型计算相关数值，再根据题意确立多个约束条件，建立单目标优化模型。

2.4 问题四的分析

问题四是利用三架无人机和其投放的一枚烟幕干扰弹，求得其遮蔽时间最大值，因此需要对遮蔽时长进行优化。我们以三架无人机的飞行方向，飞行速度以及其分别投放的一枚烟幕干扰弹的投放点，起爆点为参数，烟幕干扰弹对导弹 M1 的遮蔽时长最大为目标函数，基于第一问模型计算相关数值，再根据题意确定多个约束条件，建立单目标优化模型。

2.5 问题五的分析

问题五是利用 5 架无人机以及其投放的相应枚烟幕干扰弹，实时对三枚来袭导弹的干扰，求得其遮蔽时长最大值，因此需要对遮蔽市场进行优化。我们以五架无人机的飞行方向，飞行速度以及其分别投放的相应枚烟幕干扰弹的投放点，起爆点和来袭导弹的类型为参数，以干扰弹对着目标的遮蔽时长最大为目标函数，基于第一问中的方法求解相关数值，再根据题意建定多个约束条件，建立单目标优化模型。

三、 模型的假设

1. 假设无人机的转向时间可忽略不计。
2. 假设导弹的重力影响可忽略不计。
3. 假设天气不会对导弹，无人机，烟雾弹造成影响。
4. 假设导弹为质点。

四、 符号说明

符号	说明
t	无人机受领任务后开始运动的时间
v_0	导弹的飞行速度
r_0	真目标所在圆柱的半径
h_0	真目标所在圆柱体的高
v_1	球状烟幕云团匀速下降速度
v_{FY1}	无人机 $FY1$ 的飞行速度
t_1	烟幕干扰弹投放时间
t_2	烟幕干扰弹起炸放时间
Mi,t	导弹 Mi 在 t 时刻的位置坐标
Δt_1	$FY1$ 的一枚烟幕干扰弹对 $M1$ 的有效遮挡时长
Δt_{13}	$FY1$ 的 3 枚烟幕干扰弹对 $M1$ 的有效遮挡时长

五、 模型的建立与求解

5.1 问题一

5.1.1 问题一的模型建立

Step 1 在 t 时刻下导弹 M_1 与真目标所在圆柱侧面各点的连线方程

随着烟幕干扰技术的不断发展，在导弹来袭过程中会通过投放烟幕干扰弹尽量避免来袭导弹发现真目标。控制中心在警戒雷达发现目标时，立即向无人机指派任务。设无人机 $M1$ 受领任务后开始沿相关轨迹进行运动的时间为 t，对于 t 时刻下导弹 $M1$ 与真目标所在圆柱侧面各点的连线方程，由导弹 $M1$ 在 t 时刻的位置坐标与真目标所在圆柱侧面各点确定。

(1) 导弹 $M1$ 在 t 时刻的位置坐标

本题以假目标为原点 O，水平面为 XY 平面建立直角坐标系 XYZ. 在该坐标系下记警戒雷达发现来袭导弹时，导弹 $M1$ 初始时刻的位置记为 $M1(0)$, 其坐标为

$$(x_{M1(0)}, y_{M1(0)}, z_{M1(0)}) = (20000, 0, 2000). \tag{1}$$

在警戒雷达发现来袭导弹后，导弹 $M1$ 以飞行速度 $v_0 = 300$m/s 匀速飞向一个为掩护真目标而设置的假目标。记 t 时刻下导弹 $M1$ 的位置坐标为 $M1(t)(x_{M1(t)}, y_{M1(t)}, z_{M1(t)})$。由于已经确定导弹 $M1$ 的初始位置、运动方向与运行速度，根据直角坐标系下直线的标

准方程，导弹在 t 时刻下的轨迹方程可表示为

$$\frac{x_{M1(0)}-x_{M1t}}{x_{M1(0)}}=\frac{y_{M1(0)}-y_{M1(t)}}{x_{M1(0)}}=\frac{z_{M1(0)}-z_{M1(t)}}{z_{M1(0)}} \tag{2}$$

其参数方程为：

$$\begin{cases} x_{M1t}=x_{M1(0)}-ax_{M1(0)} \\ y_{M1t}=y_{M1(0)}-ay_{M1(0)} \\ z_{M1t}=z_{M1(0)}-az_{M1(0)} \end{cases} ,a\in[0,+\infty) \tag{3}$$

导弹 $M1$ 从初始时刻到 t 时刻向假目标进行匀速直线运动，根据匀速直线运动的位移时间公式，其在 t 时刻位置与初始时刻位置之间的距离 S_1 满足

$$S_1=v_0t \tag{4}$$

同时，根据两点间距离公式，可以得到

$$S_1=\sqrt{(x_{M1(t)}-x_{M1(0)})^2+(y_{M1(t)}-y_{M1(0)})^2+(z_{M1(t)}-z_{M1(0)})^2} \tag{5}$$

联立(3) (4) (5)，进而得到导弹 $M1(t)$ 在 t 时刻下的位置坐标

$$\begin{cases} x_{M1(t)}=x_{M1(0)}-\dfrac{x_{M1(0)}v_0t}{\sqrt{x_{M1(0)}^2+y_{M1(0)}^2+z_{M1(0)}^2}} \\ y_{M1(t)}=y_{M1(0)}-\dfrac{y_{M1(0)}v_0t}{\sqrt{x_{M1(0)}^2+y_{M1(0)}^2+z_{M1(0)}^2}} \\ z_{M1(t)}=z_{M1(0)}-\dfrac{z_{M1(0)}v_0t}{\sqrt{x_{M1(0)}^2+y_{M1(0)}^2+z_{M1(0)}^2}} \end{cases} \tag{6}$$

(2) 真目标所在圆柱侧面各点的位置坐标

由于导弹 $M1$ 与真目标所在圆柱的上底面和下底面上各点的连线，一定与圆柱侧面相交。故而，当我们考虑导弹 $M1$ 与圆柱上任一点的连线方程时，只需要考虑其与圆柱侧面上任一点的连线方程。因此，我们需要确定真目标所在圆柱侧面各点的位置坐 $N_1\left(x_l,y_l,z_s\right)$。

在直角坐标系中，真目标是半径为 $r_0=7\text{m}$、高为 $h_0=10\text{m}$ 的圆柱体，其下底面的圆心坐标为 $(0,y_0,0)=(0,200,0)$。因此，真目标圆柱侧面上任一点位置坐标 $N_1\left(x_l,y_l,z_s\right)$ 满足：

$$\Gamma_0:\begin{cases} {x_l}^2+\left(y_l-y_0\right)^2=r_0^2 \\ z_s\in[0,h_0] \end{cases} \tag{7}$$

(3) 在 t 时刻下导弹 $M1$ 与真目标所在圆柱侧面各点的连线方程

通过(6)(7)，我们得到了导弹 $M1$ 在 t 时刻的位置坐标 $M1(t)(x_{M1(t)}, y_{M1(t)}, z_{M1(t)})$ 以及真目标所在圆柱侧面任一点位置坐标 $N_1\,(x_1, y_1, z_1)$. 因此，确定了 $N_1, M1(t)$ 所在的直线方程的参数方程

$$\begin{cases} x = x_1 + k\left(x_{M1(t)} - x_1\right) \\ y = y_1 + k\left(y_{M1(t)} - y_1\right) \\ z = z_1 + k\left(z_{M1(t)} - z_1\right) \end{cases} \quad , k \in [0, +\infty) \tag{8}$$

Step 2 烟幕干扰弹起爆后形成的云团球体在 t 时刻的球面方程

烟幕干扰弹经无人机投放，并且脱离无人机后在重力作用下运动一段时间后起爆形成球状烟幕云团，云团半径已知，该烟幕云团以速度 $v_1 = 3$m/s 匀速下沉. 因此，云团球体在 t 时刻的球面方程由云团球体中心确定，也就是由烟幕干扰弹起爆时的位置确定，于是，我们需要先求解烟幕干扰弹投放时的位置坐标。

(1) 烟幕干扰弹投放时的位置坐标

通过题目给定条件可知, 在警戒雷达发现来袭导弹后，无人机 $FY1$ 以飞行速度 v_{FY1} 匀速等高朝向假目标方向飞行, 并在受领任务 t_1 秒后投放 1 枚烟幕干扰弹在来袭武器和保护目标之间形成烟幕遮蔽。因此无人机 $FY1$ 在 t_1 时刻的位置坐标，即烟幕干扰弹投放时的位置坐标 $(x_{FY1,t_1}, y_{FY1,t_1}, z_{FY1,t_1})$ 满足

$$\begin{cases} x_{FY1,t_1} = x_{FY1,0} - v_{FY1}t_1 \\ y_{FY1,t_1} = y_{FY1,0} \\ z_{FY1,t_1} = z_{FY1,0} \end{cases} \tag{9}$$

其中 $v_{FY1} = 120$m/s，$t_1 = 1.5$s。

(2) 烟幕干扰弹起爆时的位置坐标

烟幕干扰弹脱离无人机后，在重力作用下进行匀加速运动。并在 $t_2 = 5.1$s 起爆分散形成烟幕或气溶胶云团, 进而在目标前方特定空域形成遮蔽，干扰敌方导弹 $M1$. 此时，烟幕干扰弹水平方向飞行速度即无人机飞行速度 v_{FY1}，烟幕干扰弹竖直方向加速度即重力加速度 g, 因此烟幕干扰弹起爆时的位置坐标 $(x_{FY11,t_2}, y_{FY11,t_2}, z_{FY11,t_2})$ 满足

$$\begin{cases} x_{FY11,t_2} = x_{FY1,t_1} - v_{FY1}(t_2 - t_1) \\ y_{FY11,t_2} = y_{FY1,t_1} \\ z_{FY11,t_2} = z_{FY1,t_1} - \dfrac{g(t_2 - t_1)^2}{2} \end{cases} \tag{10}$$

(3) 云团球体在 t 时刻的球面方程

烟幕干扰弹起爆后瞬时形成球状烟幕云团，由于采用特定技术，该烟幕云团以速度 $v_1 = 3\text{m}$ 匀速下沉。云团中心 $r = 10\text{m}$ 范围内的烟幕浓度在起爆后一定时间内可为目标提供有效遮蔽。已知烟幕干扰弹起爆后瞬时形成的球状云团中心位置即烟幕干扰弹起爆时的位置坐标 $(x_{FY11,t_2}, y_{FY11,t_2}, z_{FY11,t_2})$，并且球状烟幕云团以速度 v_1 匀速下沉. 因此，球状烟幕云团球体中心在 t 时刻的位置坐标 $(x_{FY1,t}, y_{FY1,t}, z_{FY1,t})$ 满足

$$\begin{cases} x_{FY11,t} = x_{FY11,t_2} \\ y_{FY11,t} = y_{FY11,t_2} \\ z_{FY11,t} = z_{FY11,t_2} - v_1 (t - t_2) \end{cases} \tag{11}$$

从而可得云团球体在 t 时刻的球面方程：

$$O_{FY11,t} : (x - x_{FY11,t})^2 + (y - y_{FY11,t})^2 + (z - z_{FY11,t})^2 = r^2 \tag{12}$$

Step 3 判断 t 时刻烟幕云团是否有效遮蔽真目标

通过(8)(12)，我们得到了在 t 时刻下导弹 $M1$ 与真目标所在圆柱侧面各点的连线方程以及烟幕干扰弹起爆后形成的云团球体的球面方程。当导弹 $M1$ 与真目标所在圆柱侧面各点的连线方程与烟幕云团球面方程在 t 时刻下相交，则烟幕云团有可能遮挡真目标；反之，则烟幕云团不可能遮挡真目标。则在相交情况下，直线方程与烟幕云团球体表面的交点到真目标所在圆柱侧面上的点的距离如果都小于或等于此时导弹 $M1$ 到圆柱侧面上的点的距离，则在此时刻下形成有效遮挡；否则，在此时刻下未形成有效遮挡。

联立(7)(8)(12), 得到了 t 时刻下导弹 $M1$ 与真目标所在圆柱侧面各点的连线与云团球体的球面是否相交的判别式：

$$\Delta(x_l, y_l, z_s) = b^2 - 4ac, \tag{13}$$

其中

$$\begin{aligned} a &= (x_{M1(t)} - x_1)^2 + (y_{M1(t)} - y_1)^2 + (z_{M1(t)} - z_1)^2, \\ b &= 2(x_{M1(t)} - x_1)(x_1 - x_{FY11,t}) + 2(y_{M1(t)} - y_1)(y_1 - y_{FY11,t}) \\ &\quad + 2(z_{M1(t)} - z_1)(z_1 - z_{FY11,t}), \\ c &= (x_1 - x_{FY11,t})^2 + (y_1 - y_{FY11,t})^2 + (z_1 - z_{FY11,t})^2 - r^2 \end{aligned} \tag{14}$$

在 t 时刻下，只有当导弹 $M1$ 与真目标所在圆柱所有点的连线方程与云团球体的球面相交，才有可能进行遮挡。如图1所示

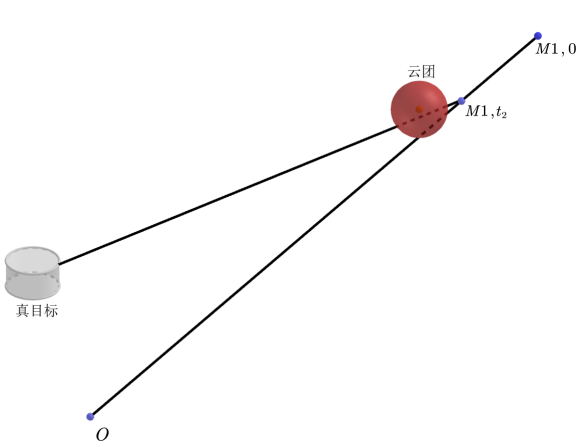

图 **1**

由于导弹 $M1$ 以速度 v_0 匀速飞行至假目标，且已知导弹 $M1$ 的初始位置 $M1(0)$ 和假目标所在位置即原点，因此可以得到时间 t 的范围

$$\begin{cases} d_{M1(0)} = \sqrt{(x_{M1(0)})^2 + (z_{M1(0)})^2} \\ 0 \leq t \leq \dfrac{d_{M1(0)}}{v_0} \end{cases} \tag{15}$$

将时间 t 进行离散化处理，从而方便计算。将时间 t 按步长 Δr 划分，得到 J 个离散时刻，使得

$$\frac{d_{M1(0)}}{v_0} = J\Delta r \tag{16}$$

其中第 w 个离散时刻记为 t_w, 计算公式如下

$$t_w = \frac{t}{J}w \quad (w = 1, \cdots, J) \tag{17}$$

为方便计算，我们将真目标所在圆柱体侧面进行离散化处理。通过题目给定条件，可以得知真目标是半径为 r_0、高为 h_0 的圆柱体，其下底面的圆心坐标为 $(0, y_0, 0)$。我们将真目标所在圆柱体的底面圆周作 n_1 等分，每一部分的圆心角 $\Delta\varphi = 2\pi/n_1$。接着将圆柱的高度 h_0 沿 z 轴进行 n_2 等分，每段的高度 $\Delta h = h_0/n_2$. 进而得到 $n_1 \times n_2$ 个离散点。记第 l 份圆心角为 φ_l, 计算公式如下：

$$\varphi_l = \frac{\varphi}{n_1}l \quad (l = 1, \cdots, n_1) \tag{18}$$

记圆柱体的第 s 份高度为 h_s, 计算公式如下：

$$h_s = \frac{h_0}{n_2}s \quad (s = 1, \cdots, n_2) \tag{19}$$

圆柱体侧面上的这些离散点位置坐标 $N_1\left(x_l, y_l, z_s\right)$ 可表示为

$$\begin{cases} x_l = r_0 \cos\left(\dfrac{2\pi}{n_1} \cdot l\right) \\ y_l = y_0 + r_0 \sin\left(\dfrac{2\pi}{n_1} \cdot l\right) \\ z_s = \dfrac{h_0}{n_2} \cdot s \end{cases} \tag{20}$$

其中 $l = 1, \cdots, n_1$，$s = 1, \cdots, n_2$。将所有点位置坐标代入判别式中，即遍历 l,s，可以得知在 t_w 时刻下导弹 $M1$ 与真目标所在圆柱所有点的连线方程与云团球体的球面是否相交。但是通过此判断式不能确定烟幕云团是否有效遮挡真目标，因此我们还需要加入其他条件。当 $\Delta\left(x_l, y_l, z_s\right) \geq 0$ 时有交点，交点参数表达式如下

$$k_1 = \frac{-b + \sqrt{\Delta}}{2a}, \quad k_2 = \frac{-b - \sqrt{\Delta}}{2a} \tag{21}$$

注意到导弹 $M1$ 与真目标所在圆柱所有点的连线方程与云团球体的球面交点到圆柱侧面上的点的距离计算公式为

$$\begin{aligned} d_1 &= \sqrt{k_1^2(x_{M1(t)} - x_l)^2 + k_1^2(y_{M1(t)} - y_l)^2 + k_1^2(z_{M1(t)} - z_s)^2}, \\ d_2 &= \sqrt{k_2^2(x_{M1(t)} - x_l)^2 + k_2^2(y_{M1(t)} - y_l)^2 + k_2^2(z_{M1(t)} - z_s)^2}, \end{aligned} \tag{22}$$

并且导弹 $M1$ 到圆柱侧面上的点的的距离：

$$\left|\overrightarrow{N1M1}\right| = \sqrt{\left(x_{M1(t)} - x_{n1}\right)^2 + \left(y_{M1(t)} - y_{n1}\right)^2 + \left(y_{M1(t)} - y_{n2}\right)^2} \tag{23}$$

只有在 t 时刻下，对于真目标所在圆柱侧面的所有离散点 (x_1, y_1, z_1) 均满足 $\Delta \geq 0$，且导弹 M_1 与真目标所在圆柱所有点的连线方程与云团球体的球面交点到原点的距离 d_1, d_2 中至少有一个比导弹到原点的距离更近（即 $d_1 \leq \left|\overrightarrow{N1M1}\right|$ 或 $d_2 \leq \left|\overrightarrow{N1M1}\right|$）时，才能形成有效遮挡；否则，未形成有效遮挡。即：

$$\begin{cases} \Delta < 0 \quad \text{未形成有效遮挡} \\ \Delta \geq 0 \begin{cases} \min\{d_1, d_2\} > \left|\overrightarrow{N1M1}\right| & \text{未形成有效遮挡} \\ \min\{d_1, d_2\} \leq \left|\overrightarrow{N1M1}\right| & \text{有效遮挡} \end{cases} \end{cases} \tag{24}$$

5.1.2 模型求解

烟幕干扰弹对导弹 $M1$ 的有效遮挡时长计算步骤如下.

