烟幕干扰弹的无人机投放策略研究

队伍编号：_____

2025 年 12 月 17 日

摘要

定点精确投放的烟幕干扰弹，可以在目标前方特定空域形成遮蔽，有效地干扰敌方导弹。本文针对无人机投放烟幕干扰弹以掩护地面目标的问题，建立了基于空气动力学和空间几何的数学模型。通过分析导弹与烟幕云团的相对运动关系，获得了不同情境下的最优投放策略，以最大化有效遮蔽时间。

针对**问题一**，我们建立了烟幕弹自由落体与扩散模型。已知无人机 $FY1$ 的飞行参数，通过计算烟幕弹起爆时的空间位置与扩散半径，结合导弹 $M1$ 的运动轨迹，求解出两者在时空上的交集，从而得出有效遮蔽时长。

针对**问题二**，这是一个单机单弹对单导弹的优化问题。以无人机飞行方向、速度、投放点和起爆点为决策变量，以遮蔽时间为目标函数，建立了非线性规划模型。利用遗传算法（Genetic Algorithm）进行求解，确定了使遮蔽时间最长的飞行参数。

针对**问题三**，在问题二的基础上引入多枚烟幕弹。考虑到烟幕生成的不连续性，建立了多波次干扰模型。通过规划三枚烟幕弹的时间间隔与空间分布，确立了“接力式”遮蔽策略，并将结果存入 `result1.xlsx`。

针对**问题四**和**问题五**，模型扩展为多机多弹协同对抗多枚导弹（M1, M2, M3）。我们将问题转化为多智能体协同任务分配（Task Assignment）与轨迹规划问题。构建了以总遮蔽效益最大化为目标的整数规划模型，给出了各架无人机的具体任务分配与投放指令。

关键词：烟幕干扰；动态几何遮蔽；多目标优化；投放策略；多机协同

目录

1 问题重述

1.1 问题背景

现代战争中，利用无人机投放烟幕干扰弹是保护地面重要目标的有效手段，具有成本低、效费比高等优点。烟幕弹在空中起爆后形成云团，形成遮蔽，干扰敌方导弹。本题要求在给定的战场环境下（包含来袭导弹位置、无人机位置、目标位置及物理参数），综合考虑多个物体的运动轨迹，设计无人机的飞行与投放策略，以实现最大化的遮蔽效果。

1.2 问题提出

- **问题 1**：对于给定单无人机 $FY1$ 的具体飞行参数，根据运动学建立数学模型计算得出导弹 $M1$ 的有效遮蔽时长。
- **问题 2**：从问题一的特殊情况出发拓展到了一般情况，优化单无人机 $FY1$ 投放 1 枚烟幕弹的策略（方向、速度、投放点、起爆点），使遮蔽时间最大。
- **问题 3**：面对单无人机 $FY1$ 投放多枚烟幕弹的情况，规划烟雾弹的投递时序，实现总遮蔽时间的最大化。
- **问题 4**：不同于单机多枚烟雾弹的情形，情况变成了多机协同（$FY1-FY3$）各投 1 枚弹对抗 $M1$ ，需要找到其投放的最优策略。
- **问题 5**：全要素协同场景，需要找到（5 架无人机，每架最多 3 枚弹）对抗 3 枚导弹（$M1-M3$）的最优策略。

2 问题分析

2.1 问题一的分析

问题一是最简单的情景，要求计算得到给定参数下的有效遮蔽时间。首先以烟幕弹爆炸瞬间的空间位置与导弹的空间位置作为初始值，后续考虑扩散半径，结合导弹运动轨迹，求解两者的交点，从而得到有效遮蔽时长。

2.2 问题二的分析

问题二是问题一的推广，需要将无人机飞行方向、速度、投放点、起爆点作为决策变量进行优化，以最大化遮蔽时间作为目标函数，获取在满足一定的约束条件下的最佳解。实现烟幕弹的最佳投放策略。

2.3 问题三的分析

问题三是问题二的优化，需要控制好烟幕弹投放的时序，完成烟幕弹的协同投放，以实现多弹接力式的遮蔽效果，从而实现最大化有效遮蔽时间。

2.4 问题四的分析

问题四是多机协同的情况，需要考虑多架无人机在时空上的协同，实现在空间与时间上的遮蔽效果的最大化。

2.5 问题五的分析

问题五是最复杂的情景，涉及多机多弹对抗多枚导弹的情况。需要综合考虑无人机的任务分配、轨迹规划以及烟幕弹的投放时序等多个因素，建立一个综合性的优化模型，以实现整体遮蔽效益的最大化。但是直接考虑所有的决策变量，时间复杂度与空间复杂度较高，无法直接求解。因而需要对问题进行简化，将问题转化为多智能体协同任务分配与投放策略问题。

3 模型假设

为简化问题，主要假设如下：

1. 假设烟幕弹在投放后忽略空气阻力，仅受重力作用做平抛运动。
2. 假设烟幕云团形成球状且半径在有效时间内保持稳定（或按题目给定速度扩散/下沉）。
3. 假设来袭导弹做匀速直线运动，不进行机动变轨。
4. 假设雷达发现目标时刻为 $t = 0$。
5. 忽略无人机调整飞行方向时的转弯半径，视为瞬时变向。
6. 假设无人机、导弹、烟幕弹、烟幕云团均为质点。

4 符号说明

本文主要使用的符号定义如下表4所示：

表 1: 符号说明（按物理意义分组排序）

符号	含义	单位
M_j	第 j 枚来袭导弹 $(j=1,2,3)$	-
FY_i	第 i 架无人机 $(i=1,2,3,4,5)$	-
V_m	导弹飞行速度 (固定 300)	m/s
v_{FY_i}	第 i 架无人机飞行速度	m/s
$\boldsymbol{v_{FY_i}}$	无人机速度向量	m/s
$\boldsymbol{n_{M_j}}$	导弹飞行方向单位向量	-
α	无人机飞行航向角	rad
g	重力加速度	m/s^2
t_{drop}	烟幕弹投放时刻	s
t_{burst}	烟幕弹起爆时刻	s
t_{delay}	烟幕弹起爆延时	s
R_{cloud}	烟幕云团有效遮蔽半径	m
v_{cloud}	烟幕云团下沉速度	m/s
$\boldsymbol{P}_{S,k}(t)$	第 k 枚烟幕弹 t 时刻位置	m
$\boldsymbol{P}_{B,k}$	第 k 枚烟幕弹起爆位置	m
$D(t)$	烟幕云团中心到导弹-目标连线距离	m
T_{cover}	有效遮蔽时间	s
$I_j(t)$	第 j 枚导弹的遮蔽指示函数	-

5 模型建立

根据题目描述首先建立统一的三维直角坐标系 $Oxyz$。以假目标中心为原点 x 轴与 y 轴位于水平面内，z 轴竖直向上, 真目标位于点 $P_T=(0,200,h_T)$ 其中 h_T 为真目标中心高度。

第 j 枚导弹的初始位置位于点

$$P_{M_j}(0)$$

第 i 架无人机的初始位置位于点

$$P_{FY_i}(0)$$

我们可以利用 matlab 绘制初始时刻的宏观战场态势图，如图1所示。

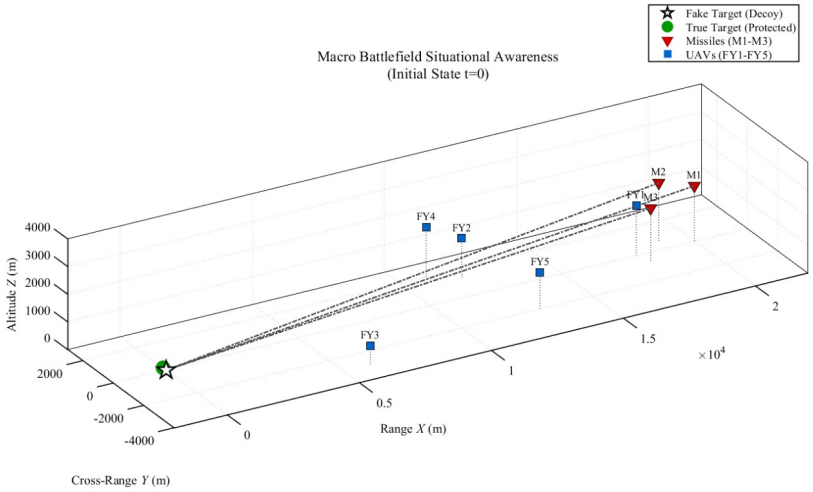

图 1: 宏观战场态势图

通过观察此图，我们可以发现，无人机群在导弹群前方按照扇形分布，形成了一个闪现拦截带，无人机 FY1 距离导弹群最近，这说明 FY1 后续可能应该优先对导弹进行拦截。同时在高度上，无人机群成梯度配置，可以形成低位前伸烟幕拦截网。

5.1 导弹运动模型构建

导弹 M_j 以速度 $V_m = 300m/s$ 直线飞向假目标 $\boldsymbol{P_O}$，其单位方向向量为：

$$\boldsymbol{n_{M_j}} = \frac{\boldsymbol{P_O} - \boldsymbol{P_{M_j}}(0)}{||\boldsymbol{P_O} - \boldsymbol{P_{M_j}}(0)||}$$

第 j 枚导弹初始位置为

$$\boldsymbol{P}_{M_j}(0)$$

导弹在任意时刻 t 的位置为：

$$\boldsymbol{P}_{M_j}(t) = \boldsymbol{P}_{M_j}(0) + V_m \boldsymbol{n}_{M_j}$$

5.2 无人机运动模型构建

第 i 枚无人机的初始位置为

$$\boldsymbol{P}_{FY_i}(0)$$

第 i 架无人机飞行速度为 v_i，航向角为 α_i ，其速度向量表示为：

$$\boldsymbol{v}_{FY_i} = v_i\left(\cos\alpha_i, \sin\alpha_i, 0\right)$$

无人机在任意时刻 t 的位置为：

$$\boldsymbol{P}_{FY_i}\left(t\right) = \boldsymbol{P}_{FY_i}\left(0\right) + \boldsymbol{v}_{FY_i}t$$

5.3 烟幕干扰弹的运动模型

设第 k 枚烟幕弹在 $t_{drop,k}$ 时刻投放，起爆延时为 $t_{delay,k}$，重力加速度为 g 起爆时刻为：

$$t_{burst,k} = t_{drop,k} + t_{delay,k}$$

起爆位置为：

$$\boldsymbol{P}_{B,k} = \boldsymbol{P}_{FY_i}(t_{drop,k}) + \boldsymbol{v}_{FY_i}t_{delay,k} - \left(0, 0, \frac{1}{2}gt_{delay,k}^2\right)$$

5.4 烟幕弹的下沉模型

烟幕弹起爆后形成半径为 R_{cloud} 的球形云团，其中心以速度 v_{cloud} 匀速下沉：

$$\boldsymbol{P}_{S,k}(t) = \boldsymbol{P}_{B,k} - (0, 0, v_{cloud}(t - t_{p,k})), \quad t \geq t_{p,k}$$

5.5 有效遮蔽判定模型

5.5.1 几何遮挡条件

根据题目，可知无人机及导弹的初始位置离真目标（原点为（0，200，0），半径为 7，高度为 10 的圆柱体）所在位置十分遥远，而题中未提到无人机和导弹的视野，那说明其不在题目考虑范围内。我们可以合理假设导弹视野为全范围，即只要在导弹正前方的事物，都处于导弹视野范围内，那对于一个距离十分远，且相对于距离量级来说大小很小的一个真目标，便于进行计算和后续的优化问题，我们可以将其简化为一个质点，进而将遮蔽条件转化为：当烟幕云团中心到“导弹与真目标质心连线”的垂直距离小于等于烟幕半径时，视为有效遮蔽。

5.5.2 点到线段距离算法

设导弹位置为点 A，真目标位置为点 B，烟幕弹中心为点 P。向量 $\overrightarrow{AB} = B - A$，向量 $\overrightarrow{AP} = P - A$。投影系数 r 计算公式为：

$$r = \frac{\overrightarrow{AP} \cdot \overrightarrow{AB}}{\|\overrightarrow{AB}\|^2}$$

