无人机烟幕干扰弹的投放策略问题

摘　要

现代战争中，无人机烟幕干扰弹投放策略受多种因素制约，对敌方导弹遮蔽效果主要由无人机飞行参数和投弹引爆时间所决定。在投放决策中，综合运用几何建模、优化算法，为无人机指挥建立直观的干扰评估模型，并结合遗传算法、局部搜索及多智能体协同优化，可以适配不同复杂度的无人机配置，最大化导弹遮蔽时间。

针对问题一，在给定条件下，根据无人机、导弹和目标的基本坐标建立几何模型，结合无人机和烟幕弹的武器参数进行运动学解算，得出引爆位置。通过烟幕团运动轨迹和导弹轨迹，判断烟幕是否有效遮蔽导弹。由于目标区域为圆柱体，分别构建**质点模型**和**圆柱模型**，对部分遮蔽和完全遮蔽分别进行讨论。最后对有效遮挡时间数值积分，得完全遮蔽结果为 1.392s。

针对问题二，一机一弹问题，建立含 4 个决策变量的无人机运动模型。初步进行暴力搜索计算，成本过高，由无人机与导弹水平飞行方向一致，缩小飞行角度搜索范围，进行**剪枝优化**。为进一步降低计算成本，采用**遗传算法**(GA)，为提高收敛速度，将暴力搜索得到的初步解作“种子”注入 GA 初始种群，快速迭代得到优解。为验证优越性，取机器学习作为反例论证。最后代入圆柱模型，得出完全遮蔽结果为 4.53s。

针对问题三，一机三弹问题，沿用问题二思路，建立含 8 个决策变量并带严格时序约束的**非线性优化模型**。考虑到投放间隔限制，为杜绝非法解产生，采用基于**相对时序**的基因编码，改进遗传算法。获得相对优解后，引入 **Nelder-Mead 局部搜索算法**精细化搜索全局最优解。最终采用 GA 结合局部搜索的混合算法，并代入圆柱模型进行验证，得出完全遮蔽结果为 7.54s。

针对问题四，三机一弹问题，为优化含 12 个决策变量的复杂模型，设计逐步进化的 GA 协同算法。考虑到无人机独立特性，引入**岛屿模型遗传算法**，各无人机通过“移民”机制，“交流”干扰策略，提升团体作战效率。同时，设计适应度函数，引入**贡献度评估**，明确个体对全局的贡献程度，最大化整体遮蔽时间。类似地，将质点模型下最优解代入圆柱模型，得到完全遮蔽结果为 11.04s。

针对问题五，多机多弹复杂决策问题，由于变量过多，先后构建了多机多弹协同拦截的**开放式系统**与**闭环式系统**。开放式系统分为三层：**战略层**接收物理数据，使用拍卖分配法输出作战指令；**战术层**接收作战指令，使用问题四最优解输出投弹策略；**执行层**接收投弹策略，使用问题三最优解驱动各无人机执行策略。因开放式系统在决策时间上优化不足，摒弃 GA，引出闭环式系统，将决策空间离散化，使复杂的几何判定结果转化为简单的 0-1 输入。闭环式系统分为两层：上层为**决策规划层**，采用闭环反馈机制，反馈迭代结果，持续优化无人机编队配置；下层为**执行验证层**，将上层方案转化具体无人机飞行策略与投放时序。层间通过闭环反馈交流，得到智能决策流程。最终得到三枚导弹的总遮蔽时间为 23.82s。

总之，该模型对实际导弹的遮蔽决策提供了一种可行的研究方向。

关键词：质点/圆柱模型　遗传算法　Nelder-Mead 局部搜索　岛屿模型　开放式/闭环式系统

一、问题重述

1.1 问题背景

现代精确制导武器技术快速发展，这些导弹凭借高速、精确、远程的打击能力，构成严重的军事威胁。为提高关键目标的生存能力，烟幕干扰弹因其结构简单、成本低廉、效费比高等优势，成为重要的防御措施之一。烟幕干扰弹干扰机理主要是在真目标与假目标之间制造遮蔽区域以实现目标隐蔽或诱导效果。

随着无人机的快速发展，无人机具备机动灵活、长航时、低成本等优势，为烟幕干扰弹的投放提供了新的手段。通过无人机挂载烟幕干扰弹，可在目标上空或目标前沿布设遮蔽云团，从而实现固定设施防护和战术集群。在实际应用中，烟幕干扰弹的投放过程需要精确的时空控制。干扰弹投放点必须位于来袭导弹飞行路径与目标之间的关键空域，以保证烟幕在有效时长内覆盖真目标。[3]

现有五架无人机、三枚导弹，导弹对假目标进行袭击，无人机投射烟幕干扰弹为真目标提供掩护。

1.2 问题提出

问题一（定量问题）：已知无人机 FY1 飞行速度、飞行方向、飞行时间，引信时间，投弹次数；导弹 M1 飞行方向、飞行速度；其他基本参数。求解有效遮蔽时间。

问题二（一机一弹）：已知无人机 FY1 投弹次数；导弹 M1 的基本参数；烟幕基本参数。求解无人机 FY1 飞行速度、飞行方向、飞行时间、引信时间，最大化对导弹 M1 的遮蔽时间。

问题三（一机三弹）：已知无人机 FY1 投弹次数；导弹 M1 的基本参数；烟幕基本参数。求解无人机 FY1 基本参数及时序，最大化对导弹 M1 的遮蔽时间。

问题四（三机一弹）：已知无人机 FY1、FY2、FY3 投弹次数；导弹 M1 的基本参数；烟幕基本参数。求解无人机 FY1、FY2、FY3 基本参数，最大化对导弹 M1 的遮蔽时间。