步骤 1: 固定时间 t_w, 通过遍历 l,s, 得到 t_w 时刻下导弹 $M1$ 与真目标所在圆柱所有点的连线方程，代入(13)式中, 得到判别式 $\Delta\left(x_l, y_l, z_s\right)$。

步骤 2: 如果判别式 $\Delta(x_l, y_l, z_s)$ 满足(25)即

$$\begin{cases} \Delta < 0 \quad \text{未形成有效遮挡} \\ \Delta \geq 0 \begin{cases} \min\{d_1, d_2\} > \left|\overrightarrow{N1M1}\right| & \text{未形成有效遮挡} \\ \min\{d_1, d_2\} \leq \left|\overrightarrow{N1M1}\right| & \text{有效遮挡} \end{cases} \end{cases} \tag{25}$$

则可以判断 t_w 时刻下，烟幕云团是否有效遮蔽真目标。

步骤 3: 遍历 w, 可以得知不同时间点烟幕云团是否有效遮蔽真目标。若遮挡真目标，则进行步骤 4。反之，则遍历下一个 w。

步骤 4: 由于只有一个烟幕干扰弹干扰导弹，因此烟幕干扰弹对导弹 $M1$ 的有效遮挡时间是连续的。从而将步骤 3 中得到的时间点的最大值减去最小值即可得到烟幕干扰弹对导弹 $M1$ 的有效遮挡时长, 即

$$\Delta t = \max t_w - \min t_w \tag{26}$$

利用 Python 遍历求解得**烟幕干扰弹对 $M1$ 的有效遮蔽时间为 1.391982s.**

5.2 问题二

5.2.1 模型建立

Step 1 目标函数

在问题一中，我们给出了无人机 $FY1$ 投放的烟幕干扰弹对 M1 的有效遮蔽时长的 Δt_{11} 的计算公式。但在本问中，我们的目标是寻找无人机 $FY1$ 与其投放的 1 枚烟幕干扰弹相关参数，使得烟幕干扰弹对导弹 $M1$ 的有效遮蔽时间尽可能长, 因此目标函数为:

$$\max_{\alpha, t_1, v_{FY1}, t_2} \Delta t_{11} \tag{27}$$

Step 2 决策变量

- **无人机 $FY1$ 的方向**: 设 α 为无人机 $FY1$ 与 x 轴正方向的夹角，逆时针为正，范围为 $[0, 2\pi]$。
- **烟幕干扰弹投放点**: 设无人机 $FY1$ 在受领任务 t_1 s 后投放 1 枚烟幕干扰弹。
- **无人机 $FY1$ 的飞行速度**: 设无人机的飞行速度为 v_{FY1}，根据问题一求得的(9)可得到

t 时刻无人机 $FY1$ 的位置坐标：

$$\begin{cases} x_{FY1,t} = x_{FY1,0} + v_{FY1} t \cos\alpha \\ y_{FY1,t} = y_{FY1,0} + v_{FY1} t \sin\alpha \\ z_{FY1,t} = z_{FY1,0} \end{cases} \tag{28}$$

- **烟幕干扰弹起爆点**: 设无人机 FY1 投放的烟幕干扰弹在无人机受领任务 t_2 s 后起爆，因此由问题一求得的(10)可得烟幕干扰弹起爆时的位置坐标：

$$\begin{cases} x_{FY11,t_2} = x_{FY1,t_1} - v_{FY1}\left(t_2 - t_1\right) \\ y_{FY11,t_2} = y_{FY1,t_1} \\ z_{FY11,t_2} = z_{FY1,t_1} - \dfrac{g\left(t_2 - t_1\right)^2}{2} \end{cases} \tag{29}$$

Step 3 约束条件

- **无人机的飞行速度**：由于无人机受领任务后，可根据需要瞬时调整飞行方向，然后以 70-140m/s 的速度等高度匀速直线飞行。因此：

$$70 \le v_{FY1} \le 140 \tag{30}$$

Step 4 优化模型

综上所述，烟幕干扰弹对导弹 $M1$ 的有效遮蔽时间单目标优化模型为：

$$\max_{\alpha, t_1, v_{FY1}, t_2} \Delta t_{11}$$

$$\begin{cases} t\text{时刻无人机的位置坐标:} \\ \quad \begin{cases} x_{FY1,t} = x_{FY1,0} + v_{FY1} t \cos\alpha \\ y_{FY1,t} = y_{FY1,0} + v_{FY1} t \sin\alpha \\ z_{FY1,t} = z_{FY1,0} \\ 0 \le \alpha \le 2\pi \end{cases} \\ \text{烟幕导弹起爆时的位置坐标:} \\ \quad \begin{cases} x_{FY11,t_2} = x_{FY1,t_1} - v_{FY1}\left(t_2 - t_1\right)\cos\lambda \\ y_{FY11,t_2} = y_{FY1,t_1} - v_{FY1}\left(t_2 - t_1\right)\sin\lambda \\ z_{FY11,t_2} = z_{FY1,t_1} - \dfrac{g\left(t_2 - t_1\right)^2}{2} \end{cases} \\ 70 \le v_{FY1} \le 140 \end{cases} \tag{31}$$

5.2.2 模型求解

我们利用模拟退火算法求解上述单目标优化模型。核心思路是：通过模拟物理退火过程的随机搜索与概率接受机制，在决策变量的可行域内寻找使有效遮蔽时间 Δt 最大化的最优解。具体步骤如下：

步骤 1 设定初始参数和离散化时间轴

根据题目要求和经验，设定干扰弹发射角度、速度及投放时间等初始参数。将导弹飞行时间段划分为细小的时间步长，便于逐点判断遮蔽状态。

步骤 2 遍历时间点并判断遮蔽

对每个时间点，计算干扰弹位置，结合导弹与目标几何关系，根据模型分析中的判断条件，判断各时间点是否满足有效遮蔽条件。收集所有满足遮蔽条件的时间点，累加得到总有效遮蔽时间。

步骤 3 应用模拟退火算法

以总遮蔽时间作为目标函数值，供优化算法调用。通过迭代生成新参数解，依据目标函数值变化和 Metropolis 准则接受或拒绝新解。

Metropolis 准则：以一定的概率接受一个新状态，即使这个新状态的能量（或目标函数值）比当前状态更低。这有助于算法跳出局部最优解，探索更广阔的状态空间。本代码中选择的比较值是 $\exp\left(\dfrac{\Delta S}{T}\right)$，其中 ΔS 为新旧有效时间之差，T 为该次循环的温度。当新解更大时，选用新解；新解更小时，以 $\exp\left(\dfrac{\Delta S}{T}\right)$ 的概率选用新解。

步骤 4 逐步优化并收敛

随着迭代进行，不断更新最优参数组合，使有效遮蔽时间逐渐增加，最终趋于稳定最优值。

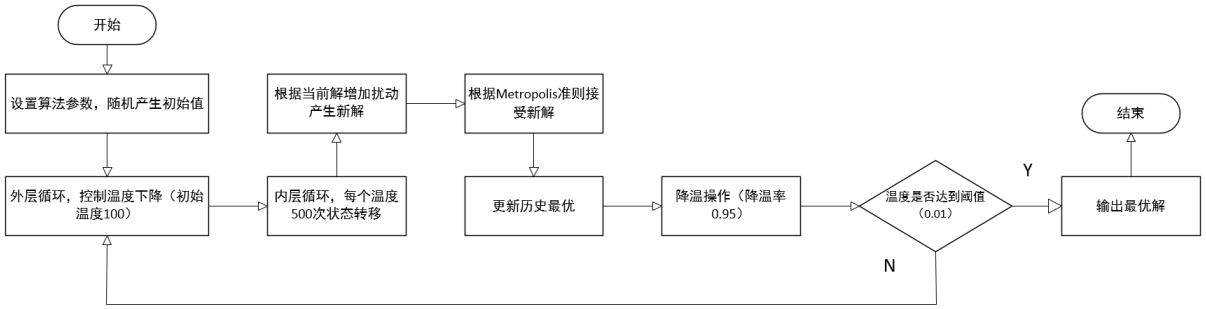

图 **2** 问题二模型求解流程图

按照上述算法思路，利用 Python 求解得当 $FY1$ 的飞行方向与 x 轴正方向的夹角 α 为 3.13rad(逆时针为正)，飞行速度为 109.78m/s，烟雾弹投放位置为

$$(17722.061437, 0.903555, 1800.000000), \tag{32}$$

烟雾弹起爆位置为

$$(17335.661803, 5.383153, 1739.287040) \tag{33}$$

时，**遮蔽时间最长为 4.602140 秒**。

5.3 问题三

5.3.1 模型建立

Step 1 目标函数

在本问中，我们的目标是通过设计无人机 $FY1$ 与其投放的 3 枚烟幕干扰弹相关参数，使得 3 枚烟幕干扰弹对导弹 $M1$ 的有效遮蔽总时间尽可能长, 因此目标函数为：

$$\max_{\alpha, t_1, v_{FY1}, t_2, i} \Delta t_{13} \tag{34}$$

Step 2 决策变量

- **无人机 $FY1$ 的方向**: 设 α 为无人机 $FY1$ 与 x 轴正方向的夹角，逆时针为正，范围为 $[0, 2\pi]$，从而确定了无人机 $FY1$ 的飞行方向。
- **烟幕干扰弹投放点**: 设无人机 $FY1$ 在受领任务 $t_{FY1,i1}$ s 后投放第 i 枚烟幕干扰弹 $(i = 1, 2, 3)$。
- **无人机 $FY1$ 的飞行速度**: 设无人机的飞行速度为 v_{FY1}，无人机受领任务后，保持等高度匀速直线运动。由问题一求得的(9)可知 t 时刻无人机 $FY1$ 的位置坐标为：

$$\begin{cases} x_{FY1,t} = x_{FY1,0} + v_{FY1} t \cos\alpha \\ y_{FY1,t} = y_{FY1,0} + v_{FY1} t \sin\alpha \\ z_{FY1,t} = z_{FY1,0} \end{cases} \tag{35}$$

- **烟幕干扰弹起爆点**: 设无人机 FY1 投放的第 i 枚烟幕干扰弹在无人机受领任务 $t_{FY1,i2}$ s 后起爆，根据问题 1 中的(10)得到投放的第 i 枚烟雾干扰弹在 $t_{FY1,i2}$ 时刻即其起爆时的位置坐标：

$$\begin{cases} x_{FY1i,t_{FY1,i2}} = x_{FY1,t_{FY1,i1}} + v_{FY1}\left(t_{FY1,i2} - t_{FY1,i1}\right)\cos\alpha \\ y_{FY1i,t_{FY1,i2}} = y_{FY1,t_{FY1,i1}} + v_{FY1}\left(t_{FY1,i2} - t_{FY1,i1}\right)\sin\alpha \\ z_{FY1i,t_{FY1,i2}} = z_{FY1,t_{FY1,i1}} - \dfrac{g\left(t_{FY1,i2} - t_{FY1,i1}\right)^2}{2} \end{cases} \tag{36}$$

通过问题 1 中的(13)，得到 t 时刻无人机 $FY1$ 投放的第 i 枚烟幕干扰弹形成的烟幕

云团是否对目标进行遮挡的判断条件 $\Delta_{FY1i}(x_l, y_l, z_s)$，代入(21)(22), 得到：

$$\begin{aligned} d_{1i} &= \sqrt{k_{1i}^2(x_{M1(t)} - x_l)^2 + k_{1i}^2(y_{M1(t)} - y_l)^2 + k_{1i}^2(z_{M1(t)} - z_s)^2}, \\ d_{2i} &= \sqrt{k_{2i}^2(x_{M1(t)} - x_l)^2 + k_{2i}^2(y_{M1(t)} - y_1)^2 + k_{2i}^2(z_{M1(t)} - z_s)^2}, \end{aligned} \tag{37}$$

并将其代入(25)中即

$$\begin{cases} \Delta < 0 \quad \text{未形成有效遮挡} \\ \Delta \geq 0 \begin{cases} \min\{d_{1i}, d_{2i}\} > \left|\overrightarrow{N1M1}\right| & \text{未形成有效遮挡} \\ \min\{d_{1i}, d_{2i}\} \leq \left|\overrightarrow{N1M1}\right| & \text{有效遮挡} \end{cases} \end{cases} \tag{38}$$

判断 t 时刻下无人机 FY1 释放的第 i 枚烟幕干扰弹形成的烟幕云团是否对真目标进行遮挡。又因为每个烟幕干扰弹形成的云团都将在 $\Delta t_0 = 20$ 秒后消散, 所以规定：

$$\Delta_{FY1i}(x_l, y_l, z_s) = \begin{cases} -1, \ t \geq t_{FY1,i2} + \Delta t_0 \\ -1, \ t \geq t_{FY1,i2} + \Delta t_0 \end{cases} \tag{39}$$

Step 3 约束条件

- **无人机的飞行速度**：由于无人机受领任务后，可根据需要瞬时调整飞行方向，然后以 70-140m/s 的速度等高度匀速直线飞行。因此：

$$70 \leq v_{FY1} \leq 140 \tag{40}$$

- **无人机投放的烟幕干扰弹的时间**：由于题目要求每架无人机投放两枚烟幕干扰弹至少间隔 1s，且据试验数据知，云团中心 10m 范围内的烟幕浓度在起爆 20s 内可为目标提供有效遮蔽。因此：

$$\begin{cases} t_{FY1,11} \in \left[0, \dfrac{x_{m1,0}}{v_0}\right] \\ t_{FY1,12} \in \left[t_{FY1,11}, \dfrac{d_{m1,0}}{v_0}\right] \\ t_{FY1,21} \in \left[t_{FY1,11} + 1, \dfrac{dx_{m1,0}}{v_0}\right] \\ t_{FY1,22} \in \left[t_{FY1,21}, \dfrac{d_{m1,0}}{v_0}\right] \\ t_{FY1,31} \in \left[t_{FY1,21} + 1, \dfrac{d_{m1,0}}{v_0}\right] \\ t_{FY1,32} \in \left[t_{FY1,21}, \dfrac{d_{m1,0}}{v_0}\right] \end{cases} \tag{41}$$

Step 4 优化模型

综上所述，无人机 $FY1$ 释放的 3 枚烟幕干扰弹有效遮蔽时间单目标优化模型为：

$$
\max_{\alpha,t_1,v_{FY1},t_2,i} \Delta t_{13}
$$

$$
\begin{cases}
t\text{时刻无人机的位置坐标:} \\
\begin{cases}
x_{FY1,t} = x_{FY1,0} + v_{FY1} t \cos\alpha \\
y_{FY1,t} = y_{FY1,0} + v_{FY1} t \sin\alpha \\
z_{FY1,t} = z_{FY1,0} \\
0 \le \alpha < 2\pi
\end{cases} \\
70 \le v_{FY1} \le 140 \\
\text{第 } i \text{ 枚烟雾干扰弹起爆时的位置坐标:} \\
\begin{cases}
x_{FY1i,t_{FY1,i2}} = x_{FY1,t_{FY1,i1}} + v_{FY1}\left(t_{FY1,i2} - t_{FY1,i1}\right)\cos\alpha \\
y_{FY1i,t_{FY1,i2}} = y_{FY1,t_{FY1,i1}} + v_{FY1}\left(t_{FY1,i2} - t_{FY1,i1}\right)\sin\alpha \\
z_{FY1i,t_{FY1,i2}} = z_{FY1,t_{FY1,i1}} - \dfrac{g\left(t_{FY1,i2} - t_{FY1,i1}\right)^2}{2}
\end{cases} \\
\begin{cases}
t_{FY1,11} \in \left[0, \dfrac{x_{m1,0}}{v_0}\right] \\
t_{FY1,12} \in \left[t_{FY1,11}, \dfrac{d_{m1,0}}{v_0}\right] \\
t_{FY1,21} \in \left[t_{FY1,11} + 1, \dfrac{dx_{m1,0}}{v_0}\right] \\
t_{FY1,22} \in \left[t_{FY1,21}, \dfrac{d_{m1,0}}{v_0}\right] \\
t_{FY1,31} \in \left[t_{FY1,21} + 1, \dfrac{d_{m1,0}}{v_0}\right] \\
t_{FY1,32} \in \left[t_{FY1,21}, \dfrac{d_{m1,0}}{v_0}\right]
\end{cases}
\end{cases}
\tag{42}
$$

5.3.2 模型求解

类似问题 2 使用模拟退火算法对问题 3 的单目标优化模型进行求解。具体步骤如下：

步骤 1　初始化参数和约束条件

设定导弹、干扰弹的初始位置与运动参数，构建三维空间中的动态几何关系。根据模型分析对发射参数设定合理边界。

步骤 2　定义遮蔽判定函数和离散化时间轴

通过模型分析中的几何条件判断干扰弹是否在某一时刻对导弹实现有效遮蔽，据此定义核心判别函数。将导弹飞行时间段划分为细小的时间步长，便于逐点判断遮蔽状态。

步骤 3　模拟退火搜索最优参数

根据发射角度、速度和时间参数，分阶段计算干扰弹的运动轨迹与云团的沉降轨迹，

并更新其随时间变化的位置。再利用判断函数判断各时刻是否有效，并统计出所有满足遮蔽条件的时间点，计算总持续时间作为目标函数。采用与问题 2 类似的自适应模拟退火算法，通过随机扰动、接受准则和温度退火机制，全局搜索最优参数组合。当优化停滞时，从历史搜索的优质解中重启搜索，增强跳出局部极值的能力。

步骤 4　输出最优遮蔽方案

返回最大遮蔽时间及对应的干扰弹投放参数，完成优化求解。

按照上述步骤，利用 Python 求解得到结果，见表1和表2.

表 **1**　问题 **3** 无人机投放策略**-1**

无人机运动方向	无人机运动速度	烟幕干扰弹编号	烟幕干扰弹投放点的 x 坐标	烟幕干扰弹投放点的 y 坐标
179.66	140.00	1	17653.00	0.88
179.66	140.00	2	17294.61	3.03
179.66	140.00	3	17112.61	4.12

表 **2**　问题 **3** 无人机投放策略**-2**

干扰弹投放点的 z 坐标	干扰弹起爆点的 x 坐标	干扰弹起爆点的 y 坐标	干扰弹起爆点的 z 坐标	有效干扰时长
1800.00	17053.81	4.47	1710.23	4.1
1800.00	16554.02	7.47	1662.88	2.7
1800.00	16290.83	9.04	1631.16	1.7

在表2条件下，某一有效时刻导弹和烟幕云团的遮挡示意图, 如图3所示.

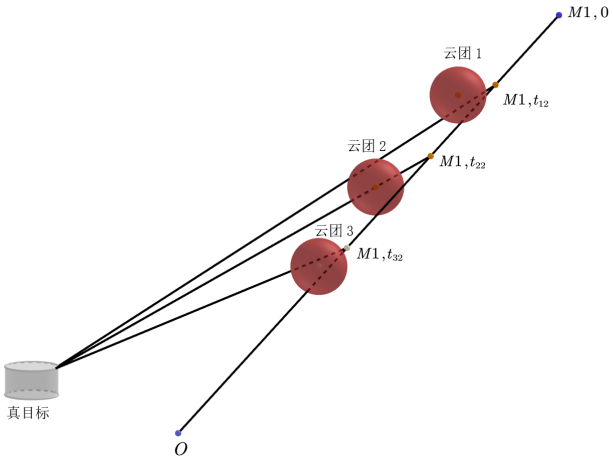

图 **3**

5.4 问题四

5.4.1 模型建立

Step 1 目标函数

在本问中，我们的目标是通过设计 3 架无人机 $FY1$，$FY2$，$FY3$ 与其分别投放的 1 枚烟幕干扰弹相关参数，使得这 3 架无人机分别释放的 3 枚烟幕干扰弹对导弹 $M1$ 的有效遮蔽总时间尽可能长，因此目标函数为：

$$\max_{\alpha_j, v_{FYj}, t_{FYj,j1}, t_{FYj,j2}} \Delta t_{31} \tag{43}$$

Step 2 决策变量

- **无人机 FYj 的方向**: 设 α_j 为无人机 FYj 与 x 轴正方向的夹角，逆时针为正，范围为 $[0, 2\pi)$，从而确定无人机 FYj 的飞行方向 $(j = 1, 2, 3)$。
- **无人机 FYj 的飞行速度**: 设无人机 FYj 的飞行速度为 v_{FYj}。
- **烟幕干扰弹投放点**: 设无人机 FYj 在受领任务 $t_{FYj,11}$ s 后投放 1 枚烟幕干扰弹。无人机受领任务后，保持等高度匀速直线运动。由问题一求得的(9)可知 t 时刻无人机 FYj 的位置坐标为：

$$\begin{cases} x_{FYj,t} = x_{FYj,0} + v_{FYj} t \cos\alpha \\ y_{FYj,t} = y_{FYj,0} + v_{FYj} t \sin\alpha \\ z_{FYi,t} = z_{FYj,0} \end{cases} \tag{44}$$

从而可得 $t_{FYj,11}$ 时刻无人机 FYj 的位置坐标, 即烟幕干扰弹投放点的位置坐标

$$\begin{cases} x_{FYj,t_{FYj,11}} = x_{FYj,0} + v_{FYj} t_{FYj,11} \cos\alpha \\ y_{FYj,t_{FYj,11}} = y_{FYj,0} + v_{FYj} t_{FYj,11} \sin\alpha \\ z_{FYi,t_{FYj,11}} = z_{FYj,0} \end{cases} \tag{45}$$

- **烟幕干扰弹起爆点**: 设无人机 FYj 投放一枚烟幕干扰弹在无人机受领任务 $t_{FYj,12}$ s 后起爆，由问题一求得的(10)可得 3 架无人机投放的烟雾干扰弹在 $t_{FYj,12}$ 时刻即其起爆时的位置坐标：

$$\begin{cases} x_{FYj,t_{FYj,12}} = x_{FYj,t_{FYj,11}} + v_{FYj}\left(t_{FYj,12} - t_{FYj,11}\right)\cos\alpha \\ y_{FYj,t_{FYj,12}} = y_{FYj,t_{FYj,11}} + v_{FYj}\left(t_{FYj,12} - t_{FYj,11}\right)\sin\alpha \\ z_{FYj,t_{FYj,12}} = z_{FYj,t_{FYj,11}} - \dfrac{g\left(t_{FYj,12} - t_{FYj,11}\right)^2}{2} \end{cases} \tag{46}$$

通过问题 1 中的(13)，得到 t 时刻无人机 FYj 投放的烟幕干扰弹形成的烟幕云团是否对目标进行遮挡的判断条件 $\Delta_{FYj1}\left(x_l, y_l, z_s\right)$。代入(21)(22), 得到：

$$\begin{aligned} d_{1j} &= \sqrt{k_{1j}^2(x_{M1(t)} - x_l)^2 + k_{1j}^2(y_{M1(t)} - y_l)^2 + k_{1j}^2(z_{M1(t)} - z_s)^2}, \\ d_{2j} &= \sqrt{k_{2j}^2(x_{M1(t)} - x_1)^2 + k_{2j}^2(y_{M1(t)} - y_l)^2 + k_{2j}^2(z_{M1(t)} - z_s)^2}, \end{aligned} \tag{47}$$

并将其代入(25)中即

$$\begin{cases} \Delta < 0 \quad \text{未形成有效遮挡} \\ \Delta \geq 0 \begin{cases} \min\left\{d_{1j}, d_{2j}\right\} > \left|\overrightarrow{N1M1}\right| & \text{未形成有效遮挡} \\ \min\left\{d_{1j}, d_{2j}\right\} \leq \left|\overrightarrow{N1M1}\right| & \text{有效遮挡} \end{cases} \end{cases} \tag{48}$$

判断 t_w 时刻下无人机 FYj 释放的烟幕干扰弹形成的烟幕云团是否对真目标进行遮挡。又因为每个烟幕弹形成的云团都将在 20s 后消散，所以规定：

$$\Delta_{FYj1}\left(x_l, y_l, z_s\right) = \begin{cases} -1, \ t \geq t_{FY1,12} + \Delta t_0 \\ -1, \ t \geq t_{FY2,12} + \Delta t_0 \end{cases} \tag{49}$$

最后，同理遍历真目标圆柱上的所有离散点 (x_l, y_l, z_s) 和离散时刻 t_w，将所有对真目标构成有效遮挡的时刻累加起来就得到该条件下的 Δt_{31}.

