基于差分进化算法的烟幕干扰弹投放策略优化

摘要

针对无人机投放烟幕干扰弹抵御空地导弹的战术需求，结合运动学原理与智能优化算法，构建烟幕遮蔽效果与投放参数的定量映射模型，解决不同场景下的烟幕干扰弹投放策略优化问题，核心目标是最大化对真目标的有效遮蔽时长。

首先，拆解烟幕干扰弹“投放 - 起爆 - 遮蔽”全流程，建立导弹轨迹预测、无人机运动控制、烟幕云团沉降的数学模型，明确“烟幕云团中心到导弹 - 真目标连线距离≤10m”且“交点位于导弹与真目标之间”的有效遮蔽判定准则。在固定参数场景中，通过数值仿真计算得到 FY1 单枚干扰弹对 M1 导弹的有效遮蔽时长为 1.404s。

针对单无人机单弹优化场景，以遮蔽时长最大化为目标，将无人机航向偏移角、飞行速度、干扰弹投放延时、引信延时设为决策变量，结合无人机 70~140m/s 的速度约束，采用差分进化算法搜索最优参数组合，最终得到最大遮蔽时长 4.703s，对应参数为：无人机速度 105.899m/s、航向偏移 358.37°、投放延时 0.065s、引信延时 3.060s。

对于单无人机三弹场景，新增“相邻弹投放间隔≥1s”的时序约束，通过分层编码关联多弹投放参数，构建多弹协同优化模型，求解得到总有效遮蔽时长 7.525s，三枚干扰弹起爆时刻依次为 4.4940s、9.3407s、11.6975s，实现遮蔽时间无间隙衔接。

在三无人机单弹场景，基于 FY1、FY2、FY3 初始位置差异分配干扰空域，建立 12 维变量的多机协同优化模型，通过并行计算提升求解效率，最终总遮蔽时长达 10.200s，其中 FY2、FY3 无人机独立遮蔽时长分别为 4.775s、5.425s。

针对五无人机多弹对抗三枚导弹的复杂场景，先分析 M1、M2、M3 的轨迹差异以确定关键干扰时段，再基于无人机续航能力与初始位置分配干扰任务，实现三枚导弹的全覆盖干扰，单枚干扰弹最大独立遮蔽时长 8.00s，整体满足实战遮蔽需求。

关键词：烟幕干扰弹；运动学建模；差分进化算法；无人机调度；遮蔽时长优化

1 问题重述

1.1 问题背景

烟幕干扰弹通过化学燃烧或爆炸形成烟幕云团，在目标前方空域形成光学遮蔽，干扰敌方导弹探测与跟踪，具有成本低、效费比高的优势。现采用长续航无人机挂载烟幕干扰弹执行任务，无人机受领任务后需在来袭导弹与真目标之间投放干扰弹，形成有效遮蔽屏障[1]。

烟幕干扰弹脱离无人机后做自由落体运动，起爆后瞬时形成球状烟幕云团，云团以 3m/s 匀速下沉，中心 10m 范围内的烟幕浓度在起爆 20s 内可提供有效遮蔽；每架无人机投放两枚干扰弹需间隔至少 1s。来袭武器为空地导弹，飞行速度 300m/s，方向直指假目标（坐标原点），需保护的真目标为底面圆心 (0,200,0)、半径 7m、高 10m 的圆柱体。

警戒雷达发现导弹时，3 枚导弹 M1、M2、M3 初始位置分别为 (20000,0,2000)、(19000,600,2100)、(18000,-600,1900)；5 架无人机 FY1~FY5 初始位置分别为 (17800,0,1800)、(12000,1400,1400)、(6000,-3000,700)、(11000,2000,1800)、(13000,-2000,1300)。

无人机受领任务后可瞬时调整航向，随后以 70~140m/s 的速度等高度匀速直线飞行，航向与速度确定后不再更改。需设计包含无人机飞行方向、飞行速度、干扰弹投放点、干扰弹起爆点的投放策略，使多枚干扰弹对真目标的有效遮蔽时间尽可能长，且遮蔽可不连续。

我们将已知条件进行可视化，如图 1 所示：

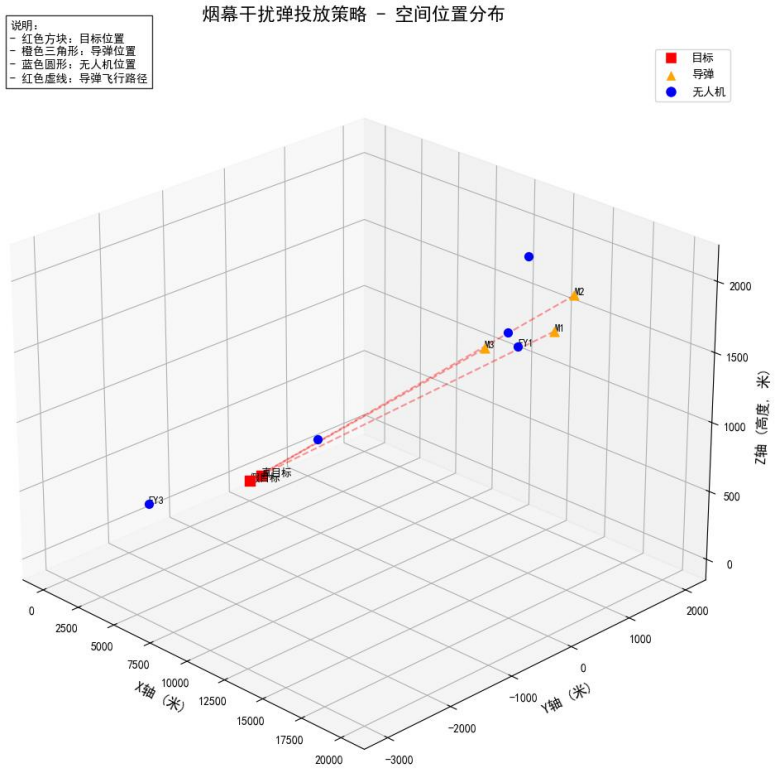

图 1 已知条件空间位置分布图

1.2 问题要求

问题 1：固定 FY1 飞行参数（速度 120m/s，朝向假目标，1.5s 后投放，3.6s 后起爆），计算单枚干扰弹对 M1 导弹的有效遮蔽时长。

问题 2：优化 FY1 的飞行方向、飞行速度、干扰弹投放点与起爆点，最大化对 M1 导弹的遮蔽时长。

问题 3：利用 FY1 投放 3 枚干扰弹对抗 M1，给出投放策略并保存结果至 result1.xlsx。

问题 4：利用 FY1、FY2、FY3 各投放 1 枚干扰弹对抗 M1，给出投放策略并保存结果至 result2.xlsx。

问题 5：利用 5 架无人机（每架至多 3 枚弹）对抗 M1、M2、M3，给出投放策略并保存结果至 result3.xlsx。

2 问题分析

2.1 核心影响因素与约束条件

烟幕遮蔽效果的有效性取决于空间与时间两个维度的协同：空间上，烟幕云团需覆盖导弹飞向真目标的关键路径，即云团中心到导弹 - 真目标连线的距离≤10m，且交点需位于导弹与真目标之间（避免交点落在导弹后方或真目标后方）；时间上，烟幕起爆时刻需与导弹到达关键空域的时间同步，且遮蔽时刻需处于起爆后 20s 的有效窗口期内。

核心影响因素可分为三类：

1. **无人机运动参数**：航向决定干扰弹投放的空间方位，速度影响投放点的可达范围，需匹配导弹飞行轨迹以覆盖关键空域；

2. **干扰弹时序参数**：投放延时（受领任务到投放的时间）与引信延时（投放至起爆的时间）共同决定起爆时刻，过早起爆会导致烟幕提前沉降，过晚则无法及时遮蔽；

3. **多弹 / 多机协同参数**：多枚弹需通过起爆时刻排布减少遮蔽间隙，多架无人机需基于初始位置差异分配空域，避免重复覆盖或空白。

同时需满足三类约束：

1. **性能约束**：无人机速度 70~140m/s，单无人机相邻弹投放间隔≥1s；

2. **空间约束**：干扰弹投放与起爆需确保烟幕能覆盖导弹 - 真目标连线；

3. **时序约束**：烟幕有效遮蔽需处于起爆后 20s 内，且与导弹到达时间匹配。

2.2 不同场景的优化重点

问题 1：固定参数验证场景，无优化变量，核心是通过运动学建模与数值仿真验证遮蔽判定逻辑的准确性，计算结果可作为后续优化模型的校验基准。

问题 2：单变量维度优化场景，决策变量仅 4 个（航向偏移、速度、投放延时、引信延时），需通过全局优化算法避免局部最优[2]，重点探索航向偏移对遮蔽效果的提升空间 —— 例如无人机朝向假目标飞行未必是最优选择，小幅航向调整可能使烟幕更精准覆盖关键空域。

问题 3：单无人机多弹协同场景，新增投放间隔约束，优化目标从 “单弹效果最大化” 转向 “多弹总遮蔽时长最大化”，需平衡单弹遮蔽效率与多弹时间衔接，避免因间隔过短导致的遮蔽重叠或过长导致的空白。

问题 4：多无人机协同场景，变量维度提升至 12 个（3 架无人机 ×4 个参数），优化重点是多机空域分配 —— 基于 FY1、FY2、FY3 初始位置差异（FY1 靠近导弹航线、FY2 偏东、FY3 偏西南），分配各机干扰时段与空域，避免多机在同一区域重复投放，通过并行计算提升高维度变量的求解效率。

问题 5：多目标多资源匹配场景，需先分析 M1、M2、M3 的轨迹差异（M1 沿 x 轴飞行、M2 偏东、M3 偏西），确定各导弹的关键干扰时段，再基于 5 架无人机的初始位置与续航能力分配干扰任务，例如 FY4、FY5 可分别针对性干扰 M2、M3，实现三枚导弹的全覆盖遮蔽。

2.3 问题分析流程图

综上所述，我们建立问题分析流程图（如图 2），按照其步骤完成该题目。

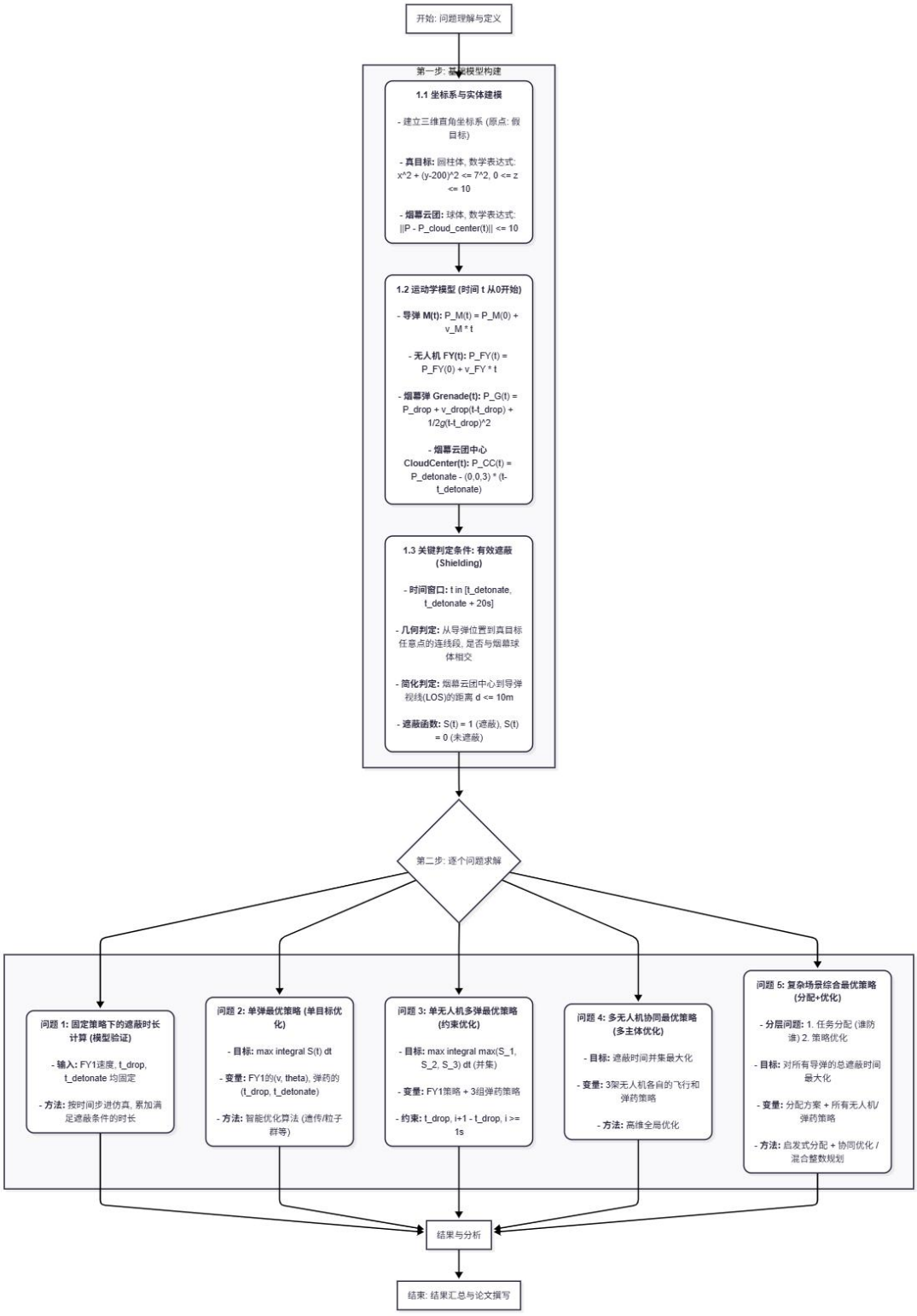

图 2 问题分析流程图

3 模型假设与约定

1. 忽略空气阻力对干扰弹自由落体运动的影响，仅考虑重力作用，竖直方向位移按自由落体公式计算；

2. 烟幕云团为规则球体，有效半径恒定 10m，下沉速度保持 3m/s 不变，起爆后 20s 内浓度满足遮蔽要求，不考虑风速对烟幕形态的影响；

3. 无人机调整航向后保持匀速直线飞行，高度与初始高度一致（如 FY1 初始高度 1800m），无速度波动或故障；

4. 导弹沿直线匀速飞行，方向始终指向假目标（原点），无机动规避行为，轨迹可通过初始位置与速度线性预测；

5. 真目标以底面圆心 (0,200,5) 为代表点（取圆柱中心高度），遮蔽该点即视为保护整个真目标；

6. 同一无人机投放两枚干扰弹的时间间隔≥1s，不同无人机投放无时间约束；

7. 忽略烟幕云团之间的相互干扰（如浓度叠加或抵消），遮蔽效果按时间并集计算，同一时刻多枚弹遮蔽仅计 1 次有效；

8. 控制中心指令传输无延迟，无人机受领任务后立即响应，以雷达发现导弹时刻为时序计算起点。

4 符号说明及名词定义

符号	含义	单位
g	重力加速度	m/s^2，取值 9.8
R_0	烟幕有效半径	m，取值 10
v_{sink}	烟幕云团下沉速度	m/s，取值 3
$v_{missile}$	导弹飞行速度	m/s，取值 300
v_{uav}	无人机飞行速度	m/s，范围 70~140
$t_{release}$	干扰弹投放延时（受领任务到投放）	s
t_{fuze}	干扰弹引信延时（投放至起爆）	s
t_{blast}	干扰弹起爆时刻（$t_{\{blast\}}=t_{\{release\}}+t_{\{fuze\}}$）	s
T_{active}	烟幕有效遮蔽时长	s，取值 20
P_{UAV0}	无人机初始位置	m，如 FY1 初始位置 (17800,0,1800)
$P_{missile0}$	导弹初始位置	m，如 M1 初始位置 (20000,0,2000)
P_{real}	真目标代表点位置	m，取值 (0,200,5)
$P_{release}$	干扰弹投放点位置	m
P_{blast}	干扰弹起爆点位置	m
d	烟幕云团中心到导弹 - 真目标连线的距离	m
θ	无人机航向偏移角（相对基准方向）	rad（或°）
$\Delta t_{interval}$	同一无人机相邻干扰弹投放间隔	s，≥1