问题五（复杂决策）：已知无人机 FY1 至 FY5 投弹次数与导弹 M1 至 M3 的基本参数；烟幕基本参数。求解无人机基本参数，最大化对导弹的遮蔽时间。

二、问题分析

2.1 问题一的分析

为求解无人机 FY1 在给定条件下对导弹 M1 的有效遮挡时间。根据条件，通过 FY1 的初始坐标、速度方向、投弹时间，可计算出烟幕干扰弹离开无人机时的坐标。在引爆间隔时间内，烟幕干扰弹做平抛运动，求得烟幕初始坐标。对烟幕运动模型和导弹运动轨迹构建线模型，分别计算导弹坐标与烟幕中心坐标，判断烟幕中心与导弹轨迹最短距离是否小于烟幕有效范围及是否处于浓度时效内。由于真目标是一个圆柱，基于此，把质点模型升级为离散化圆柱模型，分别讨论部分遮蔽和完全遮蔽。通过对各有效遮挡时段的离散化步长数值积分，得到最终的有效遮挡时间。

2.2 问题二的分析

为求解单无人机单烟幕弹的最优干扰策略，构建 4 个决策变量的优化模型。尝试探索将优化变量从飞行参数转为引爆时刻与引爆位置逆推，但对“多机多弹”问题，求解存在困难，故该方案仅作理论探讨。

继而采用暴力搜索正推，基于无人机与导弹初始位置的几何偏差，将飞行角度的搜索空间进行剪枝优化，将其约束在直飞方向 ±5° 内，得到初步可行解。考虑计算成本，选用遗传算法(GA)作为核心优化工具。为加速收敛，将暴力搜索解作“种子”注入 GA 初始种群，获得质量更高的策略。为验证该方法的优越性，进行机器学习方案的对比实验，即使采用基于已知解的种子作为训练数据，仍无法独立高效地完成寻优任务。最后确立遗传算法为问题二的核心求解方案，并将其应用于圆柱模型进行验证。

2.3 问题三的分析

为求解单无人机使用 3 枚烟幕弹对导弹 M1 实施最优干扰的策略，构建含 8 个决策变量且带严格时序约束的高维非线性优化模型。

考虑投放间隔约束，对遗传算法(GA)进行了相对时序基因编码改进，将后续投放的绝对时刻改为相对于上一次投放的时间间隔，从根本上杜绝非法解的产生，显著提升了算法的全局搜索效率。在此基础上，进一步引入 Nelder-Mead 局部搜索算法，在 GA 完成全局探索后，对所有的优良解进行高精度的局部精炼，以逼近理论最优值。最后确立该方法算法为问题三的核心求解方案，将其应用于圆柱模型进行验证。

2.4 问题四的分析

为求解 3 架无人机协同对导弹 M1 实施最优干扰的策略，沿用上述思路，构建 12 维的多智能体协同优化模型，并设计一套逐步进化的协同算法体系。

针对多智能体的结构特性，引入岛屿模型遗传算法，为每架无人机建立独立种群进行专业化进化，同时通过“移民”机制实现信息共享，相比标准 GA，极大提升寻优效率。基于此，为最大化团队利益，为适应度函数添加边际贡献度评估机制，奖励筛选无人机间形成互补和分工的最优协同策略。最后把协同进化算法找到的最优策略，代入精度最高的圆柱模型中，进行最终检验。

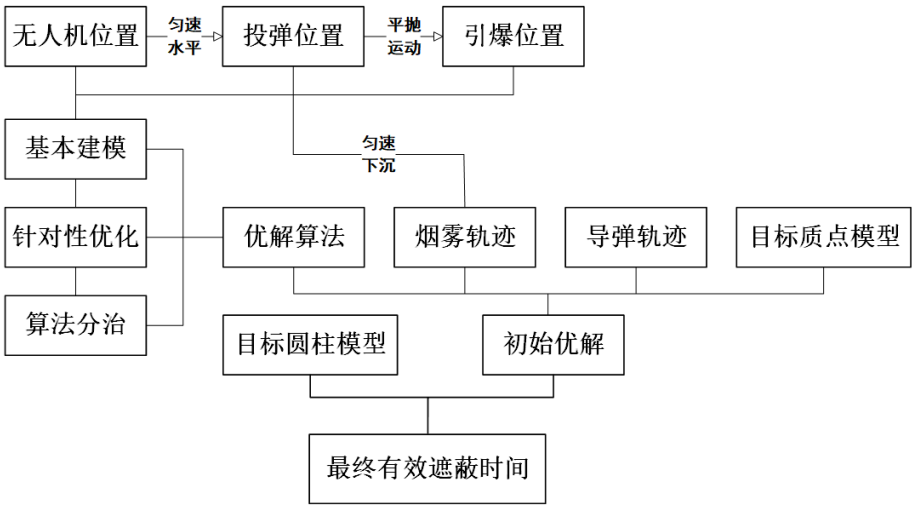
2.5 问题五的分析

为求解使用 5 架无人机对 3 枚导弹的最佳干扰策略，因变量过多，遗传算法消耗资源大，无法在短时间内求解整个问题，因此基于“分而治之”的思想分别构建多无人机多导弹协同拦截的开放式系统以及闭环式系统。