距离分三种情况讨论：

$$D(t) = \begin{cases} \|\overrightarrow{AP}\| & \text{if } r \le 0 \text{ (导弹后方)} \\ \|P - B\| & \text{if } r \ge 1 \text{ (目标后方)} \\ \frac{\|\overrightarrow{AP} \times \overrightarrow{AB}\|}{\|\overrightarrow{AB}\|} & \text{if } 0 < r < 1 \text{ (线段投影内)} \end{cases}$$

5.6 遮蔽时间定义

遮蔽时间定义为：

$$I_j(t) = \begin{cases} 1, & \text{if } D_{k,i}(t) \le R_{cloud} \text{ and } t \ge t_{p,k} \\ 0, & \text{otherwise} \end{cases}$$

其中 $D_{k,i}(t)$ 为第 k 枚烟幕弹对第 j 枚导弹在时刻 t 的点到线段距离。
导弹 j 的总有效遮蔽时长为：

$$T_j = \int_0^{T_{end}} I_j\,(t)\,dt$$

在不同问题中，优化目标统一表示为：

$$\max\ T_{cover}\ = \sum_j T_j$$

6 模型求解

6.1 问题一求解：固定参数

问题一针对的是具体场景在给定所有的参数下 (烟幕弹的投放时间，起爆时间)，求解得出单枚烟幕弹对单枚导弹的有效遮蔽时间，需要我们利用各个物体的位置函数，结合有效遮挡判据得到有效遮挡时间。针对本题，飞行策略和投弹策略为题目给定，本题求解核心在于对已知任务的仿真模拟，所以直接使用时间步长离散化仿真即可解决本题。

6.1.1 参数代入与场景设定

根据题目条件，无人机 FY1 的初始位置为 $\mathbf{P}_{FY_1}(0) = (17800,\ 0,\ 1800)$, 飞行速度为 $v_{FY_1} = 120\,\mathrm{m/s}$, 航向指向假目标 $\boldsymbol{P_O} = (0, 0, 0)$。烟幕干扰弹在受领任务后 $t_{drop} = 1.5\,\mathrm{s}$ 时投放，起爆延迟时间为 $t_{delay} = 3.6\,\mathrm{s}$. 导弹 M1 的初始位置为 $\mathbf{P}_M(0) = (20000,\ 0,\ 2000)$, 飞行速度为 $v_M = 300\,\mathrm{m/s}$, 并沿直线飞向假目标。烟幕云团半径取 $R_s = 10\,\mathrm{m}$ 下沉速度取 $v_s = 3\,\mathrm{m/s}$.

6.1.2 运动轨迹计算

由模型建立中的运动模型，可确定系统中各实体的空间位置。无人机在投放时刻的空间位置为

$$\mathbf{P}_{FY_1} = \mathbf{P}_{FY_1}(t_{drop}).$$

烟幕干扰弹的起爆时刻为

$$t_{burst} = t_{drop} + t_{delay},$$

其起爆位置为

$$\mathbf{P}_{burst} = \mathbf{P}_{FY_1}(t_{drop}) + \mathbf{v}_{FY_1} t_{delay} - \left(0,\, 0,\, \tfrac{1}{2} g t_{delay}^2\right).$$

起爆后，烟幕云团中心以恒定速度沿竖直方向下沉，其在任意时刻 t 的位置为

$$\mathbf{P}_S(t) = \mathbf{P}_{burst} - (0,\, 0,\, v_s(t - t_{burst})), \quad t \geq t_{burst}.$$

导弹 M1 在任意时刻 t 的位置为

$$\mathbf{P}_M(t) = \mathbf{P}_M(0) + v_M \mathbf{n}_M t,$$

其中 $\mathbf{n}_M$ 为导弹指向假目标的单位方向向量。

6.1.3 有效遮蔽时长计算

为计算烟幕干扰弹对导弹 M1 的有效遮蔽时长，将导弹飞行全过程时间区间 $[0, T_{end}]$ 进行离散化处理。设时间步长为 Δt，离散时间点为

$$t_k = k\Delta t, \quad k = 0, 1, \ldots, N.$$

在每一离散时刻 t_k，计算烟幕云团中心到导弹与真目标连线的最短距离 $d(t_k)$。当满足

$$d(t_k) \leq R_{cloud} \quad \text{且} \quad t_k \geq t_{burst}$$

时，认为该时刻烟幕对导弹 M1 实现有效遮蔽。定义遮蔽指示函数

$$I(t_k) = \begin{cases} 1, & d(t_k) \leq R_{cloud},\ t_k \geq t_{burst}, \\ 0, & \text{otherwise.} \end{cases}$$

则烟幕干扰弹对导弹 M1 的总有效遮蔽时长可近似表示为

$$T_{cover} \approx \sum_{k=0}^{N} I(t_k)\, \Delta t.$$

6.1.4 模型汇总

$$
\begin{cases}
\mathbf{P}_M(t) = \mathbf{P}_M(0) + V_m \mathbf{n}_M t \\
\mathbf{P}_{FY_1}(t) = \mathbf{P}_{FY_1}(0) + \mathbf{v}_{FY_1} t \\
\mathbf{P}_B = \mathbf{P}_{FY_1}(t_{drop}) + \mathbf{v}_{FY_1} t_{delay} - \left(0, 0, \frac{1}{2} g t_{delay}^2\right) \\
\mathbf{P}_S(t) = \mathbf{P}_B - (0, 0, v_{cloud}(t - t_{burst})) \\
D(t) = \begin{cases}
\|\overrightarrow{AP}\| & \text{if } r \leq 0 \text{ (导弹后方)} \\
\|P - B\| & \text{if } r \geq 1 \text{ (目标后方)} \\
\frac{\|\overrightarrow{AP} \times \overrightarrow{AB}\|}{\|\overrightarrow{AB}\|} & \text{if } 0 < r < 1 \text{ (线段投影内)}
\end{cases} \\
I(t) = \begin{cases}
1, & \text{if } D(t) \leq R_{cloud} \text{ and } t \geq t_{burst} \\
0, & \text{otherwise}
\end{cases} \\
T = \int_0^{T_{end}} I(t)\, dt
\end{cases}
$$

6.1.5 求解结果：

经计算，结果如下图2所示, 烟幕弹起爆时刻：$t = 5.10s$, 烟幕起爆初始坐标：$(17188.00, 0.00, 1736.50)$ 有效遮蔽时间段为 $8.1 - 9.5s$，总时长为 $1.4s$。

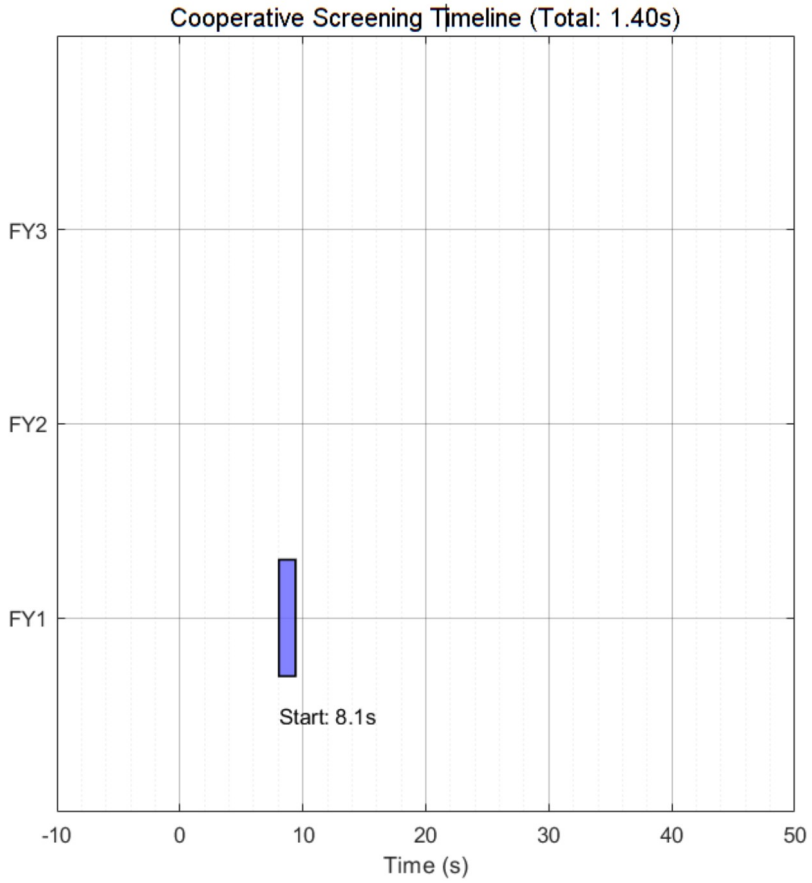

图 2: 问题一结果

6.2 问题二求解：单机单弹优化

在问题二中，无人机 FY1 的任务是通过自身的飞行参数和投弹时机，使得单枚烟雾弹对导弹 M1 的有效遮蔽时间最大化。

6.2.1 决策变量

根据题目要求，无人机一旦受领任务，其速度和航向就确定了且不再调整。因此在进行任务决策时我们需要决策以下四个变量：飞行速度 v_{FY1}, 飞行航向角 α, 投放时刻 t_{drop}, 延时起爆时间 t_{delay}, 由此定义出决策向量 X:

$$X = [v_{FY1}, \alpha, t_{drop}, t_{delay}]^T$$

6.2.2 运动状态方程

基于问题一的运动学模型，需要将无人机的速度向量参数化。无人机速度向量可表示为：

$$\boldsymbol{v_{FY_1}} = \begin{bmatrix} v_{FY1}cos\alpha \\ v_{FY_1}sin\alpha \\ 0 \end{bmatrix}$$

投放点位置和起爆点位置更新为用 X 表示的函数：

$$\boldsymbol{P}_{drop}(X) = \boldsymbol{P}_{FY_1}(0) + \boldsymbol{v_{FY_1}}(v_{FY_1}, \alpha) \cdot t_{drop}$$

$$\boldsymbol{P}_{burst}(X) = \boldsymbol{P}_{drop}(X) + \boldsymbol{v_{FY_1}}(v_{FY_1}, \alpha) \cdot t_{delay} + \left[0, 0, -\frac{1}{2}gt_{delay}^2\right]^T$$

6.2.3 目标函数

目标是最大化有效遮蔽时长，利用问题一中定义的遮蔽判定指示函数得出目标函数：

$$maxJ(X) = T_{cover} = \int_0^{T_{end}} I(t, X)\, dt$$

6.2.4 约束条件

根据题目和对于模型速度特征的分析，约束条件如下: 速度约束：

$$70 \leq v_{FY_1} \leq 140$$

时间约束：

$$0 < t_{drop} \leq 12, 1 \leq t_{delay} \leq 8$$

高度约束：

$$z_{burst}(X) > 0$$

6.2.5 求解算法

由于目标函数涉及复杂的几何运动及分段逻辑，具有高度的非线性和非凸性，且决策变量相对于一般的单变量优化问题具有高自由度。所以为了解决此类问题，本问题采用双层优化策略，外层使用网格扫描，内层使用粒子群优化算法。

算法整体架构 外层循环（针对速度，因为速度对优化结果的影响最大）：将速度在 [70，140] 区间内以 0.5m/s 为步长进行离散化扫描。内层循环（针对剩余参数）：针对每一个固定的速度，使用 PSO 算法寻找最优组合。这样分层寻优的好处是将优化问题维度从 4 维降至 3 维，降低了搜索难度，降低问题陷入局部最优的概率。

粒子群算法设计 在内层针对剩余决策变量进行寻优。设置粒子群规模为 N=30; 粒子速度与位置更新公式：

$$v_i^{k+1} = wv_i^k + c_1 r_1 \left(p_{best,i} - x_i^k\right) + c_2 r_2 \left(g_{best} - x_i^k\right)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}$$

适应度函数：

$$F\left(x\right) = -T_{cover}\left(x\right)$$

求解结果： 经过计算，得到最优投放策略如下：
无人机以 196.91° 的航向角度，72m/s 的速度，0s 时刻投放，2.4991s 后起爆，有效遮蔽时长为 4.738s 如下图3所示。