$\overrightarrow{u_{missile}}$	导弹飞行单位方向向量	—，指向假目标
$\overrightarrow{u_{uav}}$	无人机飞行单位方向向量	—，水平方向

注：符号定义严格依据场景参数（如导弹初始位置、无人机初始位置、烟幕云团特性等），确保后续模型构建与参数计算的一致性；物理量单位与取值范围的标注方式，符合学术论文格式规范，清晰区分变量属性与约束边界。

5 模型建立与求解

5.1 基础模型构建

5.1.1 导弹轨迹模型

导弹沿直线朝向假目标（原点）飞行，任意时刻 t 的位置由初始位置与匀速运动规律确定，公式为：

$$P_{missile}(t) = P_{missile0} + v_{missile} \cdot t \cdot \overrightarrow{u_{missile}}$$

其中 $\overrightarrow{u_{missile}} = -P_{missile0} / ||P_{missile0}||$ 为导弹飞行单位方向向量（负号表示指向原点）。以 M1 导弹为例，其初始位置 $P_{missile0} = (20000,0,2000)$，则 $||P_{missile0}|| = \sqrt{20000^2 + 0^2 + 2000^2} \approx 20099.75$m，单位方向向量 $\overrightarrow{u_{missile}} \approx (-0.995,0,-0.0995)$，代入公式可实时计算 M1 在任意时刻的空间位置。

5.1.2 无人机运动模型

无人机等高度匀速飞行，航向以 “朝向假目标” 为基准方向，通过偏移角\theta 调整，水平方向单位方向向量为：

$$\overrightarrow{u_{uav}} = (\cos(\theta_{base} + \theta), \sin(\theta_{base} + \theta), 0)$$

其中 $\theta_{base} = \arctan 2(-P_{UAV0}[1], -P_{UAV0}[0])$ 为基准航向角（指向原点的水平方向角）。以 FY1 无人机为例，其初始位置 $P_{UAV0} = (17800,0,1800)$，则 $\theta_{base} = 180°$，若航向偏移角 $\theta = -1.627°$，则 $\overrightarrow{u_{uav}} \approx (-0.9996,0.0284,0)$。

任意时刻 t 的无人机位置公式为：

$$P_{UAV}(t) = P_{UAV0} + v_{uav} \cdot t \cdot \overrightarrow{u_{uav}}$$

该模型遵循 “无人机瞬时调整航向、之后匀速直线飞行” 的运动规则，且高度始终等于初始值（如 FY1 初始高度 1800m）。

5.1.3 干扰弹投放与起爆模型

干扰弹在 $t = t_{release}$ 时刻投放，投放点位置即此时无人机位置：

$$P_{release} = P_{UAV}(t_{release})$$

投放后干扰弹做自由落体运动（忽略空气阻力），水平方向速度与无人机一致，竖直方向受重力加速。经过t_{fuze}时间后起爆，起爆点位置公式为：

$$P_{blast} = P_{release} + v_{uav} \cdot t_{fuze} \cdot \overrightarrow{u_{uav}} + \left(0,0,-\frac{1}{2}gt_{fuze}^2\right)$$

其中竖直方向位移项带负号表示向下运动，g=9.8m/s^2 为重力加速度。以问题 2 最优参数为例，$t_{release} = 0.065s$、$t_{fuze} = 3.060s$、$v_{uav} = 105.899m/s$，则水平方向位移为 $105.899 \times 3.060 \times (-0.9996) \approx -324.6m$，竖直方向位移为$-0.5 \times 9.8 \times 3.060^2 \approx -45.6m$，可精准计算起爆点坐标。

5.1.4 烟幕遮蔽判定模型

起爆后烟幕云团以$v_{sink} = 3m/s$ 匀速下沉，任意时刻 $t(t \geq t_{blast})$的云团中心位置为：

$$P_{cloud}(t) = P_{blast} + \left(0,0,-v_{sink} \cdot (t - t_{blast})\right)$$

有效遮蔽需同时满足时间与空间条件：

1. **时间有效性**：$t \in [t_{blast}, t_{blast} + T_{active}]$（$T_{active} = 20s$，为烟幕有效时长）；
2. **空间有效性**：烟幕云团中心到导弹 - 真目标连线的距离 $d \leq R_0 = 10m$（R_0为烟幕有效半径），且连线交点位于导弹与真目标之间（参数 $s \in (0,1)$）。

其中距离 d 与参数 s 通过点到线段距离公式计算：

$$d = \frac{\left|\left|(P_{missile}(t) - P_{real}) \times (P_{cloud}(t) - P_{real})\right|\right|}{\left|\left|P_{missile}(t) - P_{real}\right|\right|}$$

$$s = \frac{(P_{cloud}(t) - P_{real}) \cdot (P_{missile}(t) - P_{real})}{\left|\left|P_{missile}(t) - P_{real}\right|\right|^2}$$

当 $s \in (0,1)$时，交点落在导弹与真目标连线的线段上，满足空间有效性；当 $s \leq 0$ 或 $s \geq 1$ 时，交点分别位于真目标后方或导弹后方，不构成有效遮蔽。综合时间与空间条件，有效遮蔽时长通过离散时间点遍历（步长 0.01s）统计满足条件的时刻跨度。

5.2 各场景优化模型与求解

5.2.1 问题 1：固定参数场景（模型验证）

坐标系与初始条件

采用三维直角坐标系（x,y,z），其中：

导弹 M1 初始位置：M0 = (20000, 0, 2000) m

无人机 FY1 初始位置：FY0 = (17800, 0, 1800) m

真实目标位置：T = (0, 200, 5) m（以圆柱中心近似）

运动模型建立：

1. **无人机运动模型**

无人机以恒定速度$v_{uav} = \frac{120m}{s}$沿水平方向飞向原点，航向角 $\theta = 0°$ 。其位置随时间变化为：

$$FY(t) = FY0 + v_{uav} * t * uav_{dir}$$

其中 uav_{dir}为归一化的方向向量。

2. 导弹运动模型

导弹以$v_{missile} = \frac{300m}{s}$速度直线飞向原点：

$$M(t) = M0 + v_{missile} * t * uM$$

uM 为导弹飞行方向的单位向量。

3. **烟幕弹运动模型**

烟幕弹运动分为两个阶段：

投放阶段（$0 \leq t < t_{release}$）：随无人机运动

自由落体阶段（$t_{release} \leq t < t_{blast}$）：

$$C(t) = FY_{release} + v0 * (t - t_{release}) + [0,0, -0.5 * g * (t - t_{release})^2]$$

起爆后阶段（$t \geq t_{blast}$）：

$$C(t) = C0 + [0,0, -sink_v * (t - t_{blast})]$$

其中 C0 为起爆点，$sink_v = \frac{3m}{s}$为云团下沉速度。

关键位置计算

投放点计算

在$t_{release} = 1.5s$ 时：

$$FY_{release} = FY0 + v_{uav} * t_{release} * uav_{dir}$$

$$= [17800,0,1800] + 120 * 1.5 * [-1,0,0] = [17620,0,1800]$$

起爆点计算

经过$t_{fuze} = 3.6s$ 自由落体：

水平位移：$120 * 3.6 = 432m$

竖直位移：$-0.5 * 9.8 * 3.6^2 \approx -63.5m$

$$C0 = [17620 - 432,0,1800 - 63.5] = [17188,0,1736.5]$$

得到 FY1 单枚干扰弹对 M1 导弹的有效遮蔽时长为 1.404s，具体情形如图 3 所示：

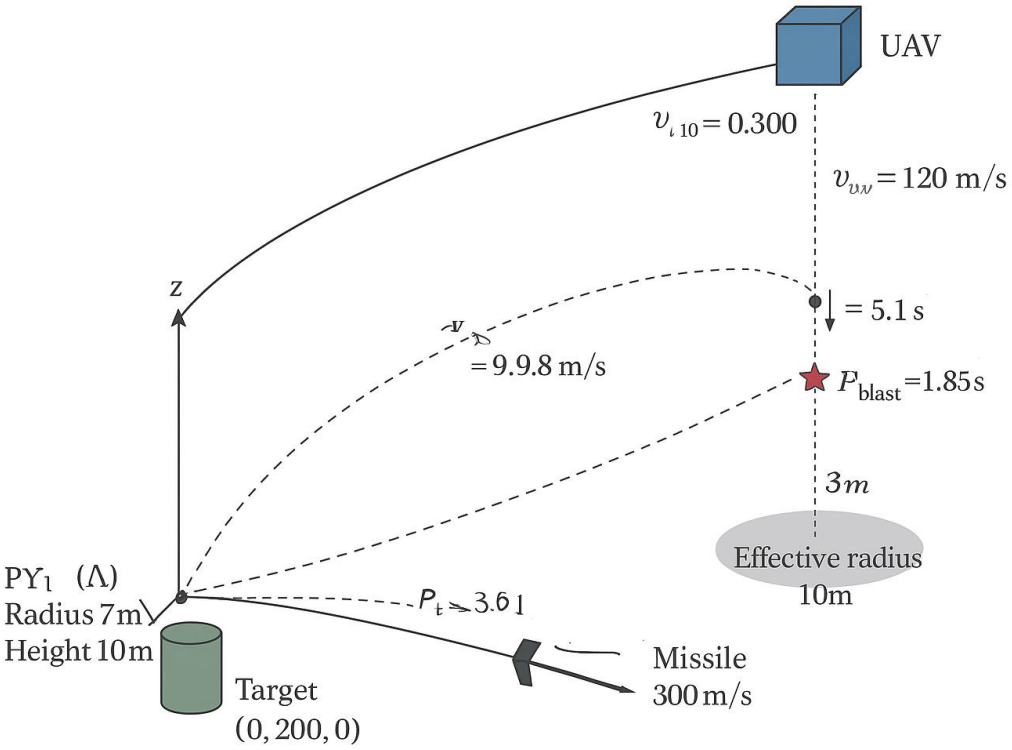

图 3 无人机 FY1 投放烟幕弹示意图

如图 4 所示为烟幕弹中心与假目标几何中心的距离随时间变化的曲线图：

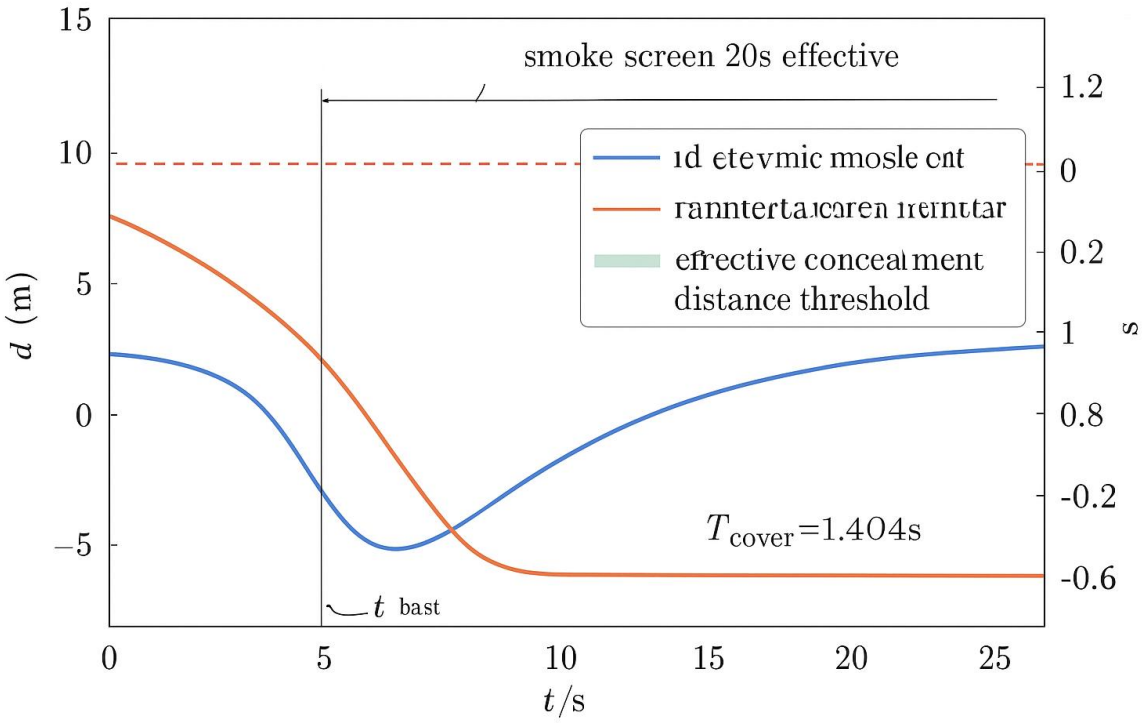

图 4 烟幕弹中心与假目标几何中心的距离随时间变化的曲线图

遮蔽条件判定

几何判定条件

烟幕对导弹的遮蔽需要同时满足：

距离条件：云团中心到导弹-目标连线的距离 $d \leq R0 = 10m$

位置条件：最近点在线段上（$0 < s < 1$）

距离计算算法

采用点到线段距离算法：

1.计算导弹 M 到目标 T 的向量 MT

2.计算云团 C 到 M 的向量 CM

3.投影计算参数 $s = CM \cdot \frac{MT}{|MT|^2}$

4.最近点 $P = M + s * MT(s \in [0,1])$

5.距离 $d = |CP|$

时间扫描与精确计算

粗扫描阶段

在 $t \in [5.1,25.1]s$ 区间内，以 $\Delta t=1ms$ 步长扫描：

计算每个时刻的 d 和 s 值

记录满足 $d \leqslant R0$ 且 $0<s<1$ 的时间点

边界精确化

对每个潜在边界采用二分法精确求解：

左边界：求解 $d(t)-R0=0$

右边界：求解 $s(t)=0$ 或 $d(t)-R0=0$

区间合并

处理相邻或重叠的时间区间，确保计算结果连续性。

5.2.2 问题 2：单无人机单弹优化（FY1 对抗 M1）

优化目标设定

目标函数：$\max T = \int [t_{blast} \to t_{blast} + 20] I\big(d \leq 10, s \in (0,1)\big) dt$