开放式系统采用三层架构：战略层基于全局任务分配和市场拍卖，只接收物理数据，输出作战指令；战术层沿用问题二及问题四的算法模式，只接收作战指令，输出投弹策略；执行层沿用问题三的算法模式，只接收投弹策略，输出引信策略及最终结果。因开放式系统内核仍为遗传算法，对时间的开销仍有优化空间，故摒弃此算法，引出闭环式系统。闭环式系统采用两层架构，上层为决策规划层，通过闭环反馈机制持续修正与优化；下层为执行验证层，将上层方案转化为具体行动路径与投放时序，层间通过闭环反馈紧密联系，得到智能决策流程。由于闭环式系统增加自下而上的反馈路径且系统封闭，因此，相较于开放式优化得到的时间更好，系统计算速度更快。

2.6 基本思维框架

图 1 问题二至四思维导图



特此声明：对于每个问题的求解，优先采用质点模型验证，得到最优模型后再使用圆柱模型。

三、基本假设

一、无人机间运动互不干扰，烟幕干扰弹间运动互不干扰，无人机投掷烟幕干扰弹后继续做匀速运动。

二、烟幕干扰弹脱离无人机后做平抛运动，忽略空气影响，且水平速度与无人机相同。重力加速度恒定为 9.8m/s^2。

三、多个烟幕干扰弹叠加遮蔽对导弹的效果与单个烟幕干扰弹遮蔽效果相同。

四、符号说明

表 1

符号	符号解释	单位
$\vec{P}_{\mathrm{X},t}$或$P_{\mathrm{X},t}$	X 在 t 时刻的位置	(m, m, m)
v_{X}	X 的速度大小	m/s
$\hat{d}_{\mathrm{X}}$或$\mathbf{u}_{\mathrm{X}}$	X 的速度单位向量	(m/s, m/s, m/s)
θ	无人机运动偏离方向	°
t_{drop}	无人机从出发至投掷烟幕干扰弹的时间	s
T_{fuse}	烟幕干扰弹从离开无人机至引爆的时间	s
g	重力加速度	m/s^2

五、模型的建立与求解

5.1 问题一的求解

问题一的核心是在给定数据下求解导弹被烟幕遮挡的时间。基于无人机初始状态和投弹时间，确定烟幕干扰弹的释放及烟幕的生成位置；结合烟幕运动模型和导弹飞行轨迹，建立坐标随时间变化的线性模型；通过计算导弹轨迹与烟幕中心的最近距离，并结合烟幕作用半径及时效性，判定导弹是否被有效遮挡；最后对各有效遮挡区间进行时间离散与积分，得到总有效遮蔽时间。

5.1.1 各物体运动模型的建立

数据准备

表 2 已知运动参数

M1 坐标	$\vec{P}_{M1,0}$=(20000, 0, 2000)	M1 速度	v_{M1}=300m/s
FY1 坐标	$\vec{P}_{FY1,0}$=(17800, 0, 1800)	FY1 速度	v_{FY1}=120m/s
投放时间	t_{drop}=1.5s	引信时间	T_{fuse}=3.6s
烟幕云半径	R_{smoke}=10m	烟幕云下沉速度	v_{sink}=3m/s

（一）导弹 M1 的运动

飞行方向的单位向量$\hat{d}_{M1}$:

$$\hat{d}_{M1}=\frac{\vec{P}_{fake}-\vec{P}_{M1,0}}{\|\vec{P}_{fake}-\vec{P}_{M1,0}\|}$$

飞行速度向量$\vec{V}_{M1}$:

$$\vec{V}_{M1}=v_{M1}\cdot\hat{d}_{M1}$$

任意时刻t的位置$\vec{P}_{M1}(t)$:

$$\vec{P}_{M1}(t)=\vec{P}_{M1,0}+\vec{V}_{M1}\cdot t$$

（二）无人机 FY1 的运动

为确保水平飞行，定义其在飞行高度上的目标点$\vec{P}'_{FY1,target}$:

$$\vec{P}'_{FY1,target}=(x_{fake},y_{fake},z_{FY1,0})=(0,0,1800)$$

水平飞行方向的单位向量$\hat{d}_{FY1}$:

$$\hat{d}_{FY1}=\frac{\vec{P}'_{FY1,target}-\vec{P}_{FY1,0}}{\|\vec{P}'_{FY1,target}-\vec{P}_{FY1,0}\|}$$

飞行速度向量$\vec{V}_{FY1}$(其 Z 分量为 0):

$$\vec{V}_{FY1}=v_{FY1}\cdot\hat{d}_{FY1}$$

任意时刻t的位置$\vec{P}_{FY1}(t)$:

$$\vec{P}_{FY1}(t)=\vec{P}_{FY1,0}+\vec{V}_{FY1}\cdot t$$

（三）烟幕云中心的运动

烟幕弹的投放位置$\vec{P}_{drop}$:

$$\vec{P}_{drop}=\vec{P}_{FY1}(t_{drop})$$

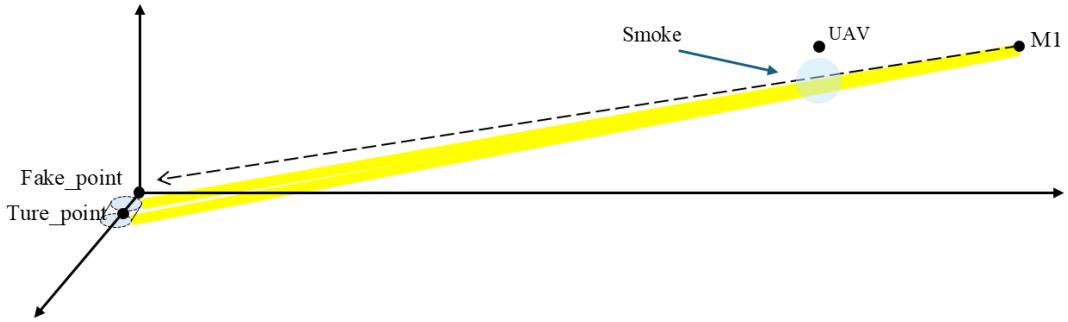
由平抛运动得，烟幕弹的引爆位置:

$$\vec{P}_{det}=\vec{P}_{drop}+\vec{V}_{FY1}\cdot T_{fuse}-\left(0,0,\frac{1}{2}gT_{fuse}^{2}\right)^{\mathrm{T}}$$

引爆时刻$t_{det}=t_{drop}+T_{fuse}$。在$t\geq t_{det}$时，烟幕云开始下沉，其中心位置$\vec{P}_{smoke}(t)$为:

$$\vec{P}_{smoke}(t)=\vec{P}_{det}-\begin{pmatrix}0\\0\\v_{sink}\end{pmatrix}\cdot(t-t_{det})$$

图 2 烟幕球体遮蔽示意图



5.1.2 有效遮蔽时长的求解

（一）有效遮蔽的判定条件

为简化计算，对所有模型先采用质点计算，再对最优模型使用圆柱计算。判断在时刻 t，烟幕球体是否与导弹位置和真目标构成的线段相交。即判断烟幕云中心到该线段的最短距离是否小于等于烟幕云半径，如图 2 所示。

定义导弹起点：$\vec{A}(t)=\vec{P}_{M1}(t)$，真目标：$\vec{B}=\vec{P}_{true}$，烟幕云球心：$\vec{P}(t)=\vec{P}_{smoke}(t)$，这里先记$\vec{B}=\vec{P}_{true}=(0,200,5)$

最短距离$d(t)$的计算如下：

①记$\begin{cases}\vec{v}=\vec{B}-\vec{A}(t)\\\vec{w}=\vec{P}(t)-\vec{A}(t)\end{cases}$

②计算球心在直线 AB 上的投影位置参数 $\lambda = \frac{\vec{w}\cdot\vec{v}}{\|\vec{v}\|^2}$。

③根据 λ 的值判断最短距离 $d_B(t)$:

$$d_B(t) = \begin{cases} \|\vec{P}(t) - \vec{A}(t)\| & \text{if } \lambda < 0 \\ \|\vec{P}(t) - \vec{B}\| & \text{if } \lambda > 1 \\ \sqrt{\|\vec{w}\|^2 - \lambda^2 \|\vec{v}\|^2} & \text{if } 0 \le \lambda \le 1 \end{cases}$$

遮蔽条件: $d_B(t) \le R_{smoke}$

定义示性函数 $I_P(t)$，当满足遮蔽条件时为 1，否则为 0:

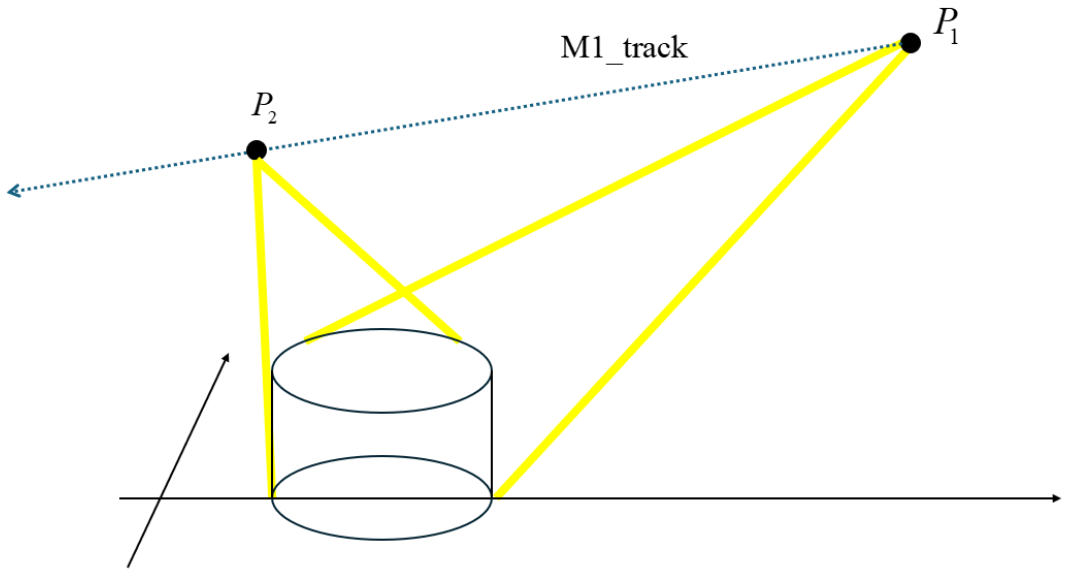
$$I_P(t) = \begin{cases} 1 & \text{if } d(t) \le R_{smoke} \\ 0 & \text{otherwise} \end{cases}$$