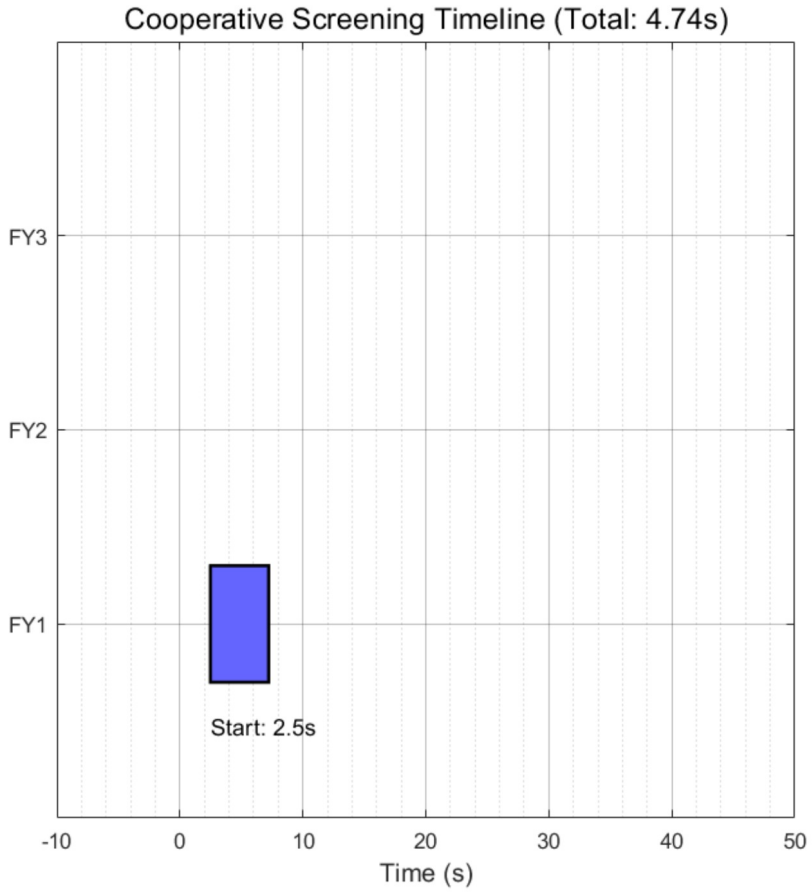

图 3: 问题二结果

6.3 问题三求解：多弹协同策略

在本题中，无人机 FY1 需要连续投放 3 枚烟幕弹以对导弹 M1 实施干扰。与投放一枚烟幕弹不同，三枚烟幕弹的投弹策略涉及时序上的配合，核心问题在对多参数的优化使得烟幕弹的遮蔽时间最大

6.3.1 决策变量

根据题目要求，无人机一旦受领任务，其速度和航向就确定了且不再调整。因此在进行任务决策时我们需要决策以下八个变量：
飞行速度

$$v_{FY_1}$$

飞行航向角

$$\alpha$$

投放时刻 1

$$t_{drop,1}$$

投放时刻 2

$$t_{drop,2}$$

投放时刻 3

$$t_{drop,3}$$

延时起爆时间 1

$$t_{delay,1}$$

延时起爆时间 2

$$t_{delay,2}$$

延时起爆时间 3

$$t_{delay,3}$$

由此定义出决策向量 X:

$$\mathbf{X} = [v_{FY_1}, \alpha, t_{drop,1}, t_{drop,2}, t_{drop,3}, t_{delay,1}, t_{delay,2}, t_{delay,3}]T$$

6.3.2 状态方程更新

基于问题一的运动学模型，第 k 枚烟幕弹的运动状态由下式描述 1. 投放点坐标：

$$P_{drop,k}(\mathbf{X}) = B_0 + \mathbf{v}_{FY1}(v_{FY1}, \alpha) \cdot t_{drop,k}$$

2. 起爆点坐标

$$P_{burst,k}(\mathbf{X}) = P_{drop,k}(\mathbf{X}) + \mathbf{v}_{FY1}(v_{FY1}, \alpha) \cdot t_{delay,k} + \left[0, 0, -\frac{1}{2} g t_{delay,k}^2\right]^T$$

3. 烟幕云团中心轨迹：起爆时刻 $t_{burst,k} = t_{drop,k} + t_{delay,k}$ 后，第 k 个云团中心位置为

$$P_{S,k}(t) = P_{burst,k} - [0, 0, v_{sink} \cdot (t - t_{burst,k})]^T \quad , \quad t \in [t_{burst,k}, t_{burst,k} + 20]$$

6.3.3 目标函数

由于 3 个烟幕团可能同时存在并产生遮挡效果，总遮蔽时间不能简单求和，而应计算时间轴上的并集。定义第 k 枚烟幕弹在 t 时刻的遮蔽指示函数 $I_k(t;\mathbf{X})$:

$$I_k(t;\mathbf{X}) = \begin{cases} 1, & \text{if } D_k(t) \le R_{cloud} \text{ and } t \in [t_{burst,k}, t_{burst,k}+20] \\ 0, & \text{otherwise} \end{cases}$$

其中 $D_k(t)$ 为第 k 个烟幕中心到“导弹-目标”视线的距离（计算方法同问题一）。

系统在 t 时刻的遮蔽状态是所有单个烟幕弹的逻辑或

$$I_{total}(t;\mathbf{X}) = \max\{I_1(t), I_2(t), I_3(t)\}$$

最终优化目标为最大化总有效遮蔽时长 T_{cover}：

$$maxJ(\mathbf{X}) = T_{cover} = \int_0^{T_{end}} I_{total}(t;\mathbf{X})dt$$

6.3.4 约束条件

根据题目和对于模型速度特征的分析，约束条件如下: 除了满足问题二中的飞行与高度约束外，本问必须严格满足多枚弹之间的操作间隔约束：
投弹间隔约束：题目要求每架无人机投放两枚烟幕弹至少间隔 1s。

$$t_{drop,k+1} - t_{drop,k} \ge 1, \quad k = 1, 2$$

基本边界约束：

$$\begin{cases} 70 \le v_{FY_1} \le 140 \\ 0 < t_{drop,1} < t_{drop,2} < t_{drop,3} \le 12 \\ 1 \le t_{delay,k} \le 8, \quad k = 1, 2, 3 \end{cases}$$

高度约束：

$$z_{burst,k}(\mathbf{X}) > 0, \quad k = 1, 2, 3$$

6.3.5 求解算法

针对决策变量维数增加（由 4 维增至 8 维）且包含时序逻辑约束的问题，传统的优化算法容易陷入停滞，因此我们采用引入灾变机制的改进粒子群算法求解。

算法整体架构 不再采用网格扫描，而是直接在 8 维解空间内进行全局寻优。算法通过监测种群的进化状态，动态调整搜索策略，平衡全局探索与局部开发能力.

改进粒子群算法设计

- 粒子编码与约束处理：

  虽然决策变量定义为绝对时刻，但在算法实现中，为了自动满足间隔约束，对时间参数采用增量编码 $(\Delta t_{12}, \Delta t_{23})$ 进行搜索，解码时再转换为绝对时间 t_{drop}。

- 灾变重启机制：
  针对多波次协同容易陷入局部最优的问题，引入" 灾变" 策略
  停滞检测：设置停滞计数器 stagnation_counter，若全局最优解 G_{best} 连续一定代数未更新，则判定算法陷入停滞。
  灾变操作：触发灾变，保留种群中前 10% 的精英粒子（历史最优个体），对其余 90% 的粒子进行完全重置（位置与速度重新随机化）。该机制能有效打破种群的聚集状态，跳出局部极值陷阱，寻找更优的多弹协同策略。

- 适应度函数：对时间轴进行离散化扫描，计算每一时刻系统总遮蔽状态 $I_{sys}(t)$，积分得到总有效时长，取负值作为适应度函数 $F(\mathbf{X}) = -T_{cover}$ 进行极小化求解.

6.3.6 求解结果

根据上述的流程，我们采用灾变机制的改进粒子群算法得到无人机 FY1 航向方向角为 180.67°, 速度大小为 95m/s 时对应的最大有效遮挡时间是 12.54241s 投放 3 枚烟幕弹干扰导弹 M1 的最优策略见下图4，三枚烟幕弹的具体投放策略见下表6.3.6。

表 2: Problem 3 最终策略

参数	数值
最大有效遮蔽时长	12.54241 s
无人机速度	95 m/s
无人机航向	3.1533 rad (180.67°)
烟幕弹时间链	
烟弹 1 投放时刻	0.0000 s
烟弹 1 起爆时刻	0.0000 s
烟弹 2 投放时刻	2.7509 s
烟弹 2 起爆时刻	6.4326 s
烟弹 3 投放时刻	8.0057 s
烟弹 3 起爆时刻	13.3706 s

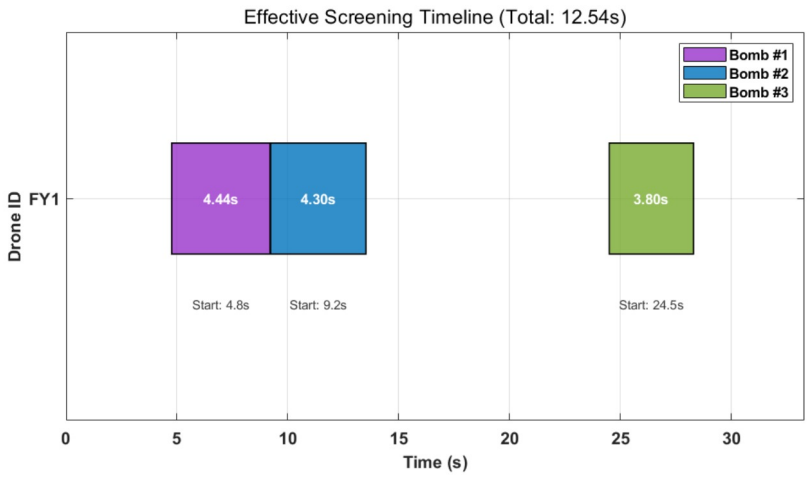

图 4: Problem 3 结果

6.4 问题四求解：多机协同优化

在问题四中，任务变更为 FY_1 FY_2 FY_3 三架无人机共同拦截导弹 $M1$，每架无人机投放一枚烟幕弹，核心问题在于协调三架处于不同空间位置的无人机的投弹策略，使得烟幕弹的遮蔽时间最大.

6.4.1 决策变量

根据题目要求，无人机一旦受领任务，其速度和航向就确定了且不再调整。因此在进行任务决策时我们需要决策 12 维变量：系统包含 3 个独立的运动实体（无人机）。定义 $i \in 1,2,3$ 分别对应 FY_1, FY_2, FY_3。每个实体的决策向量 $\mathbf{u}_i$ 包含 4 个参数：

$$\mathbf{u}_i = [v_i, \alpha_i, t_{drop,i}, t_{delay,i}]$$

全局决策向量 X 维 12 维向量：

$$\mathbf{X} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3] = [v1, 1, tdrop, 1, tdelay, 1, ..., v3, 3, tdrop, 3, tdelay, 3]^T$$

6.4.2 状态方程

各无人机的初始位置 $\mathbf{Pos}_{UAV,i}$ 不同，第 i 架无人机投放的烟幕弹在 t 刻的中心位置 $P_{S,i}(t)$ 更新公式如下投放点坐标：

$$P_{drop,i} = \mathbf{Pos}_{UAV,i} + \mathbf{v}_i(v_i, \alpha_i) \cdot t_{drop,i}$$

起爆点坐标

$$P_{burst,i} = P_{drop,i} + \mathbf{v}_i \cdot t_{delay,i} + \mathbf{G}(t_{delay,i})$$

其中 $\mathbf{G}(t) = [0, 0, -\frac{1}{2}gt^2]^T$ 为重力项。
烟幕云团中心轨迹：

$$P_{S,i}(t) = P_{burst,i} - [0, 0, v_{sink} \cdot (t - t_{burst,i})]^T$$

6.4.3 目标函数

由于 3 个烟幕团可能同时存在并产生遮挡效果，总遮蔽时间不能简单求和，而应计算时间轴上的并集。定义第 i 个烟幕云团对目标的遮蔽指示函数 $I_i(t)$:

$$I_i(t; \mathbf{X}) = \begin{cases} 1, & \text{if } \operatorname{dist}(P_{S,i}(t), \operatorname{Line}_{M1-Target}) \leq R_{cloud} \\ 0, & otherwise \end{cases} 4$$