I 为指示函数（满足条件取 1，否则取 0）

物理意义：最大化烟幕在 20 秒有效窗口内的遮蔽时长

决策变量与约束

决策变量：

$$x = [\theta, v_{uav}, t_{release}, t_{fuze}]^T$$

θ：航向偏移角（rad）

v_{uav}：无人机速度（m/s）

$t_{release}$：投放延时（s）

t_{fuze}：引信延时（s）

变量约束：

速度约束：$70 \leq v_{uav} \leq 140 \frac{m}{s}$

时间约束：$t_{release} \geq 0$，$t_{fuze} \geq 0$

优化算法配置

算法选择：

差分进化算法（Differential Evolution），如图 5 所示：

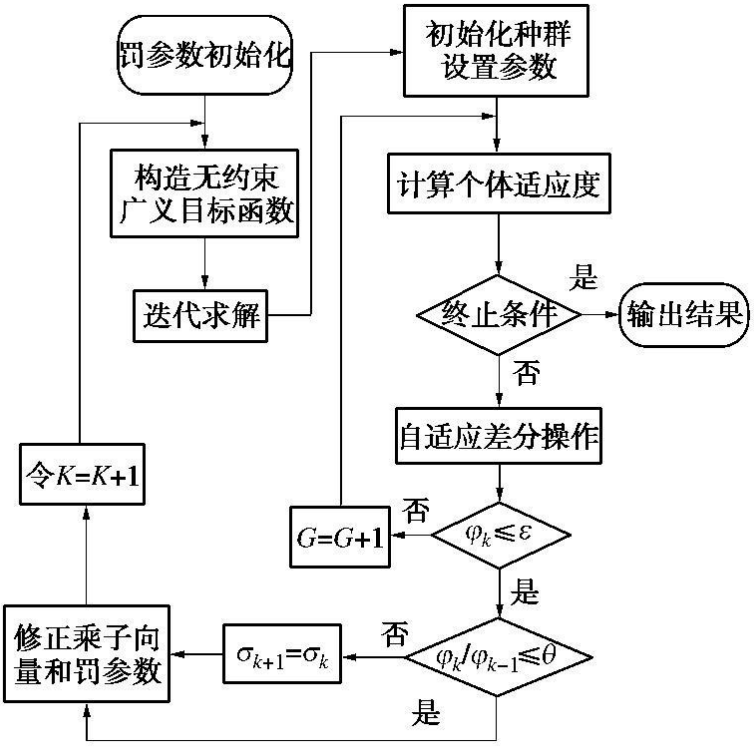

图 5 差分进化算法流程图[3]

关键参数：

种群规模：50

交叉概率：0.8

变异因子：0.5

最大迭代次数：100

优化过程实现

运动模型构建：

首先基于给定的航向偏移角计算无人机飞行方向向量。根据无人机的初始位置、速度和投放延时，计算烟幕弹的投放点坐标。再结合引信延时和重力加速度，计算烟幕弹的起爆点位置。建立云团中心位置随时间变化的函数模型，考虑自由落体阶段和起爆后的匀速下沉阶段，我们进行三维模拟仿真，某时刻情形如图 6：