为将对完整圆柱体的遮蔽判断，构建基于视觉轮廓离散化的两种遮蔽评估模型。从一个复杂的几何问题，转化为可计算的数值问题，采用了视觉轮廓离散化的方法。

a) 视觉轮廓的定义

在任意时刻t，对于一个观察点（导弹位置 P）和一个目标物体（圆柱体 C），目标的视觉轮廓 V(C, P)是指那些从 P 出发的视线与物体表面相切所形成的边缘线。若一个烟幕云能挡住目标的全部视觉轮廓，则它就可挡住整个目标。对于圆柱体，其视觉轮廓由顶部圆周、底部圆周，及两条母线（即将 P 投影至底面点 P'对圆的切线）构成，如图 3 所示。

图 3 导弹视野图



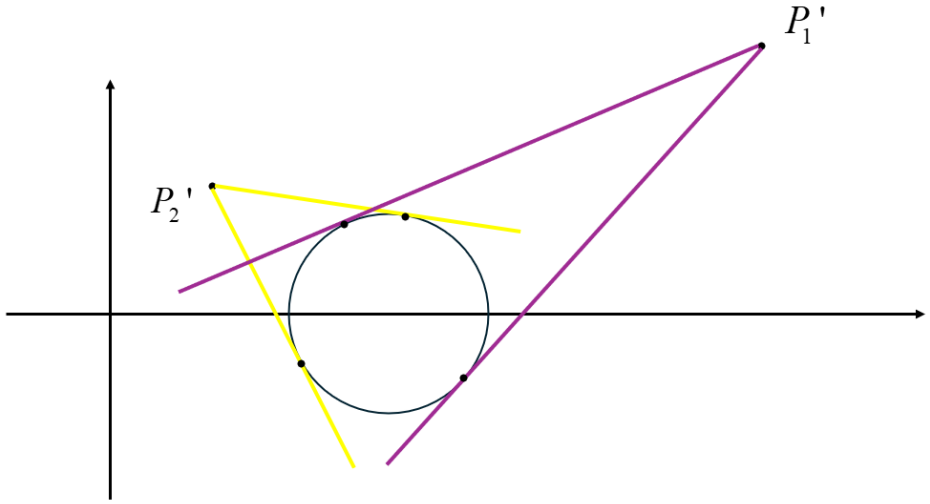
b) 轮廓的离散化

由于连续的视觉轮廓包含无限个点。采用离散化的方法，在视觉轮廓上用一个有限的点集 **V** 来近似连续的轮廓线

$$\mathbf{V} = \{v_1, v_2, v_3, ..., v_k\}$$

在每个时间步 t，将导弹位置 P 和圆柱体在底面平面上进行投影，通过几何计算，计算出此时从导弹投影点看向圆柱投影圆的两个切点。根据这两个切点，求出两条竖直母线的坐标，如下图所示。

图 4 导弹视野投影图



在这两条动态变化的竖直母线，以及顶部和底部的圆周上，均匀地采集成一个包含 k 个点的集合 **V**。当采样点数 k 足够大时，集合 **V** 则可逼近真实轮廓线。

c) 两种遮蔽标准的定义

对于轮廓点集 **V** 中的每一个点 v_i，首先定义其是否被遮挡的示性函数 $B(v_i,t)$：

$$B(v_i,t)=\begin{cases}1 & \text{if } d_{v_i}(t)\le R_{smoke}\\ 0 & \text{otherwise}\end{cases}$$

基于此，定义两个遮蔽指示函数：

部分遮蔽指示函数 $I_{partial}(t)$：至少一个轮廓点被遮挡，即部分遮蔽，。

$$I_{partial}(t)=\begin{cases}1 & \text{if } \sum_{i=1}^{k}B(v_i,t)\ge 1\\ 0 & \text{otherwise}\end{cases}$$

完全遮蔽指示函数 $I_{total}(t)$：所有轮廓点都被遮挡，即完全遮蔽。

$$I_{total}(t)=\begin{cases}1 & \text{if } \sum_{i=1}^{k}B(v_i,t)=k\\ 0 & \text{otherwise}\end{cases}$$

（二）利用数值积分计算有效遮蔽时长

由于，仿真从烟幕云诞生 t_{det} 开始，到烟幕云失效($t_{det}+T_{duration}$)或导弹击中目标 t_{impact} 结束，取其较早者。故积分上限为 $T_{end}=\min\left(t_{det}+T_{duration},\dfrac{\|\vec{P}_{fake}-\vec{P}_{M1,0}\|}{v_{M1}}\right)$；积分下限为 t_{det}.

总有效遮蔽时长即为指示函数在有效时间区间上的积分：

$$T_{total} = \int_{t_{det}}^{T_{end}} I(t)dt$$

由于 $I(t)$是一个复杂的分段函数，难以求出解析解。采用数值积分，将时间以步长 Δt 离散化，通过求和来近似积分结果：

$$T_{total} \approx \sum_{t=t_{det}}^{T_{end}} I(t) \cdot \Delta t$$

Δt 取 0.001 得：

表 3 不同类型有效遮蔽时间结果对比

评估模型	遮蔽类型	有效时长(s)
质点模型	-	1.435
圆柱模型	部分遮蔽	1.485
圆柱模型	完全遮蔽	1.392

为简化计算，此后 Δt 取 0.01。

5.2 问题二的求解

问题二的核心是求解无人机多个参数。首先构建4个决策变量的几何模型，注意到，每一个坐标点对应一组参数，可遍历坐标点，但因适配度低，舍去。

然后，编写程序暴力遍历四个变量进行正向推导，得到初步结论。考虑其较高的计算成本，进一步选用遗传算法，发现开局零值较多。故为加速收敛，将暴力遍历所得结论作为种子注入，得到质点模型下的优解，并带入圆柱模型得到最终结论。