系统总遮蔽状态为各子状态的逻辑或：

$$Isys(t) = \max_{i=1,2,3} I_i(t)$$

最终优化目标函数定义为最小化代价 $J(\mathbf{X})$：

$$\min J(\mathbf{X}) = -\left(\int_0^{T_{end}} I_{sys}(t)dt \right) + \lambda \cdot \sum_{i=1}^{3} \max(d_{min,i}, R_{cloud})$$

其中：第一项为总遮蔽时长（取负求最小），第二项为惩罚项，$d_{min,i}$ 为第 i 枚烟幕弹在其生命周期内距离视线的最近距离，$\lambda = 0.1$ 为权重系数。当烟幕完全脱靶时，该项通过惩罚距离迫使解向视线靠拢。

6.4.4 约束条件

速度约束：

$$70 \leq v_i \leq 140, \quad \forall i$$

时间窗约束：我们通过物理知识可知，遮蔽的理想状态是烟幕弹之间无缝接力，所以设置时间窗口约束，减小搜索空间：

$$t_{drop,1} \leq 15, \quad t_{drop,2} \leq 35, \quad t_{drop,3} \leq 55$$

起爆延时约束：

$$1 \leq t_{delay,i} \leq 8$$

高度约束：

$$z_{burst,k}(\mathbf{X}) > 0, \quad k = 1, 2, 3$$

6.4.5 求解算法

由于决策变量维数增加为 12 维，而且各维度之间差异较大，所以第四问我们选择鲁棒性强的差分进化算法（DE）求解

算法流程设计

- 种群初始化：生成 NP=200 个个体。针对航向角 α_i，基于各无人机到目标的初始视线角 $\theta_{base,i}$ 进行正态分布初始化，范围控制在 $\theta_{base,i} \pm 45°$ 内，提高搜索效率
- 变异: 采用 DE/rand/1 策略生成变异向量 $\mathbf{V}_g$：

$$\mathbf{V}_{i,g} = \mathbf{X}_{r1,g} + F \cdot (\mathbf{X}_{r2,g} - \mathbf{X}_{r3,g})$$

  其中缩放因子 F 采用自适应策略，随迭代次数从 0.5 线性衰减，平衡全局探索与局部开发。

- 交叉: 对变异向量与目标向量进行二项式交叉，生成试验向量 $\mathbf{U}_{i,g}$。交叉概率 $CR = 0.9$。
- 选择: 计算试验向量与目标向量的适应度，选择适应度更优的个体进入下一代。采用贪婪策略，若试验向量的适应度优于目标向量，则在下一代中保留试验向量：

$$\mathbf{X}_{i,g+1} = \begin{cases} \mathbf{U}_{i,g}, & \text{if } J(\mathbf{U}_{i,g}) < J(\mathbf{X}_{i,g}) \\ \mathbf{X}_{i,g}, & \text{otherwise} \end{cases}$$

- 终止条件: 最大迭代次数 $G_{max} = 800$ 或适应度不再显著提升

6.4.6 求解结果

我们通过差分进化算法（DE）用 matlab 求解得到了最大的总遮蔽有效时间 15s，并且烟幕弹的遮蔽时间衔接良好，提供较好的遮蔽效果。对应的三机协同投放策略如下表6.4.6所示，效果如下图5所示。

表 3: 问题 4 三机协同立体封锁最终策略

机号	初始位置 (m)	速度 (m/s)	航向 (°)	投放时刻 (s)	延时 (s)	起爆时刻 (s)
FY1	(17800, 0, 1800)	78.65	178.42	0.67	2.87	3.55
FY2	(12000, 1400, 1400)	119.78	-129.29	6.69	7.84	14.53
FY3	(6000, -3000, 700)	135.11	106.94	18.00	6.09	24.09

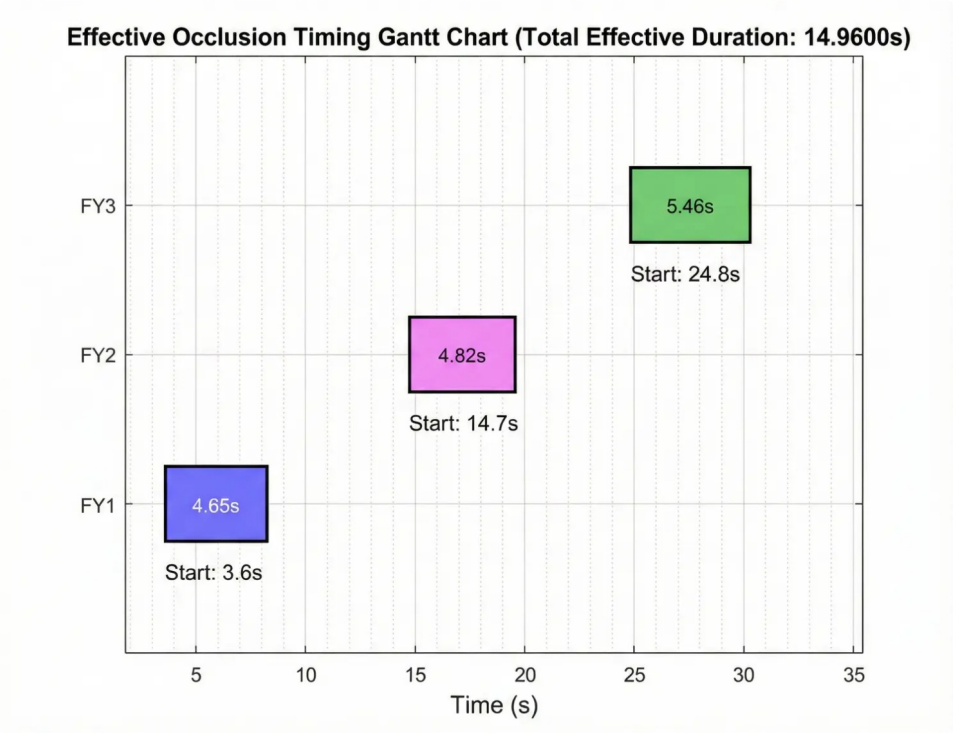

图 5: 问题 4 三机协同立体封锁结果

6.5 问题五求解：多机多目标协同拦截

在问题五中，作战场景扩展为 5 架无人机 $FY_1 - FY_5$ 协同拦截 3 枚来袭导弹 $M_1 - -M_3$。每架无人机最多可携带 3 枚烟幕弹。问题的核心在于如何在复杂的空间分布下，合理分配无人机的飞行策略和弹药资源，使得对所有导弹的总有效遮蔽时间最大化。

6.5.1 决策变量体系

考虑到问题规模显著扩大，且受到前几问优化算法的启发，我们将决策变量分为战略层（飞行参数）和战术层（投弹参数）。战略决策变量：定义全局飞行决策向量，包含 5 架无人机的速度和航向：

$$X_{fly} = [v_1, \alpha_1, v_2, \alpha_2, \ldots, v_5, \alpha_5]^T$$

战术决策变量：对于第 i 架无人机，其携带的第 k 枚烟幕弹 $k = 1, 2, 3$ 的投放参数由算法在内层生成，不直接作为外层优化变量，但需满足物理约束：

$$u_{i,k} = \left[t_{drop}^{(i,k)}, t_{delay}^{(i,k)} \right]$$

6.5.2 状态空间

多导弹运动模型的建立参考第一问，因为每个导弹的飞行特性为一致，只有初始位置不同。多无人机运动模型亦可由之前类比得到。

6.5.3 目标函数

由于存在 3 个独立的来袭目标，系统总效益定义为所有导弹被有效遮蔽时长的总和：

$$S_j(t) = \bigvee_{i=1}^{5} \bigvee_{k=1}^{3} I_{i,k,j}(t)$$

最终优化目标为最大化总效益 J：

$$\max J(X_{fly}) = \sum_{j=1}^{3} \left(\int_0^{T_{end}} S_j(t) dt \right)$$

6.5.4 约束条件

最大载弹量约束：每架无人机投放数量 N<=3；
投弹间隔约束：

$$|t_{drop}^{(i,k)} - {}_{drop}^{(i,k-1)}| \geq 1, \forall k > 1$$

飞行边界约束：

$$70 \leq v_i \leq 140, 0 \leq \alpha_i \leq 2\pi$$

6.5.5 求解算法

针对 5 机 3 弹的高维组合优化问题，依旧采用双层优化结构。外层使用遗传算法搜索无人机的最优飞行参数，内层使用贪心启发式算法快速计算给定航迹下的最优投弹方案。

- 外层：遗传算法全局寻优将 5 架无人机的速度和航向编码为一条染色体。采用实数编码，操作算子采用迷你二进制交叉和多项式变异。实现编码 X_{fly}，则适应度函数为：

$$Fitness(X_{fly}) = -J(X_{fly})$$

- 内层：贪心策略解算投弹方案对于种群中每一个体确定的飞行轨迹，内层算法通过以下步骤计算适应度：

  Step 1: 生成候选打击集遍历第 i 架无人机飞行路径上的离散时间点 t，对于每一个 t，遍历可能的延时起爆时间，检测生成的烟幕云团能否拦截 M1, M2, M3 中的任意一枚。若能拦截，记录该候选方案。

  Step 2: 贪心选择对候选方案集按遮蔽时长降序排列。依次选择遮蔽贡献最大的方案加入最终策略，需满足：该无人机已选弹药数 < 3；与该无人机已选方案的时间间隔 $>= 1s$。

  Step 3: 计算并集效益汇总所有无人机的投弹方案，分别计算对 M1, M2, M3 的遮蔽时间区间并集，求和得到总分。此方法的优势在于将复杂的时序配合问题解耦：外层负责“站位”（规划航线），内层负责“输出”（寻找最佳开火时机），在保证解的质量的同时大幅降低了计算复杂度。

6.5.6 求解结果

通过该算法求解，得到 5 机 3 弹协同拦截 3 枚导弹的最优策略如下表6.5.6所示，总有效遮蔽时间为 28.763s，效果如图6所示。

表 4: 五机协同的详细方案

Drone	BombID	DropTime (s)	ExpTime (s)	Target	CoverDuration (s)	Interval
FY1: 88.19 m/s, 178.35°(3.11 rad)						
FY1	#1	0.00	3.00	M1	3.80	[3.0–6.8]
FY1	#2	1.00	4.00	M	3.80	[4.6–8.4]
FY2: 110.93 m/s, 283.65°(4.95 rad)						
FY2	#1	6.00	9.00	M2	4.00	[9.0–13.0]
FY2	#2	8.00	13.00	M1	0.80	[24.8–25.6]
FY3: 108.05 m/s, 121.72°(2.12 rad)						
FY3	#1	28.00	35.00	M2	3.40	[37.8–41.2]
FY3	#2	25.00	32.00	M3	3.00	[35.6–38.6]
FY3	#3	26.00	33.00	M	1.60	[49.4–51.0]
FY4: 135.80 m/s, 262.62°(4.58 rad)						
FY4	#1	1.00	12.00	M2	4.40	[15.0–19.4]
FY5: 134.90 m/s, 123.38°(2.15 rad)						
FY5	#1	13.00	18.00	M1	4.00	[19.6–23.6]

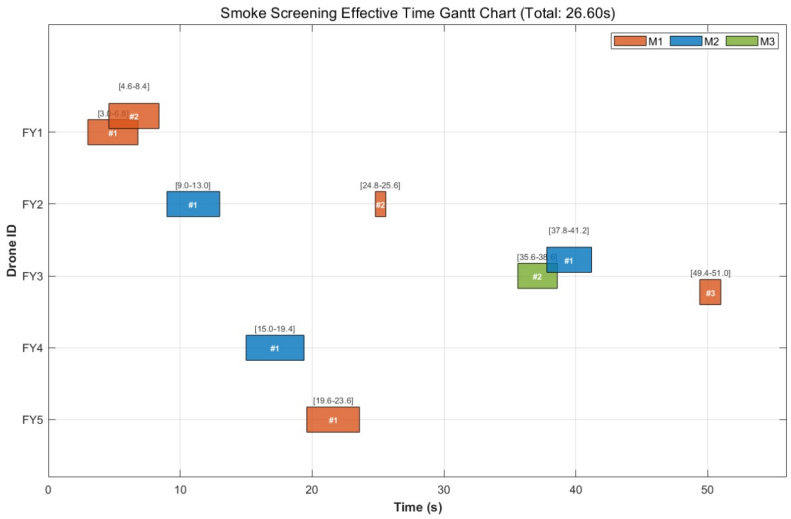

图 6: 五机协同的最优方案

7 模型评价

7.1 模型的优点

- 考虑了烟幕弹的物理沉降特性，符合实际战场环境。
- 算法具有较强的通用性，可扩展至更多数量的无人机集群。

7.2 模型的改进方向

- 当前模型假设导弹不机动，未来可增加导弹末端机动的对抗策略。

参考文献

[1] 作者. 题目. 期刊名, 年份, 卷 (期): 页码.