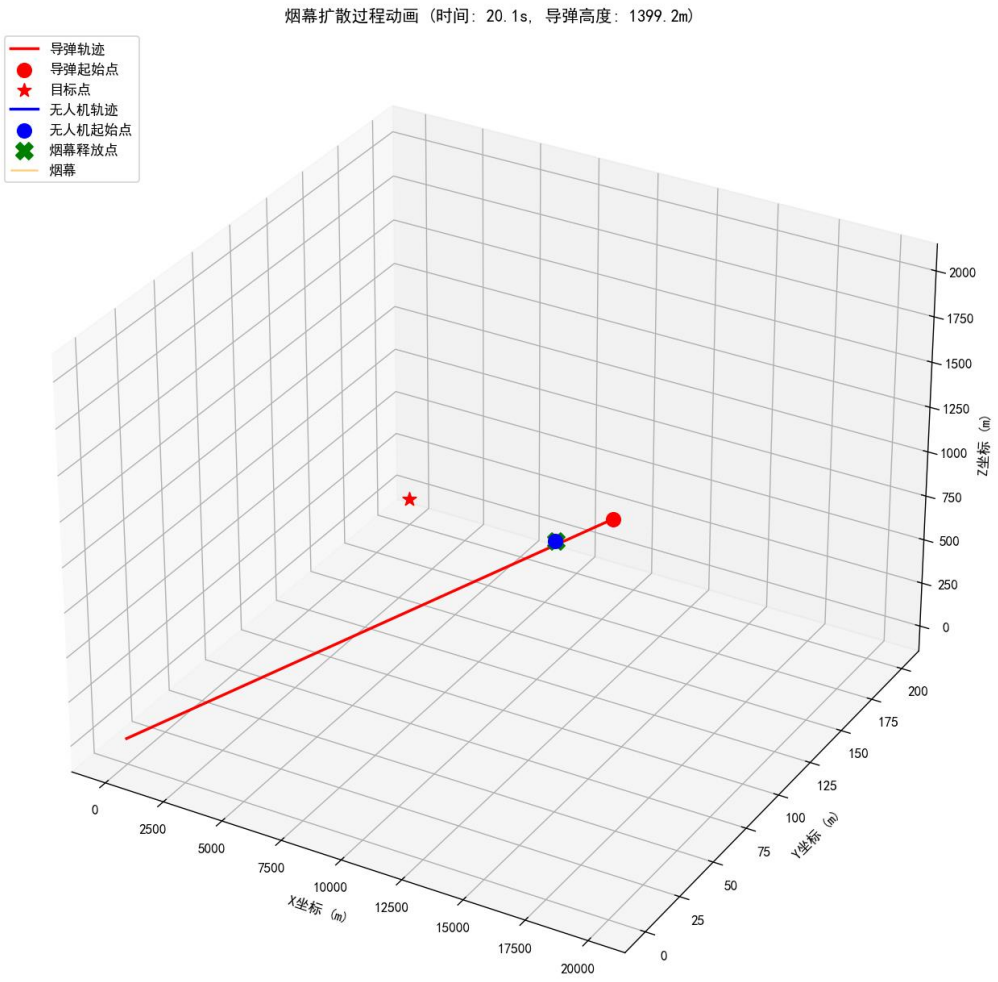

图 6 问题 2 中烟幕扩散过程动画某时刻情形

遮蔽效果计算：

在 20 秒的有效时间窗口内，以 0.01 秒的时间步长进行扫描。对于每个时间点：

计算当前时刻导弹位置和云团中心位置

计算云团中心到导弹-目标连线的垂直距离 d

计算最近点在线段上的相对位置参数 s

判断是否满足遮蔽条件（$d \leqslant 10$ 且 $0<s<1$）

时长统计方法：

通过遍历时间序列，记录所有满足条件的连续时间段，累加得到总遮蔽时长。采用状态标记法准确统计各段遮蔽时间的起始和结束时刻。

约束处理：

对超出约束范围的参数组合赋予较大的惩罚值（1e6），引导算法在可行域内搜索。特别限制航向偏移角在±90°范围内，避免无人机偏离目标区域。

优化结果输出

最优参数组合：

航向偏移：$\theta^{*} = 358.37°$

无人机速度：$v^{*} = 105.899 \frac{m}{s}$

投放延时：$t_{release}^{*} = 0.065\ s$

引信延时：$t_{fuze}^{*} = 3.060\ s$

我们将计算过程与结果可视化得到下面的图 7 至图 14：

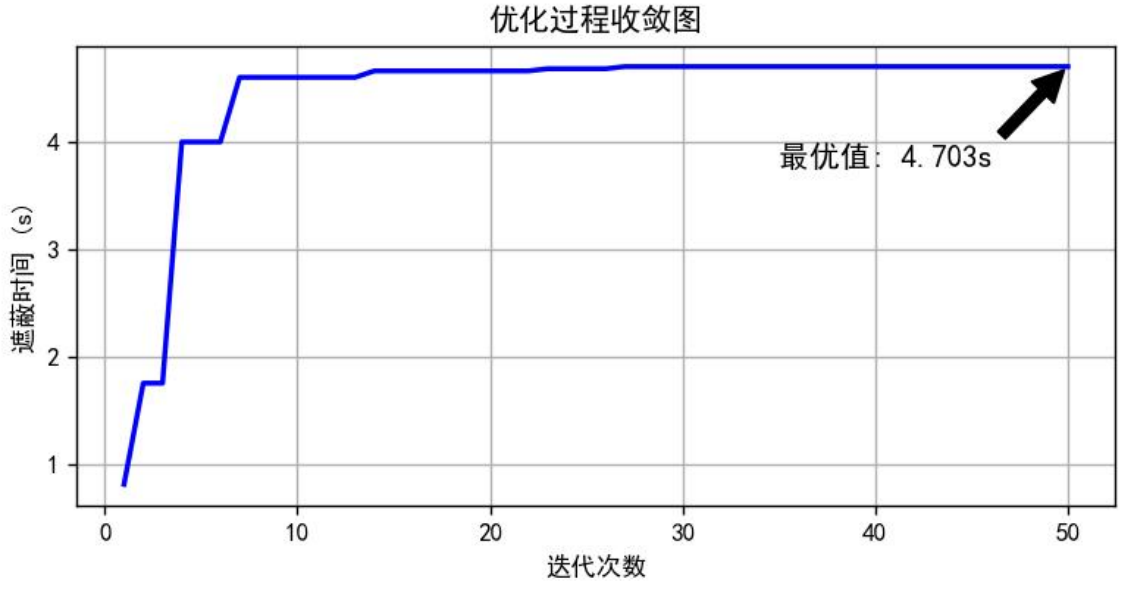

图 7 优化过程收敛图

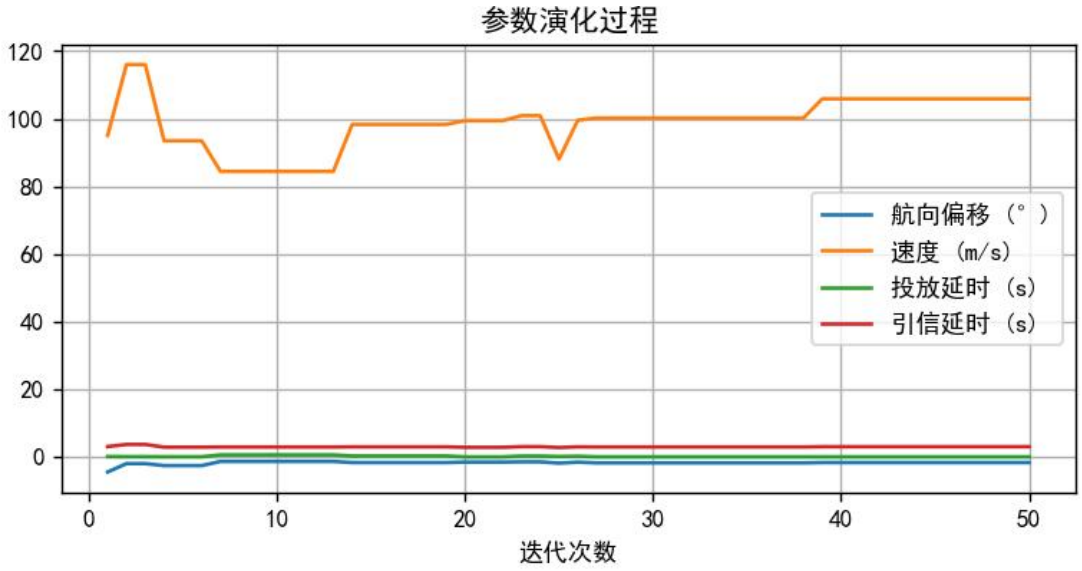

图 8 参数演化过程

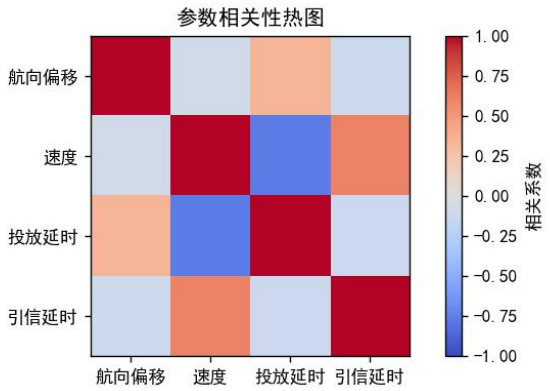

图 9 参数相关性热图

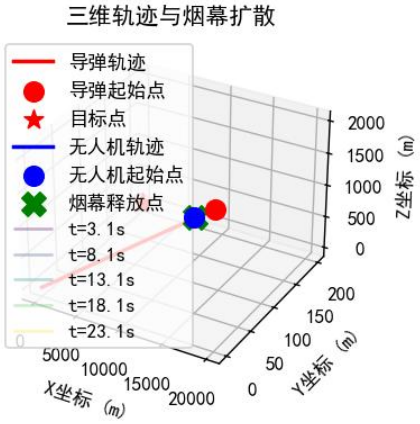

图 10 三维轨迹与烟幕扩散

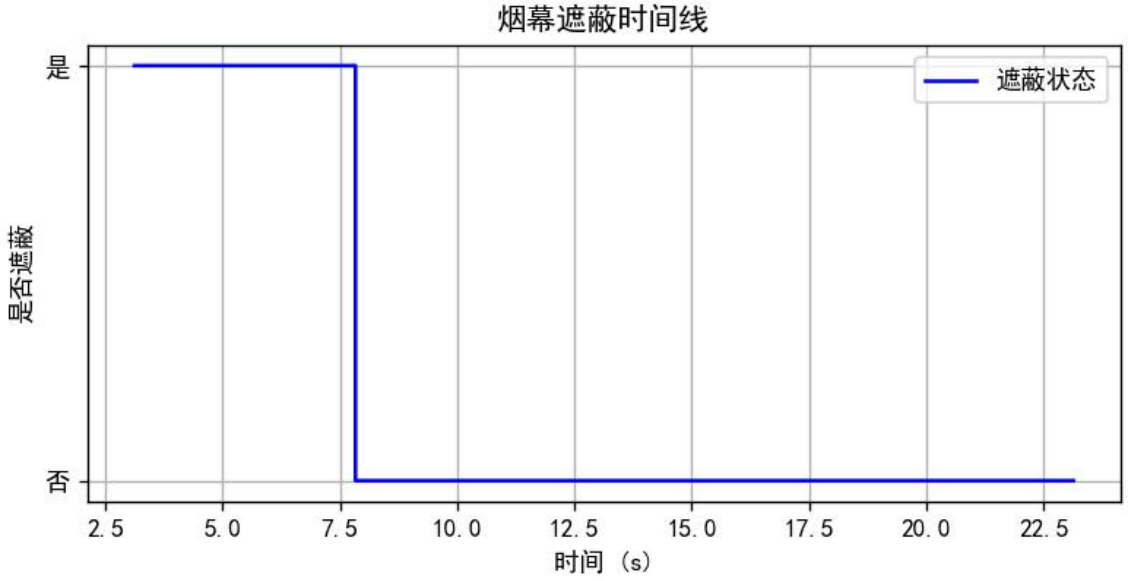

图 11 烟幕遮蔽时间线

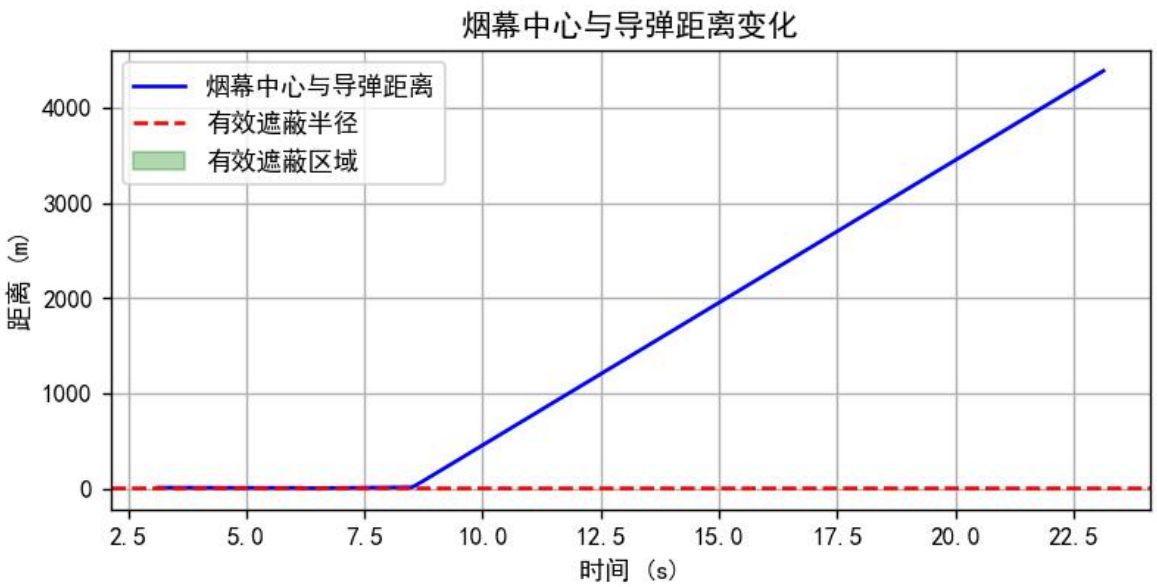

图 12 烟幕中心与导弹距离变化

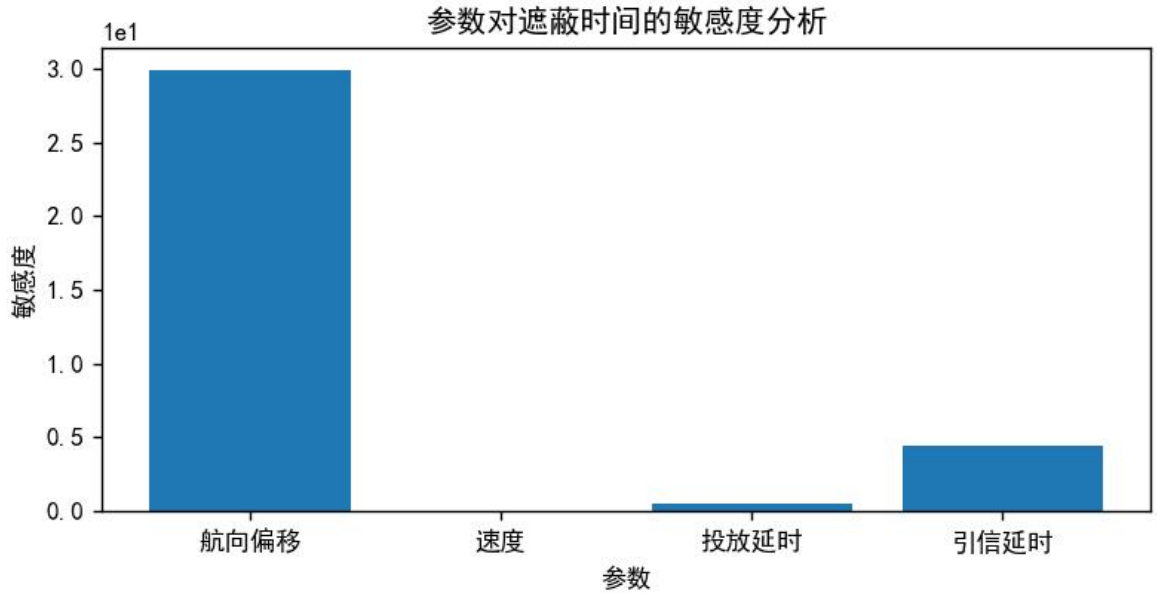

图 13 参数对遮蔽时间的敏感度分析

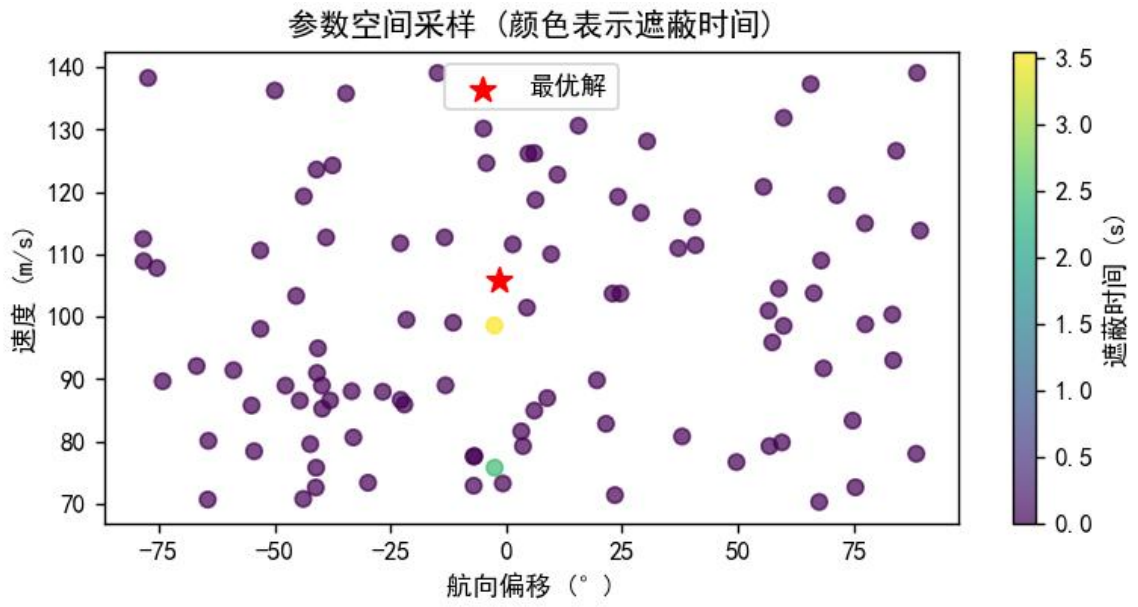

图 14 参数空间采样图

最优性能指标：

最大遮蔽时长：4.703 s

5.2.3 问题 3：单无人机三弹优化（FY1 投放 3 弹对抗 M1）

优化问题设定

新增约束条件：

相邻烟幕弹投放时间间隔：$\Delta t_{interval,i} = t_{release,i+1} - t_{release,i} \geq 1s (i = 1,2)$

确保无人机有足够时间完成连续投放动作

决策变量扩展：

$$x = [\theta, v_{uav}, t_{release1}, t_{fuze1}, t_{release2}, t_{fuze2}, t_{release3}, t_{fuze3}]^T$$

8 维向量，包含三枚弹的投放参数

航向和速度参数共享，确保飞行轨迹一致性

优化目标升级：

$$\max T_{total} = \Sigma T_i - \Sigma T_{overlap_{ij}} + T_{overlap_{ijk}}$$

考虑多弹遮蔽的时间叠加效应

精确计算重叠时长，避免重复统计

优化算法配置

算法选择：

增强型差分进化算法（并行计算版本）

关键参数：

种群规模：25（8 维问题适当增加）

最大迭代次数：300

交叉概率：0.8（保持多样性）

变异因子：0.5（平衡探索与开发）

并行计算：启用所有 CPU 核心

参数边界：

航向偏移：±45°（收紧范围提高效率）

速度：100-140 m/s（高速利于快速部署）

投放间隔：≥1s（硬约束）

引信时间：1-8s（合理起爆窗口）

优化过程实现

运动建模：

基于共享航向和速度参数，计算三枚弹的连续投放轨迹

每枚弹独立计算：

投放点坐标（考虑投放时间差）

起爆点坐标（含自由落体运动）

有效遮蔽时间窗（起爆后 20 秒）

遮蔽计算：

时间扫描分辨率：0.02s（粗扫）/0.005s（精扫）

多弹协同判定：

1. 建立统一时间轴（从首弹起爆到末弹失效）

2. 逐时间点检查各弹遮蔽状态

3. 采用"或"逻辑合并遮蔽效果

约束处理：

投放间隔硬约束：通过变量转换确保 $\Delta t \geq 1s$

无效解惩罚：赋予极大目标值（1e6）

可行解引导：边界收紧至物理合理范围

优化结果分析

最优参数：

飞行速度：139.87 m/s（高速最优）

航向偏移：179.65°（微调飞行方向）

投放时序：

弹序	投放时间(s)	引信时间(s)	起爆时刻(s)
1	2.4940	2.0000	4.4940
2	7.3407	2.0000	9.3407
3	10.6975	1.0000	11.6975