Step 3 约束条件

- **无人机的飞行速度**：由于无人机受领任务后，可根据需要瞬时调整飞行方向，然后以 70-140m/s 的速度等高度匀速直线飞行。因此：

$$70 \leq v_{FY1} \leq 140 \tag{50}$$

Step 4 优化模型

综上所述，3 架无人机 FYj 释放的烟幕遮挡弹的有效遮蔽总时间单目标优化模型

为：

$$\max_{\alpha_j, v_{FYj}, t_{FYj,j1}, t_{FYj,j2}} \Delta t_{31}$$

$$\begin{cases} t\text{时刻无人机的位置坐标:} \\ \begin{cases} x_{FYj,t} = x_{FYj,0} + v_{FYj} t \cos\alpha \\ y_{FYj,t} = y_{FYj,0} + v_{FYj} t \sin\alpha \\ z_{FYi,t} = z_{FYj,0} \\ 0 \le \alpha < 2\pi \end{cases} \\ 70 \le v_{FY1} \le 140 \\ \text{无人机} FYj \text{投放的烟雾干扰弹起爆时的位置坐标:} \\ \begin{cases} x_{FYj,t_{FYj,12}} = x_{FYj,t_{FYj,11}} + v_{FYj}\left(t_{FYj,12} - t_{FYj,11}\right)\cos\alpha \\ y_{FYj,t_{FYj,12}} = y_{FYj,t_{FYj,11}} + v_{FYj}\left(t_{FYj,12} - t_{FYj,11}\right)\sin\alpha \\ z_{FYj,t_{FYj,12}} = z_{FYj,t_{FYj,11}} - \dfrac{g\left(t_{FYj,12} - t_{FYj,11}\right)^2}{2} \end{cases} \end{cases} \tag{51}$$

5.4.2 模型求解

步骤 1：初始化系统参数与几何约束

设定导弹、无人机、防御目标及烟幕云团的物理参数，构建三维空间中的动态运动模型与几何关系，并通过经验与物理限制为优化变量设定合理边界。

步骤 2：构建多无人机协同遮蔽动态仿真模型

建立三维空间中导弹与目标之间的视线向量模型，结合无人机投放烟幕的空间分布，采用时间离散化对整个过程进行精细化推进，每一时刻判断视线段是否与至少一个有效烟幕区域发生空间重叠。根据当前优化参数，在仿真时序中同步更新三架无人机的烟幕释放状态及其空间覆盖范围。逐时刻调用几何遮挡判定函数，判断是否实现对导弹视线的完全遮蔽。

步骤 3：构建优化目标函数并调用差分进化算法

利用差分进化算法自动完成初始化种群、变异、交叉、选择、迭代进化等步骤，利用差分进化算法全局搜索最优参数组合。

步骤 4：输出最优遮蔽方案与部署参数

解析优化结果，输出可执行的无人机投放策略和预期效果。

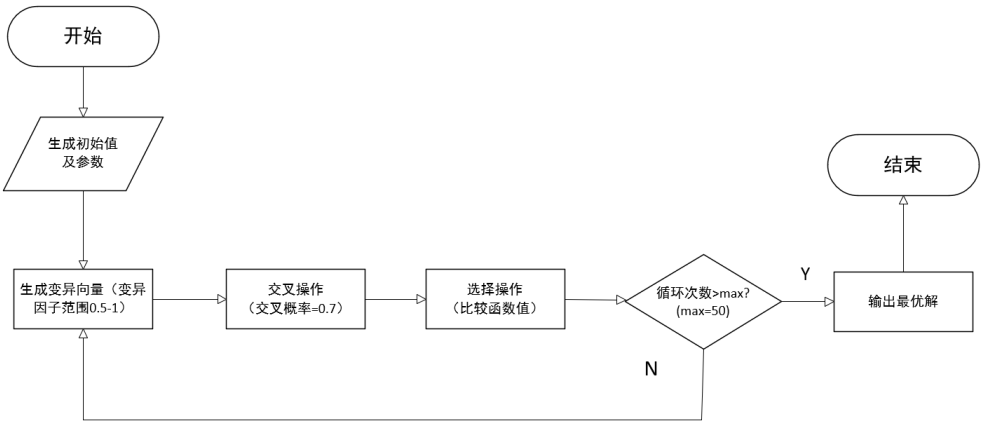

图 **4**　问题四模型求解流程图

按照上述算法思路, 利用 Python 求解得到结果，见表3和表4 .

表 **3**　问题 **4** 投放策略**-1**

无人机编号	无人机运动方向	无人机运动速度	烟幕干扰弹投放点的 x 坐标	烟幕干扰弹投放点的 y 坐标
FY1	4.86	96.37	17855.44	4.72
FY2	246.86	87.25	11643.20	565.03
FY3	100.85	93.93	5505.75	-420.55

表 **4**　问题 **4** 投放策略**-2**

干扰弹投放点的 z 坐标	干扰弹起爆点的 x 坐标	干扰弹起爆点的 y 坐标	干扰弹起爆点的 z 坐标	有效干扰时长
1800.00	17905.80	9.00	1798.65	4.60
1400.00	11405.11	7.84	1163.68	2.95
700.00	5411.79	69.82	561.54	3.10

在表3和表4条件下，某一有效时刻导弹和烟幕云团的遮挡示意图, 如图5所示.

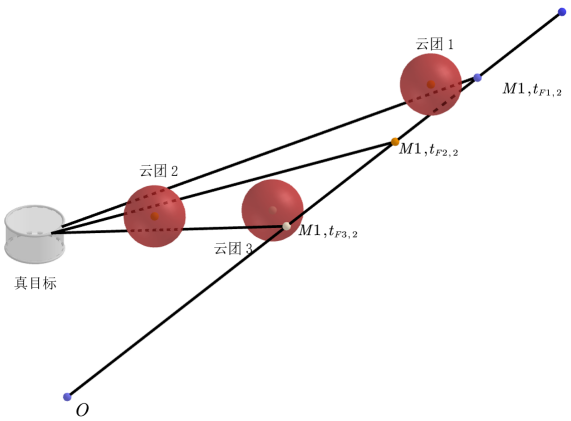

图 **5**

5.4.3 灵敏度分析

调整导弹的飞行速度，利用问题 4 的模型对三架无人机的投弹策略重新进行优化。同理使用遗传算法，利用 Python 求解，结果见图6。

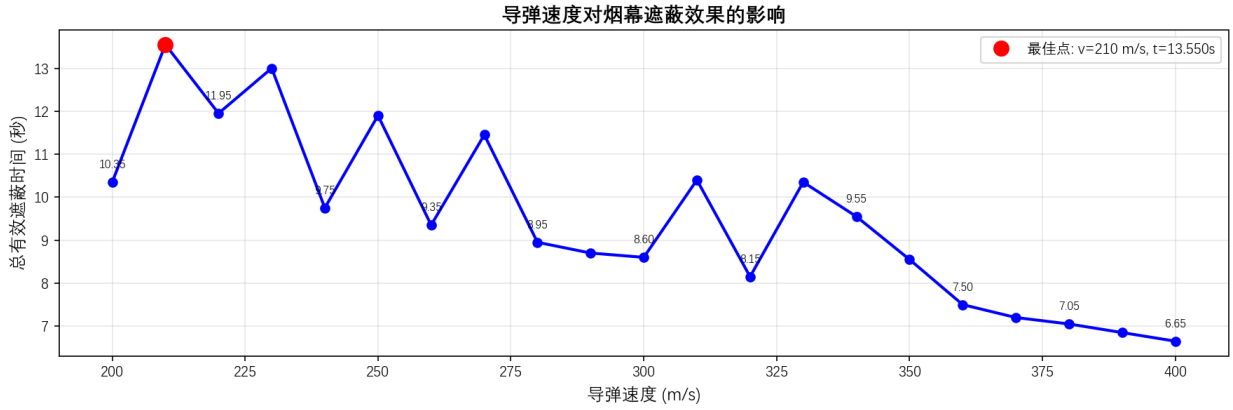

图 **6**　导弹速度与有效时间关系

由图6可知**有效时间对导弹的飞行速度的变化是敏感的**, 并且随着导弹速度增加有效时间逐渐减小。

5.5 问题五

5.5.1 模型建立

Step 1 目标函数

本问中，我们的目标是通过设计 5 架无人机 FYj 与其分别投放的烟幕干扰弹相关参数，使得烟幕干扰弹对导弹 $M1$，$M2$，$M3$ 的有效遮蔽总时间尽可能长，因此目标函数为：

$$\max_{\alpha_j, v_{FYj}, t_{FYj,i1}, t_{FYj,i2}} \Delta t \tag{52}$$

Step 2 决策变量

- **无人机 FYj 的方向**: 设 α_j 为无人机 FYj 与 x 轴正方向的夹角，逆时针为正，范围为 $[0, 2\pi]$，从而确定无人机 FYi 的飞行方向 $(i = 1, 2, 3, 4, 5. j = 1, 2, 3)$。
- **来袭导弹**: 由问题一同理可知导弹 $Mk(k = 1, 2, 3)$ 在 t 时刻的位置坐标 $Mk\,(t)$ 为:

$$\begin{cases} x_{Mk,t} = x_{Mk,0} - \dfrac{x_{Mk,0} v_0 t}{\sqrt{x_{Mk,0}^2 + y_{Mk,0}^2 + z_{Mk,0}^2}} \\ y_{Mk,t} = y_{Mk,0} - \dfrac{y_{Mk,0} v_0 t}{\sqrt{x_{Mk,0}^2 + y_{Mk,0}^2 + z_{Mk,0}^2}} \\ z_{Mk,t} = z_{Mk,0} - \dfrac{z_{Mk,0} v_0 t}{\sqrt{x_{Mk,0}^2 + y_{Mk,0}^2 + z_{Mk,0}^2}} \end{cases} \tag{53}$$

- **无人机 FYj 的飞行速度**: 设无人机的飞行速度为 v_{FYj}，无人机受领任务后，保持等高度匀速直线运动。由第一问可知 t 时刻无人机 FYj 的位置坐标为：

$$\begin{cases} x_{FYj,t} = x_{FYj,0} + v_{FYj} t \cos\alpha \\ y_{FYj,t} = y_{FYj,0} + v_{FYj} t \sin\alpha \\ z_{FYj,t} = z_{FYj,0} \end{cases} \tag{54}$$

- **烟幕干扰弹投放点**: 设无人机 FYj 在受领任务 $t_{FYj,i1}$ s 后投放第 i 枚烟幕干扰弹。由(54)式可知, 无人机 FYj 投放的第 i 枚烟幕干扰弹的投放点位置坐标为

$$\begin{cases} x_{FYj,t_{FYj,i1}} = x_{FYj,0} + v_{FYj} t_{FYj,i1} \cos\alpha \\ y_{FYj,t_{FYj,i1}} = y_{FYj,0} + v_{FYj} t_{FYj,i1} \sin\alpha \\ z_{FYj,t_{FYj,i1}} = z_{FYj,0} \end{cases} \tag{55}$$

- **烟幕干扰弹起爆点**: 设无人机 FYj 投放的第 i 枚烟幕干扰弹在无人机受领任务 $t_{FYj,i2}$ 秒后起爆，因此由问题一可得投放的第 i 枚烟雾干扰弹在 $t_{FYj,i2}$ 时刻即其起爆时的位置坐标：

$$\begin{cases} x_{FYji,t_{FYj,i2}} = x_{FYj,t_{FYj,i1}} + v_{FYj}\left(t_{FYj,i2} - t_{FYj,i1}\right)\cos\alpha \\ y_{FYji,t_{FYj,i2}} = y_{FYj,t_{FYj,i1}} + v_{FYj}\left(t_{FYj,i2} - t_{FYj,i1}\right)\sin\alpha \\ z_{FYji,t_{FYj,i2}} = z_{FYj,t_{FYj,i1}} - \dfrac{g\left(t_{FYj,i2} - t_{FYj,i1}\right)^2}{2} \end{cases} \tag{56}$$

代入问题 1 中，得到 t 时刻 ($t \in \left[\min\limits_{i=1,2,3,4,5}\{t_{FYj,12}\}, \max\limits_{i=1,2,3,4,5}\{t_{FYj,32}\} + \Delta t_0\right]$) 第 i 枚烟幕干扰弹形成的烟幕云团是否对目标进行遮挡，避免导弹 Mk 发现真目标的判别式 $\Delta_{FYjik}(x_l, y_l, z_s)$。代入(21)(22), 得到：

$$\begin{aligned} d_{1ji} &= \sqrt{k_{1ji}^2 (x_{Mk(t)} - x_l)^2 + k_{1ji}^2 (y_{Mk(t)} - y_l)^2 + k_{1ji}^2 (z_{Mk(t)} - z_s)^2}, \\ d_{2ji} &= \sqrt{k_{2ji}^2 (x_{Mk(t)} - x_l)^2 + k_{2ji}^2 (y_{Mk(t)} - y_1)^2 + k_{2ji}^2 (z_{Mk(t)} - z_s)^2}, \end{aligned} \tag{57}$$

其中导弹 $M1$ 到圆柱侧面上的点的的距离：

$$\left|\overrightarrow{N1Mk}\right| = \sqrt{\left(x_{Mk(t)} - x_{n1}\right)^2 + \left(y_{Mk(t)} - y_{n1}\right)^2 + \left(y_{Mk(t)} - y_{n2}\right)^2} \tag{58}$$

只有在 t 时刻，对于 3 枚导弹同时遮挡，才有可能看作有效遮挡，结合问题一中的(25), 则此问中的有效遮挡判断为：

$$\begin{cases} \min\{\Delta_{FYji1},\Delta_{FYji2},\Delta_{FYji3}\}<0 & \text{未形成有效遮挡} \\ \min\{\Delta_{FYji1},\Delta_{FYji2},\Delta_{FYji3}\}\geq 0 \begin{cases} \min\{d_{1ji},d_{2ji}\}>\left|\overrightarrow{N1Mk}\right| & \text{未形成有效遮挡} \\ \min\{d_{1ji},d_{2ji}\}\leq\left|\overrightarrow{N1Mk}\right| & \text{有效遮挡} \end{cases} \end{cases} \tag{59}$$

判断 t 时刻下无人机 FYj 释放的第 i 枚烟幕干扰弹形成的烟幕云团是否对真目标进行遮挡。又因为每个烟幕干扰弹形成的云团都将在 $\Delta t_0=20$ 秒后消散, 所以规定：

$$\Delta_{FYjik}(x_l,y_l,z_s)=-1,\ t\geq t_{FYj,i2}+\Delta t_0 \tag{60}$$

(61)

Step 3 约束条件

- **无人机的飞行速度**：由于无人机受领任务后，可根据需要瞬时调整飞行方向，然后以 70-140m/s 的速度等高度匀速直线飞行。因此：

$$70\leq v_{\text{FY1}}\leq 140 \tag{62}$$

- **无人机投放的烟幕干扰弹的时间**：
  由于题目要求每架无人机投放两枚烟幕干扰弹至少间隔 1s，且据试验数据知，云团中心 10m 范围内的烟幕浓度在起爆 20s 内可为目标提供有效遮蔽。因此：

$$\begin{cases} t_{FYj,11}\in\left[0,\dfrac{x_{mk,0}}{v_0}\right] \\ t_{FYj,12}\in\left[t_{FYj,11},\dfrac{d_{mk,0}}{v_0}\right] \\ t_{FYj,21}\in\left[t_{FYj,11}+1,\dfrac{dx_{mk,0}}{v_0}\right] \\ t_{FYj,22}\in\left[t_{FYj,21},\dfrac{d_{mk,0}}{v_0}\right] \\ t_{FYj,31}\in\left[t_{FYj,21}+1,\dfrac{d_{mk,0}}{v_0}\right] \\ t_{FYj,32}\in\left[t_{FYj,21},\dfrac{d_{mk,0}}{v_0}\right] \end{cases} \tag{63}$$

Step 4 优化模型

综上所述，5 架无人机释放的相应的烟幕干扰弹对真目标的有效遮蔽总时间单目标

优化模型为：

$$
\begin{aligned}
&\max_{\alpha_j, v_{FYj}, t_{FYj,i1}, t_{FYj,i2}} \Delta t \\
&\left\{
\begin{array}{l}
\text{导弹 } Mk \text{ 在 } t \text{ 时刻的位置坐标:} \\
\left\{
\begin{array}{l}
x_{Mk,t} = x_{Mk,0} - \dfrac{x_{Mk,0} v_0 t}{\sqrt{x_{Mk,0}^2 + y_{Mk,0}^2 + z_{Mk,0}^2}} \\
y_{Mk,t} = y_{Mk,0} - \dfrac{y_{Mk,0} v_0 t}{\sqrt{x_{Mk,0}^2 + y_{Mk,0}^2 + z_{Mk,0}^2}} \\
z_{Mk,t} = z_{Mk,0} - \dfrac{z_{Mk,0} v_0 t}{\sqrt{x_{Mk,0}^2 + y_{Mk,0}^2 + z_{Mk,0}^2}}
\end{array}
\right. \\
t\text{时刻无人机的位置坐标:} \\
\left\{
\begin{array}{l}
x_{FYj,t} = x_{FYj,0} + v_{FYj} t \cos\alpha \\
y_{FYj,t} = y_{FYj,0} + v_{FYj} t \sin\alpha \\
z_{FYj,t} = z_{FYj,0} 0 \le \alpha \le 2\pi
\end{array}
\right. \\
70 \le v_{\mathrm{FY1}} \le 140 \\
\text{第 } j \text{ 架无人机的第 } i \text{ 枚烟雾干扰弹起爆时的位置坐标:} \\
\left\{
\begin{array}{l}
x_{FYji,t_{FYj,i2}} = x_{FYj,t_{FYj,i1}} + v_{FYj} \left(t_{FYj,i2} - t_{FYj,i1}\right) \cos\alpha \\
y_{FYji,t_{FYj,i2}} = y_{FYj,t_{FYj,i1}} + v_{FYj} \left(t_{FYj,i2} - t_{FYj,i1}\right) \sin\alpha \\
z_{FYji,t_{FYj,i2}} = z_{FYj,t_{FYj,i1}} - \dfrac{g\left(t_{FYj,i2} - t_{FYj,i1}\right)^2}{2}
\end{array}
\right. \\
\left\{
\begin{array}{l}
t_{FYj,11} \in \left[0, \dfrac{x_{mk,0}}{v_0}\right] \\
t_{FYj,12} \in \left[t_{FYj,11}, \dfrac{d_{mk,0}}{v_0}\right] \\
t_{FYj,21} \in \left[t_{FYj,11} + 1, \dfrac{dx_{mk,0}}{v_0}\right] \\
t_{FYj,22} \in \left[t_{FYj,21}, \dfrac{d_{mk,0}}{v_0}\right] \\
t_{FYj,31} \in \left[t_{FYj,21} + 1, \dfrac{d_{mk,0}}{v_0}\right] \\
t_{FYj,32} \in \left[t_{FYj,21}, \dfrac{d_{mk,0}}{v_0}\right]
\end{array}
\right.
\end{array}
\right.
\end{aligned}
\tag{64}
$$