特别地，为验证遗传算法优越性，进行机器学习方案对比实验，因其结论高度依赖于其他算法的优解，不具备独立性，易陷入局部最优解，舍去。

5.2.1 思路探索

有效遮蔽时长由球形云团运动轨迹和经历时间直接决定，因此只需找到一个无人机策略 **S**，使烟幕弹在时刻 t_{det}、位置 $P_{\text{det,ideal}}$ 引爆，使有效遮蔽时间最长。对此时刻 t_{det}、位置 $P_{\text{det,ideal}}$，可确定 FY1 的飞行方向、飞行速度、烟幕干扰弹投放点、烟幕干扰弹起爆点。[1]

反解无人机速度参数

定义起爆时刻 $t_{\text{det}}=t_{\text{drop}}+T_{\text{fuse}}$，起爆位置 $P_{\text{det}}=(x_{\text{d}},y_{\text{d}},z_{\text{d}})$

烟幕弹在释放后竖向为自由落体则释放高度为无人机高度 z_{FY1}，释放到起爆的时间等于引信时间 $T_{\underline{fuse}}$。释放高度与起爆高度关系为：

$$z_d = z_{FY1} - \frac{1}{2} g T_{fuse}^2$$

其中 $T_{fuse} \geq 0$ 且 $z_d \leq z_{FY1}$。

无人机在等高匀速直线飞行，设其水平单位航向向量为 $d_{FY1,xy}$，速度为 v_{FY1}。无人机初始水平位置为 $P_{FY1,0,xy}$。起爆时刻 t_{det} 时无人机对应的水平位置为：

$$.(x_d, y_d) = \vec{P}_{FY1,0,xy} + v_{FY1} d_{FY1,xy} t_{det}.$$

航向单位向量与速度：

$$d_{FY1,xy}=\frac{(x_d,y_d)-\vec{P}_{FY1,0,xy}}{|(x_d,y_d)-\vec{P}_{FY1,0,xy}|}$$

$$v_{FY1}=\frac{|(x_d,y_d)-\vec{P}_{FY1,0,xy}|}{t_{det}}.$$

投放时刻与起爆时刻关系由引信时间给出：

$$t_{drop}=t_{det}-T_{fuse}$$

投放点（释放位置）为无人机在 t_{drop} 时的位置：

$$\vec{P}_{drop,xy}=\vec{P}_{FY1,0,xy}+v_{FY1}d_{FY1,xy}t_{drop}.$$

各动态物体的运动状态是整个仿真的基础，其位置向量 $\mathbf{P}(t)$ 关于时间 t 的参数方程如下：

导弹 M1：

$$\vec{V}_{M1}=v_{M1}\cdot\frac{\vec{P}_{fake}-\vec{P}_{M1,0}}{\|\vec{P}_{fake}-\vec{P}_{M1,0}\|}.$$

$$\vec{P}_{M1}(t)=\vec{P}_{M1,0}+\vec{V}_{M1}\cdot t$$

无人机 FY1：

策略向量为 $\vec{S}=[\theta,v,t_{drop},t_{fuse}]$

$$\vec{V}_{FY1}(\theta,v)=v\cdot(\cos\theta,\sin\theta,0)$$

$$\vec{P}_{FY1}(t,\vec{S})=\vec{P}_{FY1,0}+\vec{V}_{FY1}\cdot t$$

烟幕云中心：

$$\vec{P}_{drop}(\vec{S})=\vec{P}_{FY1}(t_{drop},\vec{S})$$

$$\vec{P}_{det}(\vec{S})=\vec{P}_{drop}+\vec{V}_{FY1}\cdot t_{fuse}-(0,0,\frac{1}{2}gt_{fuse}^2)$$

$$\vec{P}_{smoke}(t,\vec{S})=\vec{P}_{det}-(0,0,v_{sink})\cdot(t-(t_{drop}+t_{fuse}))$$

计算得：有效遮蔽时长达到 2.41s。

但因物理过程求解复杂，对于多机多弹问题难以解决，因此该方案仅作理论探讨，实际运用效果并不好，由此引出正推基于程序的算法。

5.2.2 正向求解

为寻找最优策略，对多种优化范式进行了探索与比较。

（一）基准解法——剪枝优化暴力搜索

为确立后续算法的性能基准，首先采用了暴力网格搜索方法。基于无人机与导弹的几何洞察，将飞行角度的搜索空间进行剪枝优化。[4]

由于 M1(20000, 0, 2000) 和 FY1(17800, 0, 1800)在同一平面，真目标(0, 200, 0)。

$$|\theta|\le\arcsin\frac{200}{\sqrt{20000^2+2000^2}}\approx5.72°$$

为计算方便，约束在直飞假目标方向±5°内。

该方法通过遍历决策变量离散化后形成的搜索空间

$$\Omega = \{\theta\} \times \{v\} \times \{t_{\text{drop,k}}\} \times \{t_{\text{fuse,l}}\}$$

在对质点模型进行计算后，耗时 174.44s，获得了一个有效的基准解飞行策略: θ=177.50°，v =80m/s, $t_{\text{drop,k}}$ =1.00s，$t_{\text{fuse,l}}$ =3.00s，有效遮蔽时长为 3.21s。该结果验证了模型的可行性，但其巨大的计算成本也凸显了寻求更高效智能算法的必要性。

（二）种子注入遗传算法

考虑到暴力搜索的局限性，选用遗传算法作为核心优化工具。将一个可行策略定义为染色体 $\vec{S} = [\theta, v, t_{drop}, t_{fuse}]^T$。为加速收敛并保证寻优的有效性，选取暴力搜索得到的 3.21s，作为一个种子预置到 GA 的初始种群中。

种群初始化：

初始种群 P_0 由一个种子解 $\vec{S}_{seed}$ 和 $N-1$ 个在可行域内随机生成的解构成：

$$P_0 = \{\vec{S}_{seed}\} \bigcup \{\vec{S}_{rand,i}\}_{i=1}^{N-1}$$

适应度函数：

个体的适应度 $f(\vec{S})$ 直接由质点模型计算的遮蔽总时长 $T_{total}(\vec{S})$ 决定。.