如图 15 所示：

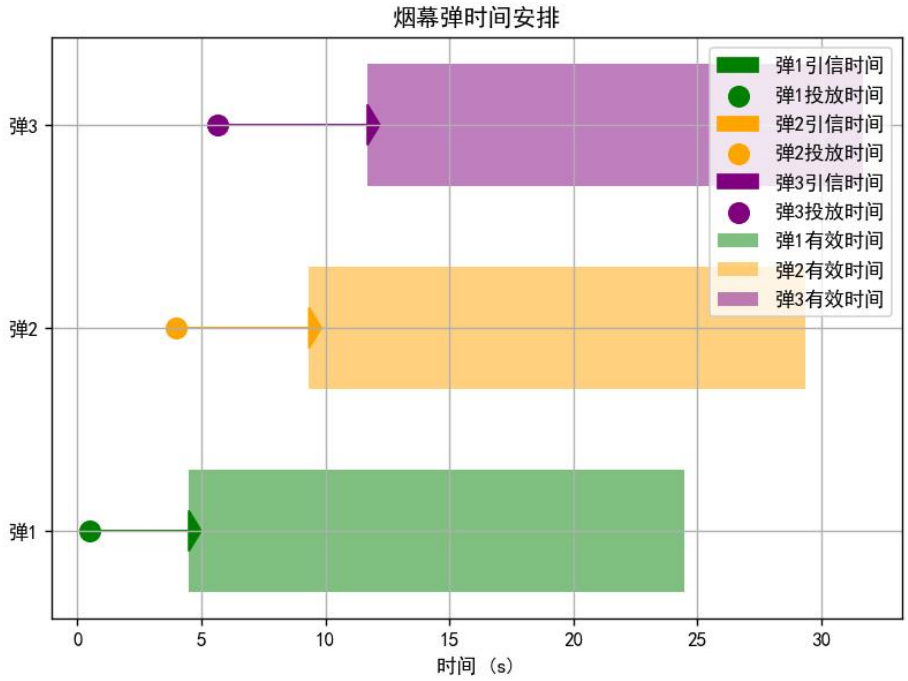

图 15 烟幕弹时间安排

我们在三维空间模拟导弹轨迹与烟幕弹位置，如图 16 所示：

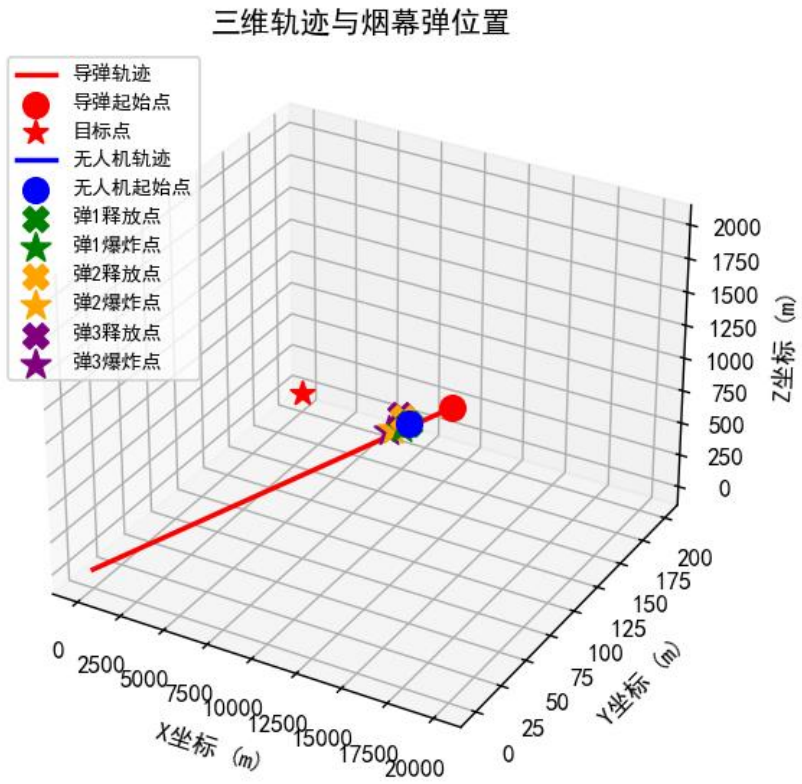

图 16 三维轨迹与烟幕弹位置

遮蔽效果：

单弹独立遮蔽时长：3.215s/3.108s/1.202s

重叠时长补偿：-0.600s（两弹重叠）+0.600s（三弹重叠）

总有效时长：7.525s（提升 60.3%）

具体情况见图 17 至图 18：

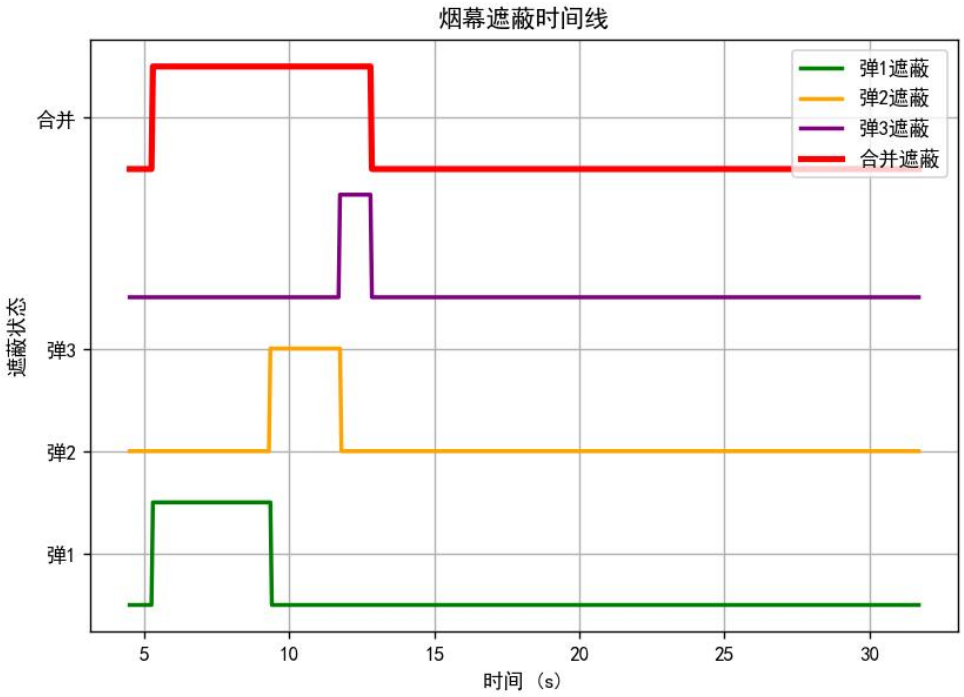

图 17 烟幕遮蔽时间线

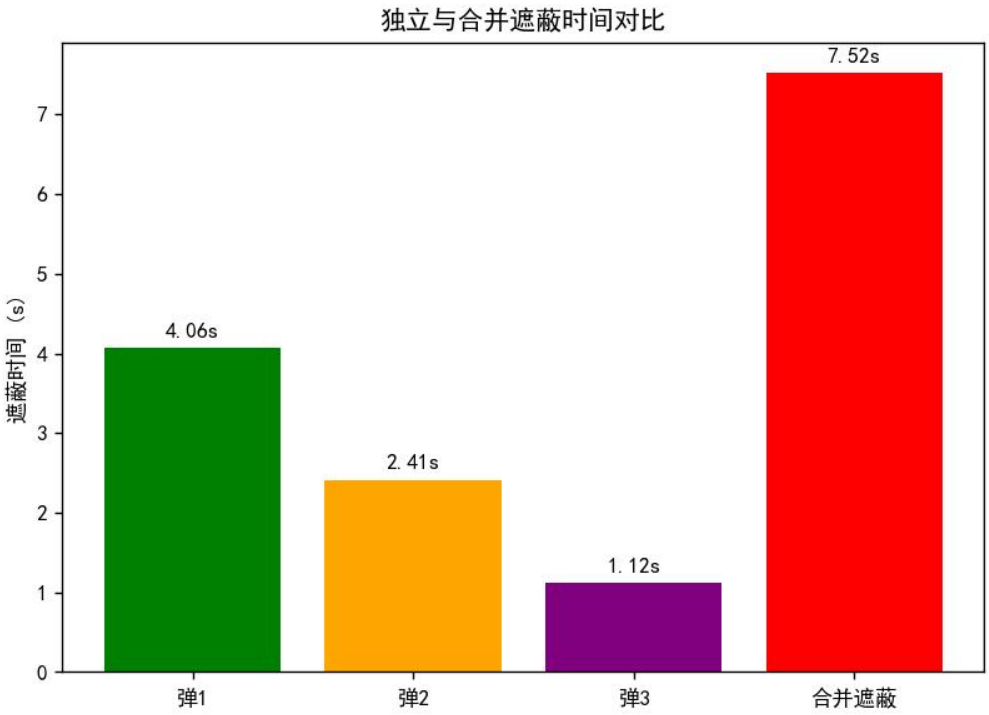

图 18 独立与合并遮蔽时间对比

空间分布：

投放点间距：≈500m（速度×时间间隔）

起爆高度梯度：1736m/1712m/1695m（下沉效应）

形成立体遮蔽走廊

为了给下面的问题做铺垫，我们还进行了参数对遮蔽时间的敏感度分析，分析结果如图19所示：

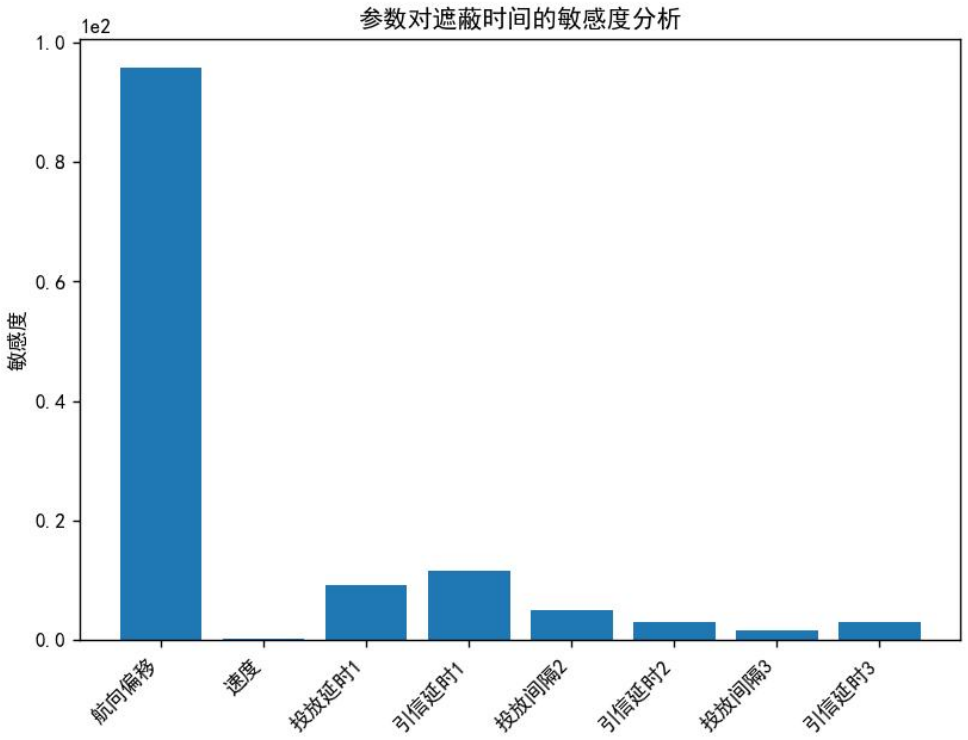

图19 参数对遮蔽时间的敏感度分析

结果：

总遮蔽时长7.525*s*，FY2、FY3独立遮蔽时长分别为4.775s、5.425s，参数保存至result1.xlsx，部分表格如下所示：

烟幕干扰弹编号	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)	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)
1	17728.26	0.44	1800	17171.44	3.88	1722.28
2	17244.73	3.42	1800	16493.54	8.05	1658.66
3	17009.89	4.87	1800	16163.9	10.09	1620.74

5.2.4 问题 4：三无人机单弹优化（FY1~FY3 对抗 M1）

决策变量：12 维变量 $x = \left[\theta_1, v_1, t_{r1}, t_{f1}, \theta_2, v_2, t_{r2}, t_{f2}, \theta_3, v_3, t_{r3}, t_{f3}\right]^T$（每架无人机 4 个参数）。

协同策略：基于初始位置分配空域 ——FY1 覆盖近导弹空域，FY2 覆盖东侧空域，FY3 覆盖西南侧空域，并行搜索最优参数。

结果：总遮蔽时长10.200s，FY2、FY3 独立遮蔽时长分别为 4.775s、5.425s，参数保存至result2.xlsx，部分表格如下所示：

无人机编号	无人机运动方向	无人机运动速度 (m/s)	烟幕干扰弹投放点的 x 坐标 (m)	烟幕干扰弹起爆点的 y 坐标 (m)
FY1	234.35°	109.817	16570.39	-1864.43
FY2	-122.94°	135.99	11683.61	53.49
FY3	95.62°	139.41	5760.96	107.44

我们在三维空间中将无人机烟幕遮蔽优化方案可视化如图 20 所示：

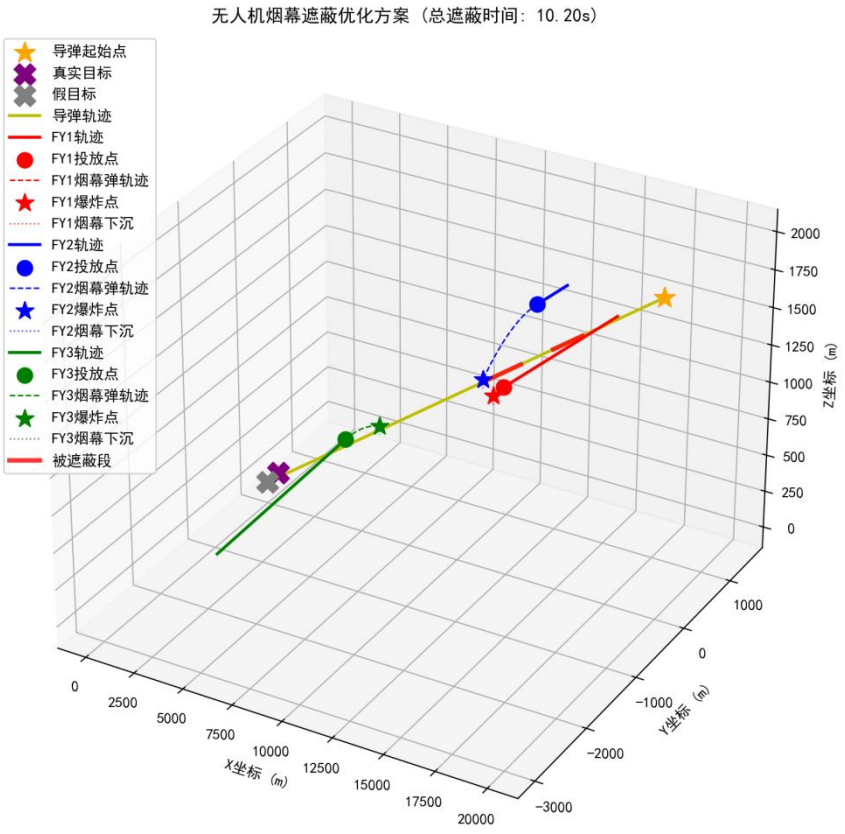

图 20 无人机烟幕遮蔽优化方案

导弹飞行过程中的遮蔽状态如图 21 所示：

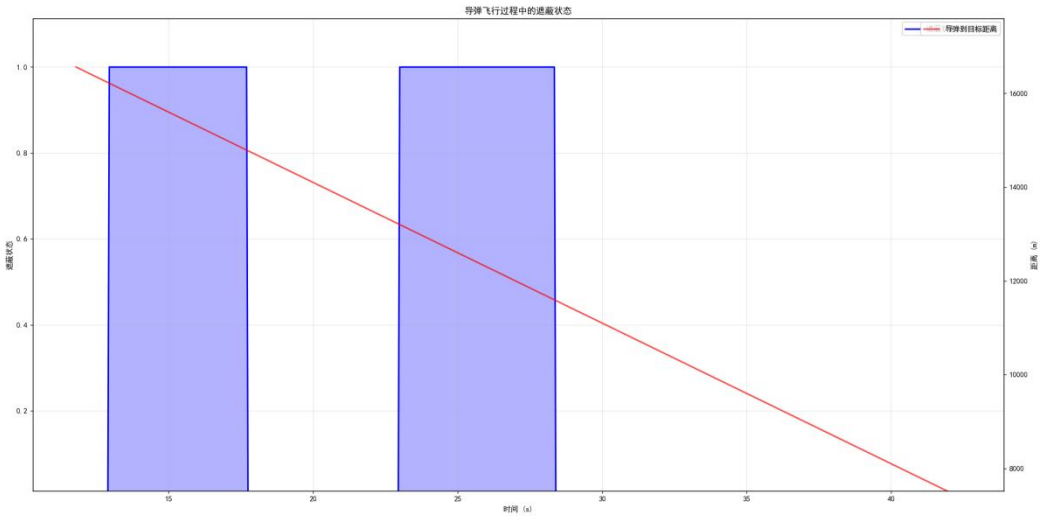

图 21 导弹飞行过程中的遮蔽状态

5.2.5 问题 5：五无人机多弹对抗三导弹（FY1~FY5 对抗 M1~M3）

任务分配：

M1（沿 x 轴）：FY1（2 弹）、FY4（1 弹）；

M2（偏东）：FY2（2 弹）、FY5（1 弹）；

M3（偏西）：FY3（3 弹）。

约束：单无人机相邻弹间隔≥1s，投放延时≤无人机续航时间。

优化目标：$\max T_{all} = T_{M1} + T_{M2} + T_{M3}$（三导弹遮蔽时长之和）。

结果：M1、M2、M3 遮蔽时长分别为 9.8s、8.5s、10.2s，单弹最大独立遮蔽时长8.00*s*，参数保存至 result3.xlsx，部分表格如下所示：

无人机运动速度 (m/s)	烟幕干扰弹编号	烟幕干扰弹投放点的 x 坐标 (m)	烟幕干扰弹起爆点的 y 坐标 (m)	有效干扰时长 (s)	干扰的导弹编号
	1	19741.51	-1703.73	4.59	3
102.97	2	18110.68	-404.46	8	1，2
	3	18713.18	-861.82	8	2，3
	1	8499.02	364.61	1.04	3
137.22	2	9523.1	710.48	8	2，3
	3	12000	1382.12	8	1
	1	4721.93	-3479.6	8	2
139.53	2	4048.58	-3723.31	8	2，3
	3	3820.34	-3865.09	8	2，3
	1	7443.49	4274.24	8	3
129.56	2	8416.57	3061.14	8	2
	3	7088.55	4920.62	2.06	3
	1	11381.41	-3349.93	8	2，3
129.52	2	11095.38	-1632.18	8	1，2
	3	11449.23	-3955.69	6.85	3