5.5.2 模型求解

由于问题 5 的变量较多，直接使用启发式搜索算法进行优化非常容易陷入局部最优解中，并且与实际最优解差距过大。因此，我们采用的优化策略是先逐个优化一个无人机对三个导弹的投放策略，再将得到的有效时间合并 (去掉重复有效时间)。反复操作，得到多组不同的投放策略和有效时间，再对比这些投放策略，选取其中有效时间最长的一组作为最优投放策略。以下是核心步骤：

步骤 1：设定初始参数与离散化时间轴

首先定义导弹、无人机、烟幕弹及目标的物理参数，包括速度、加速度、尺寸等。同时设定仿真时间范围与步长，构建离散化时间轴，为后续动态模拟提供基础。

步骤 2：建立目标模型与遮蔽判断机制

通过在真圆柱体上采样关键点，构建目标几何模型。利用辅助函数判断导弹视线是否被烟幕云团（球体）遮挡，即导弹-目标连线是否与烟幕球相交，作为有效遮蔽的核心判据。

步骤 3：模拟烟幕弹投放与云团演化过程

根据无人机飞行方向、速度、投放时间及延迟起爆时间，计算每枚烟幕弹的投放点与爆炸点位置。结合重力与下沉速度，动态更新烟幕云团在空中的三维坐标随时间的变化，并判断其在有效生命周期内是否存在，形成时空分布的遮蔽区域。

步骤 4：评估综合遮蔽效果与时间贡献

在整个时间轴上逐点判断各导弹是否被有效遮蔽。通过累计连续遮蔽区间，计算总有效遮蔽时间及各导弹的遮蔽时长。同时记录每架无人机单独作用时的贡献，用于分析协同效能。

步骤 5：采用差分进化算法进行独立优化

对每架无人机的决策变量（方向角、速度、投放时序、延迟）进行独立优化。以最大化其单独贡献的遮蔽时间为目标函数，多次运行取最优解，确保搜索充分，避免陷入局部极值。

步骤 6：结果整合分析与最优结果输出

将各无人机优化后的参数合并，评估整体协同遮蔽效果。

按照上式步骤，利用 Pytho 求解得到结果，见表5、表6和表7。

表 **5**　问题 **5** 投放策略**-1**

无人机编号	烟幕干扰弹编号	无人机运动方向 (度)	无人机运动速度 (m/s)
FY1	1	7.45	72.00
FY1	2	7.45	72.00
FY1	3	7.45	72.00
FY2	1	295.07	127.02
FY2	2	295.07	127.02
FY2	3	295.07	127.02
FY3	1	79.64	82.34
FY3	2	79.64	82.34
FY3	3	79.64	82.34
FY4	1	0	70.00
FY4	2	0	70.00
FY4	3	0	70.00
FY5	1	120.32	104.78
FY5	2	120.32	104.78
FY5	3	120.32	104.78

表 **6**　问题 **5** 投放策略**-2**

烟幕干扰弹投放点的 x 坐标 (m)	烟幕干扰弹投放点的 y 坐标 (m)	烟幕干扰弹投放点的 z 坐标 (m)	烟幕干扰弹起爆点的 x 坐标 (m)
17800.00	0.00	1800.00	17800.00
17871.39	9.33	1800.00	17903.52
19121.47	172.77	1800.00	19180.73
12425.78	489.95	1400.00	12458.62
12707.30	-111.76	1400.00	12851.56
13363.47	-1514.23	1400.00	13757.49
6502.06	-253.36	700.00	6509.76
6516.87	-172.36	700.00	6555.07
6531.68	-91.36	700.00	6564.69
11064.40	2000.00	1800.00	11099.40
11134.40	2000.00	1800.00	11228.20
11204.40	2000.00	1800.00	11239.40
12151.52	-549.23	1300.00	12064.24
12069.53	-409.04	1300.00	11821.44
11858.99	-49.06	1300.00	11759.02

表 **7**　问题 **5** 投放策略**-3**（烟幕弹起爆点补充 + 有效遮蔽）

烟幕干扰弹起爆点的 y 坐标 (m)	烟幕干扰弹起爆点的 z 坐标 (m)	有效干扰时长 (s)	干扰的导弹编号
0.00	1800.00	2.60	M1
13.53	1799.01	4.50	M1
180.51	1796.62	0.00	M3
419.77	1398.18	4.12	M1
-420.10	1364.81	3.38	M2
-2356.40	1137.45	0.00	M3
-211.24	698.68	1.75	M1
36.61	667.38	2.77	M2
89.26	675.63	0.00	M3
2000.00	1798.78	0.00	M1
2000.00	1791.20	0.00	M2
2000.00	1798.78	0.00	M3
-399.99	1286.66	1.04	M1
15.16	1192.22	3.51	M2
121.89	1282.50	0.00	M3

问题 5 部分结果可视化见图7和图8.

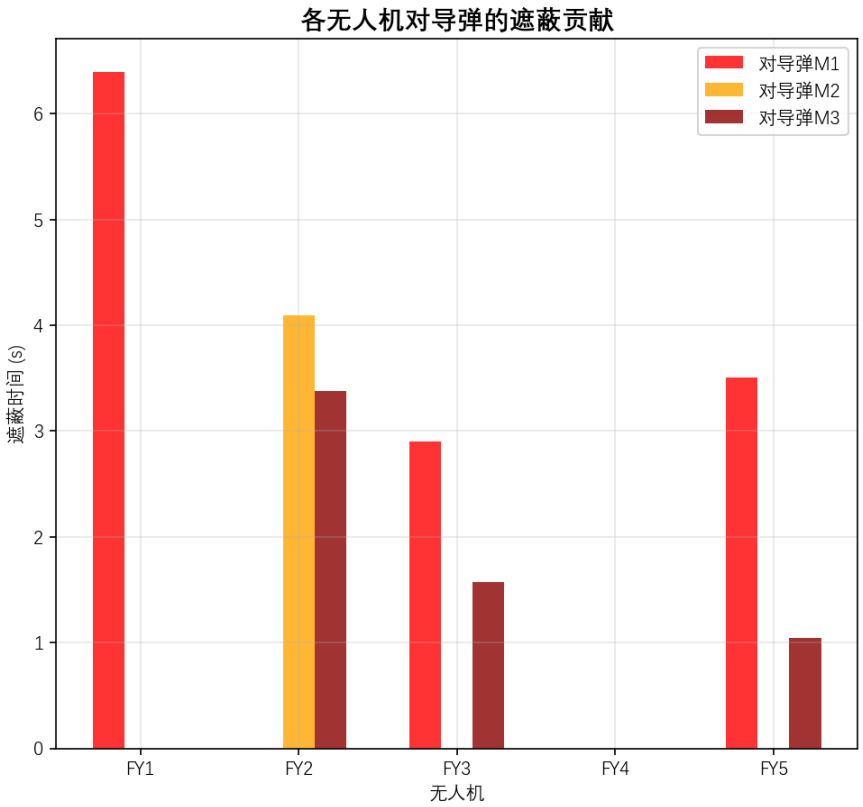

图 7　各无人机对导弹的遮蔽贡献

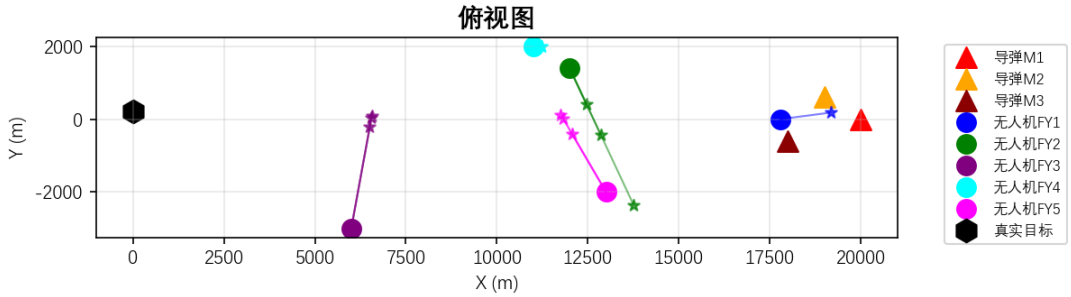

图 8　无人机、导弹和烟雾轨迹俯视图

六、 模型的评价

6.1 模型的优点 [2]

1. 在模型求解过程中，针对不同问题采用了合适的优化算法。如问题二和问题三使用模拟退火算法，问题五采用先逐个优化单无人机投放策略再合并的方法。这些算法有助于在复杂的参数空间中寻找最优解，提高了模型的实用性和有效性。模拟退火算法通过模拟物理退火过程，能够在一定程度上避免陷入局部最优解，在问题三的求解中，通过设

置初始温度、降温系数等参数，在决策变量的可行域内搜索使有效遮蔽时间最大化的最优解。

2. 在问题四的求解中，对导弹飞行速度进行了灵敏度分析。通过调整导弹飞行速度，重新优化无人机投弹策略并观察有效时间的变化，这有助于了解模型中关键参数的变化对结果的影响，为进一步优化模型和制定策略提供了参考依据。发现有效时间对导弹飞行速度敏感且随速度增加而减小，这对于根据导弹实际速度调整无人机干扰策略具有重要意义。

6.2 模型的缺点 [2]

1. 计算复杂度较高：随着问题规模的增加，模型的复杂度迅速上升。在问题五中，需要考虑 5 架无人机、每架最多 3 枚烟幕弹以及 3 枚导弹的情况，决策变量众多，计算量巨大。这不仅增加了计算时间和资源消耗，还可能导致在实际应用中难以快速得到结果，限制了模型的实时性和应用范围。

2. 未充分考虑动态因素：模型主要基于固定的参数和静态的场景进行构建，对于战场环境中的动态因素考虑不足。战场上，导弹和无人机的运动方向、速度可能随时发生变化，烟幕云团的扩散和消散过程也可能受到外界因素干扰而动态变化。模型未能有效处理这些动态变化，使其在面对复杂多变的实际战场情况时适应性较差。