进化迭代过程：

下一代种群 P_{g+1} 由当前种群 P_g 经过选择、交叉、变异等一系列遗传算子 $\mathcal{G}$ 作用生成：

$$P_{g+1} = \mathcal{G}(P_g) = \text{Mutate}(\text{Crossover}(\text{Select}(P_g)))$$

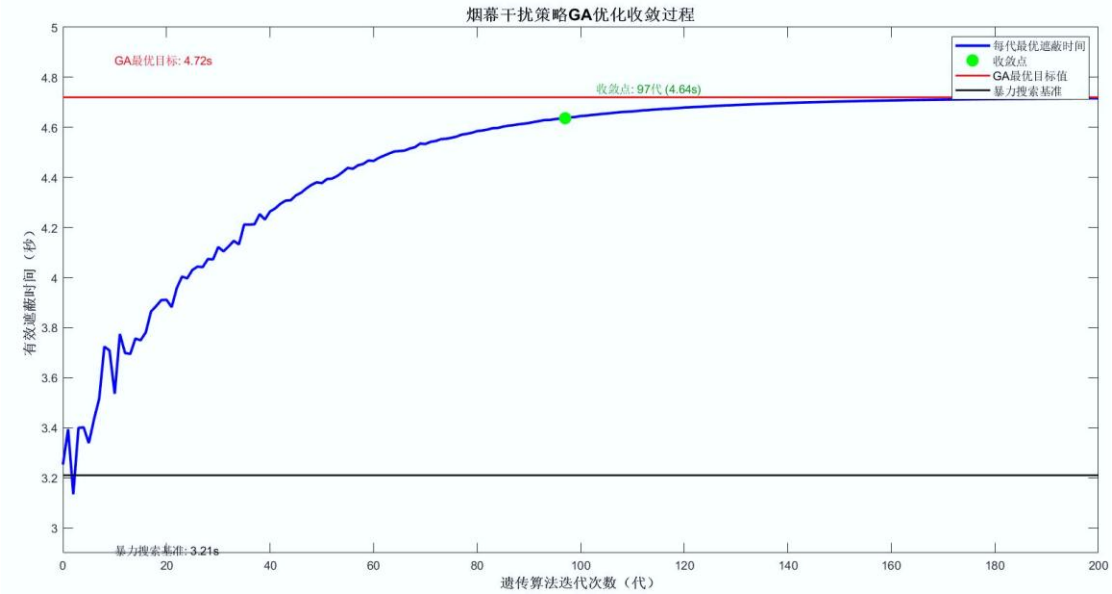
选择：采用锦标赛选择，从种群中随机选取 k 个个体，将适应度最高的个体作为父代。交叉(Crossover)：对父代 $\vec{p}_1, \vec{p}_2$ 采用模拟二进制交叉(SBX)，根据随机变量 $u \in [0,1]$ 和分布指数 η_c 生成扩展因子 β，产生子代 $\vec{c}_1, \vec{c}_2$：

$$\vec{c}_{1,2} = 0.5 \cdot \left[\left(\vec{p}_1 + \vec{p}_2 \right) \mp \beta \left| \vec{p}_2 - \vec{p}_1 \right| \right]$$

变异：对基因 s_k 采用多项式变异，根据随机变量 $r \in [0,1]$ 和分布指数 η_m 生成扰动 δ_k，产生新基因 s_k'：

$$s_k' = s_k + \delta_k (s_{k,\text{upper}} - s_{k,\text{lower}})$$

图 5 烟幕干扰策略 GA 优化收敛图



如图 5，通过该方法，算法在极短的时间内便找到了一个远优于基准解的策略，在质点模型上的有效遮蔽时长达到 4.64s。

此时飞行策略为：θ=178.04°，v_{j} =82.21m/s, $t_{\text{drop,k}}$ =0.11，$t_{\text{fuse,l}}$ =2.64s

（三）对比方法：机器学习可行性探究

为验证 GA 的优越性，进行了机器学习方案的对比实验。尝试训练一个随机森林回归模型 f_{ML}，以逼近真实的物理仿真函数 $T_{total}(\vec{S})$。即便采用基于 GA 优解的采样策略，得有效遮蔽时长达 4.62s。