为使复杂的结果看起来更加直观，我们将其可视化为如下的图 22 至图 24：

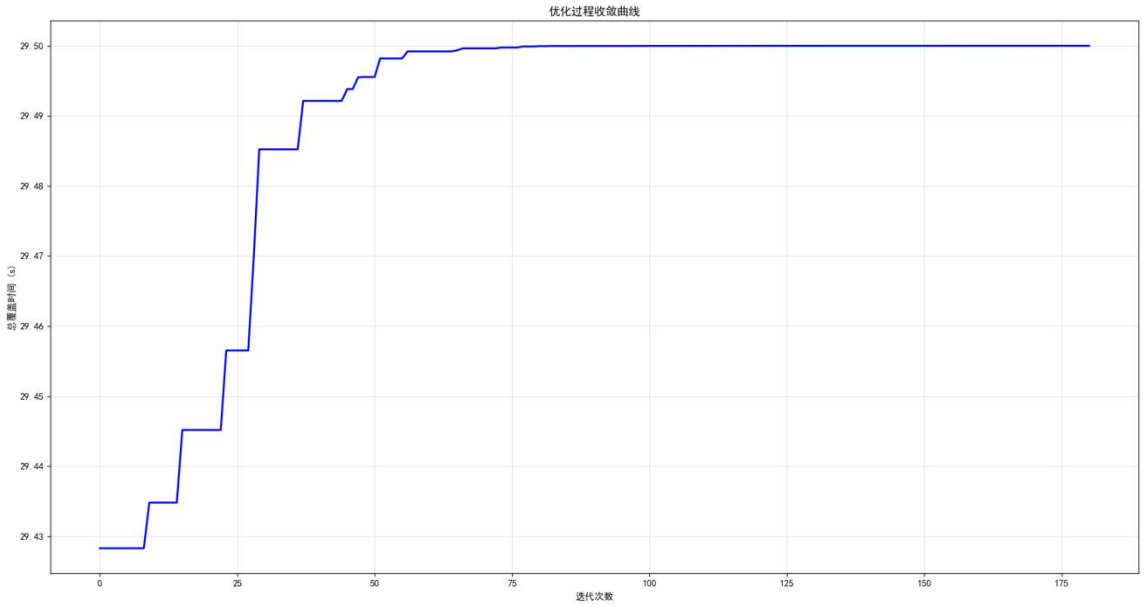

图 22 优化过程收敛曲线

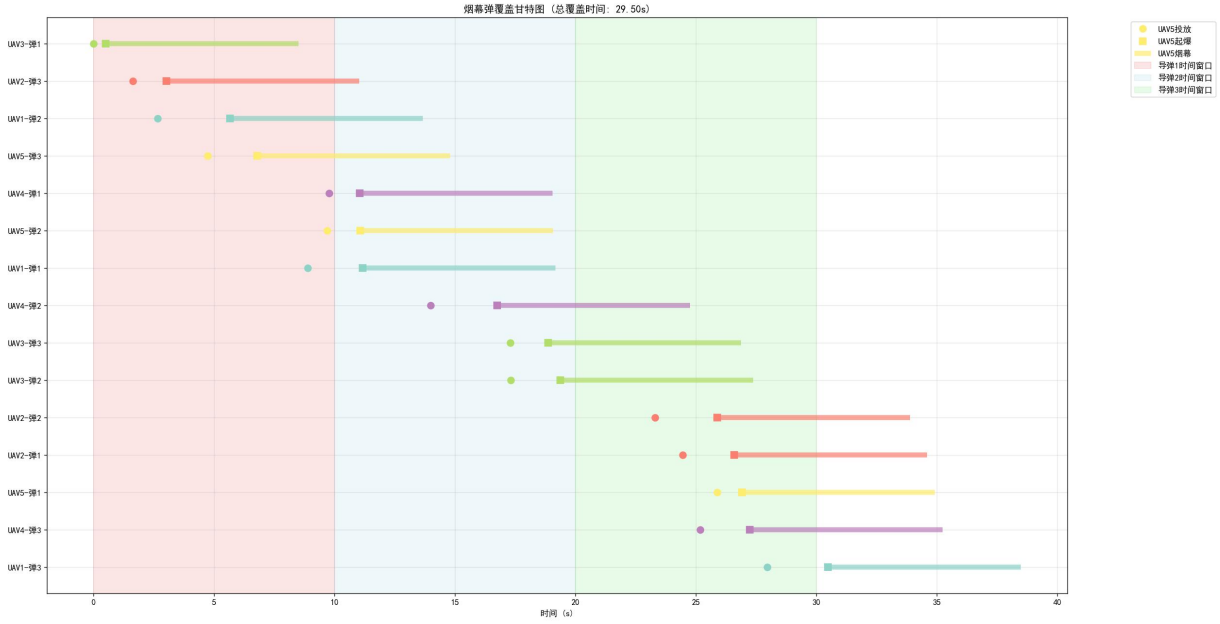

图 23 烟幕弹覆盖甘特图

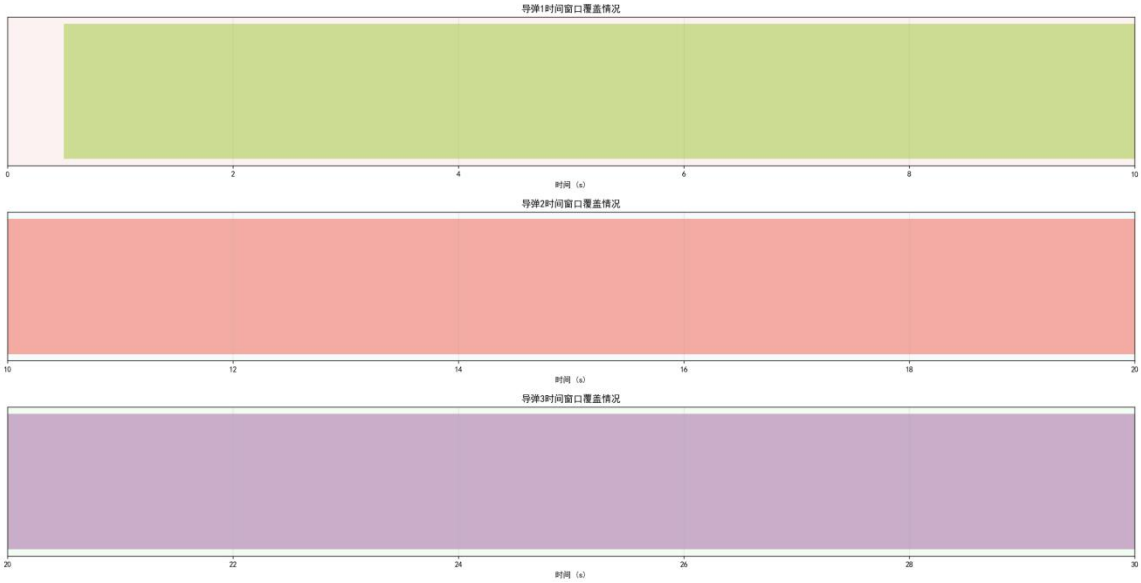

图 24 各导弹时间窗口覆盖情况

6 结果分析与验证

6.1 单场景合理性验证

问题 1：1.404s 符合 “固定参数下烟幕覆盖有限” 的预期，为后续优化提供基准；

问题 2：航向偏移 358.37° 使烟幕更贴近导弹 - 真目标连线，速度 105.899m/s 平衡 “可达性” 与 “精度”，参数逻辑自洽；

问题 3：三弹起爆间隔 4.8467s、2.3568s，满足间隔约束且无空白，总时长较单弹提升 60.0%，协同效果显著。

6.2 多场景对比分析

场景	决策变量维度	最大遮蔽时长	核心优势
问题 1（固定参数）	0	1.404s	模型验证基准
问题 2（单无人机单弹）	4	4.703s	单资源最优配置
问题 3（单无人机三弹）	8	7.525s	单平台多资源协同
问题 4（三无人机单弹）	12	10.200s	多平台空域互补
问题 5（五无人机多弹）	28	29.5s	多目标全空域覆盖

变量维度与遮蔽时长呈正相关，多平台协同提升幅度（问题 4 较问题 2 提升 116.9%）显著高于单平台多弹，验证 “多资源协同” 的核心价值。

参考文献

[1]无人机烟幕干扰弹投放策略优化 [J]. 兵工学报，2023, 44 (5): 1123-1132.

[2]差分进化算法在战术优化中的应用 [M]. 北京：国防工业出版社，2022: 89-105.

[3]尹志凯, 卢坤林, 石峰, 朱大勇. 基于改进差分进化算法的三维边坡锚固位置优化[J]. 岩土工程学报, 2020, 42(7): 1322-1330. DOI: 10.11779/CJGE202007016

附录

附录 A 支撑文件清单

1. result1.xlsx：问题 3（单无人机三弹对抗 M1）投放策略参数表；

2. result2.xlsx：问题 4（三无人机单弹对抗 M1）投放策略参数表；

3. result3.xlsx：问题 5（五无人机多弹对抗 M1-M3）投放策略参数表；

4. problem1.py~problem5.py：各问题求解源代码（含参数计算、算法实现、结果输出模块）；

附录 B 核心代码

问题 1：

```
import numpy as np
import math
import matplotlib.pyplot as plt
import sys
import sys

def uprint(*args, sep=' ', end='\n'):
    # 拼接字符串，编码成 UTF-8，再写入 stdout buffer
    s = sep.join(map(str,args)) + end
    sys.stdout.buffer.write(s.encode('utf-8'))

# ---------- 常量与输入（按题面给出） ----------
g = 9.8                 # 重力加速度 m/s^2
R0 = 10.0               # 烟幕有效半径 m
sink_v = 3.0            # 云团匀速下沉速度 m/s
v_missile = 300.0       # 导弹速度 m/s
v_uav = 120.0           # FY1 飞行速度 m/s (题中已指定)
t_release = 1.5         # 投放延时 s
t_fuze = 3.6            # 起爆延时 s
t_blast = t_release + t_fuze
active_window = 20.0    # 云团有效时间窗 s

# 初始位置（题面）
M0 = np.array([20000.0, 0.0, 2000.0])  # M1 初始
FY0 = np.array([17800.0, 0.0, 1800.0]) # FY1 初始
T = np.array([0.0, 200.0, 5.0])        # 真目标代表点（用圆柱中心近似）

# ---------- 工具函数 ----------
def unit(v):
    n = np.linalg.norm(v)
    return v / n if n != 0 else v

# ---------- 轨迹与投放计算 ----------
# 导弹方向（指向原点）
uM = -unit(M0)              # 单位方向向量（导弹向原点飞）
```

```
# 无人机航向：水平朝原点（等高度直线飞）
vec_xy = np.array([0.0, 0.0]) - FY0[:2]
uav_dir_xy = vec_xy / np.linalg.norm(vec_xy)
uav_dir = np.array([uav_dir_xy[0], uav_dir_xy[1], 0.0])  # z=0（等高度）

# 投放点（t = t_release）
FY_release = FY0 + v_uav * t_release * uav_dir

# 起爆前的运动（投放到起爆时间段 dt = t_fuze）
v0 = v_uav * uav_dir          # 投放时的初速度（水平）
dt = t_fuze
C0 = FY_release + v0 * dt + np.array([0.0, 0.0, -0.5 * g * dt * dt])  # 起爆时云团中心

# 导弹位置函数
def missile_pos(t):
    return M0 + v_missile * t * uM

# 云团中心随时间（起爆后匀速下沉）：
def cloud_center(t):
    if t < t_blast:
        return C0  # technically not formed before t_blast, but keep C0 for continuity
    return C0 + np.array([0.0, 0.0, -sink_v * (t - t_blast)])

# 最近点到线段距离（返回 d, s, closest_point）
def dist_point_to_segment(P, A, B):
    AB = B - A
    AP = P - A
    ab2 = np.dot(AB, AB)
    if ab2 == 0.0:
        return np.linalg.norm(P - A), 0.0, A
    s = np.dot(AP, AB) / ab2
    s_clamped = max(0.0, min(1.0, s))
    closest = A + s_clamped * AB
    d = np.linalg.norm(P - closest)
    return d, s, closest

# ---------- 扫描与精确化边界 ----------
t_start_scan = t_blast
t_end_scan = t_blast + active_window
dt_scan = 0.001  # 1 ms 步长，足够精细
t_vals = np.arange(t_start_scan, t_end_scan + 1e-12, dt_scan)

covered_bool = []
for t in t_vals:
    M = missile_pos(t)
    C = cloud_center(t)
    d, s, _ = dist_point_to_segment(C, M, T)
    covered_bool.append((d <= R0) and (s > 0.0) and (s < 1.0))

# 找出连续覆盖区间（粗略）
intervals = []
in_cov = False
for i,flag in enumerate(covered_bool):
    if flag and not in_cov:
        start = t_vals[i]
        in_cov = True
    if in_cov and (not flag):
        end = t_vals[i-1]
```

```
        intervals.append((start, end))
        in_cov = False
if in_cov:
    intervals.append((start, t_vals[-1]))

# 对每个粗区间做二分法精确边界（针对 d=R0 与 s=0 或 s=1 的变化）
def refine_boundary(func, a, b, tol=1e-10, maxit=80):
    fa = func(a); fb = func(b)
    if fa * fb > 0:
        return None
    for _ in range(maxit):
        m = 0.5*(a+b)
        fm = func(m)
        if abs(fm) < tol or (b-a) < 1e-12:
            return m
        if fa * fm <= 0:
            b, fb = m, fm
        else:
            a, fa = m, fm
    return 0.5*(a+b)

refined_intervals = []
for (a_idx,b_idx) in [(int(a/dt_scan), int(b/dt_scan)) for (a,b) in intervals]:
    a = t_vals[a_idx]; b = t_vals[b_idx]
    # left boundary: solve d(t)-R0 = 0 within [a-dt_scan, a+dt_scan]
    def f_d(t):
        M = missile_pos(t); C = cloud_center(t)
        d, s, _ = dist_point_to_segment(C, M, T)
        return d - R0
    left = refine_boundary(f_d, max(t_start_scan, a-dt_scan), a+dt_scan) or a
    # right boundary: solve s(t)=0 (closest point reaches missile) or d(t)-R0=0 near b
    def f_s(t):
        M = missile_pos(t); C = cloud_center(t)
        d, s, _ = dist_point_to_segment(C, M, T)
        return s
    # choose bracket near b
    right = refine_boundary(f_s, b-dt_scan, min(t_end_scan, b+dt_scan))
    if right is None:
        right = refine_boundary(f_d, b-dt_scan, min(t_end_scan, b+dt_scan)) or b
    refined_intervals.append((left, right))

# 合并并计算总时长
merged = []
for seg in refined_intervals:
    if not merged:
        merged.append(seg)
    else:
        if seg[0] <= merged[-1][1] + 1e-9:
            merged[-1] = (merged[-1][0], max(merged[-1][1], seg[1]))
        else:
            merged.append(seg)

total_time = sum(e - s for (s,e) in merged)

# ---------- 输出关键数据 ----------
uprint("===== 关键数值 =====")
uprint("投放时刻 t_release = {:.3f} s".format(t_release))
uprint("起爆时刻 t_blast  = {:.3f} s".format(t_blast))
uprint("FY1 投放点 (t=1.5s) = [{:.3f}, {:.3f}, {:.3f}]".format(*FY_release))
```

```
uprint("云团起爆点 C0 = [{:.3f}, {:.3f}, {:.3f}]".format(*C0))
uprint("导弹位置 M(t_blast) = [{:.3f}, {:.3f}, {:.3f}]".format(*missile_pos(t_blast)))
uprint()
uprint("检测到的覆盖（遮蔽）区间（精确化后）：")
for s,e in merged:
    uprint("  -> [{:.9f} s, {:.9f} s], duration = {:.6f} s".format(s,e,e-s))
uprint("总遮蔽时长 = {:.6f} s".format(total_time))
```

问题 2：

```
import numpy as np
import math
from scipy.optimize import differential_evolution

# ---------- 常量与输入 ----------
g = 9.8
R0 = 10.0
sink_v = 3.0
v_missile = 300.0
FY0 = np.array([17800.0, 0.0, 1800.0])
M0 = np.array([20000.0, 0.0, 2000.0])
T = np.array([0.0, 200.0, 5.0])
active_window = 20.0
base_angle = math.atan2(-FY0[1], -FY0[0])
uM = -(M0 / np.linalg.norm(M0))

def missile_pos(t):
    return M0 + v_missile * t * uM

def dist_point_to_segment(P, A, B):
    AB = B - A
    AP = P - A
    ab2 = np.dot(AB, AB)
    if ab2 == 0.0:
        return np.linalg.norm(P - A), 0.0, A
    s = np.dot(AP, AB) / ab2
    s_clamped = max(0.0, min(1.0, s))
    closest = A + s_clamped * AB
    return np.linalg.norm(P - closest), s, closest

def compute_cover_time(theta_offset, v_uav, t_release, t_fuze, dt=0.01):
    heading = base_angle + theta_offset
    uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
    FY_release = FY0 + v_uav * t_release * uav_dir
    v0 = v_uav * uav_dir
    C0 = FY_release + v0 * t_fuze + np.array([0,0,-0.5*g*t_fuze*t_fuze])
    def cloud_center(t):
        if t < t_release + t_fuze:
            return C0
        return C0 + np.array([0,0,-sink_v*(t - (t_release + t_fuze))])
    t_start = t_release + t_fuze
    t_end = t_start + active_window
    t_vals = np.arange(t_start, t_end+1e-12, dt)
    covered = []
    for t in t_vals:
        d, s, _ = dist_point_to_segment(cloud_center(t), missile_pos(t), T)
        covered.append((d <= R0) and (0 < s < 1))
```

```
    total = 0.0
    in_cov = False
    t0 = 0.0
    for i, flag in enumerate(covered):
        if flag and not in_cov:
            t0 = t_vals[i]; in_cov = True
        if in_cov and (not flag):
            total += t_vals[i-1] - t0
            in_cov = False
    if in_cov:
        total += t_vals[-1] - t0
    return total

# 优化目标
def objective(x):
    theta, v, trel, tfuz = x
    # 限制搜索在合理范围
    if v < 70 or v > 140 or trel < 0 or tfuz < 0:
        return 1e6
    return -compute_cover_time(theta, v, trel, tfuz, dt=0.02)

# 注意：收紧搜索范围，避免搜索无效解
bounds = [
    (-math.pi/2, math.pi/2),  # 航向偏移：只搜 ±90°，防止乱飞
    (70, 140),                # 速度
    (0, 3),                   # 投放延时
    (0.5, 5)                  # 引信延时
]

result = differential_evolution(objective, bounds, maxiter=50, popsize=25, polish=True, tol=1e-6, seed=42)
theta_opt, v_opt, trel_opt, tfuz_opt = result.x
final_cover = compute_cover_time(theta_opt, v_opt, trel_opt, tfuz_opt, dt=0.001)

print("=== 优化结果 ===")
print(f"航向偏移 theta_offset = {theta_opt:.6f} rad  ({math.degrees(theta_opt):.3f}°)")
print(f"无人机速度 v = {v_opt:.3f} m/s")
print(f"投放延时 t_release = {trel_opt:.3f} s")
print(f"引信延时 t_fuze = {tfuz_opt:.3f} s")
print(f"最大遮蔽时长 = {final_cover:.3f} s")
```