七、 参考文献

[1] 司守奎，孙玺菁，数学建模算法与应用 [M]. 北京：国防工业出版社，2022

[2] Doubao，1.5-pro, 字节跳动,2025-09-05

```python
# 问题一

import numpy as np
import math

# 初始参数设置
M1_0_position = np.array([20000.0, 0.0, 2000.0])
FY1_0_position = np.array([17800.0, 0.0, 1800.0])
v_fy1 = 120 # m/s
v_missile = 300 # m/s
fake_target = np.array([0.0, 0.0, 0.0]) # 原点(假目标)
true_target = np.array([0.0, 200.0, 0.0]) # 真目标
delayed_release = 1.5 # s
delayed_detonation = 3.6 # s
effective_radius = 10 # 烟雾有效半径 m
effective_time = 20 # s
sinking_speed = 3 # m/s
g = 9.8 # 重力加速度m/s²

# 圆柱面参数
y0 = 200
r0 = 7
h0 = 10

dt = 0.1

# 1.5s时无人机投下烟雾弹时的位置
FY1_1_5_position = np.array([
    FY1_0_position[0] - v_fy1 * 1.5,
    FY1_0_position[1],
    FY1_0_position[2]
])

# 5.1s时烟雾弹爆炸前的位置
FY11_5_1_position = np.array([
    FY1_1_5_position[0] - v_fy1 * 3.6,
    FY1_1_5_position[1],
    FY1_1_5_position[2] - g * 3.6 ** 2 / 2
])

M1_to_fake_vec = fake_target - M1_0_position
M1_sum = np.linalg.norm(M1_to_fake_vec)
M1_dir_unit = M1_to_fake_vec / M1_sum

# 定义所有测试点
angles = [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4, np.pi]
```

```
heights = [0, h0 / 2, h0]


def get_valid_point(angle, z):
    x1 = r0 * np.cos(angle)
    y1 = y0 + r0 * np.sin(angle)
    z1 = z
    return [x1, y1, z1]


def get_all():
    """获取所有测试点"""
    test_points = []
    for angle in angles:
        for z in heights:
            test_points.append(get_valid_point(angle, z))
    return test_points


def data(vars, M1_pos, FY11_pos):
    x1, y1, z1 = vars
    x_M1_t, y_M1_t, z_M1_t = M1_pos
    x_FY11_t, y_FY11_t, z_FY11_t = FY11_pos

    term1 = (x_M1_t - x1) * (x1 - x_FY11_t)
    term2 = (y_M1_t - y1) * (y1 - y_FY11_t)
    term3 = (z_M1_t - z1) * (z1 - z_FY11_t)
    sum_terms = term1 + term2 + term3
    part1 = 4 * math.pow(sum_terms, 2)

    squared_sum1 = math.pow(x_M1_t - x1, 2) + math.pow(y_M1_t - y1, 2) + math.pow(z_M1_t - z1,
        2)
    squared_sum2 = math.pow(x1 - x_FY11_t, 2) + math.pow(y1 - y_FY11_t, 2) + math.pow(z1 -
        z_FY11_t, 2) - math.pow(
        effective_radius, 2)
    part2 = 4 * squared_sum1 * squared_sum2

    delta = part1 - part2
    return delta, squared_sum1, 2 * sum_terms


def valid_for_point(delta, a, b):
    if delta < 0:
        return False
    k1 = (-b + np.sqrt(delta)) / (2 * a)
    k2 = (-b - np.sqrt(delta)) / (2 * a)
    return (0 <= k1 <= 1) or (0 <= k2 <= 1)
```

```
def check_validity_all_points(t, M1_pos, FY11_pos):
    test_points = get_all()

    for test_point in test_points:
        delta, a, b = data(test_point, M1_pos, FY11_pos)
        if not (delta >= 0 and valid_for_point(delta, a, b)):
            return False

    return True


def check(t):
    if t < 5.1:
        return False

    M1_pos = M1_0_position + v_missile * t * M1_dir_unit
    FY11_pos = np.array([
        FY11_5_1_position[0],
        FY11_5_1_position[1],
        FY11_5_1_position[2] - sinking_speed * (t - 5.1)
    ])

    return check_validity_all_points(t, M1_pos, FY11_pos)

record = []

for t in np.arange(6.0, 10.0, dt):
    M1_t_position = M1_0_position + v_missile * t * M1_dir_unit

    if t < 5.1:
        continue
    FY11_t_position = np.array([
        FY11_5_1_position[0],
        FY11_5_1_position[1],
        FY11_5_1_position[2] - sinking_speed * (t - 5.1)
    ])

    valid_flag = False
    for angle in angles:
        for z in heights:
            test_point = get_valid_point(angle, z)
            delta, a, b = data(test_point, M1_t_position, FY11_t_position)
            if delta >= 0 and valid_for_point(delta, a, b):
                valid_flag = True
                break
```

```
        if valid_flag:
            break

    if valid_flag:
        record.append(t)

if record:
    def binary_search_earliest(t_low, t_high):
        tol = 1e-6
        max_iter = 100

        valid_low = check(t_low)
        valid_high = check(t_high)

        if valid_low:
            return t_low

        if not valid_high:
            return None

        for iteration in range(max_iter):
            if abs(t_high - t_low) < tol:
                return t_high

            t_mid = (t_low + t_high) / 2
            valid_mid = check(t_mid)

            if valid_mid:
                t_high = t_mid
            else:
                t_low = t_mid

        return t_high


    def binary_search_latest(t_low, t_high):
        tol = 1e-6
        max_iter = 100

        valid_low = check(t_low)
        valid_high = check(t_high)

        if valid_high:
            return t_high

        if not valid_low:
            return None
```

```
    for iteration in range(max_iter):
        if abs(t_high - t_low) < tol:
            return t_low

        t_mid = (t_low + t_high) / 2
        valid_mid = check(t_mid)

        if valid_mid:
            t_low = t_mid
        else:
            t_high = t_mid

    return t_low

if record:
    earliest_value = min(record)
    latest_values = max(record)
    search_margin = 0.5

    earliest_search_low = max(5.1, earliest_value - search_margin)
    earliest_search_high = earliest_value + 0.2
    pea = binary_search_earliest(earliest_search_low, earliest_search_high)

    latest_search_low = latest_values - 0.2
    latest_search_high = latest_values + search_margin
    pla = binary_search_latest(latest_search_low, latest_search_high)

test_points = get_all()

individual_windows = []

for i, test_point in enumerate(test_points):
    angle_idx = i // len(heights)
    height_idx = i % len(heights)

    point_record = []
    for t in np.arange(6.0, 10.0, dt):
        if t < 5.1:
            continue

        M1_pos = M1_0_position + v_missile * t * M1_dir_unit
        FY11_pos = np.array([
            FY11_5_1_position[0],
            FY11_5_1_position[1],
            FY11_5_1_position[2] - sinking_speed * (t - 5.1)
        ])
```

```
            delta, a, b = data(test_point, M1_pos, FY11_pos)
            if delta >= 0 and valid_for_point(delta, a, b):
                point_record.append(t)

        if point_record:
            earliest = min(point_record)
            latest = max(point_record)
            duration = latest - earliest
            individual_windows.append((earliest, latest, duration))
        else:
            individual_windows.append((None, None, 0))

    # 找到最短的时间窗口
    valid_windows = [(start, end, duration) for start, end, duration in individual_windows if
        start is not None]
    if valid_windows:
        shortest_window = min(valid_windows, key=lambda x: x[2])
        longest_window = max(valid_windows, key=lambda x: x[2])

        if pea is not None and pla is not None:
            all_points_window = pla - pea
            print(f"所有点同时被遮蔽的窗口长度: {all_points_window:.6f}s")
```

```
# 问题二

import numpy as np

M1_0_position = np.array([20000.0, 0.0, 2000.0])
FY1_0_position = np.array([17800.0, 0.0, 1800.0])
v_missile = 300 # m/s
fake_target = np.array([0.0, 0.0, 0.0]) # 原点(假目标)
true_target = np.array([0.0, 200.0, 0.0]) # 真目标
effective_radius = 7 # m
sinking_speed = 3 # m/s
g = 9.8 # 重力加速度m/s
r0 = 10
r1 = 7
h0 = 10
y0 = 200

param_ranges = {
    'alpha': (3.13, 3.15),
    'v': (105, 110),
    't1': (0.5, 1.0),
    't2': (4.15, 4.45)
```

```
}


def get_valid_point(angle, z):
    x1 = r1 * np.cos(angle)
    y1 = y0 + r1 * np.sin(angle)
    z1 = z
    return [x1, y1, z1]


def obj(t, target, FY11_t_position):
    M1_sum = np.sqrt(M1_0_position[0] **2 + M1_0_position[1]** 2 + M1_0_position[2] ** 2)
    unit_dir = -M1_0_position / M1_sum
    M1_t_position = M1_0_position + v_missile * t * unit_dir
    OM1_length = np.sqrt(M1_t_position[0] **2 + M1_t_position[1]** 2 + M1_t_position[2] ** 2)

    x1, y1, z1 = target
    x_M1_t, y_M1_t, z_M1_t = M1_t_position
    x_FY11_t, y_FY11_t, z_FY11_t = FY11_t_position

    term1 = (x_M1_t - x1) * (x1 - x_FY11_t)
    term2 = (y_M1_t - y1) * (y1 - y_FY11_t)
    term3 = (z_M1_t - z1) * (z1 - z_FY11_t)
    dot_product = term1 + term2 + term3
    part1 = 4 * (dot_product **2)

    a_sq = (x_M1_t - x1)** 2 + (y_M1_t - y1) **2 + (z_M1_t - z1)** 2
    b_sq = (x1 - x_FY11_t) **2 + (y1 - y_FY11_t)** 2 + (z1 - z_FY11_t) ** 2
    part2 = 4 * a_sq * (b_sq - r0 **2)

    delta = part1 - part2

    return delta, a_sq, 2 * dot_product, x_M1_t, x1, y_M1_t, y1, z_M1_t, z1, OM1_length,
        M1_t_position


def valid(delta, a, b, xm, x1, ym, y1, zm, z1):
    if delta < 0:
        return False

    k1 = (-b + np.sqrt(delta)) / (2 * a)
    k2 = (-b - np.sqrt(delta)) / (2 * a)
    d1 = np.sqrt(pow(k1 * (xm - x1), 2) + pow(k1 * (ym - y1), 2) + pow(k1 * (zm - z1), 2))
    d2 = np.sqrt(pow(k2 * (xm - x1), 2) + pow(k2 * (ym - y1), 2) + pow(k2 * (zm - z1), 2))
    OM = np.sqrt((xm - x1) **2 + (ym - y1)** 2 + (zm - z1) ** 2)
    return d1 <= OM or d2 <= OM
```

```
def calculate_effective_time(params, return_details=False):
    alpha, v, t1, t2 = params

    if t2 <= t1:
        if return_details:
            return 0, [], None, None, None, None
        return 0, []

    flag = -1
    valid_times = []
    delta_records = []
    time_step = 1e-5
    max_time = min(t2 + 20, 50)

    earliest_time = None
    latest_time = None
    M1_earliest_pos = None
    FY11_earliest_pos = None
    M1_latest_pos = None
    FY11_latest_pos = None

    for t in np.arange(t2 + 0.1, max_time, time_step):
        FY1_t1_position = np.array([
            FY1_0_position[0] + v * t1 * np.cos(alpha),
            FY1_0_position[1] + v * t1 * np.sin(alpha),
            FY1_0_position[2]
        ])

        FY11_t2_position = np.array([
            FY1_t1_position[0] + v * (t2 - t1) * np.cos(alpha),
            FY1_t1_position[1] + v * (t2 - t1) * np.sin(alpha),
            FY1_t1_position[2] - g * (t2 - t1) **2 / 2
        ])

        FY11_t_position = np.array([
            FY11_t2_position[0],
            FY11_t2_position[1],
            FY11_t2_position[2] - sinking_speed * (t - t2)
        ])

        angles = np.arange(0, 2 * np.pi, np.pi / 8)
        heights = np.arange(0, h0, h0 / 5)
        is_valid = False
        max_delta = -np.inf

        for angle in angles:
```

```
        for z in heights:
            target = get_valid_point(angle, z)
            result = obj(t, target, FY11_t_position)
            delta, a_sq, b_val, xm, x1, ym, y1, zm, z1, OM1_length, M1_t_position = result

            if delta > max_delta:
                max_delta = delta

            if delta >= 0:
                if valid(delta, a_sq, b_val, xm, x1, ym, y1, zm, z1):
                    is_valid = True
                    break
        if is_valid:
            break

    delta_records.append((t, max_delta))

    if is_valid:
        if flag < 0:
            flag = flag * -1
            earliest_time = t
            M1_earliest_pos = M1_t_position.copy()
            FY11_earliest_pos = FY11_t_position.copy()
        valid_times.append(t)
        latest_time = t
        M1_latest_pos = M1_t_position.copy()
        FY11_latest_pos = FY11_t_position.copy()
    else:
        if flag > 0:
            flag = flag * -1
            break

try:
    if len(valid_times) < 2:
        if return_details:
            return 0, delta_records, None, None, None, None
        return 0, delta_records

    time_window = max(valid_times) - min(valid_times)

    if return_details:
        return (time_window, delta_records,
                earliest_time, latest_time,
                (M1_earliest_pos, FY11_earliest_pos),
                (M1_latest_pos, FY11_latest_pos))
    return time_window, delta_records
except:
```

```
        if return_details:
            return 0, delta_records, None, None, None, None
        return 0, delta_records

def objective_function(params):
    time_window, _ = calculate_effective_time(params)
    return -time_window

def simulated_annealing(initial_params, max_iter=500, initial_temp=100,
     final_temp=0.01,cooling_rate=0.95):
    current_params = np.array(initial_params)
    current_score, _ = calculate_effective_time(current_params)
    best_params = current_params.copy()
    best_score = current_score

    best_details = None

    scores = [current_score]
    temp = initial_temp
    count = 0
    patience = 50

    while temp>final_temp:
        for i in range(max_iter):
            scale = 0.1 + 0.9 * (temp / initial_temp)

            new_params = current_params.copy()
            new_params[0] += np.random.normal(0, 0.1 * scale)
            new_params[1] += np.random.normal(0, 2 * scale)
            new_params[2] += np.random.normal(0, 0.2 * scale)

            new_params[3] += np.random.normal(0, 0.2 * scale)
            if new_params[3] <= new_params[2]:
                new_params[3] = new_params[2] + 0.1 + np.random.random() * 0.5 * scale

            new_params[0] = np.clip(new_params[0], *param_ranges['alpha'])
            new_params[1] = np.clip(new_params[1], *param_ranges['v'])
            new_params[2] = np.clip(new_params[2], *param_ranges['t1'])
            new_params[3] = np.clip(new_params[3], *param_ranges['t2'])

            new_score, _ = calculate_effective_time(new_params)

            score_diff = new_score - current_score

            if score_diff > 0 or np.random.random() < np.exp(score_diff / temp):
                current_params = new_params
                current_score = new_score
```

```
            if current_score > best_score:
                best_params = current_params.copy()
                best_score = current_score
                count = 0

                _, _, earliest, latest, earliest_pos, latest_pos = calculate_effective_time(
                    best_params, return_details=True)
                best_details = (earliest, latest, earliest_pos, latest_pos)
            else:
                count += 1
        else:
            count += 1

        if count >= patience:
            break
        scores.append(current_score)
    temp *= cooling_rate

  return best_params, best_score, scores, best_details

initial_alpha = 3.13
initial_v = 109.78
initial_t1 = 0.71
initial_t2 = 4.23
initial_params = [initial_alpha, initial_v, initial_t1, initial_t2]

(initial_time_window, initial_delta_data,
 initial_earliest, initial_latest,
 initial_earliest_pos, initial_latest_pos) = calculate_effective_time(initial_params,
     return_details=True)

if initial_earliest and initial_latest:
  M1_earliest_init, FY11_earliest_init = initial_earliest_pos
  M1_latest_init, FY11_latest_init = initial_latest_pos

best_params, best_score, scores, best_details = simulated_annealing(
  initial_params,
  max_iter=500,
  initial_temp=10,
  final_temp=0.01,
  cooling_rate=0.95
)


initial_alpha = 3.13
initial_v = 109.78
```

```
initial_t1 = 0.71
initial_t2 = 4.23
initial_params = [initial_alpha, initial_v, initial_t1, initial_t2]

(initial_time_window, initial_delta_data,
 initial_earliest, initial_latest,
 initial_earliest_pos, initial_latest_pos) = calculate_effective_time(initial_params,
     return_details=True)

print(f"最优有效遮蔽时间: {initial_time_window:.6f}秒")
```

```
# 问题三

import numpy as np
import matplotlib.pyplot as plt
from math import cos, sin, pi
import random
import math

plt.rc("font", family='DengXian')
plt.rcParams['axes.unicode_minus'] = False

M1_0_position = np.array([20000.0, 0.0, 2000.0])
FY1_0_position = np.array([17800.0, 0.0, 1800.0])
v_missile = 300 # m/s
fake_target = np.array([0.0, 0.0, 0.0]) # 原点(假目标)
true_target = np.array([0.0, 200.0, 0.0]) # 真目标
effective_radius = 10 # m
effective_time = 20 # s
sinking_speed = 3 # m/s
g = 9.8 # 重力加速度m/s
r0 = 7
h0 = 10
y0 = 200

def obj(t, target, FY1i_t_position):
    unit_dir = -M1_0_position / np.linalg.norm(M1_0_position)
    M1_t_position = M1_0_position + v_missile * t * unit_dir
    OM1_length = np.linalg.norm(M1_t_position)

    x1, y1, z1 = target
    x_M1_t, y_M1_t, z_M1_t = M1_t_position
    x_FY11_t, y_FY11_t, z_FY11_t = FY1i_t_position

    term1 = (x_M1_t - x1) * (x1 - x_FY11_t)
    term2 = (y_M1_t - y1) * (y1 - y_FY11_t)
```

```
    term3 = (z_M1_t - z1) * (z1 - z_FY11_t)
    dot_product = term1 + term2 + term3
    part1 = 4 * (dot_product ** 2)

    a_sq = (x_M1_t - x1) ** 2 + (y_M1_t - y1) ** 2 + (z_M1_t - z1) ** 2
    b_sq = (x1 - x_FY11_t) ** 2 + (y1 - y_FY11_t) ** 2 + (z1 - z_FY11_t) ** 2
    part2 = 4 * a_sq * (b_sq - effective_radius ** 2)

    delta = part1 - part2
    if delta < 0:
        return 0, False

    b = 2 * dot_product
    k1 = (-b + np.sqrt(delta)) / (2 * a_sq)
    k2 = (-b - np.sqrt(delta)) / (2 * a_sq)
    d1 = np.sqrt(pow(k1 * (x_M1_t - x1) + x1, 2) + pow(k1 * (y_M1_t - y1) + y1, 2) + pow(k1 *
         (z_M1_t - z1) + z1, 2))
    d2 = np.sqrt(pow(k2 * (x_M1_t - x1) + x1, 2) + pow(k2 * (y_M1_t - y1) + y1, 2) + pow(k2 *
         (z_M1_t - z1) + z1, 2))
    check = d1 <= OM1_length or d2 <= OM1_length

    return delta, check


def calculate_bomb_position(alpha, v, ta, tb, t):
    FY1_ta_position = np.array([
        FY1_0_position[0] + v * ta * cos(alpha),
        FY1_0_position[1] + v * ta * sin(alpha),
        FY1_0_position[2]
    ])

    FY1_tb_position = np.array([
        FY1_ta_position[0] + v * (tb - ta) * cos(alpha),
        FY1_ta_position[1] + v * (tb - ta) * sin(alpha),
        FY1_ta_position[2] - g * (tb - ta) ** 2 / 2
    ])

    if t >= tb:
        FY1_t_position = np.array([
            FY1_tb_position[0],
            FY1_tb_position[1],
            FY1_tb_position[2] - sinking_speed * (t - tb)
        ])
    else:
        FY1_t_position = np.array([
            FY1_ta_position[0] + v * (t - ta) * cos(alpha),
            FY1_ta_position[1] + v * (t - ta) * sin(alpha),
```

```
            FY1_ta_position[2] - g * (t - ta) ** 2 / 2
        ])

    return FY1_t_position

def check_constraints(params):
    alpha, v, t1a, t2a, t3a, t1b, t2b, t3b = params

    if v < 70 or v > 140:
        return False

    if not (0 < t1a < t2a < t3a and t2a - t1a >= 1 and t3a - t2a >= 1):
        return False

    if not (t1b > t1a and t2b > t2a and t3b > t3a):
        return False

    if alpha < 0 or alpha > 2 * pi:
        return False

    return True

def function(params, debug=False):
    if not check_constraints(params):
        return -1

    alpha, v, t1a, t2a, t3a, t1b, t2b, t3b = params
    time_step = 0.01
    target = get_valid_initial_point()

    missile_time_to_target = np.linalg.norm(M1_0_position) / v_missile

    covered_time_points = []
    coverage_details = []

    bomb_params = [(t1a, t1b), (t2a, t2b), (t3a, t3b)]

    for t in np.arange(0, min(missile_time_to_target, 100), time_step):
        is_covered = False
        covering_bombs = []

        for bomb_idx, (ta, tb) in enumerate(bomb_params):
            if tb <= t <= tb + effective_time:
                bomb_position = calculate_bomb_position(alpha, v, ta, tb, t)
                delta, check = obj(t, target, bomb_position)

                if delta > 0 and check:
```

```
                covering_bombs.append(bomb_idx + 1)
                if not is_covered:
                    is_covered = True

        if is_covered:
            covered_time_points.append(t)
            if debug:
                coverage_details.append({
                    'time': t,
                    'covering_bombs': covering_bombs,
                    'count': len(covering_bombs)
                })

    total_coverage_time = total_coverage(covered_time_points, time_step)

    return total_coverage_time

def get_valid_initial_point():
    angle = np.pi / 4
    x1_init = r0 * np.cos(angle)
    y1_init = y0 + r0 * np.sin(angle)
    z1_init = h0 / 2
    return [x1_init, y1_init, z1_init]

def total_coverage(time_points, dt):
    if len(time_points) == 0:
        return 0

    if len(time_points) == 1:
        return dt

    simple_total = len(time_points) * dt

    return simple_total


def analyze_bomb_contributions(params):
    if not check_constraints(params):
        return

    alpha, v, t1a, t2a, t3a, t1b, t2b, t3b = params
    time_step = 0.1
    target = get_valid_initial_point()
    missile_time_to_target = np.linalg.norm(M1_0_position) / v_missile

    bomb_params = [(t1a, t1b), (t2a, t2b), (t3a, t3b)]
```

```
    individual_contributions = []

    for bomb_idx, (ta, tb) in enumerate(bomb_params):
        covered_times = []

        for t in np.arange(0, min(missile_time_to_target, 100), time_step):
            if tb <= t <= tb + effective_time:
                bomb_position = calculate_bomb_position(alpha, v, ta, tb, t)
                delta, check = obj(t, target, bomb_position)

                if delta > 0 and check:
                    covered_times.append(t)

        contribution = len(covered_times) * time_step
        individual_contributions.append(contribution)

    total = function(params, debug=False)

    return individual_contributions, total

def objective_function(params):
    return function(params, debug=False)


def adaptive_simulated_annealing(initial_params=None, max_iterations=10000, initial_temp=100,
     final_temp=0.01,
                             restart_threshold=500):
    param_ranges = [
        (0, 2 * pi),
        (70, 140),
        (0.1, 5),