[2] 张三. 无人机协同作战理论. 北京: 科学出版社, 2020.

[3] CUMCM 组委会. 202x 年全国大学生数学建模竞赛赛题.

附录：主要代码

MATLAB 轨迹仿真代码

Listing 1: Problem 1

```
clc; clear; close all;
%% 1. 参数设置
% 烟幕弹参数
R_smoke = 10;            % 有效遮蔽半径（阈值）
Time_smoke_last = 25;    % 延长一点仿真时间以便看全曲线
V_smoke_sink = 3;        % 烟雾团下沉速度
% 物理常数
g = 9.8;
% 目标位置
Pos_FakeTarget = [0, 0, 0];           % 假目标（导弹飞行的目的地）
Pos_TrueTarget_Center = [0, 200, 5]; % 真目标（我们要保护的对象）
%% 2. 计算烟幕弹起爆初始位置
% FY1 初始状态
Pos_FY1_0 = [17800, 0, 1800];
V_FY1 = 120;
% 时间节点
t_drop = 1.5;            % 投放时刻
t_delay = 3.6;           % 延时时长
t_pop = t_drop + t_delay; % 起爆时刻 (5.1s)
% 1. 投放点位置 (FY1 飞行 1.5s)
Pos_Drop = Pos_FY1_0 + [-1, 0, 0] * V_FY1 * t_drop;
% 2. 起爆点初始位置 (平抛运动 3.6s)
Delta_X = -1 * V_FY1 * t_delay;
Delta_Z = -0.5 * g * t_delay^2;
Pos_Smoke_Init = Pos_Drop + [Delta_X, 0, Delta_Z];
fprintf('烟幕弹起爆时刻: t = %.2f s\n', t_pop);
fprintf('烟幕起爆初始坐标: (%.2f, %.2f, %.2f)\n', Pos_Smoke_Init);
%% 3. 计算有效遮蔽时长（并记录数据用于画图）
% M1 初始状态
Pos_M1_0 = [20000, 0, 2000];
V_M1 = 300;
Vec_M1 = Pos_FakeTarget - Pos_M1_0; % 方向指向假目标
Dist_M1_Total = norm(Vec_M1);
Dir_M1 = Vec_M1 / Dist_M1_Total;
% 时间积分设置
```

```
dt = 0.05; % 稍微调大一点点dt，画图不需要太密集
valid_time = 0;
% --- 初始化数据记录数组 ---
t_log = [];    % 记录时间
dist_log = []; % 记录距离
fprintf('正在模拟动态遮挡过程...\n');
% 从 t=0 开始记录导弹位置，但 t<t_pop 时距离设为 NaN 或 到初始点的距离
Total_Sim_Time = 30; % 总仿真时长 (对应画图横坐标 0-30s)
for t_current = 0 : dt : Total_Sim_Time

    % A. 更新导弹位置 (一直飞)
    dist_flown = V_M1 * t_current;
    if dist_flown >= Dist_M1_Total
        Current_M1_Pos = Pos_FakeTarget; % 已经击中假目标
    else
        Current_M1_Pos = Pos_M1_0 + Dir_M1 * dist_flown;
    end

    % B. 更新烟雾位置 & 计算距离
    % 只有在时间超过起爆时间后，烟雾才存在
    if t_current >= t_pop
        t_relative = t_current - t_pop;

        % 烟雾下沉
        if t_relative <= Time_smoke_last
            Sink_Dist = V_smoke_sink * t_relative;
            Current_Smoke_Pos = Pos_Smoke_Init - [0, 0, Sink_Dist];

            % --- 计算遮挡距离 (核心算法) ---
            P1 = Current_M1_Pos;
            P2 = Pos_TrueTarget_Center;
            Q = Current_Smoke_Pos;

            v = P2 - P1;
            w = Q - P1;
            c1 = dot(w, v);
            c2 = dot(v, v);

            if c1 <= 0
```

```
                dist = norm(Q - P1);
            elseif c2 <= c1
                dist = norm(Q - P2);
            else
                b = c1 / c2;
                Pb = P1 + b * v;
                dist = norm(Q - Pb);
            end

            % 统计有效时间
            if dist <= R_smoke
                valid_time = valid_time + dt;
            end

        else
            % 烟雾消失了
            dist = NaN;
        end
    else
        % 还没起爆，计算导弹到"未来起爆点"的距离
        dist = norm(Pos_Smoke_Init - Current_M1_Pos);
    end

    % C. 记录数据
    t_log = [t_log, t_current];
    dist_log = [dist_log, dist];
end
fprintf('----------------------------------------\n');
fprintf('>>> 最终结果：有效遮蔽时长为：%.4f 秒 <<<\n', valid_time);
%% 4. 绘图（完全复刻之前的样式）
figure('Color', 'w', 'Name', '烟幕干扰弹有效遮蔽时间分析');
hold on; grid on; box on;
% --- A. 准备填充区域数据 ---
% 找到距离小于阈值且时间大于起爆时间的索引
mask = (dist_log <= R_smoke) & (t_log >= t_pop);
if any(mask)
    % 提取需要填充的时间段和距离
    t_fill_data = t_log(mask);
    dist_fill_data = dist_log(mask);

```

```
    % 构造多边形：上边缘是阈值线，下边缘是实际曲线
    % 顶点顺序：[左上 -> 曲线点集 -> 右上]
    X_poly = [t_fill_data, fliplr(t_fill_data)];
    Y_poly = [dist_fill_data, ones(size(dist_fill_data)) * R_smoke];

    % 绘制淡黄色填充
    h_area = fill(X_poly, Y_poly, [1 1 0.8], 'EdgeColor', 'none', '
        FaceAlpha', 0.5);

    % 找到有效结束时间 (遮蔽区间的最后一个时刻)
    t_end_effective = t_fill_data(end);
else
    t_end_effective = t_pop; % 如果没遮住，结束时间就标在起爆点
    h_area = plot(NaN, NaN, 'y'); % 占位符防止legend报错
end
% --- B. 绘制主要曲线 ---
h_dist = plot(t_log, dist_log, 'b-', 'LineWidth', 2);
% --- C. 绘制辅助线 (阈值、起爆、结束) ---
% 1. 阈值红虚线
h_thresh = plot([0, 30], [R_smoke, R_smoke], 'r--', 'LineWidth', 1);
% 2. 起爆时间绿虚线
h_start = xline(t_pop, 'g--', 'LineWidth', 1.5);
% 3. 有效结束绿虚线 (如果有遮蔽的话)
if any(mask)
    h_end = xline(t_end_effective, 'g--', 'LineWidth', 1.5);
    % 标注文字：有效结束
    text(t_end_effective + 0.5, 60, '有效结束', 'Color', 'g', '
        Rotation', 90, 'FontSize', 10);
else
    h_end = plot(NaN,NaN,'g--'); % 占位
end
% --- D. 添加文字标注 ---
% 起爆时间
text(t_pop - 0.8, 60, '起爆时间', 'Color', 'g', 'Rotation', 90, '
    FontSize', 10);
text(25, R_smoke + 2, ['阈值␣' num2str(R_smoke) 'm'], 'Color', 'r', '
    FontSize', 10);
% --- E. 图例与坐标轴设置 ---
title('烟幕干扰弹有效遮蔽时间分析', 'FontSize', 12);
xlabel('时间␣(s)', 'FontSize', 11);
```

```
ylabel('到导弹路径的垂直距离␣(m)', 'FontSize', 11);
xlim([0, 30]);
ylim([0, 70]); % 根据你的图片要求定在70，如果数据很大可以改大
legend([h_dist, h_thresh, h_start, h_end, h_area], ...
    {'距离曲线', '阈值', '起爆时间', '有效结束', '有效遮蔽区间'}, ...
    'Location', 'north', 'Orientation', 'vertical');
hold off;
```

Listing 2: Problem 2

```
function Full_Range_PSO_Sweep()
clc; close all;

%% 1. 场景基础参数
Env.g = 9.8;
Env.Pos_FakeTarget = [0, 0, 0];
Env.Pos_TrueTarget = [0, 200, 5];
Env.Pos_M1_Init = [20000, 0, 2000];
Env.V_M1 = 300;
Env.Vec_M = Env.Pos_FakeTarget - Env.Pos_M1_Init;
Env.Dir_M1 = Env.Vec_M / norm(Env.Vec_M);
Env.Dist_Total_M1 = norm(Env.Vec_M);
Env.Pos_FY1_Init = [17800, 0, 1800];
Env.V_smoke_sink = 3;
Env.R_smoke = 10;
Env.Time_smoke_last = 25;

% === 仿真精度设置 ===
% 在PSO搜索阶段使用 1ms 精度，足够区分优劣且速度可控
Env.dt = 0.001;

%% 2. 扫描设置
v_scan_list = 70 : 0.5 : 140; % 您的要求：70-140，步长0.5
num_scan = length(v_scan_list);

% 结果存储
results_score = zeros(num_scan, 1);
results_params = zeros(num_scan, 3); % [航向，投放，延时]

fprintf('=======================================================\n
    ');
```

```
    fprintf('␣␣␣␣␣␣FY1␣全速域␣PSO␣暴力扫描␣(Grid-PSO)\n');
    fprintf('␣␣␣␣␣␣速度:␣70-140␣m/s␣|␣步长:␣0.5␣m/s␣|␣核心:␣PSO\n');
    fprintf('=========================================================\n
        ');

    try
        if isempty(gcp('nocreate')), parpool; end
        fprintf('>>>␣并行计算池已就绪，全速运算中...\n');
    catch
        fprintf('>>>␣并行启动失败，切换为单核运算...\n');
    end

    timer_start = tic;

    %% 3. 并行扫描循环
    parfor i = 1 : num_scan
        v_curr = v_scan_list(i);

        % --- PSO 配置 ---
        % 在固定的速度下，寻找最佳的 [航向，投放，延时]
        % 变量维度: 3
        % 搜索范围:
        %   航向: 基准 +/- 30度
        %   投放: 0 - 12秒
        %   延时: 1 - 8秒

        % 计算基准航向
        Vec_Base = Env.Pos_TrueTarget - Env.Pos_FY1_Init;
        Base_Angle = rad2deg(atan2(Vec_Base(2), Vec_Base(1)));

        LB = [Base_Angle-30, 0, 1.0];
        UB = [Base_Angle+30, 12, 8.0];

        % 运行内置的微型 PSO
        % 粒子数: 30 (对3维问题足够)
        % 迭代: 50
        [best_x, best_val] = Run_Micro_PSO(v_curr, LB, UB, Env);

        % 记录
        results_score(i) = best_val;
```

```
70          results_params(i, :) = best_x;
71
72          if mod(i, 10) == 0
73              fprintf('␣␣␣␣...完成速度␣%.1f␣m/s␣(最佳遮蔽:␣%.4fs)\n',
                    v_curr, best_val);
74          end
75      end
76
77      total_time = toc(timer_start);
78      fprintf('\n>>>␣扫描完成！总耗时:␣%.2f␣秒\n', total_time);
79
80      %% 4. 结果分析与可视化
81      [max_score, idx_best] = max(results_score);
82      best_v = v_scan_list(idx_best);
83      best_p = results_params(idx_best, :);
84
85      fprintf('\n
            ========================================================\n');
86      fprintf('␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣全局最优解␣(Global␣Optimum)\n');
87      fprintf('========================================================\n
            ');
88      fprintf('>>>␣冠军速度:␣%.1f␣m/s\n', best_v);
89      fprintf('>>>␣极限遮蔽:␣%.5f␣秒\n', max_score);
90      fprintf('>>>␣战术参数:␣航向␣%.2f度␣|␣投放␣%.4fs␣|␣延时␣%.4fs\n',
            best_p(1), best_p(2), best_p(3));
91
92      % --- 绘制“速度-性能”全景图 ---
93      figure('Color', 'w', 'Name', 'Speed␣vs␣Shield␣Time');
94      plot(v_scan_list, results_score, 'b-', 'LineWidth', 1.5); hold on
            ;
95      scatter(best_v, max_score, 100, 'r', 'filled', 'p');
96
97      % 标记关键点
98      xlabel('飞行速度␣(m/s)', 'FontSize', 12);
99      ylabel('最大有效遮蔽时长␣(s)', 'FontSize', 12);
100     title('全速域战术性能分布图␣(PSO搜索)', 'FontSize', 14);
101     grid on;
102
103     % 自动寻找局部峰值（比如高速区的峰值）
104     [pks, locs] = findpeaks(results_score);
```

```
    if ~isempty(pks)
        % 标记除了全局最优外的另一个显著峰值（如果有）
        sorted_pks = sort(pks, 'descend');
        if length(sorted_pks) > 1 && sorted_pks(2) > 4.0
            second_pk = sorted_pks(2);
            % 找到对应速度
            idx_sec = find(results_score == second_pk, 1);
            sec_v = v_scan_list(idx_sec);
            if abs(sec_v - best_v) > 10 % 确保两个峰隔得够远
                plot(sec_v, second_pk, 'ko', 'MarkerFaceColor', 'y');
                text(sec_v, second_pk+0.1, sprintf('次优解\n%.1fm/s',
                    sec_v), 'HorizontalAlignment', 'center');
            end
        end
    end

    text(best_v, max_score + 0.15, sprintf('全局最优\nV=%.1f\nT=%.4fs
        ', best_v, max_score), ...
        'Color', 'r', 'HorizontalAlignment', 'center', 'FontWeight',
            'bold');

    %% 5. 最终验证（使用 0.0001s 航天级精度复核冠军参数）
    fprintf('\n>>> 正在使用 0.1ms 精度复核冠军方案...\n');
    final_check_score = Tactical_Sim_Engine(best_v, best_p(1), best_p
        (2), best_p(3), 0.0001, Env);
    fprintf('>>> 最终复核得分: %.6f 秒\n', -final_check_score); % 注
        意引擎返回负值

    % 绘制冠军轨迹
    Plot_3D_Final(best_v, best_p, Env);
end

%% --- 内部微型 PSO 求解器 ---
function [best_pos, best_val] = Run_Micro_PSO(v, lb, ub, Env)
    % 粒子群参数
    n_part = 30;
    n_iter = 50;
    w = 0.6; c1 = 1.5; c2 = 1.5;
    n_vars = 3;