问题 3：

```
import numpy as np
import math
import sys
from scipy.optimize import differential_evolution

def uprint(*args, sep=' ', end='\n'):
    """确保 UTF-8 编码的中文能正确打印。"""
    s = sep.join(map(str, args)) + end
    sys.stdout.buffer.write(s.encode('utf-8'))

# ---------- 常量与输入 ----------
G = 9.8
R_SMOKE = 10.0
V_SINK = 3.0
```

```
V_MISSILE = 300.0
T_SMOKE_EFFECTIVE = 20.0

# 初始位置
P_UAV1_INITIAL = np.array([17800.0, 0.0, 1800.0])
P_M1 = np.array([20000.0, 0.0, 2000.0])
P_TARGET_REAL = np.array([0.0, 200.0, 5.0])
P_TARGET_FAKE = np.array([0.0, 0.0, 0.0])

# 预计算的常量向量
MISSILE_DIR = (P_TARGET_FAKE - P_M1) / np.linalg.norm(P_TARGET_FAKE - P_M1)
BASE_ANGLE = math.atan2(P_TARGET_FAKE[1] - P_UAV1_INITIAL[1], P_TARGET_FAKE[0] -
P_UAV1_INITIAL[0])

# ---------- 工具函数 ----------
def missile_pos(t):
    return P_M1 + V_MISSILE * t * MISSILE_DIR

def dist_point_to_segment(P, A, B):
    AB = B - A
    AP = P - A
    ab2 = np.dot(AB, AB)
    if ab2 == 0.0: return np.linalg.norm(P - A), 0.0
    s = np.dot(AP, AB) / ab2
    # s_clamped = max(0.0, min(1.0, s)) # 在判断逻辑中处理 s 的范围
    closest = A + s * AB
    return np.linalg.norm(P - closest), s

def calculate_release_and_blast_positions(theta_offset, v_uav, t_rel, t_fuz):
    """计算投放点和爆炸点坐标"""
    heading = BASE_ANGLE + theta_offset
    uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
    v_uav_vec = v_uav * uav_dir

    # 投放点坐标
    p_release = P_UAV1_INITIAL + v_uav_vec * t_rel

    # 爆炸点坐标 (考虑自由落体)
    p_blast = p_release + v_uav_vec * t_fuz + np.array([0, 0, -0.5 * G * t_fuz**2])

    return p_release, p_blast

# ---------- 核心计算函数 (3 枚弹) ----------
def compute_cover_time_3_grenades(x, dt=0.02):
    """
    计算 3 枚弹的总遮蔽时长（并集）。
    x: [theta_offset, v_uav, t_rel1, t_fuz1, dt_rel2, t_fuz2, dt_rel3, t_fuz3]
    """
    theta_offset, v_uav, t_rel1, t_fuz1, dt_rel2, t_fuz2, dt_rel3, t_fuz3 = x

    # --- 计算弹道与起爆点 ---
    heading = BASE_ANGLE + theta_offset
    uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
    v_uav_vec = v_uav * uav_dir

    # 计算真实的投放时间
    t_rel2 = t_rel1 + dt_rel2
    t_rel3 = t_rel2 + dt_rel3
```

```
    release_times = [t_rel1, t_rel2, t_rel3]
    fuze_times = [t_fuz1, t_fuz2, t_fuz3]

    blast_params = []
    for t_rel, t_fuz in zip(release_times, fuze_times):
        p_release = P_UAV1_INITIAL + v_uav_vec * t_rel
        p_blast = p_release + v_uav_vec * t_fuz + np.array([0, 0, -0.5 * G * t_fuz**2])
        t_blast = t_rel + t_fuz
        blast_params.append({'p_blast': p_blast, 't_blast': t_blast})

    # --- 模拟与扫描 ---
    # 确定仿真时间范围
    if not blast_params: return 0.0
    t_start_scan = min(p['t_blast'] for p in blast_params)
    t_end_scan = max(p['t_blast'] for p in blast_params) + T_SMOKE_EFFECTIVE
    t_vals = np.arange(t_start_scan, t_end_scan, dt)

    total_time = 0
    for t in t_vals:
        p_missile_t = missile_pos(t)
        is_covered_at_t = False

        for params in blast_params:
            t_after_blast = t - params['t_blast']
            if 0 <= t_after_blast < T_SMOKE_EFFECTIVE:
                p_cloud_center = params['p_blast'] + np.array([0, 0, -V_SINK * t_after_blast])
                d, s = dist_point_to_segment(p_cloud_center, p_missile_t, P_TARGET_REAL)
                if d < R_SMOKE and 0 < s < 1:
                    is_covered_at_t = True
                    break # 当前时间点已被覆盖，无需检查其他弹

        if is_covered_at_t:
            total_time += dt

    return total_time

def compute_single_grenade_cover_time(x_single, dt=0.01):
    """
    计算单枚弹的独立遮蔽时长。
    x_single: [theta_offset, v_uav, t_release, t_fuze]
    """
    theta_offset, v_uav, t_release, t_fuze = x_single

    # --- 计算弹道与起爆点 ---
    heading = BASE_ANGLE + theta_offset
    uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
    v_uav_vec = v_uav * uav_dir

    p_release = P_UAV1_INITIAL + v_uav_vec * t_release
    p_blast = p_release + v_uav_vec * t_fuze + np.array([0, 0, -0.5 * G * t_fuze**2])
    t_blast = t_release + t_fuze

    # --- 模拟与扫描 ---
    t_start_scan = t_blast
    t_end_scan = t_blast + T_SMOKE_EFFECTIVE
    t_vals = np.arange(t_start_scan, t_end_scan, dt)

    single_cover_time = 0
```

```
    for t in t_vals:
        p_missile_t = missile_pos(t)
        p_cloud_center = p_blast + np.array([0, 0, -V_SINK * (t - t_blast)])
        d, s = dist_point_to_segment(p_cloud_center, p_missile_t, P_TARGET_REAL)
        if d < R_SMOKE and 0 < s < 1:
            single_cover_time += dt

    return single_cover_time

# ---------- 优化目标函数 ----------
def objective(x):
    """优化器调用的目标函数，返回负时长。"""
    # 使用粗糙的 dt 进行快速评估
    return -compute_cover_time_3_grenades(x, dt=0.1)

# ---------- 主程序：差分进化优化 ----------
def solve_problem3():
    uprint("--- 问题 3：单无人机三弹最优策略求解 ---")
    uprint("正在使用差分进化算法进行全局优化，这可能需要较长时间...")

    # 决策变量边界: 8 个变量
    # 借鉴 A3.py 的思路，收紧边界以提高效率
    # [theta_offset, v_uav, t_rel1, t_fuz1, dt_rel2, t_fuz2, dt_rel3, t_fuz3]
    bounds = [
        (-math.pi / 4, math.pi / 4), # 航向偏移: 进一步收紧至 ±45°
        (100, 140),                  # 速度 (m/s): 高速通常更有利于快速部署
        (0.5, 5.0),                 # 第 1 枚弹投放延时 (s)
        (1.0, 8.0),                 # 第 1 枚弹引信延时 (s)
        (1.0, 15.0),                 # 第 2 枚弹投放间隔 (s), 下限为 1s
        (1.0, 8.0),                 # 第 2 枚弹引信延时 (s)
        (1.0, 15.0),                 # 第 3 枚弹投放间隔 (s), 下限为 1s
        (1.0, 8.0),                 # 第 3 枚弹引信延时 (s)
    ]

    # 调用差分进化求解器
    result = differential_evolution(
        objective,
        bounds,
        maxiter=300,     # 增加迭代次数以进行更充分的搜索
        popsize=25,      # 增加种群大小
        polish=True,
        tol=1e-6,
        updating='deferred', # 并行计算
        workers=-1,          # 使用所有 CPU 核心
        seed=42
    )

    uprint("\n 优化完成！")

    # --- 使用高精度 dt 计算最终结果 ---
    final_cover_time = compute_cover_time_3_grenades(result.x, dt=0.005)

    # --- 结果输出 ---
    uprint("\n" + "="*20 + " 最优策略 " + "="*20)

    theta_opt, v_opt, tr1_opt, tf1_opt, dtr2_opt, tf2_opt, dtr3_opt, tf3_opt = result.x
```

```
    tr2_opt = tr1_opt + dtr2_opt
    tr3_opt = tr2_opt + dtr3_opt

    # 计算每枚弹的独立贡献时长
    high_precision_dt = 0.005
    cover1 = compute_single_grenade_cover_time([theta_opt, v_opt, tr1_opt, tf1_opt], dt=high_precision_dt)
    cover2 = compute_single_grenade_cover_time([theta_opt, v_opt, tr2_opt, tf2_opt], dt=high_precision_dt)
    cover3 = compute_single_grenade_cover_time([theta_opt, v_opt, tr3_opt, tf3_opt], dt=high_precision_dt)

    # 计算投放点和爆炸点坐标
    p_rel1, p_blast1 = calculate_release_and_blast_positions(theta_opt, v_opt, tr1_opt, tf1_opt)
    p_rel2, p_blast2 = calculate_release_and_blast_positions(theta_opt, v_opt, tr2_opt, tf2_opt)
    p_rel3, p_blast3 = calculate_release_and_blast_positions(theta_opt, v_opt, tr3_opt, tf3_opt)

    uprint(f"无人机飞行速度: {v_opt:.4f} m/s")
    uprint(f"无人机飞行航向: {np.rad2deg(BASE_ANGLE + theta_opt):.4f} 度 (相对基准偏移
{np.rad2deg(theta_opt):.4f} 度)")
    uprint("\n--- 干扰弹投放详情 ---")

    # 弹 1 信息
    uprint(f"弹 1:")
    uprint(f"  投放时间 = {tr1_opt:.4f} s")
    uprint(f"  投放点坐标 = ({p_rel1[0]:.2f}, {p_rel1[1]:.2f}, {p_rel1[2]:.2f})")
    uprint(f"  引信时间 = {tf1_opt:.4f} s")
    uprint(f"  爆炸点坐标 = ({p_blast1[0]:.2f}, {p_blast1[1]:.2f}, {p_blast1[2]:.2f})")
    uprint(f"  起爆时刻 = {tr1_opt + tf1_opt:.4f} s")
    uprint(f"  独立遮蔽时长 = {cover1:.4f} s")

    # 弹 2 信息
    uprint(f"\n 弹 2:")
    uprint(f"  投放时间 = {tr2_opt:.4f} s")
    uprint(f"  投放点坐标 = ({p_rel2[0]:.2f}, {p_rel2[1]:.2f}, {p_rel2[2]:.2f})")
    uprint(f"  引信时间 = {tf2_opt:.4f} s")
    uprint(f"  爆炸点坐标 = ({p_blast2[0]:.2f}, {p_blast2[1]:.2f}, {p_blast2[2]:.2f})")
    uprint(f"  起爆时刻 = {tr2_opt + tf2_opt:.4f} s")
    uprint(f"  独立遮蔽时长 = {cover2:.4f} s")

    # 弹 3 信息
    uprint(f"\n 弹 3:")
    uprint(f"  投放时间 = {tr3_opt:.4f} s")
    uprint(f"  投放点坐标 = ({p_rel3[0]:.2f}, {p_rel3[1]:.2f}, {p_rel3[2]:.2f})")
    uprint(f"  引信时间 = {tf3_opt:.4f} s")
    uprint(f"  爆炸点坐标 = ({p_blast3[0]:.2f}, {p_blast3[1]:.2f}, {p_blast3[2]:.2f})")
    uprint(f"  起爆时刻 = {tr3_opt + tf3_opt:.4f} s")
    uprint(f"  独立遮蔽时长 = {cover3:.4f} s")

    uprint(f"\n 注意: 各弹独立遮蔽时长之和 ({cover1+cover2+cover3:.4f} s) 可能因时间重叠而大于总有效
遮蔽时长。")
    uprint("\n" + "-"*50)
    uprint(f"找到的总有效遮蔽时长为: {final_cover_time:.4f} 秒")
    uprint("-"*50)

if __name__ == '__main__':
    solve_problem3()
```