        (1.1, 6),
        (3.1, 8),
        (0.2, 10),
        (2.2, 12),
        (4.2, 14)
    ]

    # 初始参数
    if initial_params is None:
        initial_params = np.array([
            3.1356, 140, 1.05, 3.61, 4.91, 5.33, 8.9, 10.78
        ])

    current_params = initial_params.copy()
    current_score = objective_function(current_params)
    best_params = current_params.copy()
```

```
best_score = current_score
best_scores = [best_score]

history_best = [(current_params.copy(), current_score)]

temp = initial_temp
no_improve_count = 0
iteration = 0

cooling_rate = (final_temp / initial_temp) ** (1 / max_iterations)

while iteration < max_iterations and temp > final_temp:
    scale = 0.5 * (1 + temp / initial_temp)

    new_params = current_params.copy()
    param_index = random.randint(0, len(new_params) - 1)

    if param_index == 0:
        range_width = param_ranges[param_index][1] - param_ranges[param_index][0]
        new_params[param_index] += random.uniform(-0.05 * scale, 0.05 * scale) * range_width
    elif param_index == 1:
        new_params[param_index] += random.uniform(-3 * scale, 3 * scale)
    else:
        range_width = param_ranges[param_index][1] - param_ranges[param_index][0]
        new_params[param_index] += random.uniform(-0.3 * scale, 0.3 * scale) * range_width


    new_params[param_index] = np.clip(new_params[param_index],
                                param_ranges[param_index][0],
                                param_ranges[param_index][1])

    if new_params[2] >= new_params[3]:
        new_params[3] = new_params[2] + 1 + random.uniform(0, 0.5)
    if new_params[3] >= new_params[4]:
        new_params[4] = new_params[3] + 1 + random.uniform(0, 0.5)

    if new_params[5] <= new_params[2]:
        new_params[5] = new_params[2] + 0.1 + random.uniform(0, 0.5)
    if new_params[6] <= new_params[3]:
        new_params[6] = new_params[3] + 0.1 + random.uniform(0, 0.5)
    if new_params[7] <= new_params[4]:
        new_params[7] = new_params[4] + 0.1 + random.uniform(0, 0.5)

    new_score = objective_function(new_params)

    if new_score > current_score:
        current_params = new_params
```

```
            current_score = new_score
            no_improve_count = 0

            if new_score > best_score:
                best_params = new_params.copy()
                best_score = new_score
                history_best.append((best_params.copy(), best_score))

        else:
            if best_score > 0:
                quality_factor = min(1.0, current_score / best_score)
            else:
                quality_factor = 1.0

            denominator = temp * quality_factor
            if abs(denominator) < 1e-10:
                prob = 0.0
            else:
                prob = math.exp((new_score - current_score) / denominator)

            if random.random() < prob:
                current_params = new_params
                current_score = new_score

            no_improve_count += 1

        if no_improve_count >= restart_threshold:
            idx = random.randint(0, len(history_best) - 1)
            current_params, current_score = history_best[idx]

            for i in range(len(current_params)):
                if random.random() < 0.3:
                    if i == 0:
                        current_params[i] += random.uniform(-0.05, 0.05)
                    elif i == 1:
                        current_params[i] += random.uniform(-2, 2)
                    else:
                        current_params[i] += random.uniform(-0.2, 0.2)

            current_score = objective_function(current_params)
            no_improve_count = 0
            temp = max(temp * 1.5, final_temp * 2)

        progress = iteration / max_iterations
        if progress < 0.3:
            temp *= (cooling_rate ** 0.5)
        elif progress < 0.7:
```

```
            temp *= cooling_rate
        else:
            temp *= (cooling_rate ** 2)

        if temp < final_temp:
            temp = final_temp

        best_scores.append(best_score)
        iteration += 1

    return best_params, best_score


if __name__ == "__main__":
    best_params, best_score = adaptive_simulated_annealing(max_iterations=5000)
    print("最长有效遮蔽时间: ",best_score)
```

```
# 问题四

import numpy as np
from scipy.optimize import differential_evolution

g = 9.8
v_M1 = 300
R_cloud = 10
v_sink = 3
cloud_lifetime = 20
r0 = 7
h0 = 10

r_M1_0 = np.array([20000, 0, 2000])
O = np.array([0, 0, 0])
T_base = np.array([0, 200, 0])

r_FY1_0 = np.array([17800, 0, 1800])
r_FY2_0 = np.array([12000, 1400, 1400])
r_FY3_0 = np.array([6000, -3000, 700])

e_M1 = (O - r_M1_0) / np.linalg.norm(O - r_M1_0)

t_start = 0
t_end = 100
dt = 0.05
t_range = np.arange(t_start, t_end + dt, dt)
n_t = len(t_range)
```

```
n_points = 2
key_points = np.zeros((2 * n_points + 2, 3))

key_points[0, :] = T_base
key_points[1, :] = T_base + np.array([0, 0, h0])

for i in range(n_points):
    angle = 2 * np.pi * i / n_points
    key_points[2 + i, :] = T_base + r0 * np.array([np.cos(angle), np.sin(angle), 0])

for i in range(n_points):
    angle = 2 * np.pi * i / n_points
    key_points[2 + n_points + i, :] = T_base + r0 * np.array([np.cos(angle), np.sin(angle), 0])
        + np.array(
        [0, 0, h0])

dist_to_target = np.linalg.norm(r_M1_0 - T_base)
est_flight_time = dist_to_target / v_M1

def check_intersect(p1, p2, center, radius):
    d = p2 - p1
    a = p1 - center

    a_dot_d = np.dot(a, d)
    d_dot_d = np.dot(d, d)
    a_dot_a = np.dot(a, a)
    r_sq = radius ** 2

    discriminant = a_dot_d ** 2 - d_dot_d * (a_dot_a - r_sq)

    if discriminant < 0:
        return False

    sqrt_discriminant = np.sqrt(discriminant)

    t1 = (-a_dot_d + sqrt_discriminant) / d_dot_d
    t2 = (-a_dot_d - sqrt_discriminant) / d_dot_d

    if (0 <= t1 <= 1) or (0 <= t2 <= 1):
        return True
    return False

def cal_mul_time(theta, r_M1_0, e_M1, v_M1, r_FY1_0, r_FY2_0, r_FY3_0, g, R_cloud, v_sink,
                                    cloud_lifetime, key_points, t_range, dt):
    angle1, speed1, t_release1, t_delay1 = theta[0:4]
    angle2, speed2, t_release2, t_delay2 = theta[4:8]
    angle3, speed3, t_release3, t_delay3 = theta[8:12]
```

```
t_explosion1 = t_release1 + t_delay1
t_explosion2 = t_release2 + t_delay2
t_explosion3 = t_release3 + t_delay3

e_FY1 = np.array([np.cos(angle1), np.sin(angle1), 0])
e_FY2 = np.array([np.cos(angle2), np.sin(angle2), 0])
e_FY3 = np.array([np.cos(angle3), np.sin(angle3), 0])

r_FY1_release = r_FY1_0 + speed1 * t_release1 * e_FY1
r_S1_explosion = r_FY1_release + t_delay1 * speed1 * e_FY1 - 0.5 * g * t_delay1 ** 2 *
    np.array([0, 0, 1])

r_FY2_release = r_FY2_0 + speed2 * t_release2 * e_FY2
r_S2_explosion = r_FY2_release + t_delay2 * speed2 * e_FY2 - 0.5 * g * t_delay2 ** 2 *
    np.array([0, 0, 1])

r_FY3_release = r_FY3_0 + speed3 * t_release3 * e_FY3
r_S3_explosion = r_FY3_release + t_delay3 * speed3 * e_FY3 - 0.5 * g * t_delay3 ** 2 *
    np.array([0, 0, 1])

n_t = len(t_range)

r_M1 = np.zeros((n_t, 3))
r_C = np.zeros((n_t, 3, 3))
is_effective = np.zeros(n_t, dtype=bool)
has_cloud = np.zeros((n_t, 3), dtype=bool)

for i in range(n_t):
    t = t_range[i]
    r_M1[i, :] = r_M1_0 + v_M1 * t * e_M1

    any_cloud_exists = False

    if t >= t_explosion1 and t <= t_explosion1 + cloud_lifetime:
        has_cloud[i, 0] = True
        any_cloud_exists = True
        r_C[i, 0, :] = r_S1_explosion - v_sink * (t - t_explosion1) * np.array([0, 0, 1])

    if t >= t_explosion2 and t <= t_explosion2 + cloud_lifetime:
        has_cloud[i, 1] = True
        any_cloud_exists = True
        r_C[i, 1, :] = r_S2_explosion - v_sink * (t - t_explosion2) * np.array([0, 0, 1])

    if t >= t_explosion3 and t <= t_explosion3 + cloud_lifetime:
        has_cloud[i, 2] = True
        any_cloud_exists = True
```

```
            r_C[i, 2, :] = r_S3_explosion - v_sink * (t - t_explosion3) * np.array([0, 0, 1])

        if any_cloud_exists:
            is_effective_current = True

            for j in range(key_points.shape[0]):
                any_intersection = False

                for k in range(3):
                    if has_cloud[i, k]:
                        cloud_center = r_C[i, k, :]
                        if check_intersect(r_M1[i, :], key_points[j, :], cloud_center, R_cloud):
                            any_intersection = True
                            break

                if not any_intersection:
                    is_effective_current = False
                    break

            is_effective[i] = is_effective_current

    effective_idx = np.where(is_effective)[0]
    if len(effective_idx) == 0:
        return 0

    effective_intervals = []
    interval_start = t_range[effective_idx[0]]
    prev_idx = effective_idx[0]

    for j in range(1, len(effective_idx)):
        if effective_idx[j] - prev_idx > 1:
            interval_end = t_range[prev_idx]
            effective_intervals.append([interval_start, interval_end])
            interval_start = t_range[effective_idx[j]]
        prev_idx = effective_idx[j]

    interval_end = t_range[effective_idx[-1]]
    effective_intervals.append([interval_start, interval_end])
    effective_intervals = np.array(effective_intervals)

    effective_time = np.sum(effective_intervals[:, 1] - effective_intervals[:, 0])
    return effective_time

def fitness_function(theta):
    return -cal_mul_time(theta,r_M1_0, e_M1, v_M1, r_FY1_0, r_FY2_0, r_FY3_0, g, R_cloud,
        v_sink,
                    cloud_lifetime,key_points, t_range, dt)
```

```
lb = [0.05, 80, 0.5, 0.3, 4, 70, 10, 5, 1.6, 80, 20, 5]
ub = [0.15, 120, 1, 0.6, 4.5, 90, 15, 9, 2.2, 120, 30, 8]

bounds = list(zip(lb, ub))

options = {
    'maxiter': 50,
    'popsize': 20,
    'tol': 1e-6,
    'mutation': (0.5, 1),
    'recombination': 0.7,
    'disp': True,
    'polish': True
}

result = differential_evolution(fitness_function, bounds, **options)
optimal_params = result.x
fval = result.fun

angle1 = optimal_params[0]
angle1_deg = angle1 * 180 / np.pi
speed1 = optimal_params[1]
release_time1 = optimal_params[2]
delay_time1 = optimal_params[3]
explosion_time1 = release_time1 + delay_time1

angle2 = optimal_params[4]
angle2_deg = angle2 * 180 / np.pi
speed2 = optimal_params[5]
release_time2 = optimal_params[6]
delay_time2 = optimal_params[7]
explosion_time2 = release_time2 + delay_time2

angle3 = optimal_params[8]
angle3_deg = angle3 * 180 / np.pi
speed3 = optimal_params[9]
release_time3 = optimal_params[10]
delay_time3 = optimal_params[11]
explosion_time3 = release_time3 + delay_time3

e_FY1 = np.array([np.cos(angle1), np.sin(angle1), 0])
e_FY2 = np.array([np.cos(angle2), np.sin(angle2), 0])
e_FY3 = np.array([np.cos(angle3), np.sin(angle3), 0])

release_pos1 = r_FY1_0 + speed1 * release_time1 * e_FY1
explosion_pos1 = release_pos1 + delay_time1 * speed1 * e_FY1 - 0.5 * g * delay_time1 ** 2 *
```

```
    np.array([0, 0, 1])

release_pos2 = r_FY2_0 + speed2 * release_time2 * e_FY2
explosion_pos2 = release_pos2 + delay_time2 * speed2 * e_FY2 - 0.5 * g * delay_time2 ** 2 *
    np.array([0, 0, 1])

release_pos3 = r_FY3_0 + speed3 * release_time3 * e_FY3
explosion_pos3 = release_pos3 + delay_time3 * speed3 * e_FY3 - 0.5 * g * delay_time3 ** 2 *
    np.array([0, 0, 1])

total_effective_time = -fval
print(f'总有效遮蔽时间: {total_effective_time:.2f} 秒')
```

```
# 问题四灵敏度分析可视化

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import differential_evolution
import time
plt.rc("font",family='DengXian')
plt.rcParams['axes.unicode_minus'] = False
g = 9.8
R_cloud = 10
v_sink = 3
cloud_lifetime = 20
cylinder_radius = 7
cylinder_height = 10
r_M1_0 = np.array([20000, 0, 2000])
O = np.array([0, 0, 0])
T_base = np.array([0, 200, 0])

r_FY1_0 = np.array([17800, 0, 1800])
r_FY2_0 = np.array([12000, 1400, 1400])
r_FY3_0 = np.array([6000, -3000, 700])

e_M1 = (O - r_M1_0) / np.linalg.norm(O - r_M1_0)

t_start = 0
t_end = 100
dt = 0.05
t_range = np.arange(t_start, t_end + dt, dt)
n_t = len(t_range)

n_points = 2
key_points = np.zeros((2 * n_points + 2, 3))
```

```
key_points[0, :] = T_base
key_points[1, :] = T_base + np.array([0, 0, cylinder_height])

for i in range(n_points):
    angle = 2 * np.pi * i / n_points
    key_points[2 + i, :] = T_base + cylinder_radius * np.array([np.cos(angle), np.sin(angle),
        0])

for i in range(n_points):
    angle = 2 * np.pi * i / n_points
    key_points[2 + n_points + i, :] = T_base + cylinder_radius * np.array([np.cos(angle),
        np.sin(angle), 0]) + np.array(
        [0, 0, cylinder_height])

def is_line_segment_intersect_sphere(p1, p2, center, radius):
    d = p2 - p1
    a = p1 - center
    a_dot_d = np.dot(a, d)
    d_dot_d = np.dot(d, d)
    a_dot_a = np.dot(a, a)
    r_sq = radius ** 2
    discriminant = a_dot_d ** 2 - d_dot_d * (a_dot_a - r_sq)

    if discriminant < 0:
        return False

    sqrt_discriminant = np.sqrt(discriminant)
    t1 = (-a_dot_d + sqrt_discriminant) / d_dot_d
    t2 = (-a_dot_d - sqrt_discriminant) / d_dot_d

    if (0 <= t1 <= 1) or (0 <= t2 <= 1):
        return True
    return False

def calculate_multiple_UAVs_shielding_time(theta, v_M1_current):
    angle1, speed1, t_release1, t_delay1 = theta[0:4]
    angle2, speed2, t_release2, t_delay2 = theta[4:8]
    angle3, speed3, t_release3, t_delay3 = theta[8:12]

    t_explosion1 = t_release1 + t_delay1
    t_explosion2 = t_release2 + t_delay2
    t_explosion3 = t_release3 + t_delay3

    e_FY1 = np.array([np.cos(angle1), np.sin(angle1), 0])
    e_FY2 = np.array([np.cos(angle2), np.sin(angle2), 0])
    e_FY3 = np.array([np.cos(angle3), np.sin(angle3), 0])
```

```
r_FY1_release = r_FY1_0 + speed1 * t_release1 * e_FY1
r_S1_explosion = r_FY1_release + t_delay1 * speed1 * e_FY1 - 0.5 * g * t_delay1 ** 2 *
    np.array([0, 0, 1])

r_FY2_release = r_FY2_0 + speed2 * t_release2 * e_FY2
r_S2_explosion = r_FY2_release + t_delay2 * speed2 * e_FY2 - 0.5 * g * t_delay2 ** 2 *
    np.array([0, 0, 1])

r_FY3_release = r_FY3_0 + speed3 * t_release3 * e_FY3
r_S3_explosion = r_FY3_release + t_delay3 * speed3 * e_FY3 - 0.5 * g * t_delay3 ** 2 *
    np.array([0, 0, 1])

n_t = len(t_range)
is_effective = np.zeros(n_t, dtype=bool)
has_cloud = np.zeros((n_t, 3), dtype=bool)

for i in range(n_t):
    t = t_range[i]
    r_M1 = r_M1_0 + v_M1_current * t * e_M1

    any_cloud_exists = False
    r_C = np.zeros((3, 3))

    if t >= t_explosion1 and t <= t_explosion1 + cloud_lifetime:
        has_cloud[i, 0] = True
        any_cloud_exists = True
        r_C[0, :] = r_S1_explosion - v_sink * (t - t_explosion1) * np.array([0, 0, 1])

    if t >= t_explosion2 and t <= t_explosion2 + cloud_lifetime:
        has_cloud[i, 1] = True
        any_cloud_exists = True
        r_C[1, :] = r_S2_explosion - v_sink * (t - t_explosion2) * np.array([0, 0, 1])

    if t >= t_explosion3 and t <= t_explosion3 + cloud_lifetime:
        has_cloud[i, 2] = True
        any_cloud_exists = True
        r_C[2, :] = r_S3_explosion - v_sink * (t - t_explosion3) * np.array([0, 0, 1])

    if any_cloud_exists:
        is_effective_current = True

        for j in range(key_points.shape[0]):
            any_intersection = False

            for k in range(3):
                if has_cloud[i, k]:
                    cloud_center = r_C[k, :]
```

```
                    if is_line_segment_intersect_sphere(r_M1, key_points[j, :], cloud_center,
                        R_cloud):
                        any_intersection = True
                        break

            if not any_intersection:
                is_effective_current = False
                break

        is_effective[i] = is_effective_current

    effective_idx = np.where(is_effective)[0]
    if len(effective_idx) == 0:
        return 0

    effective_intervals = []
    interval_start = t_range[effective_idx[0]]
    prev_idx = effective_idx[0]

    for j in range(1, len(effective_idx)):
        if effective_idx[j] - prev_idx > 1:
            interval_end = t_range[prev_idx]
            effective_intervals.append([interval_start, interval_end])
            interval_start = t_range[effective_idx[j]]
        prev_idx = effective_idx[j]

    interval_end = t_range[effective_idx[-1]]
    effective_intervals.append([interval_start, interval_end])
    effective_intervals = np.array(effective_intervals)

    effective_time = np.sum(effective_intervals[:, 1] - effective_intervals[:, 0])
    return effective_time

def fitness_function(theta, v_M1_current):
    return -calculate_multiple_UAVs_shielding_time(theta, v_M1_current)

def run_optimization_for_velocity(v_M1_current):
    lb = [0.05, 80, 0.5, 0.3, 4, 70, 10, 5, 1.6, 80, 20, 5]
    ub = [0.15, 120, 1, 0.6, 4.5, 90, 15, 9, 2.2, 120, 30, 8]
    bounds = list(zip(lb, ub))

    options = {
        'maxiter': 30,
        'popsize': 15,
        'tol': 1e-6,
        'mutation': (0.5, 1),
        'recombination': 0.7,
```

```
        'disp': False,
        'polish': True
    }

    def fitness_wrapper(theta):
        return fitness_function(theta, v_M1_current)

    result = differential_evolution(fitness_wrapper, bounds, **options)
    optimal_params = result.x
    fval = result.fun

    total_effective_time = -fval

    return total_effective_time, optimal_params

def sensitivity_analysis():

    velocities = np.arange(200, 401, 10)

    effective_times = []
    optimal_parameters = []

    start_time = time.time()

    for i, v_missile in enumerate(velocities):

        try:
            eff_time, opt_params = run_optimization_for_velocity(v_missile)
            effective_times.append(eff_time)
            optimal_parameters.append(opt_params)

        except Exception as e:
            effective_times.append(0)
            optimal_parameters.append(None)

    total_time = time.time() - start_time

    return velocities, effective_times, optimal_parameters

def plot_sensitivity_results(velocities, effective_times):

    plt.figure(figsize=(12, 8))
    plt.subplot(2, 1, 1)
    plt.plot(velocities, effective_times, 'b-o', linewidth=2, markersize=6)
    plt.xlabel('导弹速度 (m/s)', fontsize=12)
    plt.ylabel('总有效遮蔽时间 (秒)', fontsize=12)
    plt.title('导弹速度对烟幕遮蔽效果的影响', fontsize=14, fontweight='bold')
```

```
    plt.grid(True, alpha=0.3)

    for i, (v, t) in enumerate(zip(velocities, effective_times)):
        if i % 2 == 0:
            plt.annotate(f'{t:.2f}', (v, t), textcoords="offset points",
                         xytext=(0, 10), ha='center', fontsize=8)

    max_time_idx = np.argmax(effective_times)
    best_velocity = velocities[max_time_idx]
    best_time = effective_times[max_time_idx]

    plt.plot(best_velocity, best_time, 'ro', markersize=10,
             label=f'最佳点: v={best_velocity} m/s, t={best_time:.3f}s')
    plt.legend()

    plt.subplot(2, 1, 2)
    velocity_changes = np.diff(effective_times) / np.diff(velocities)
    velocity_mid = (velocities[1:] + velocities[:-1]) / 2

    plt.plot(velocity_mid, velocity_changes, 'r-s', linewidth=2, markersize=4)
    plt.xlabel('导弹速度 (m/s)', fontsize=12)
    plt.ylabel('遮蔽时间变化率 (秒/(m/s))', fontsize=12)
    plt.title('遮蔽效果对速度的敏感性', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.axhline(y=0, color='k', linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    velocities, effective_times, optimal_parameters = sensitivity_analysis()
    plot_sensitivity_results(velocities, effective_times)
```

```
# 问题五

import numpy as np
from scipy.optimize import differential_evolution
import time
from datetime import datetime

def main_combined_evaluation():
    g = 9.8
    v_missile = 300
    R_cloud = 10
    v_sink = 3
    shadow_t = 20
```

```
cylinder_radius = 7
cylinder_height = 10
min_interval = 1
max_flares_per_K = 3

r_M1_0 = np.array([20000, 0, 2000])
r_M2_0 = np.array([19000, 600, 2100])
r_M3_0 = np.array([18000, -600, 1900])
r_missile0 = np.vstack([r_M1_0, r_M2_0, r_M3_0])

O = np.array([0, 0, 0])
T_base = np.array([0, 200, 0])

r_FY1_0 = np.array([17800, 0, 1800])
r_FY2_0 = np.array([12000, 1400, 1400])
r_FY3_0 = np.array([6000, -3000, 700])
r_FY4_0 = np.array([11000, 2000, 1800])
r_FY5_0 = np.array([13000, -2000, 1300])
r_Ks0 = np.vstack([r_FY1_0, r_FY2_0, r_FY3_0, r_FY4_0, r_FY5_0])

e_M1 = (O - r_M1_0) / np.linalg.norm(O - r_M1_0)
e_M2 = (O - r_M2_0) / np.linalg.norm(O - r_M2_0)
e_M3 = (O - r_M3_0) / np.linalg.norm(O - r_M3_0)
e_missiles = np.vstack([e_M1, e_M2, e_M3])
t_start = 0
t_end = 100
dt = 0.01
t_range = np.arange(t_start, t_end + dt, dt)
n_t = len(t_range)

n_points = 2
keypoints = np.zeros((2 * n_points + 2, 3))
keypoints[0, :] = T_base
keypoints[1, :] = T_base + np.array([0, 0, cylinder_height])
for i in range(1, n_points + 1):
    angle = 2 * np.pi * (i - 1) / n_points
    keypoints[2 + i - 1, :] = T_base + cylinder_radius * np.array([np.cos(angle),
        np.sin(angle), 0])

for i in range(1, n_points + 1):
    angle = 2 * np.pi * (i - 1) / n_points
    keypoints[2 + n_points + i - 1, :] = T_base + cylinder_radius * np.array(
        [np.cos(angle), np.sin(angle), 0]) + np.array([0, 0, cylinder_height])