```

```
    % 初始化
    pos = repmat(lb, n_part, 1) + rand(n_part, n_vars) .* repmat(ub-
        lb, n_part, 1);

    % [种子注入] 针对不同速度段注入经验值，加速收敛
    if v < 90
        pos(1,:) = [176.88, 0.01, 2.5]; % 低速经验
    else
        pos(1,:) = [178.46, 0.01, 3.3]; % 高速经验
    end

    vel = zeros(n_part, n_vars);

    pbest_pos = pos;
    pbest_val = zeros(n_part, 1);

    gbest_pos = zeros(1, n_vars);
    gbest_val = -1e9; % 初始化为很小的值

    % 评估初始种群
    for i = 1:n_part
        % 注意：Sim引擎返回的是负值(为了fminsearch)，这里我们取相反数
            变回正值方便比较
        val = -Tactical_Sim_Engine(v, pos(i,1), pos(i,2), pos(i,3),
            Env.dt, Env);
        pbest_val(i) = val;
        if val > gbest_val
            gbest_val = val;
            gbest_pos = pos(i,:);
        end
    end

    % 迭代
    for t = 1:n_iter
        for i = 1:n_part
            r1 = rand(1, n_vars); r2 = rand(1, n_vars);
            vel(i,:) = w*vel(i,:) + c1*r1.*(pbest_pos(i,:)-pos(i,:))
                + c2*r2.*(gbest_pos-pos(i,:));
            pos(i,:) = pos(i,:) + vel(i,:);

```

```
            % 边界限制
            pos(i,:) = max(pos(i,:), lb);
            pos(i,:) = min(pos(i,:), ub);

            val = -Tactical_Sim_Engine(v, pos(i,1), pos(i,2), pos(i
                ,3), Env.dt, Env);

            if val > pbest_val(i)
                pbest_val(i) = val;
                pbest_pos(i,:) = pos(i,:);
            end
            if val > gbest_val
                gbest_val = val;
                gbest_pos = pos(i,:);
            end
        end
    end

    best_pos = gbest_pos;
    best_val = gbest_val;
end

%% --- 仿真核函数 (Cost Function) ---
function score = Tactical_Sim_Engine(v, a, td, ty, dt, Env)
    % 返回负值用于最小化，正值代表遮蔽时长

    % 几何解算
    ang_rad = deg2rad(a);
    Dir_Vec = [cos(ang_rad), sin(ang_rad), 0];
    Vel_Vec = Dir_Vec * v;

    P_Drop = Env.Pos_FY1_Init + Vel_Vec * td;
    P_Pop = P_Drop + Vel_Vec * ty;
    P_Pop(3) = P_Pop(3) - 0.5 * Env.g * ty^2;

    % 剪枝：地下
    if P_Pop(3) < 0
        score = 0; return;
    end

```

```
    t_pop = td + ty;
    t_start = max(0, t_pop);
    t_end = min(Env.Dist_Total_M1/Env.V_M1, t_pop + Env.
        Time_smoke_last);

    if t_start >= t_end
        score = 0; return;
    end

    % 矢量化计算
    t_vec = t_start : dt : t_end;
    if isempty(t_vec), score=0; return; end

    d_m_vec = Env.V_M1 * t_vec;
    P_M_mat = Env.Pos_M1_Init' + Env.Dir_M1' * d_m_vec;
    P_Smk_mat = P_Pop' - [0;0;Env.V_smoke_sink] * (t_vec - t_pop);

    V_LOS_mat = Env.Pos_TrueTarget' - P_M_mat;
    W_mat = P_Smk_mat - P_M_mat;

    c1 = sum(W_mat .* V_LOS_mat, 1);
    c2 = sum(V_LOS_mat .* V_LOS_mat, 1);

    b = c1 ./ c2;

    % 垂足修正
    Pb = P_M_mat + V_LOS_mat .* b;
    idx_less = b < 0; if any(idx_less), Pb(:, idx_less) = P_M_mat(:,
        idx_less); end
    idx_more = b > 1; if any(idx_more), Pb(:, idx_more) = repmat(Env.
        Pos_TrueTarget', 1, sum(idx_more)); end

    dists_sq = sum((P_Smk_mat - Pb).^2, 1);

    count = sum(dists_sq <= Env.R_smoke^2);
    score = -(count * dt); % 返回负值
end

%% --- 绘图 ---
function Plot_3D_Final(v, params, Env)
```

```
    a=params(1); td=params(2); ty=params(3);
    ang_rad = deg2rad(a);
    Vel_Vec = [cos(ang_rad), sin(ang_rad), 0] * v;
    P_Drop = Env.Pos_FY1_Init + Vel_Vec * td;
    P_Pop = P_Drop + Vel_Vec * ty;
    P_Pop(3) = P_Pop(3) - 0.5 * Env.g * ty^2;

    figure('Color','w'); hold on; grid on; axis equal; view([-20,
        25]);
    plot3(Env.Pos_TrueTarget(1), Env.Pos_TrueTarget(2), Env.
        Pos_TrueTarget(3), 'go', 'MarkerSize', 10, 'MarkerFaceColor','
        g');
    plot3(Env.Pos_FakeTarget(1), Env.Pos_FakeTarget(2), Env.
        Pos_FakeTarget(3), 'kp', 'MarkerSize', 10, 'MarkerFaceColor','
        k');
    plot3([Env.Pos_FY1_Init(1) P_Drop(1)],[Env.Pos_FY1_Init(2) P_Drop
        (2)],[Env.Pos_FY1_Init(3) P_Drop(3)],'b-');
    plot3([P_Drop(1) P_Pop(1)],[P_Drop(2) P_Pop(2)],[P_Drop(3) P_Pop
        (3)],'k:');
    plot3(P_Pop(1), P_Pop(2), P_Pop(3), 'rh', 'MarkerFaceColor','r','
        MarkerSize',14);
    line([Env.Pos_M1_Init(1) Env.Pos_TrueTarget(1)], [Env.Pos_M1_Init
        (2) Env.Pos_TrueTarget(2)], [Env.Pos_M1_Init(3) Env.
        Pos_TrueTarget(3)], 'Color', [0.7 0.7 0.7], 'LineStyle', '--')
        ;
    title(['Global␣Optimal:␣V=' num2str(v) '␣m/s']);
    legend('True','Fake','FY1','Bomb','Pop','LOS');
end
```

Listing 3: Problem 3

```
clc; clear; close all;

%% 1. 参数初始化
% 粒子群算法参数
n_particles = 200;       % 增加粒子数以覆盖更大的搜索空间(8维)
n_iterations = 300;      % 增加迭代次数
c1 = 2.0; c2 = 2.0;
w_max = 0.9; w_min = 0.4;

% 变量设计 (8维):
```

```
% x(1): 速度 v (70-140)
% x(2): 航向 theta (0-2pi)
% x(3): 第1枚投放时刻 t1 (0-60)
% x(4): 第1枚与第2枚的时间间隔 dt12 (1-30) -> 保证了间隔>=1s
% x(5): 第2枚与第3枚的时间间隔 dt23 (1-30) -> 保证了间隔>=1s
% x(6): 第1枚起爆延时 delay1 (0-15)
% x(7): 第2枚起爆延时 delay2 (0-15)
% x(8): 第3枚起爆延时 delay3 (0-15)

lb = [70,  0,    0,  1,  1,  0,  0,  0];
ub = [140, 2*pi, 60, 20, 20, 15, 15, 15];

% 速度限制
v_max = 0.15 * (ub - lb);
v_min = -v_max;

%% 2. 种群初始化
particles = repmat(lb, n_particles, 1) + rand(n_particles, 8) .* (ub
    - lb);
velocities = zeros(n_particles, 8);

pbest_pos = particles;
pbest_val = inf(n_particles, 1);
gbest_pos = zeros(1, 8);
gbest_val = inf;

stagnation_counter = 0; % 停滞计数器，用于判断是否陷入局部最优

%% 3. PSO 主循环
disp('开始多烟幕弹协同优化␣(引入灾变机制防止局部最优)...');
tic;

for iter = 1:n_iterations
    w = w_max - (w_max - w_min) * iter / n_iterations;

    current_gbest_improved = false;

    for i = 1:n_particles
        current_x = particles(i, :);
        current_x(1) = round(current_x(1)); % 速度取整
```

```
50
51          % 边界处理
52          current_x = max(current_x, lb);
53          current_x = min(current_x, ub);
54
55          % 计算适应度（核心修改：支持3枚弹）
56          fitness = calculate_fitness_multi(current_x);
57
58          % 更新个体最优
59          if fitness < pbest_val(i)
60              pbest_val(i) = fitness;
61              pbest_pos(i, :) = current_x;
62          end
63
64          % 更新全局最优
65          if fitness < gbest_val
66              gbest_val = fitness;
67              gbest_pos = current_x;
68              current_gbest_improved = true;
69          end
70      end
71
72      % --- 灾变机制（防止陷入局部最优） ---
73      if current_gbest_improved
74          stagnation_counter = 0;
75      else
76          stagnation_counter = stagnation_counter + 1;
77      end
78
79      % 如果连续20代没有进步，触发“小灾变”：重置部分粒子
80      if stagnation_counter > 20 && iter < n_iterations - 50
81          disp(['␣␣␣␣>>␣检测到局部最优停滞，触发灾变重置␣(Iter␣'
                num2str(iter) ')']);
82          % 保留前 10% 的精英，重置其余 90%
83          n_keep = round(n_particles * 0.1);
84          [~, sorted_idx] = sort(pbest_val);
85          for k = n_keep+1 : n_particles
86              idx = sorted_idx(k);
87              particles(idx, :) = lb + rand(1, 8) .* (ub - lb);
88              velocities(idx, :) = zeros(1, 8);
```

```
            pbest_val(idx) = inf; % 重置历史记忆
        end
        stagnation_counter = 0;
    end

    % 粒子更新公式
    for i = 1:n_particles
        r1 = rand(1, 8);
        r2 = rand(1, 8);

        velocities(i,:) = w * velocities(i,:) ...
            + c1 * r1 .* (pbest_pos(i,:) - particles(i,:)) ...
            + c2 * r2 .* (gbest_pos - particles(i,:));

        velocities(i,:) = max(min(velocities(i,:), v_max), v_min);
        particles(i,:) = particles(i,:) + velocities(i,:);

        % 变异操作 (5%概率)
        if rand < 0.05
            dim = randi(8);
            particles(i, dim) = lb(dim) + rand * (ub(dim) - lb(dim));
        end
        particles(i,:) = max(min(particles(i,:), ub), lb);
    end

    % 进度显示
    if mod(iter, 10) == 0 || iter == 1
        real_time = 0;
        if gbest_val < -1000, real_time = -(gbest_val + 10000); end
        fprintf('迭代 %d/%d, 当前最佳遮蔽时长: %.5f 秒 (停滞: %d)\n',
            ...
            iter, n_iterations, real_time, stagnation_counter);
    end
end
toc;

%% 4. 结果解析与输出
best_params = gbest_pos;
best_params(1) = round(best_params(1));