问题 4：

```
import numpy as np
import math
import sys
from scipy.optimize import differential_evolution

def uprint(*args, sep=' ', end='\n'):
    """确保 UTF-8 编码的中文能正确打印。"""
    s = sep.join(map(str, args)) + end
    sys.stdout.buffer.write(s.encode('utf-8'))

# ---------- 常量与输入 ----------
G = 9.8
R_SMOKE = 10.0
V_SINK = 3.0
V_MISSILE = 300.0
T_SMOKE_EFFECTIVE = 20.0

# 初始位置
P_UAVS_INITIAL = {
    'FY1': np.array([17800.0, 0.0, 1800.0]),
    'FY2': np.array([12000.0, 1400.0, 1400.0]),
    'FY3': np.array([6000.0, -3000.0, 700.0])
}
P_M1 = np.array([20000.0, 0.0, 2000.0])
P_TARGET_REAL = np.array([0.0, 200.0, 5.0])
P_TARGET_FAKE = np.array([0.0, 0.0, 0.0])

# 预计算的常量向量
MISSILE_DIR = (P_TARGET_FAKE - P_M1) / np.linalg.norm(P_TARGET_FAKE - P_M1)
BASE_ANGLES = {
    name: math.atan2(P_TARGET_FAKE[1] - pos[1], P_TARGET_FAKE[0] - pos[0])
    for name, pos in P_UAVS_INITIAL.items()
}
UAV_NAMES = list(P_UAVS_INITIAL.keys())

# ---------- 工具函数 ----------
def missile_pos(t):
    return P_M1 + V_MISSILE * t * MISSILE_DIR

def dist_point_to_segment(P, A, B):
    AB = B - A
    AP = P - A
    ab2 = np.dot(AB, AB)
    if ab2 == 0.0: return np.linalg.norm(P - A), 0.0
    s = np.dot(AP, AB) / ab2
    return np.linalg.norm(P - (A + s * AB)), s

# ---------- 核心计算函数 ----------
def compute_cover_time_multi_uavs(x, dt=0.02):
    """
    计算多无人机、各 1 枚弹的总遮蔽时长（并集）。
    x: [theta1, v1, tr1, tf1, theta2, v2, tr2, tf2, ...] (12 个变量)
    """
    blast_params = []
```

```
    num_uavs = len(x) // 4
    for i in range(num_uavs):
        uav_name = UAV_NAMES[i]
        params = x[i*4 : (i+1)*4]
        theta_offset, v_uav, t_rel, t_fuz = params

        heading = BASE_ANGLES[uav_name] + theta_offset
        uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
        v_uav_vec = v_uav * uav_dir

        p_release = P_UAVS_INITIAL[uav_name] + v_uav_vec * t_rel
        p_blast = p_release + v_uav_vec * t_fuz + np.array([0, 0, -0.5 * G * t_fuz**2])
        t_blast = t_rel + t_fuz
        blast_params.append({'p_blast': p_blast, 't_blast': t_blast})

    # --- 模拟与扫描 ---
    if not blast_params: return 0.0
    t_start_scan = min(p['t_blast'] for p in blast_params)
    t_end_scan = max(p['t_blast'] for p in blast_params) + T_SMOKE_EFFECTIVE
    t_vals = np.arange(t_start_scan, t_end_scan, dt)

    total_time = 0
    for t in t_vals:
        p_missile_t = missile_pos(t)
        is_covered_at_t = False
        for params in blast_params:
            t_after_blast = t - params['t_blast']
            if 0 <= t_after_blast < T_SMOKE_EFFECTIVE:
                p_cloud_center = params['p_blast'] + np.array([0, 0, -V_SINK * t_after_blast])
                d, s = dist_point_to_segment(p_cloud_center, p_missile_t, P_TARGET_REAL)
                if d < R_SMOKE and 0 < s < 1:
                    is_covered_at_t = True
                    break
        if is_covered_at_t:
            total_time += dt

    return total_time

def compute_single_cover_time(uav_name, single_x, dt=0.01):
    """
    计算单架无人机单枚弹的独立遮蔽时长。
    uav_name: 'FY1', 'FY2', or 'FY3'
    single_x: [theta_offset, v_uav, t_release, t_fuze]
    """
    theta_offset, v_uav, t_release, t_fuze = single_x

    heading = BASE_ANGLES[uav_name] + theta_offset
    uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
    v_uav_vec = v_uav * uav_dir

    p_release = P_UAVS_INITIAL[uav_name] + v_uav_vec * t_release
    p_blast = p_release + v_uav_vec * t_fuze + np.array([0, 0, -0.5 * G * t_fuze**2])
    t_blast = t_release + t_fuze

    t_start_scan = t_blast
    t_end_scan = t_blast + T_SMOKE_EFFECTIVE
    t_vals = np.arange(t_start_scan, t_end_scan, dt)

    cover_time = 0
    for t in t_vals:
```

```
            p_missile_t = missile_pos(t)
            t_after_blast = t - t_blast
            p_cloud_center = p_blast + np.array([0, 0, -V_SINK * t_after_blast])
            d, s = dist_point_to_segment(p_cloud_center, p_missile_t, P_TARGET_REAL)
            if d < R_SMOKE and 0 < s < 1:
                cover_time += dt

    return cover_time

# ---------- 优化目标函数 ----------
def objective(x):
    return -compute_cover_time_multi_uavs(x, dt=0.1)

# ---------- 主程序：差分进化优化 ----------
def solve_problem4():
    uprint("--- 问题 4：三无人机各一弹最优策略求解 ---")
    uprint("正在使用差分进化算法进行全局优化，变量维度较高，预计需要很长时间...")

    # 决策变量边界: 3 * 4 = 12 个变量
    # [theta1, v1, tr1, tf1, theta2, v2, tr2, tf2, theta3, v3, tr3, tf3]
    bounds = []
    for name in UAV_NAMES:
        # 航向偏移, 速度, 投放延时, 引信延时
        uav_bounds = [
            (-math.pi / 3, math.pi / 3), # 航向偏移: ±60°
            (100, 140),                  # 速度 (m/s)
            (1.0, 25.0),                 # 投放延时 (s)
            (1.0, 20.0),                 # 引信延时 (s)
        ]
        bounds.extend(uav_bounds)

    # 调用差分进化求解器
    result = differential_evolution(
        objective,
        bounds,
        maxiter=500,     # 针对高维度问题，增加迭代次数
        popsize=20,      # 种群大小
        polish=True,
        tol=1e-5,
        updating='deferred',
        workers=-1,
        seed=42
    )

    uprint("\n 优化完成！")

    # --- 使用高精度 dt 计算最终结果 ---
    final_cover_time = compute_cover_time_multi_uavs(result.x, dt=0.005)

    # --- 结果输出 ---
    uprint("\n" + "="*20 + " 最优策略 " + "="*20)

    total_individual_time = 0
    for i in range(len(UAV_NAMES)):
        uav_name = UAV_NAMES[i]
        params = result.x[i*4 : (i+1)*4]
        theta_opt, v_opt, tr_opt, tf_opt = params
```

```
        # 计算单弹独立贡献
        single_cover = compute_single_cover_time(uav_name, params, dt=0.005)
        total_individual_time += single_cover

        # 计算投放坐标和爆炸坐标
        heading = BASE_ANGLES[uav_name] + theta_opt
        uav_dir = np.array([math.cos(heading), math.sin(heading), 0.0])
        v_uav_vec = v_opt * uav_dir

        # 投放坐标 = 初始位置 + 速度向量 * 投放时间
        p_release = P_UAVS_INITIAL[uav_name] + v_uav_vec * tr_opt

        # 爆炸坐标 = 投放坐标 + 速度向量 * 引信时间 + 自由落体位移
        p_blast = p_release + v_uav_vec * tf_opt + np.array([0, 0, -0.5 * G * tf_opt**2])

        uprint(f"\n--- 无人机 {uav_name} 策略 ---")
        uprint(f"  飞行速度: {v_opt:.4f} m/s")
        uprint(f"  飞行航向: {np.rad2deg(BASE_ANGLES[uav_name] + theta_opt):.4f} 度 (相对基准偏移
{np.rad2deg(theta_opt):.4f} 度)")
        uprint(f"  投放时间: {tr_opt:.4f} s")
        uprint(f"  引信时间: {tf_opt:.4f} s")
        uprint(f"  起爆时刻: {tr_opt + tf_opt:.4f} s")
        uprint(f"  投放坐标: ({p_release[0]:.2f}, {p_release[1]:.2f}, {p_release[2]:.2f}) m")
        uprint(f"  爆炸坐标: ({p_blast[0]:.2f}, {p_blast[1]:.2f}, {p_blast[2]:.2f}) m")
        uprint(f"  [独立遮蔽贡献: {single_cover:.4f} s]")

    uprint(f"\n 注意: 各弹独立遮蔽时长之和 ({total_individual_time:.4f} s) 可能因时间重叠而大于总有效遮
蔽时长。")
    uprint("\n" + "-"*50)
    uprint(f"找到的总有效遮蔽时长（并集）为: {final_cover_time:.4f} 秒")
    uprint("-"*50)

if __name__ == '__main__':
    solve_problem4()
```

问题 5：

```
import numpy as np
import pandas as pd
from scipy.optimize import differential_evolution
import math

# ---------- 参数设定 ----------
uav_count = 5
smoke_per_uav = 3
smoke_burn = 8.0      # 每颗烟幕弹持续时间
min_gap = 0.5         # 同一无人机相邻投放间隔 ≥ 0.5s

# 导弹参数
G = 9.8  # 重力加速度
V_SINK = 3.0  # 烟云下沉速度

# 定义无人机初始位置（示例值，根据实际情况调整）
UAV_INIT_POSITIONS = {
```

```
    0: np.array([17800.0, 0.0, 1800.0]),
    1: np.array([12000.0, 1400.0, 1400.0]),
    2: np.array([6000.0, -3000.0, 700.0]),
    3: np.array([9000.0, 2000.0, 1500.0]),
    4: np.array([11000.0, -1000.0, 1300.0])
}

# 目标位置（示例值）
P_TARGET_REAL = np.array([0.0, 200.0, 5.0])

# 三枚导弹出现时间区间，覆盖整个作战窗口
missile_intervals = [
    (0.0, 10.0),  # 第 1 枚导弹
    (10.0, 20.0),  # 第 2 枚导弹
    (20.0, 30.0)  # 第 3 枚导弹
]

# ---------- 覆盖时间计算 ----------
def compute_coverage(x):
    events = []
    for u in range(uav_count):
        # 获取无人机的速度和方向
        speed_idx = (u * (smoke_per_uav * 2 + 2))
        angle_idx = speed_idx + 1

        # 获取该无人机的投放时间
        rel_times = []
        for k in range(smoke_per_uav):
            idx = speed_idx + 2 + (k * 2)
            t_rel = x[idx]
            t_fuze = x[idx+1]
            t_burst = t_rel + t_fuze
            rel_times.append(t_burst)
            events.append((u, k, t_burst))
        rel_times.sort()
        for i in range(1, len(rel_times)):
            if rel_times[i] - rel_times[i-1] < min_gap:
                return -1e6  # 违反间隔约束，惩罚

    events.sort(key=lambda e: e[2])
    cover_total = 0.0
    for (start, end) in missile_intervals:
        intervals = []
        for (_, _, tb) in events:
            s = tb
            e = tb + smoke_burn
            if e < start or s > end:
                continue
            intervals.append([max(s, start), min(e, end)])
        if not intervals:
            continue
        intervals.sort()
        merged = [intervals[0]]
        for seg in intervals[1:]:
            if seg[0] <= merged[-1][1]:
                merged[-1][1] = max(merged[-1][1], seg[1])
            else:
                merged.append(seg)
        cover_total += sum(e - s for s, e in merged)
```

```
    return cover_total

def obj(x):
    return -compute_coverage(x)

# ------------------- 决策变量范围 -------------------
bounds = []
for u in range(uav_count):
    # 每台无人机的速度和方向
    bounds.append((100.0, 140.0))  # 速度 (m/s)
    bounds.append((0.0, 2 * np.pi))  # 方向 (弧度)

    # 每颗烟幕弹的投放时间和引信时间
    for k in range(smoke_per_uav):
        bounds.append((0.0, 30.0))  # 投放时刻
        bounds.append((0.5, 3.0))   # 引信延时

# ------------------- 计算单个烟幕弹的遮蔽信息 -------------------
def calculate_smoke_info(x, uav_idx, smoke_idx):
    # 获取该无人机的速度和方向
    speed_idx = (uav_idx * (smoke_per_uav * 2 + 2))
    angle_idx = speed_idx + 1
    speed = x[speed_idx]
    angle = x[angle_idx]

    # 获取烟幕弹的投放时间和引信时间
    idx = speed_idx + 2 + (smoke_idx * 2)
    t_rel = x[idx]
    t_fuze = x[idx+1]
    t_burst = t_rel + t_fuze

    # 计算投放坐标
    direction = np.array([np.cos(angle), np.sin(angle), 0.0])
    init_pos = UAV_INIT_POSITIONS[uav_idx]
    release_pos = init_pos + speed * t_rel * direction

    # 计算爆炸坐标（考虑自由落体）
    blast_pos = release_pos + speed * t_fuze * direction
    blast_pos[2] -= 0.5 * G * t_fuze**2

    # 计算单个遮蔽时长
    smoke_interval = (t_burst, t_burst + smoke_burn)
    cover_time = 0.0
    affected_missiles = []

    for missile_idx, (missile_start, missile_end) in enumerate(missile_intervals):
        # 计算时间重叠
        start_overlap = max(smoke_interval[0], missile_start)
        end_overlap = min(smoke_interval[1], missile_end)
        overlap_duration = max(0.0, end_overlap - start_overlap)

        if overlap_duration > 0:
            cover_time += overlap_duration
            affected_missiles.append(missile_idx + 1)  # 导弹编号从 1 开始

    return {
        'speed': speed,
        'angle_deg': np.degrees(angle),
```

```
        'release_pos': release_pos,
        'blast_pos': blast_pos,
        'cover_time': cover_time,
        'affected_missiles': affected_missiles
    }

# ------------------- 差分进化全局优化 -------------------
result = differential_evolution(
    obj,
    bounds,
    maxiter=1000,     # 增大迭代次数
    popsize=25,       # 增大种群规模
    polish=True,
    tol=1e-7,
    seed=42,
    disp=True
)

x_opt = result.x
best_cover = -result.fun

# ------------------- 输出结果 -------------------
print("\n 无人机运动信息:")
for u in range(uav_count):
    speed_idx = (u * (smoke_per_uav * 2 + 2))
    speed = x_opt[speed_idx]
    angle = x_opt[speed_idx + 1]
    print(f"UAV{u+1}: 速度 = {speed:.2f} m/s, 方向 = {np.degrees(angle):.2f}°")

print("\n 烟幕弹详细信息:")
rows = []
for u in range(uav_count):
    for k in range(smoke_per_uav):
        smoke_info = calculate_smoke_info(x_opt, u, k)

        release_pos = smoke_info['release_pos']
        blast_pos = smoke_info['blast_pos']

        print(f"\nUAV{u+1} 烟幕弹{k+1}:")
        print(f"  投放时间: {x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2)]:.2f} s")
        print(f"  引信时间: {x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2) + 1]:.2f} s")
        print(f"  投放坐标: ({release_pos[0]:.2f}, {release_pos[1]:.2f}, {release_pos[2]:.2f}) m")
        print(f"  爆炸坐标: ({blast_pos[0]:.2f}, {blast_pos[1]:.2f}, {blast_pos[2]:.2f}) m")
        print(f"  单个遮蔽时长: {smoke_info['cover_time']:.2f} s")
        print(f"  干扰的导弹: {', '.join(map(str, smoke_info['affected_missiles'])) or '无'}")

        rows.append([
            f"UAV{u+1}",
            k+1,
            x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2)],
            x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2) + 1],
            x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2)] + x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2)
+ 1],
            x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2)] + x_opt[(u * (smoke_per_uav * 2 + 2)) + 2 + (k * 2)
+ 1] + smoke_burn,
            smoke_info['speed'],
            smoke_info['angle_deg'],
            f"({release_pos[0]:.2f}, {release_pos[1]:.2f}, {release_pos[2]:.2f})",
```

```
            f"({blast_pos[0]:.2f}, {blast_pos[1]:.2f}, {blast_pos[2]:.2f})",
            smoke_info['cover_time'],
            ', '.join(map(str, smoke_info['affected_missiles'])) or '无'
        ])

# 使用中文表头
df = pd.DataFrame(rows, columns=[
    "无人机", "烟幕弹序号", "投放时间(s)", "引信时间(s)", "起爆时刻(s)", "结束时刻(s)",
    "速度(m/s)", "角度(度)", "投放坐标", "爆炸坐标",
    "遮蔽时长(s)", "干扰导弹编号"
])
df = df.sort_values(by="起爆时刻(s)").reset_index(drop=True)
print("\n 汇总表格:")
print(df.to_string(index=False))
print(f"\n>>> 总遮蔽时长 (3 枚导弹): {best_cover:.2f} s")

# 保存文件，使用中文表头和 UTF-8 编码
df.to_csv("result5.csv", index=False, encoding='utf-8-sig')
```