dist_to_target = [np.linalg.norm(r_M1_0 - T_base),
                  np.linalg.norm(r_M2_0 - T_base),
                  np.linalg.norm(r_M3_0 - T_base)]
```

```
est_flight_time = np.array(dist_to_target) / v_missile

K_params = [
    {
        'direction_angle': 0.13,
        'speed': 72.00,
        'release_time': np.array([0.00, 1.00, 18.51]),
        'delay_time': np.array([0.00, 0.45, 0.83]),
        'explosion_time': np.array([0.00, 1.00, 18.51])
    },
    # 无人机FY2
    {
        'direction_angle': 5.15,
        'speed': 127.02,
        'release_time': np.array([7.91, 13.14, 25.33]),
        'delay_time': np.array([0.61, 2.68, 7.32]),
        'explosion_time': np.array([8.52, 13.75, 25.94])
    },
    # 无人机FY3
    {
        'direction_angle': 1.39,
        'speed': 82.34,
        'release_time': np.array([33.91, 34.91, 35.91]),
        'delay_time': np.array([0.52, 2.58, 2.23]),
        'explosion_time': np.array([34.43, 35.43, 36.43])
    },
    # 无人机FY4
    {
        'direction_angle': 0.00,
        'speed': 70.00,
        'release_time': np.array([0.92, 1.92, 2.92]),
        'delay_time': np.array([0.50, 1.34, 0.50]),
        'explosion_time': np.array([1.42, 2.42, 3.42])
    },
    # 无人机FY5
    {
        'direction_angle': 2.10,
        'speed': 104.78,
        'release_time': np.array([16.04, 17.59, 21.57]),
        'delay_time': np.array([1.65, 4.69, 1.89]),
        'explosion_time': np.array([17.69, 19.24, 23.22])
    }
]

for i in range(len(K_params)):
    K_params[i]['direction_angle_deg'] = np.mod(K_params[i]['direction_angle'] * 180 /
        np.pi, 360)
```

```
num_Ks = 5
K_contributions = np.zeros(num_Ks)
missile_contributions = np.zeros((num_Ks, 3))
release_po_all = [np.zeros((3, 3)) for _ in range(num_Ks)]
explositon_po_all = [np.zeros((3, 3)) for _ in range(num_Ks)]
explosion_time_all = [np.zeros(3) for _ in range(num_Ks)]

for K_idx in range(num_Ks):
    K_param = K_params[K_idx]
    e_K = np.array([np.cos(K_param['direction_angle']),
                    np.sin(K_param['direction_angle']), 0])

    release_po = np.zeros((3, 3))
    explositon_po = np.zeros((3, 3))
    explosion_time = np.zeros(3)

    for j in range(3):
        release_po[j, :] = r_Ks0[K_idx, :] + K_param['speed'] * K_param['release_time'][j] *
            e_K
        explosion_time[j] = K_param['explosion_time'][j]

        explositon_po[j, :] = (release_po[j, :] +
                               K_param['speed'] * K_param['delay_time'][j] * e_K -
                               0.5 * g * (K_param['delay_time'][j] ** 2) * np.array([0, 0, 1]))

    release_po_all[K_idx] = release_po
    explositon_po_all[K_idx] = explositon_po
    explosion_time_all[K_idx] = explosion_time

for K_idx in range(num_Ks):
    temp_K_params = [{} for _ in range(num_Ks)]
    for i in range(num_Ks):
        if i == K_idx:
            temp_K_params[i] = K_params[i].copy()
        else:
            temp_K_params[i] = {
                'direction_angle': 0,
                'direction_angle_deg': 0,
                'speed': 100,
                'release_time': np.array([1000, 1000, 1000]),
                'delay_time': np.array([1, 1, 1]),
                'explosion_time': np.array([1001, 1001, 1001])
            }

    total_contribution, missile_times, _ = evaluate_combined_solution(
        temp_K_params, r_missile0, e_missiles, v_missile,
```

```
        r_Ks0, g, R_cloud, v_sink, shadow_t,
        keypoints, t_range, dt
    )
    K_contributions[K_idx] = total_contribution
    missile_contributions[K_idx, :] = missile_times

total_time, missile_times, effective_intervals = evaluate_combined_solution(
    K_params, r_missile0, e_missiles, v_missile,
    r_Ks0, g, R_cloud, v_sink, shadow_t,
    keypoints, t_range, dt
)

for i in range(3):
    print(f"导弹M{i + 1}有效遮蔽时间: {missile_times[i]:.2f} 秒")
    if effective_intervals[i].size > 0:
        print(" 有效遮蔽区间: ")
        for j in range(effective_intervals[i].shape[0]):
            start = effective_intervals[i][j, 0]
            end = effective_intervals[i][j, 1]
            print(f"  {start:.2f}秒 - {end:.2f}秒 (持续{end - start:.2f}秒)")
    else:
        print(" 无有效遮蔽区间")

for K_idx in range(num_Ks):
    K_param = K_params[K_idx]
    for j in range(3):
        flare_params = (
            explositon_po_all[K_idx][j, :],
            explosion_time_all[K_idx][j],
        )
        flare_t = evaluate_single_flare_contribution(
            flare_params, r_missile0, e_missiles, v_missile,
            g, R_cloud, v_sink, shadow_t,
            keypoints, t_range, dt
        )
        print(f"\n无人机FY{K_idx + 1} 烟幕弹 #{j + 1}:")
        print(f" 对导弹M1的遮蔽时间: {flare_t[0]:.2f} 秒")
        print(f" 对导弹M2的遮蔽时间: {flare_t[1]:.2f} 秒")
        print(f" 对导弹M3的遮蔽时间: {flare_t[2]:.2f} 秒")

for i in range(num_Ks):
    print(f"\n无人机FY{i + 1}:")
    print(
        f" 飞行方向角: {K_params[i]['direction_angle_deg']:.2f} 度
            ({K_params[i]['direction_angle']:.2f} 弧度)")
    print(f" 飞行速度: {K_params[i]['speed']:.2f} m/s")
    print(
```

```
            f" 对各导弹的贡献: M1={missile_contributions[i, 0]:.2f}s,
                M2={missile_contributions[i, 1]:.2f}s, M3={missile_contributions[i, 2]:.2f}s")

        for j in range(max_flares_per_K):
            print(f" 烟幕干扰弹 #{j + 1}:")
            print(f"  投放时间: {K_params[i]['release_time'][j]:.2f} s")
            print(f"  延迟时间: {K_params[i]['delay_time'][j]:.2f} s")
            print(f"  起爆时间: {K_params[i]['explosion_time'][j]:.2f} s")
            print(
                f"   投放点坐标: ({release_po_all[i][j, 0]:.2f}, {release_po_all[i][j, 1]:.2f},
                    {release_po_all[i][j, 2]:.2f}) m")
            print(
                f"   起爆点坐标: ({explositon_po_all[i][j, 0]:.2f}, {explositon_po_all[i][j,
                    1]:.2f}, {explositon_po_all[i][j, 2]:.2f}) m")

        print(f" 总贡献的有效遮蔽时间: {K_contributions[i]:.2f} s")

def main_independent_optimization():
    g = 9.8
    v_missile = 300
    R_cloud = 10
    v_sink = 3
    shadow_t = 20
    cylinder_radius = 7
    cylinder_height = 10
    min_interval = 1
    max_flares_per_K = 3

    r_M1_0 = np.array([20000, 0, 2000])
    r_M2_0 = np.array([19000, 600, 2100])
    r_M3_0 = np.array([18000, -600, 1900])
    r_missile0 = np.vstack([r_M1_0, r_M2_0, r_M3_0])

    O = np.array([0, 0, 0])
    T_base = np.array([0, 200, 0])

    r_FY1_0 = np.array([17800, 0, 1800])
    r_FY2_0 = np.array([12000, 1400, 1400])
    r_FY3_0 = np.array([6000, -3000, 700])
    r_FY4_0 = np.array([11000, 2000, 1800])
    r_FY5_0 = np.array([13000, -2000, 1300])
    r_Ks0 = np.vstack([r_FY1_0, r_FY2_0, r_FY3_0, r_FY4_0, r_FY5_0])

    e_M1 = (O - r_M1_0) / np.linalg.norm(O - r_M1_0)
    e_M2 = (O - r_M2_0) / np.linalg.norm(O - r_M2_0)
    e_M3 = (O - r_M3_0) / np.linalg.norm(O - r_M3_0)
    e_missiles = np.vstack([e_M1, e_M2, e_M3])
```

```
t_start = 0
t_end = 100
dt = 0.1
t_range = np.arange(t_start, t_end + dt, dt)
n_t = len(t_range)

n_points = 2
keypoints = np.zeros((2 * n_points + 2, 3))
keypoints[0, :] = T_base
keypoints[1, :] = T_base + np.array([0, 0, cylinder_height])
for i in range(1, n_points + 1):
    angle = 2 * np.pi * (i - 1) / n_points
    keypoints[2 + i - 1, :] = T_base + cylinder_radius * np.array([np.cos(angle),
        np.sin(angle), 0])
for i in range(1, n_points + 1):
    angle = 2 * np.pi * (i - 1) / n_points
    keypoints[2 + n_points + i - 1, :] = T_base + cylinder_radius * np.array(
        [np.cos(angle), np.sin(angle), 0]) + np.array([0, 0, cylinder_height])

dist_to_target = [np.linalg.norm(r_M1_0 - T_base),
                  np.linalg.norm(r_M2_0 - T_base),
                  np.linalg.norm(r_M3_0 - T_base)]
est_flight_time = np.array(dist_to_target) / v_missile
print(f"估计导弹飞行时间(M1, M2, M3): {est_flight_time} 秒")

effective_time_window = [0, np.max(est_flight_time) * 1.2]
max_release_t = effective_time_window[1] - 5

vars_per_K = 8
num_Ks = 5

K_opt_params = [{} for _ in range(num_Ks)]
K_contributions = np.zeros(num_Ks)
missile_contributions = np.zeros((num_Ks, 3))
release_po_all = [np.zeros((3, 3)) for _ in range(num_Ks)]
explositon_po_all = [np.zeros((3, 3)) for _ in range(num_Ks)]
explosion_time_all = [np.zeros(3) for _ in range(num_Ks)]

num_runs = 5

for K_idx in range(num_Ks):
    print(f"======== 开始优化无人机 {K_idx + 1} ========")
    best_fval = -np.inf
    best_x_opt = None

    for run in range(num_runs):
        print(f"-- 运行 {run + 1} / {num_runs} --")
```

```
        lb = np.array([
            0,
            70,
            0,
            min_interval,
            min_interval,
            0.0,
            0.0,
            0.0
        ])
        ub = np.array([
            2 * np.pi,
            140,
            max_release_t,
            20,
            20,
            10.0,
            10.0,
            10.0
        ])

        def obj_fun(x):
            return -evaluate_single_K(
                x, K_idx, r_missile0, e_missiles, v_missile,
                r_Ks0, g, R_cloud, v_sink, shadow_t,
                keypoints, t_range, dt
            )

        start_time = time.time()
        result = differential_evolution(
            obj_fun, bounds=list(zip(lb, ub)),
            popsize=20, maxiter=50, disp=False
        )
        optimization_time = time.time() - start_time
        print(f"优化完成，耗时: {optimization_time:.2f} 秒")

        current_solution = -result.fun
        if current_solution > best_fval:
            best_fval = current_solution
            best_x_opt = result.x
            print(f"发现更好的解，适应度值: {best_fval:.2f}")

    x_opt = best_x_opt
    K_param = single_solution_to_params(x_opt)
    K_opt_params[K_idx] = K_param

    e_K = np.array([np.cos(K_param['direction_angle']),
```

```
                np.sin(K_param['direction_angle']), 0])
    release_po = np.zeros((3, 3))
    explositon_po = np.zeros((3, 3))
    explosion_time = np.zeros(3)

    for j in range(3):
        release_po[j, :] = r_Ks0[K_idx, :] + K_param['speed'] * K_param['release_time'][j] *
            e_K
        explosion_time[j] = K_param['explosion_time'][j]
        explositon_po[j, :] = (release_po[j, :] +
                               K_param['speed'] * K_param['delay_time'][j] * e_K -
                               0.5 * g * (K_param['delay_time'][j] ** 2) * np.array([0, 0, 1]))

    release_po_all[K_idx] = release_po
    explositon_po_all[K_idx] = explositon_po
    explosion_time_all[K_idx] = explosion_time

    total_contribution, missile_times, _ = evaluate_single_K_contribution(
        K_param, K_idx, r_missile0, e_missiles, v_missile,
        r_Ks0, g, R_cloud, v_sink, shadow_t,
        keypoints, t_range, dt
    )
    K_contributions[K_idx] = total_contribution
    missile_contributions[K_idx, :] = missile_times

    print(f"无人机{K_idx + 1}总贡献: {total_contribution:.2f} 秒")
    print(f"对导弹M1贡献: {missile_times[0]:.2f} 秒")
    print(f"对导弹M2贡献: {missile_times[1]:.2f} 秒")
    print(f"对导弹M3贡献: {missile_times[2]:.2f} 秒")
    print(f"飞行方向角: {K_param['direction_angle_deg']:.2f} 度")
    print(f"飞行速度: {K_param['speed']:.2f} m/s")
    for j in range(3):
        print(f"烟幕干扰弹 #{j + 1}:")
        print(f" 投放时间: {K_param['release_time'][j]:.2f} s")
        print(f" 延迟时间: {K_param['delay_time'][j]:.2f} s")
        print(f" 起爆时间: {K_param['explosion_time'][j]:.2f} s")
        print(f" 投放点坐标: ({release_po[j, 0]:.2f}, {release_po[j, 1]:.2f}, {release_po[j,
            2]:.2f}) m")
        print(f" 起爆点坐标: ({explositon_po[j, 0]:.2f}, {explositon_po[j, 1]:.2f},
            {explositon_po[j, 2]:.2f}) m")

    for j in range(3):
        flare_params = (
            explositon_po[j, :],
            explosion_time[j],
        )
        flare_t = evaluate_single_flare_contribution(
```

```
            flare_params, r_missile0, e_missiles, v_missile,
            g, R_cloud, v_sink, shadow_t,
            keypoints, t_range, dt
        )
        print(f" 烟幕弹 #{j + 1} 对导弹M1的遮蔽时间: {flare_t[0]:.2f} 秒")
        print(f" 烟幕弹 #{j + 1} 对导弹M2的遮蔽时间: {flare_t[1]:.2f} 秒")
        print(f" 烟幕弹 #{j + 1} 对导弹M3的遮蔽时间: {flare_t[2]:.2f} 秒")

total_time, missile_times, effective_intervals = evaluate_combined_solution(
    K_opt_params, r_missile0, e_missiles, v_missile,
    r_Ks0, g, R_cloud, v_sink, shadow_t,
    keypoints, t_range, dt
)
print(f"总有效遮蔽时间: {total_time:.2f} 秒")
for i in range(3):
    print(f"导弹M{i + 1}有效遮蔽时间: {missile_times[i]:.2f} 秒")
    if effective_intervals[i].size > 0:
        print(" 有效遮蔽区间: ")
        for j in range(effective_intervals[i].shape[0]):
            start = effective_intervals[i][j, 0]
            end = effective_intervals[i][j, 1]
            print(f"  {start:.2f}秒 - {end:.2f}秒 (持续{end - start:.2f}秒)")
    else:
        print(" 无有效遮蔽区间")

for i in range(num_Ks):
    print(f"\n无人机FY{i + 1}:")
    print(
        f" 飞行方向角: {K_opt_params[i]['direction_angle_deg']:.2f} 度
            ({K_opt_params[i]['direction_angle']:.2f} 弧度)")
    print(f" 飞行速度: {K_opt_params[i]['speed']:.2f} m/s")
    print(
        f" 对各导弹的贡献: M1={missile_contributions[i, 0]:.2f}s,
            M2={missile_contributions[i, 1]:.2f}s, M3={missile_contributions[i, 2]:.2f}s")

    for j in range(max_flares_per_K):
        print(f" 烟幕干扰弹 #{j + 1}:")
        print(f"  投放时间: {K_opt_params[i]['release_time'][j]:.2f} s")
        print(f"  延迟时间: {K_opt_params[i]['delay_time'][j]:.2f} s")
        print(f"  起爆时间: {K_opt_params[i]['explosion_time'][j]:.2f} s")
        print(
            f"   投放点坐标: ({release_po_all[i][j, 0]:.2f}, {release_po_all[i][j, 1]:.2f},
                {release_po_all[i][j, 2]:.2f}) m")
        print(
            f"   起爆点坐标: ({explositon_po_all[i][j, 0]:.2f}, {explositon_po_all[i][j,
                1]:.2f}, {explositon_po_all[i][j, 2]:.2f}) m")
```

```
        print(f" 总贡献的有效遮蔽时间: {K_contributions[i]:.2f} s")

def isLineSegmentIntersectSphere(p1, p2, center, radius):
    p1 = p1.flatten()
    p2 = p2.flatten()
    center = center.flatten()

    dist1 = np.linalg.norm(p1 - center)
    dist2 = np.linalg.norm(p2 - center)
    if dist1 <= radius or dist2 <= radius:
        return True

    v = p2 - p1
    a = np.dot(v, v)
    b = 2 * np.dot(p1 - center, v)
    c = np.dot(p1 - center, p1 - center) - radius ** 2
    discriminant = b ** 2 - 4 * a * c

    if discriminant < 0:
        return False

    t1 = (-b + np.sqrt(discriminant)) / (2 * a)
    t2 = (-b - np.sqrt(discriminant)) / (2 * a)

    if (0 <= t1 <= 1) or (0 <= t2 <= 1):
        return True
    else:
        return False


def evaluate_combined_solution_with_status(K_params, r_missile0, e_missiles, v_missile,
                                           r_Ks0, g, R_cloud, v_sink, shadow_t,
                                           keypoints, t_range, dt):
    num_Ks = len(K_params)
    n_t = len(t_range)
    num_missiles = 3

    e_Ks = np.zeros((num_Ks, 3))
    for i in range(num_Ks):
        e_Ks[i, :] = np.array([np.cos(K_params[i]['direction_angle']),
                               np.sin(K_params[i]['direction_angle']), 0])

    release_po = [np.zeros((3, 3)) for _ in range(num_Ks)]
    explositon_po = [np.zeros((3, 3)) for _ in range(num_Ks)]
    for i in range(num_Ks):
        for j in range(3):
            release_po[i][j, :] = (r_Ks0[i, :] +
```

```
                            K_params[i]['speed'] * K_params[i]['release_time'][j] * e_Ks[i,
                                :])
        explositon_po[i][j, :] = (release_po[i][j, :] +
                            K_params[i]['speed'] * K_params[i]['delay_time'][j] * e_Ks[i,
                                :] -
                            0.5 * g * (K_params[i]['delay_time'][j] ** 2) * np.array([0,
                                0, 1]))

r_missiles = np.zeros((n_t, num_missiles, 3))
r_clouds = [[np.zeros((n_t, 3)) for _ in range(3)] for __ in range(num_Ks)]
is_effective = np.zeros((n_t, num_missiles))
has_cloud = np.zeros((n_t, num_Ks, 3))

for i in range(n_t):
    t = t_range[i]
    for j in range(num_missiles):
        r_missiles[i, j, :] = r_missile0[j, :] + v_missile * t * e_missiles[j, :]

    for j in range(num_Ks):
        for k in range(3):
            if (t >= K_params[j]['explosion_time'][k]) and (
                    t <= K_params[j]['explosion_time'][k] + shadow_t):
                has_cloud[i, j, k] = 1
                cloud_center = (explositon_po[j][k, :] -
                                v_sink * (t - K_params[j]['explosion_time'][k]) * np.array([0,
                                    0, 1]))
                r_clouds[j][k][i, :] = cloud_center

    for j in range(num_missiles):
        all_points_covered = True
        for k in range(keypoints.shape[0]):
            point_covered = False
            for m in range(num_Ks):
                for n in range(3):
                    if has_cloud[i, m, n] == 1:
                        cloud_center = r_clouds[m][n][i, :]
                        if isLineSegmentIntersectSphere(
                                r_missiles[i, j, :].reshape(-1, 1),
                                keypoints[k, :].reshape(-1, 1),
                                cloud_center, R_cloud
                        ):
                            point_covered = True
                            break
                if point_covered:
                    break
            if not point_covered:
                all_points_covered = False
```

```
                break
        is_effective[i, j] = all_points_covered

    effective_intervals = [np.array([]) for _ in range(num_missiles)]
    for j in range(num_missiles):
        effective_idx = np.where(is_effective[:, j] == 1)[0]
        if len(effective_idx) == 0:
            effective_intervals[j] = np.array([])
            continue

        intervals = []
        interval_start = t_range[effective_idx[0]]
        prev_idx = effective_idx[0]

        for k in range(1, len(effective_idx)):
            if effective_idx[k] - prev_idx > 1:
                interval_end = t_range[prev_idx]
                intervals.append([interval_start, interval_end])
                interval_start = t_range[effective_idx[k]]
            prev_idx = effective_idx[k]

        interval_end = t_range[effective_idx[-1]]
        intervals.append([interval_start, interval_end])
        effective_intervals[j] = np.array(intervals)

    missile_times = np.zeros(num_missiles)
    for j in range(num_missiles):
        if effective_intervals[j].size > 0:
            missile_times[j] = np.sum(effective_intervals[j][:, 1] - effective_intervals[j][:,
                0])

    total_time = np.sum(missile_times)
    return total_time, missile_times, effective_intervals, is_effective

def evaluate_combined_solution(K_params, r_missile0, e_missiles, v_missile,
                           r_Ks0, g, R_cloud, v_sink, shadow_t,
                           keypoints, t_range, dt):
    total_time, missile_times, effective_intervals, _ = evaluate_combined_solution_with_status(
        K_params, r_missile0, e_missiles, v_missile,
        r_Ks0, g, R_cloud, v_sink, shadow_t,
        keypoints, t_range, dt
    )
    return total_time, missile_times, effective_intervals


def single_solution_to_params(x):
    direction_angle = x[0]
```

```
    direction_angle_deg = np.mod(direction_angle * 180 / np.pi, 360)
    speed = x[1]

    release_time = np.zeros(3)
    release_time[0] = x[2]
    release_time[1] = release_time[0] + x[3]
    release_time[2] = release_time[1] + x[4]

    delay_time = x[5:8]

    explosion_time = release_time + delay_time

    return {
        'direction_angle': direction_angle,
        'direction_angle_deg': direction_angle_deg,
        'speed': speed,
        'release_time': release_time,
        'delay_time': delay_time,
        'explosion_time': explosion_time
    }

def evaluate_single_K(x, K_idx, r_missile0, e_missiles, v_missile,
                      r_Ks0, g, R_cloud, v_sink, shadow_t,
                      keypoints, t_range, dt):
    K_param = single_solution_to_params(x)
    total_contribution, _, _ = evaluate_single_K_contribution(
        K_param, K_idx, r_missile0, e_missiles, v_missile,
        r_Ks0, g, R_cloud, v_sink, shadow_t,
        keypoints, t_range, dt
    )
    return total_contribution

def evaluate_single_K_contribution(K_param, K_idx, r_missile0, e_missiles, v_missile,
                                   r_Ks0, g, R_cloud, v_sink, shadow_t,
                                   keypoints, t_range, dt):
    num_Ks = 5
    temp_K_params = [{} for _ in range(num_Ks)]
    for i in range(num_Ks):
        if i == K_idx:
            temp_K_params[i] = K_param.copy()
        else:
            temp_K_params[i] = {
                'direction_angle': 0,
                'direction_angle_deg': 0,
                'speed': 100,
                'release_time': np.array([1000, 1000, 1000]),
                'delay_time': np.array([1, 1, 1]),
```

```
            'explosion_time': np.array([1001, 1001, 1001])
        }

    total_contribution, missile_times, _ = evaluate_combined_solution(
        temp_K_params, r_missile0, e_missiles, v_missile,
        r_Ks0, g, R_cloud, v_sink, shadow_t,
        keypoints, t_range, dt