```

```
% 解码时间参数
t_drop1 = best_params(3);
t_drop2 = t_drop1 + best_params(4);
t_drop3 = t_drop2 + best_params(5);
t_delay1 = best_params(6);
t_delay2 = best_params(7);
t_delay3 = best_params(8);

% 最终高精度验证
final_time = calculate_masking_pure_multi(best_params, 0.00001);

fprintf('\n----------------␣最终优化结果␣(Problem␣3)␣
    ----------------\n');
fprintf('最大有效遮蔽时长:␣%.5f␣秒\n', final_time);
fprintf('\n策略详情: \n');
fprintf('无人机速度:␣%d␣m/s\n', best_params(1));
fprintf('无人机航向:␣%.4f␣rad␣(%.2f␣度)\n', best_params(2), rad2deg(
    best_params(2)));
fprintf('\n[烟弹1]␣投放时刻:␣%6.4f␣s,␣起爆延时:␣%6.4f␣s,␣起爆时刻:␣
    %6.4f␣s\n', t_drop1, t_delay1, t_drop1+t_delay1);
fprintf('[烟弹2]␣投放时刻:␣%6.4f␣s,␣起爆延时:␣%6.4f␣s,␣起爆时刻:␣%6.4
    f␣s\n', t_drop2, t_delay2, t_drop2+t_delay2);
fprintf('[烟弹3]␣投放时刻:␣%6.4f␣s,␣起爆延时:␣%6.4f␣s,␣起爆时刻:␣%6.4
    f␣s\n', t_drop3, t_delay3, t_drop3+t_delay3);
fprintf('注: 投放间隔分别为␣%.4f␣s␣和␣%.4f␣s␣(均满足>=1s约束)\n',
    best_params(4), best_params(5));


%% ================= 子函数定义区域 =================

function fitness = calculate_fitness_multi(x)
    % 1. 解码所有参数
    v = x(1); theta = x(2);
    t_d1 = x(3);
    t_d2 = t_d1 + x(4); % 确保间隔
    t_d3 = t_d2 + x(5); % 确保间隔
    del1 = x(6); del2 = x(7); del3 = x(8);

    % 2. 计算3枚弹的运动学参数
    [init_pos_m1, v_vec_m1, target_pos] = get_env_params(v, theta);
```

```
    [exp1, pos1] = get_bomb_kinematics(v, theta, t_d1, del1);
    [exp2, pos2] = get_bomb_kinematics(v, theta, t_d2, del2);
    [exp3, pos3] = get_bomb_kinematics(v, theta, t_d3, del3);

    % 物理约束：任何一枚在地下爆炸都给予惩罚
    if pos1(3)<0 || pos2(3)<0 || pos3(3)<0
        fitness = 1e6; return;
    end

    % 3. 确定仿真时间范围
    smoke_dur = 20;
    t_hit = norm(init_pos_m1) / 300;

    % 整个遮蔽过程是3个区间的并集，我们取最早起爆到最晚结束
    t_start_global = min([exp1, exp2, exp3]);
    t_end_global = min(max([exp1, exp2, exp3]) + smoke_dur, t_hit);

    if t_start_global >= t_end_global
        fitness = 1e5; return;
    end

    % --- 智能扫描策略 ---

    % A. 粗扫描 (0.1s)
    dt_coarse = 0.1;
    T_coarse = t_start_global : dt_coarse : t_end_global;
    if isempty(T_coarse), fitness = 1e5; return; end

    min_dist_global = inf;
    potential_times = [];

    % 预计算导弹位置
    Pm_c = init_pos_m1 + v_vec_m1 .* T_coarse(:);
    Pt = target_pos;

    for k = 1:length(T_coarse)
        t = T_coarse(k);
        pm = Pm_c(k,:);
        vec_line = Pt - pm;
```

```
        norm_line = norm(vec_line);

        dist_min_k = inf;

        % 检查3个烟雾团
        clouds = [exp1, pos1; exp2, pos2; exp3, pos3]; % 3x4 matrix
        for m = 1:3
            t_exp = clouds(m, 1);
            p_exp = clouds(m, 2:4);

            % 只有在烟雾存续期内才计算
            if t >= t_exp && t <= t_exp + smoke_dur
                ps = p_exp;
                ps(3) = ps(3) - 3 * (t - t_exp); % 下沉

                vec_point = ps - pm;
                d = norm(cross(vec_point, vec_line)) / norm_line;
                if d < dist_min_k, dist_min_k = d; end
            end
        end

        if dist_min_k < min_dist_global, min_dist_global = dist_min_k
            ; end
        if dist_min_k < 18, potential_times = [potential_times; t];
            end
    end

    % B. 精细计算
    if isempty(potential_times)
        fitness = min_dist_global;
    else
        dt_fine = 0.0001;
        window = 0.06;
        all_fine_times = [];

        for t_center = potential_times'
            t_sub = (max(t_start_global, t_center - window) : dt_fine
                : min(t_end_global, t_center + window))';
            t_sub = round(t_sub / dt_fine) * dt_fine; % 网格对齐
            all_fine_times = [all_fine_times; t_sub];
```

```
        end

        if isempty(all_fine_times), fitness = min_dist_global; return; end
        T_fine = unique(all_fine_times);

        Pm_f = init_pos_m1 + v_vec_m1 .* T_fine;

        mask_count = 0;

        % 向量化比较麻烦，这里用循环处理这3个球
        % 为了加速，把3个球的信息展开
        exp_times = [exp1, exp2, exp3];
        pos_starts = [pos1; pos2; pos3];

        for j = 1:length(T_fine)
            t = T_fine(j);
            pm = Pm_f(j,:);
            vec_line = Pt - pm;
            norm_line = norm(vec_line);

            is_masked = false;

            % 遍历3个球
            for m = 1:3
                if t >= exp_times(m) && t <= exp_times(m) + smoke_dur
                    ps = pos_starts(m, :);
                    ps(3) = ps(3) - 3 * (t - exp_times(m));

                    vec_point = ps - pm;
                    % 快速判断距离平方
                    cp = cross(vec_point, vec_line);
                    d2 = sum(cp.^2) / (norm_line^2);

                    if d2 <= 100 % 10^2
                        is_masked = true;
                        break; % 只要被任意一个遮住就算遮住
                    end
                end
            end
```

```
278
279             if is_masked
280                 mask_count = mask_count + 1;
281             end
282         end
283
284         fitness = -(mask_count * dt_fine) - 10000;
285     end
286 end
287
288 function mask_time = calculate_masking_pure_multi(x, dt)
289     % 解码
290     v = x(1); theta = x(2);
291     t_d1 = x(3); t_d2 = t_d1 + x(4); t_d3 = t_d2 + x(5);
292     del1 = x(6); del2 = x(7); del3 = x(8);
293
294     [init_pos_m1, v_vec_m1, target_pos] = get_env_params(v, theta);
295     [exp1, pos1] = get_bomb_kinematics(v, theta, t_d1, del1);
296     [exp2, pos2] = get_bomb_kinematics(v, theta, t_d2, del2);
297     [exp3, pos3] = get_bomb_kinematics(v, theta, t_d3, del3);
298
299     smoke_dur = 20;
300     t_hit = norm(init_pos_m1) / 300;
301     t_start = min([exp1, exp2, exp3]);
302     t_end = min(max([exp1, exp2, exp3]) + smoke_dur, t_hit);
303
304     if t_start >= t_end, mask_time=0; return; end
305
306     T = t_start : dt : t_end;
307     count = 0;
308
309     exp_times = [exp1, exp2, exp3];
310     pos_starts = [pos1; pos2; pos3];
311
312     for k = 1:length(T)
313         t = T(k);
314         pm = init_pos_m1 + v_vec_m1 * t;
315         vec_line = target_pos - pm;
316         norm_line = norm(vec_line);
317
```

```
        masked = false;
        for m = 1:3
            if t >= exp_times(m) && t <= exp_times(m) + smoke_dur
                ps = pos_starts(m, :);
                ps(3) = ps(3) - 3 * (t - exp_times(m));

                vec_point = ps - pm;
                d = norm(cross(vec_point, vec_line)) / norm_line;
                if d <= 10
                    masked = true; break;
                end
            end
        end
        if masked, count = count + 1; end
    end
    mask_time = count * dt;
end

function [init_pos_m1, v_vec_m1, target_pos] = get_env_params(v,
   theta)
    init_pos_m1 = [20000, 0, 2000];
    fake_target = [0, 0, 0];
    dir_m1 = (fake_target - init_pos_m1) / norm(fake_target -
       init_pos_m1);
    v_vec_m1 = dir_m1 * 300;
    target_pos = [0, 200, 0];
end

function [t_explode, explode_pos] = get_bomb_kinematics(v_mag, theta,
    t_drop, t_delay)
    init_pos_uav = [17800, 0, 1800];
    v_vec_uav = [v_mag * cos(theta), v_mag * sin(theta), 0];
    drop_pos = init_pos_uav + v_vec_uav * t_drop;

    g = 9.8;
    explode_pos = drop_pos;
    explode_pos(1) = drop_pos(1) + v_vec_uav(1) * t_delay;
    explode_pos(2) = drop_pos(2) + v_vec_uav(2) * t_delay;
    explode_pos(3) = drop_pos(3) - 0.5 * g * t_delay^2;