    )
    return total_contribution, missile_times, _


def evaluate_single_flare_contribution(flare_params, r_missile0, e_missiles, v_missile,
                                       g, R_cloud, v_sink, shadow_t,
                                       keypoints, t_range, dt):
    num_missiles = 3
    n_t = len(t_range)
    flare_explositon_po, flare_explosion_time = flare_params

    is_effective = np.zeros((n_t, num_missiles))

    for i in range(n_t):
        t = t_range[i]
        missile_positions = np.zeros((num_missiles, 3))
        for j in range(num_missiles):
            missile_positions[j, :] = r_missile0[j, :] + v_missile * t * e_missiles[j, :]

        if (t >= flare_explosion_time) and (t <= flare_explosion_time + shadow_t):
            cloud_center = flare_explositon_po - v_sink * (t - flare_explosion_time) *
                np.array([0, 0, 1])
            for j in range(num_missiles):
                all_points_covered = True
                for k in range(keypoints.shape[0]):
                    if not isLineSegmentIntersectSphere(
                            missile_positions[j, :].reshape(-1, 1),
                            keypoints[k, :].reshape(-1, 1),
                            cloud_center, R_cloud
                    ):
                        all_points_covered = False
                        break
                is_effective[i, j] = all_points_covered
        else:
            is_effective[i, :] = 0

    missile_times = np.zeros(num_missiles)
    for j in range(num_missiles):
        effective_idx = np.where(is_effective[:, j] == 1)[0]
        if len(effective_idx) == 0:
```

```
            continue

        intervals = []
        interval_start = t_range[effective_idx[0]]
        prev_idx = effective_idx[0]
        for k in range(1, len(effective_idx)):
            if effective_idx[k] - prev_idx > 1:
                interval_end = t_range[prev_idx]
                intervals.append([interval_start, interval_end])
                interval_start = t_range[effective_idx[k]]
            prev_idx = effective_idx[k]
        interval_end = t_range[effective_idx[-1]]
        intervals.append([interval_start, interval_end])
        missile_times[j] = np.sum(np.array(intervals)[:, 1] - np.array(intervals)[:, 0])
    return missile_times

if __name__ == "__main__":
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"results_{timestamp}.txt"

    import sys


    class Tee:
        def __init__(self, file_name):
            self.file = open(file_name, 'w')
            self.stdout = sys.stdout
            sys.stdout = self

        def write(self, data):
            self.file.write(data)
            self.stdout.write(data)

        def flush(self):
            self.file.flush()
            self.stdout.flush()

        def close(self):
            sys.stdout = self.stdout
            self.file.close()

    tee = Tee(filename)

    print("=" * 50)
    print("开始执行综合最优参数评估")
    print("=" * 50)
    main_combined_evaluation()
```

```
    print("\n" + "=" * 50)
    print("开始执行无人机独立优化")
    print("=" * 50)
    main_independent_optimization()

    tee.close()
    print(f"\n结果已保存到 {filename}")
```

```
# 问题五结果可视化

import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

plt.rc("font", family='DengXian')
plt.rcParams['axes.unicode_minus'] = False

def create_comprehensive_visualization( r_missiles_0,r_UAVs_0,
                                key_points, UAV_contributions,
                                missile_contributions, release_pos_all,
                                explosion_pos_all):
    fig = plt.figure(figsize=(16, 7))

    fig.subplots_adjust(
        left=0.08,
        right=0.95,
        top=0.90,
        bottom=0.15,
        wspace=0.35,
        hspace=0.3
    )

    ax2 = fig.add_subplot(1, 2, 1)
    plot_2d_overview(ax2, r_missiles_0, r_UAVs_0, key_points, release_pos_all,
        explosion_pos_all)
    ax4 = fig.add_subplot(1, 2, 2)
    plot_uav_contributions(ax4, UAV_contributions, missile_contributions)
    fig.suptitle('导弹防御系统分析报告', fontsize=16, fontweight='bold', y=0.95)
    plt.show()

def plot_2d_overview(ax, r_missiles_0, r_UAVs_0, key_points, release_pos_all,
    explosion_pos_all):
    """绘制2D俯视图"""
```

```
    colors_missiles = ['red', 'orange', 'darkred']
    for i in range(3):
        ax.scatter(r_missiles_0[i, 0], r_missiles_0[i, 1],
                   color=colors_missiles[i], s=120, marker='^', label=f'导弹M{i + 1}')

    colors_uav = ['blue', 'green', 'purple', 'cyan', 'magenta']
    for i in range(5):
        ax.scatter(r_UAVs_0[i, 0], r_UAVs_0[i, 1],
                   color=colors_uav[i], s=100, marker='o', label=f'无人机FY{i + 1}')

        for j in range(3):
            ax.plot([r_UAVs_0[i, 0], release_pos_all[i][j, 0], explosion_pos_all[i][j, 0]],
                    [r_UAVs_0[i, 1], release_pos_all[i][j, 1], explosion_pos_all[i][j, 1]],
                    color=colors_uav[i], alpha=0.5, linewidth=1)
            ax.scatter(explosion_pos_all[i][j, 0], explosion_pos_all[i][j, 1],
                       color=colors_uav[i], s=40, marker='*', alpha=0.7)

    target_circle = Circle((key_points[0, 0], key_points[0, 1]), 7,
                           fill=False, color='black', linewidth=2)
    ax.add_patch(target_circle)
    ax.scatter(key_points[0, 0], key_points[0, 1], color='black', s=150, marker='h',
               label='真实目标')

    ax.set_xlabel('X (m)')
    ax.set_ylabel('Y (m)')
    ax.set_title('俯视图 - 战场部署', fontsize=14, fontweight='bold')
    ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=8)
    ax.grid(True, alpha=0.3)
    ax.set_aspect('equal')

def plot_uav_contributions(ax, UAV_contributions, missile_contributions):
    """绘制无人机贡献条形图"""

    uav_names = [f'FY{i + 1}' for i in range(5)]
    x = np.arange(len(uav_names))
    width = 0.2

    colors = ['red', 'orange', 'darkred']
    for i in range(3):
        ax.bar(x + i * width, missile_contributions[:, i], width,
               label=f'对导弹M{i + 1}', color=colors[i], alpha=0.8)

    ax.set_xlabel('无人机')
    ax.set_ylabel('遮蔽时间 (s)')
    ax.set_title('各无人机对导弹的遮蔽贡献', fontsize=14, fontweight='bold')
    ax.set_xticks(x + width)
    ax.set_xticklabels(uav_names)
```

```
    ax.legend()
    ax.grid(True, alpha=0.3)

def main_combined_evaluation():
    g = 9.8
    v_missile = 300
    R_cloud = 10
    v_sink = 3
    cloud_lifetime = 20
    cylinder_radius = 7
    cylinder_height = 10
    min_interval = 1
    max_flares_per_uav = 3

    r_M1_0 = np.array([20000, 0, 2000])
    r_M2_0 = np.array([19000, 600, 2100])
    r_M3_0 = np.array([18000, -600, 1900])
    r_missiles_0 = np.vstack([r_M1_0, r_M2_0, r_M3_0])

    O = np.array([0, 0, 0])
    T_base = np.array([0, 200, 0])

    r_FY1_0 = np.array([17800, 0, 1800])
    r_FY2_0 = np.array([12000, 1400, 1400])
    r_FY3_0 = np.array([6000, -3000, 700])
    r_FY4_0 = np.array([11000, 2000, 1800])
    r_FY5_0 = np.array([13000, -2000, 1300])
    r_UAVs_0 = np.vstack([r_FY1_0, r_FY2_0, r_FY3_0, r_FY4_0, r_FY5_0])
    e_M1 = (O - r_M1_0) / np.linalg.norm(O - r_M1_0)
    e_M2 = (O - r_M2_0) / np.linalg.norm(O - r_M2_0)
    e_M3 = (O - r_M3_0) / np.linalg.norm(O - r_M3_0)
    e_missiles = np.vstack([e_M1, e_M2, e_M3])
    t_start = 0
    t_end = 100
    dt = 0.01
    t_range = np.arange(t_start, t_end + dt, dt)
    n_t = len(t_range)

    n_points = 2
    key_points = np.zeros((2 * n_points + 2, 3))
    key_points[0, :] = T_base
    key_points[1, :] = T_base + np.array([0, 0, cylinder_height])
    for i in range(1, n_points + 1):
        angle = 2 * np.pi * (i - 1) / n_points
        key_points[2 + i - 1, :] = T_base + cylinder_radius * np.array([np.cos(angle),
            np.sin(angle), 0])
```

```
for i in range(1, n_points + 1):
    angle = 2 * np.pi * (i - 1) / n_points
    key_points[2 + n_points + i - 1, :] = T_base + cylinder_radius * np.array(
        [np.cos(angle), np.sin(angle), 0]) + np.array([0, 0, cylinder_height])

dist_to_target = [np.linalg.norm(r_M1_0 - T_base),
                  np.linalg.norm(r_M2_0 - T_base),
                  np.linalg.norm(r_M3_0 - T_base)]
est_flight_time = np.array(dist_to_target) / v_missile

UAV_params = [
    {
        'direction_angle': 0.13,
        'speed': 72.00,
        'release_time': np.array([0.01, 1.00, 18.51]),
        'delay_time': np.array([0.01, 0.45, 0.83]),
        'explosion_time': np.array([0.02, 1.00, 18.51])
    },
    {
        'direction_angle': 5.15,
        'speed': 127.02,
        'release_time': np.array([7.9, 13.14, 25.33]),
        'delay_time': np.array([0.6, 2.68, 7.32]),
        'explosion_time': np.array([8.5, 13.75, 25.94])
    },
    {
        'direction_angle': 1.39,
        'speed': 82.34,
        'release_time': np.array([33.9, 34.945, 35.91]),
        'delay_time': np.array([0.51, 2.5845, 2.23]),
        'explosion_time': np.array([34.5, 35.429, 36.43])
    },
    {
        'direction_angle': 0.00,
        'speed': 70.00,
        'release_time': np.array([0.92, 1.92, 2.92]),
        'delay_time': np.array([0.50, 1.34, 0.50]),
        'explosion_time': np.array([1.42, 2.42, 3.42])
    },
    {
        'direction_angle': 2.10,
        'speed': 104.78,
        'release_time': np.array([16.04, 17.59, 21.57]),
        'delay_time': np.array([1.65, 4.69, 1.89]),
        'explosion_time': np.array([17.69, 19.24, 23.22])
    }
]
```

```
for i in range(len(UAV_params)):
    UAV_params[i]['direction_angle_deg'] = np.mod(UAV_params[i]['direction_angle'] * 180 /
        np.pi, 360)

num_UAVs = 5
UAV_contributions = np.zeros(num_UAVs)
missile_contributions = np.zeros((num_UAVs, 3))
release_pos_all = [np.zeros((3, 3)) for _ in range(num_UAVs)]
explosion_pos_all = [np.zeros((3, 3)) for _ in range(num_UAVs)]
explosion_time_all = [np.zeros(3) for _ in range(num_UAVs)]

for uav_idx in range(num_UAVs):
    UAV_param = UAV_params[uav_idx]
    e_UAV = np.array([np.cos(UAV_param['direction_angle']),
                      np.sin(UAV_param['direction_angle']), 0])

    release_pos = np.zeros((3, 3))
    explosion_pos = np.zeros((3, 3))
    explosion_time = np.zeros(3)

    for j in range(3):
        release_pos[j, :] = r_UAVs_0[uav_idx, :] + UAV_param['speed'] *
            UAV_param['release_time'][j] * e_UAV
        explosion_time[j] = UAV_param['explosion_time'][j]
        explosion_pos[j, :] = (release_pos[j, :] +
                               UAV_param['speed'] * UAV_param['delay_time'][j] * e_UAV -
                               0.5 * g * (UAV_param['delay_time'][j] ** 2) * np.array([0, 0,
                                   1]))

    release_pos_all[uav_idx] = release_pos
    explosion_pos_all[uav_idx] = explosion_pos
    explosion_time_all[uav_idx] = explosion_time

for uav_idx in range(num_UAVs):
    temp_UAV_params = [{} for _ in range(num_UAVs)]
    for i in range(num_UAVs):
        if i == uav_idx:
            temp_UAV_params[i] = UAV_params[i].copy()
        else:
            temp_UAV_params[i] = {
                'direction_angle': 0,
                'direction_angle_deg': 0,
                'speed': 100,
                'release_time': np.array([1000, 1000, 1000]),
                'delay_time': np.array([1, 1, 1]),
                'explosion_time': np.array([1001, 1001, 1001])
            }
```

```
        total_contribution, missile_times, _ = evaluate_combined_solution(
            temp_UAV_params, r_missiles_0, e_missiles, v_missile,
            r_UAVs_0, g, R_cloud, v_sink, cloud_lifetime,
            key_points, t_range, dt
        )
        UAV_contributions[uav_idx] = total_contribution
        missile_contributions[uav_idx, :] = missile_times

    total_time, missile_times, effective_intervals = evaluate_combined_solution(
        UAV_params, r_missiles_0, e_missiles, v_missile,
        r_UAVs_0, g, R_cloud, v_sink, cloud_lifetime,
        key_points, t_range, dt
    )

    for uav_idx in range(num_UAVs):
        UAV_param = UAV_params[uav_idx]
        for j in range(3):
            flare_params = (
                explosion_pos_all[uav_idx][j, :],
                explosion_time_all[uav_idx][j],
            )
            flare_times = evaluate_single_flare_contribution(
                flare_params, r_missiles_0, e_missiles, v_missile,
                g, R_cloud, v_sink, cloud_lifetime,
                key_points, t_range, dt
            )
    create_comprehensive_visualization(
         r_missiles_0,
        r_UAVs_0,
        key_points, UAV_contributions,
        missile_contributions, release_pos_all,
        explosion_pos_all
    )

    return UAV_params, total_time, missile_times, effective_intervals

def isLineSegmentIntersectSphere(p1, p2, center, radius):
    """判断线段与球体是否相交（修正向量维度）"""
    p1 = p1.flatten()
    p2 = p2.flatten()
    center = center.flatten()

    dist1 = np.linalg.norm(p1 - center)
    dist2 = np.linalg.norm(p2 - center)
    if dist1 <= radius or dist2 <= radius:
        return True
```

```
    v = p2 - p1
    a = np.dot(v, v)
    b = 2 * np.dot(p1 - center, v)
    c = np.dot(p1 - center, p1 - center) - radius ** 2
    discriminant = b ** 2 - 4 * a * c

    if discriminant < 0:
        return False

    t1 = (-b + np.sqrt(discriminant)) / (2 * a)
    t2 = (-b - np.sqrt(discriminant)) / (2 * a)
    if (0 <= t1 <= 1) or (0 <= t2 <= 1):
        return True
    else:
        return False


def evaluate_combined_solution_with_status(UAV_params, r_missiles_0, e_missiles, v_missile,
                                           r_UAVs_0, g, R_cloud, v_sink, cloud_lifetime,
                                           key_points, t_range, dt):
    """带遮蔽状态的合并效果评估（原文档逻辑）"""
    num_UAVs = len(UAV_params)
    n_t = len(t_range)
    num_missiles = 3

    e_UAVs = np.zeros((num_UAVs, 3))
    for i in range(num_UAVs):
        e_UAVs[i, :] = np.array([np.cos(UAV_params[i]['direction_angle']),
                                 np.sin(UAV_params[i]['direction_angle']), 0])

    release_pos = [np.zeros((3, 3)) for _ in range(num_UAVs)]
    explosion_pos = [np.zeros((3, 3)) for _ in range(num_UAVs)]
    for i in range(num_UAVs):
        for j in range(3):
            release_pos[i][j, :] = (r_UAVs_0[i, :] +
                                    UAV_params[i]['speed'] * UAV_params[i]['release_time'][j] *
                                        e_UAVs[i, :])
            explosion_pos[i][j, :] = (release_pos[i][j, :] +
                                      UAV_params[i]['speed'] * UAV_params[i]['delay_time'][j] *
                                          e_UAVs[i, :] -
                                      0.5 * g * (UAV_params[i]['delay_time'][j] ** 2) * np.array([0,
                                          0, 1]))

    r_missiles = np.zeros((n_t, num_missiles, 3))
    r_clouds = [[np.zeros((n_t, 3)) for _ in range(3)] for __ in range(num_UAVs)]
    is_effective = np.zeros((n_t, num_missiles))
```

```
has_cloud = np.zeros((n_t, num_UAVs, 3))

for i in range(n_t):
    t = t_range[i]
    for j in range(num_missiles):
        r_missiles[i, j, :] = r_missiles_0[j, :] + v_missile * t * e_missiles[j, :]

    for j in range(num_UAVs):
        for k in range(3):
            if (t >= UAV_params[j]['explosion_time'][k]) and (
                    t <= UAV_params[j]['explosion_time'][k] + cloud_lifetime):
                has_cloud[i, j, k] = 1
                cloud_center = (explosion_pos[j][k, :] -
                                v_sink * (t - UAV_params[j]['explosion_time'][k]) * np.array([0,
                                    0, 1]))
                r_clouds[j][k][i, :] = cloud_center

    for j in range(num_missiles):
        all_points_covered = True
        for k in range(key_points.shape[0]):
            point_covered = False
            for m in range(num_UAVs):
                for n in range(3):
                    if has_cloud[i, m, n] == 1:
                        cloud_center = r_clouds[m][n][i, :]
                        if isLineSegmentIntersectSphere(
                                r_missiles[i, j, :].reshape(-1, 1),
                                key_points[k, :].reshape(-1, 1),
                                cloud_center, R_cloud
                        ):
                            point_covered = True
                            break
                if point_covered:
                    break
            if not point_covered:
                all_points_covered = False
                break
        is_effective[i, j] = all_points_covered

effective_intervals = [np.array([]) for _ in range(num_missiles)]
for j in range(num_missiles):
    effective_idx = np.where(is_effective[:, j] == 1)[0]
    if len(effective_idx) == 0:
        continue

    intervals = []
    interval_start = t_range[effective_idx[0]]
```

```
        prev_idx = effective_idx[0]
        for k in range(1, len(effective_idx)):
            if effective_idx[k] - prev_idx > 1:
                interval_end = t_range[prev_idx]
                intervals.append([interval_start, interval_end])
                interval_start = t_range[effective_idx[k]]
            prev_idx = effective_idx[k]
        interval_end = t_range[effective_idx[-1]]
        intervals.append([interval_start, interval_end])
        effective_intervals[j] = np.array(intervals)

    missile_times = np.zeros(num_missiles)
    for j in range(num_missiles):
        if effective_intervals[j].size > 0:
            missile_times[j] = np.sum(effective_intervals[j][:, 1] - effective_intervals[j][:,
                0])
    total_time = np.sum(missile_times)

    return total_time, missile_times, effective_intervals, is_effective

def evaluate_combined_solution(UAV_params, r_missiles_0, e_missiles, v_missile,
                           r_UAVs_0, g, R_cloud, v_sink, cloud_lifetime,
                           key_points, t_range, dt):
    """合并效果评估（调用带状态评估，屏蔽状态输出）"""
    total_time, missile_times, effective_intervals, _ = evaluate_combined_solution_with_status(
        UAV_params, r_missiles_0, e_missiles, v_missile,
        r_UAVs_0, g, R_cloud, v_sink, cloud_lifetime,
        key_points, t_range, dt
    )
    return total_time, missile_times, effective_intervals


def evaluate_single_flare_contribution(flare_params, r_missiles_0, e_missiles, v_missile,
                                   g, R_cloud, v_sink, cloud_lifetime,
                                   key_points, t_range, dt):
    num_missiles = 3
    n_t = len(t_range)
    flare_explosion_pos, flare_explosion_time = flare_params

    is_effective = np.zeros((n_t, num_missiles))

    for i in range(n_t):
        t = t_range[i]
        missile_positions = np.zeros((num_missiles, 3))
        for j in range(num_missiles):
            missile_positions[j, :] = r_missiles_0[j, :] + v_missile * t * e_missiles[j, :]
```

```
        if (t >= flare_explosion_time) and (t <= flare_explosion_time + cloud_lifetime):
            cloud_center = flare_explosion_pos - v_sink * (t - flare_explosion_time) *
                np.array([0, 0, 1])
            for j in range(num_missiles):
                all_points_covered = True
                for k in range(key_points.shape[0]):
                    if not isLineSegmentIntersectSphere(
                            missile_positions[j, :].reshape(-1, 1),
                            key_points[k, :].reshape(-1, 1),
                            cloud_center, R_cloud
                    ):
                        all_points_covered = False
                        break
                is_effective[i, j] = all_points_covered
        else:
            is_effective[i, :] = 0

    missile_times = np.zeros(num_missiles)
    for j in range(num_missiles):
        effective_idx = np.where(is_effective[:, j] == 1)[0]
        if len(effective_idx) == 0:
            continue
        intervals = []
        interval_start = t_range[effective_idx[0]]
        prev_idx = effective_idx[0]
        for k in range(1, len(effective_idx)):
            if effective_idx[k] - prev_idx > 1:
                interval_end = t_range[prev_idx]
                intervals.append([interval_start, interval_end])
                interval_start = t_range[effective_idx[k]]
            prev_idx = effective_idx[k]
        interval_end = t_range[effective_idx[-1]]
        intervals.append([interval_start, interval_end])
        missile_times[j] = np.sum(np.array(intervals)[:, 1] - np.array(intervals)[:, 0])
    return missile_times

if __name__ == "__main__":
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"results_{timestamp}.txt"

    import sys


    class Tee:
        def __init__(self, file_name):
            self.file = open(file_name, 'w')
            self.stdout = sys.stdout
```

```
        sys.stdout = self

    def write(self, data):
        self.file.write(data)
        self.stdout.write(data)

    def flush(self):
        self.file.flush()
        self.stdout.flush()

    def close(self):
        sys.stdout = self.stdout
        self.file.close()

tee = Tee(filename)

try:
    main_combined_evaluation()
except Exception as e:
    import traceback

    traceback.print_exc()

tee.close()
```