```

```
355     t_explode = t_drop + t_delay;
356 end
```

Listing 4: Problem 4

```
1  function Problem4_MultiUAV_Strict()
2      clc; close all;
3
4      %% 1. 全局环境参数 (Env)
5      Env.g = 9.8;
6      Env.Pos_True = [0, 200, 0];
7      Env.Pos_M1   = [20000, 0, 2000];
8      Env.V_M1     = 300;
9      Env.Vec_M    = [0, 0, 0] - Env.Pos_M1;
10     Env.Dist_M1  = norm(Env.Vec_M);
11     Env.Dir_M1   = Env.Vec_M / Env.Dist_M1;
12     Env.V_sink   = 3;
13     Env.R_smk    = 10;
14     Env.T_last   = 20;
15
16     Env.Pos_UAVs = [
17         17800, 0,    1800;  % FY1
18         12000, 1400, 1400;  % FY2
19         6000, -3000, 700    % FY3
20     ];
21
22     %% 2. 优化参数设置 (已修正速度约束)
23
24     % 估算基准航向
25     Base_Ang1 = rad2deg(atan2(200 - 0,    0 - 17800));
26     Base_Ang2 = rad2deg(atan2(200 - 1400, 0 - 12000));
27     Base_Ang3 = rad2deg(atan2(200 + 3000, 0 - 6000));
28
29     % === [修正点]: 将速度下界从 60 改为 70 ===
30     % 变量顺序: [V, Ang, T_drop, T_delay] * 3架
31     LB = [70, Base_Ang1-45, 0, 1,   70, Base_Ang2-45, 0, 1,   70,
           Base_Ang3-45, 0, 1];
32     UB = [140, Base_Ang1+45, 15, 8, 140, Base_Ang2+45, 35, 8, 140,
           Base_Ang3+45, 55, 8];
33
34     de_opts.NP = 200;
```

```
    de_opts.MaxIter = 800;
    de_opts.F = 0.5;
    de_opts.CR = 0.9;

    fprintf('============================================================\n');
    fprintf('      问题4: 三机协同立体封锁(速度修正版70-140)\n');
    fprintf('============================================================\n');

    try, if isempty(gcp('nocreate')), parpool; end; end

    CostFunc = @(x) CostFunc_3UAV(x, Env);

    tic;
    [best_x, best_val, curve] = Run_DE_3UAV(CostFunc, LB, UB, de_opts);
    total_time = toc;

    %% 3. 结果解析与输出
    [~, real_shield_time] = CostFunc_3UAV(best_x, Env);
    Res = Parse_Params(best_x);

    fprintf('\n>>> 优化完成! 耗时: %.2f 秒\n', total_time);
    fprintf('>>>  最终最大遮蔽时长: %.4f 秒\n', real_shield_time);

    fprintf('\n[战术指令表]\n');
    fprintf('| 机号 | 初始位置(X,Y,Z) | 速度(m/s) | 航向(deg) | 投放(s) | 延时(s) | 起爆(s) |\n');
    fprintf('|------|-----------------|-----------|-----------|---------|---------|---------|\n');
    fprintf('| FY1  | (%5d,%5d,%4d) | %9.2f | %9.2f | %7.2f | %7.2f | %7.2f |\n', ...
        Env.Pos_UAVs(1,:), Res(1).v, Res(1).a, Res(1).td, Res(1).ty, Res(1).tp);
    fprintf('| FY2  | (%5d,%5d,%4d) | %9.2f | %9.2f | %7.2f | %7.2f | %7.2f |\n', ...
        Env.Pos_UAVs(2,:), Res(2).v, Res(2).a, Res(2).td, Res(2).ty, Res(2).tp);
```

```
    fprintf('| FY3 | (%5d,%5d,%4d) | %9.2f | %9.2f | %7.2f | %7.2f | %7.2f |\n', ...
        Env.Pos_UAVs(3,:), Res(3).v, Res(3).a, Res(3).td, Res(3).ty, Res(3).tp);

    %% 4. 绘图
    Plot_3UAV_Scenario(Res, real_shield_time, curve, Env);
    Save_To_Excel(Res, 'result2.xlsx');
end

%% ========================================================================
%  局部函数库 (保持不变, 仅 CostFunc_3UAV 内部也加一道保险)
% ========================================================================

function Res = Parse_Params(x)
    for i = 1:3
        idx = (i-1)*4;
        Res(i).v  = x(idx+1);
        Res(i).a  = x(idx+2);
        Res(i).td = x(idx+3);
        Res(i).ty = x(idx+4);
        Res(i).tp = Res(i).td + Res(i).ty;
    end
end

function [score, pure_time] = CostFunc_3UAV(x, Env)
    Res = Parse_Params(x);

    % --- [保险修正]: 并在内部逻辑中再次检查边界 ---
    for i=1:3
        if Res(i).v < 70 || Res(i).v > 140 % 严格限制 70-140
            score = 1e6; pure_time = 0; return;
        end
    end

    % ... (其余物理计算逻辑与之前完全一致, 此处省略以节省篇幅) ...
    % 您之前复制的 CostFunc_3UAV 代码逻辑在这里保持不变

    % ---------------- 粘贴之前的 CostFunc_3UAV 逻辑 ----------------
    P_Pop = zeros(3, 3);
```

```
    Times_Pop = zeros(3, 1);
    for i = 1:3
        ang_rad = deg2rad(Res(i).a);
        Vel_Vec = [cos(ang_rad), sin(ang_rad), 0] * Res(i).v;
        P_Drop = Env.Pos_UAVs(i,:) + Vel_Vec * Res(i).td;
        Disp = Vel_Vec * Res(i).ty;
        Disp(3) = Disp(3) - 0.5 * 9.8 * Res(i).ty^2;
        P_Pop(i,:) = P_Drop + Disp;
        Times_Pop(i) = Res(i).tp;
        if P_Pop(i,3) < 0, score = 1e5; pure_time = 0; return; end
    end

    t_start = min(Times_Pop);
    t_end = min(Env.Dist_M1/Env.V_M1, max(Times_Pop) + Env.T_last);
    if t_start >= t_end, score = 1e5; pure_time = 0; return; end

    Target_Points = [0, 200, 5; 0, 200, 10; 0, 200, 0; -7, 200, 5; 7,
        200, 5];
    num_pts = 5;
    dt = 0.05; t_vec = t_start : dt : t_end;
    total_coverage = 0;
    penalty_dist = 0;
    min_dists = [1e9, 1e9, 1e9];

    for t = t_vec
        P_M = Env.Pos_M1 + Env.Dir_M1 * (Env.V_M1 * t);
        blocked_pts_count = 0;
        for p = 1:num_pts
            TP = Target_Points(p,:);
            v_los = TP - P_M;
            len_los_sq = sum(v_los.^2);
            is_pt_blocked = false;
            for k = 1:3
                if t >= Times_Pop(k) && t <= Times_Pop(k) + Env.
                   T_last
                    P_Smk = P_Pop(k,:) - [0, 0, Env.V_sink * (t -
                       Times_Pop(k))];
                    w_vec = P_Smk - P_M;
                    c1 = dot(w_vec, v_los);
                    if c1 > 0
```

```
                        b = max(0, min(1, c1 / len_los_sq));
                        Pb = P_M + v_los * b;
                        d_sq = sum((P_Smk - Pb).^2);
                        dist = sqrt(d_sq);
                        if p == 1 && dist < min_dists(k), min_dists(k
                            ) = dist; end
                        if d_sq <= Env.R_smk^2, is_pt_blocked = true;
                             end
                    end
                end
            end
            if is_pt_blocked, blocked_pts_count = blocked_pts_count +
                1; end
        end
        total_coverage = total_coverage + (blocked_pts_count /
            num_pts) * dt;
    end
    pure_time = total_coverage;
    for k=1:3, penalty_dist = penalty_dist + max(0, min_dists(k) -
        Env.R_smk); end
    score = -total_coverage + 0.1 * penalty_dist;
end

%% --- DE 求解器 (修正种子速度) ---
function [best_mem, best_val, curve] = Run_DE_3UAV(cost_func, lb, ub,
    opts)
    NP = opts.NP; D = length(lb);
    pop = repmat(lb, NP, 1) + rand(NP, D) .* repmat(ub-lb, NP, 1);

    % === [修正点]: 种子的速度也必须符合 70-140 约束 ===
    % FY1 修正为 73 (低速流最优), FY2/FY3 为 120 (高速流)
    pop(1,:) = [73, 180, 0, 3,   120, 190, 15, 4,   120, 150, 35, 5];
    pop(2,:) = [140, 180, 0, 3,  140, 190, 20, 3,   140, 150, 40, 3];

    val = zeros(NP, 1);
    parfor i=1:NP, val(i) = cost_func(pop(i,:)); end
    [best_val, idx] = min(val);
    best_mem = pop(idx, :);
    curve = zeros(opts.MaxIter, 1);

```

```
    h = waitbar(0, 'Multi-UAV Optimization...');
    for gen = 1 : opts.MaxIter
        F = opts.F * (1 - 0.3 * gen/opts.MaxIter);
        pop_new = pop; val_new = val;
        parfor i = 1 : NP
            r = randperm(NP, 3);
            mutant = best_mem + F * (pop(r(1),:) - pop(r(2),:));
            trial = pop(i, :);
            j_rand = randi(D);
            for j = 1 : D
                if rand < opts.CR || j == j_rand, trial(j) = mutant(j
                    ); end
            end
            trial = max(trial, lb); trial = min(trial, ub);
            t_v = feval(cost_func, trial);
            if t_v < val(i), pop_new(i,:) = trial; val_new(i) = t_v;
                end
        end
        pop = pop_new; val = val_new;
        [c_best, idx] = min(val);
        if c_best < best_val, best_val = c_best; best_mem = pop(idx,
            :); end
        curve(gen) = -best_val;
        if mod(gen, 50) == 0
            [~, t_real] = feval(cost_func, best_mem);
            waitbar(gen/opts.MaxIter, h, sprintf('Iter %d: %.2fs',
                gen, t_real));
            fprintf('    Iter %d: 引导得分 %.4f | 真实遮蔽 %.4fs\n',
                gen, best_val, t_real);
        end
    end
    close(h);
end

%% --- 绘图与保存（保持不变） ---
function Plot_3UAV_Scenario(Res, score, curve, Env)
    figure('Color','w', 'Position', [50, 50, 1200, 600]);
    subplot(1, 2, 1); hold on; grid on; axis equal; view([-30, 30]);
    plot3(Env.Pos_True(1), Env.Pos_True(2), Env.Pos_True(3), 'go', '
        MarkerSize', 12, 'MarkerFaceColor','g');
```

```
    plot3(Env.Pos_M1(1), Env.Pos_M1(2), Env.Pos_M1(3), 'rv', '
        MarkerSize', 10, 'MarkerFaceColor','r');
    colors = {'b', 'm', [0 0.5 0]};
    for i = 1:3
        ang = deg2rad(Res(i).a);
        V = [cos(ang), sin(ang), 0] * Res(i).v;
        P0 = Env.Pos_UAVs(i,:);
        P_Drop = P0 + V * Res(i).td;
        plot3([P0(1) P_Drop(1)], [P0(2) P_Drop(2)], [P0(3) P_Drop(3)
            ], '-', 'Color', colors{i}, 'LineWidth', 1.5);
        text(P0(1), P0(2), P0(3)+200, sprintf('FY%d',i), 'Color',
            colors{i});
        P_Pop = P_Drop + V * Res(i).ty - [0,0,0.5*9.8*Res(i).ty^2];
        plot3(P_Pop(1), P_Pop(2), P_Pop(3), 'h', 'MarkerSize', 14, '
            MarkerFaceColor', colors{i});
        plot3([P_Drop(1) P_Pop(1)], [P_Drop(2) P_Pop(2)], [P_Drop(3)
            P_Pop(3)], 'k:');
    end
    line([Env.Pos_M1(1) Env.Pos_True(1)], [Env.Pos_M1(2) Env.Pos_True
        (2)], [Env.Pos_M1(3) Env.Pos_True(3)], ...
         'Color', [0.6 0.6 0.6], 'LineStyle', '--');
    xlabel('X'); ylabel('Y'); zlabel('Z');
    title('三机协同立体封锁示意图');
    subplot(1, 2, 2); hold on; grid on;
    for i=1:3
        tp = Res(i).tp;
        fill([tp, tp+Env.T_last, tp+Env.T_last, tp], [i-0.3, i-0.3, i
            +0.3, i+0.3], colors{i}, 'FaceAlpha', 0.6);
        text(tp, i, sprintf('␣Start:␣%.1fs', tp));
    end
    title(['协同遮蔽时序␣(总长:␣' num2str(score) 's)']);
    xlabel('时间␣(s)'); ylim([0 4]); yticks([1 2 3]); yticklabels({'
        FY1','FY2','FY3'});
end

function Save_To_Excel(Res, filename)
    data = zeros(3, 5);
    for i=1:3
        data(i,1) = Res(i).v;
        data(i,2) = Res(i).a;
```

```
240         data(i,3) = Res(i).td;
241         data(i,4) = Res(i).ty;
242         data(i,5) = Res(i).tp;
243     end
244     try, writematrix(data, filename); catch, end
245 end
```