其实验结果尽管接近 GA 解，但仍不理想。其根本缺陷在于：

①模型独立性缺失，其性能严重依赖于由其他优化算法预先提供的高质量数据源；

②本质错配，其作为“预测器”而非“优化器”的本质，决定了其寻优流程本质上是“高级插值+大规模随机猜测”，无法独立、高效地完成寻优任务。

最终验证与结果分析

综合对比，最终选择种子注入遗传算法作为问题二的核心求解方案。将 GA 在质点模型上找到的最优策略，代入精度最高的圆柱模型中，进行最终的仿真检验。

表 4 不同类型有效遮蔽时间结果对比

评估模型	遮蔽类型	最终有效时长(s)
质点模型(GA 寻优)	-	4.72
圆柱模型	部分遮蔽	4.80
圆柱模型	完全遮蔽	4.53

最终，确定问题二的最优策略如下，该策略能在高保真模型下，产生 4.80s 的部分遮蔽，4.53s 的完全遮蔽。

表 5 无人机结果参数

参数	最优值
飞行角度 θ	177.71°
飞行速度 v	80.34m/s
投放时间 t_{drop}	0.16s
引信时间 t_{fuse}	2.68s

5.3 问题三、问题四的求解

问题三的核心是一架无人机可投放三次烟幕弹，且每个烟幕弹间至少间隔 1s 条件下，求解运动参数和投放时序。首先沿用问题二的遗传算法，但因对时间不敏感，会产生大量冗余解，导致算法收敛速度过慢。因此，在编码上，采用相对时序的方式改进，从而有效杜绝非法解的产生。经改进编码后的解答范围相对稳定，考虑在此基础上引入 Nelder-Mead 局部搜索算法，对一定范围内的合法解进行精炼，从而得到更好的解。

类似地，问题四是在三架无人机各可投放一枚烟幕弹的条件下求解最优投放策略。沿用问题二的遗传算法，但因其无法兼顾各无人机间的时序配合，具有较强的封闭性，因此，在此基础上引入岛屿模型，优化无人机团队相互配合的效果，每隔一段时间将最优的子代循环送入其他岛屿，打破一定的封闭性。然而该方案对个体的关注不明显，导致循环过程中的最优子代不一定是对整体团队贡献度最高的。故引入贡献度评估机制，从而筛选出最利于团队分工互补的协同策略。

特别地，对于问题三和问题四的求解，还进行了一定的灵敏度分析。

5.3.1 问题三方法提要

在问题二的基础上，问题三进一步要求为单架无人机规划一个包含三枚烟幕干扰弹的投放方案，以实现对导弹 M1 的最大化遮蔽。此时，模型的决策变量数量由 4 个扩展至 8 个：

$$\mathbf{x} = [\theta, v, t_{d,1}, t_{e,1}, t_{d,2}, t_{e,2}, t_{d,3}, t_{e,3}],$$

其中 θ 为飞行方向，v 为飞行速度，$t_{d,i}$ 为第 i 枚干扰弹的投放时刻，$t_{e,i}$ 为其引信起爆时刻。

该模型需满足以下时序约束：

$$t_{d,2} - t_{d,1} \geq 1, \quad t_{d,3} - t_{d,2} \geq 1,$$

以及速度约束：$70 \leq v \leq 140\ \text{m/s}, \quad t_{e,i} \geq t_{d,i}, i = 1,2,3.$

（一）标准遗传算法

沿用问题二的思路，直接将上述 8 个变量直接编码为一条染色体，构建标准遗传算法，统一进化，实现简单，作为基准方法；但大量进化资源浪费在无效个体上，使得算法收敛速度明显变慢；合法解在搜索空间中比例过低，GA 容易早熟收敛，导致解质量受限。经测试，标准 GA 能找到约 5.78s 的有效遮蔽时长，验证了三枚弹协同投放策略的可行性。

（二）相对时序编码遗传算法

由于标准 GA 的时序性极差，因此，从编码机制上直接消除非法解，提高进化效率。

$$\mathbf{x}' = [\theta, v, t_{d,1}, \Delta t_2, \Delta t_3, \tau_1, \tau_2, \tau_3],$$

其中：$t_{d,2} = t_{d,1} + \Delta t_2, \quad t_{d,3} = t_{d,2} + \Delta t_3, \quad \tau_i = t_{e,i} - t_{d,i}.$

设定约束：$\Delta t_2 \geq 1, \quad \Delta t_3 \geq 1, \quad \tau_i \geq 0,$

与标准 GA 相比，该方法存在以下优点：

编码空间全为合法解，节省了适应度评估资源；搜索空间维度保持一致，但收敛速度显著加快；最优值提升：在相同的计算预算下，有效遮蔽时间由 5.78 s 提升至 6.1s。因此，相对时序编码 GA 为后续进一步优化提供了坚实基础。

（三）精英局部搜索混合遗传算法

为进一步提升解的精度，在相对时序编码 GA 的基础上，引入局部搜索算子。具体做法为：在每一代 GA 迭代完成后，选取种群中的精英个体 $\mathbf{x}^*$，并调用 Nelder–Mead 算法在连续变量空间中进行局部优化：

$$\mathbf{x}^{(k+1)} = \text{NelderMead}(\mathbf{x}^*),$$

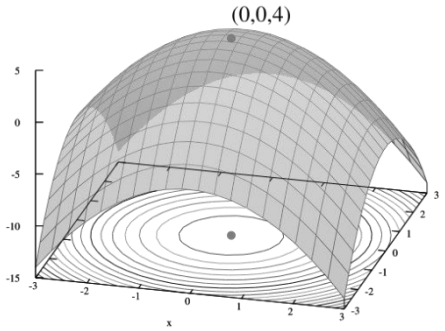


图 6 局部搜索示意图[2]

①全局搜索阶段：采用相对时序编码 GA 生成种群并进化，得到当前种群的精英解 $\mathbf{x}^*$。

②局部优化阶段：对精英解 $\mathbf{x}^*$ 调用 Nelder–Mead 单纯形法进行局部搜索。

在 n 维空间中构造 n+1 个点组成的单纯形；

$$\mathbf{x}_c = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

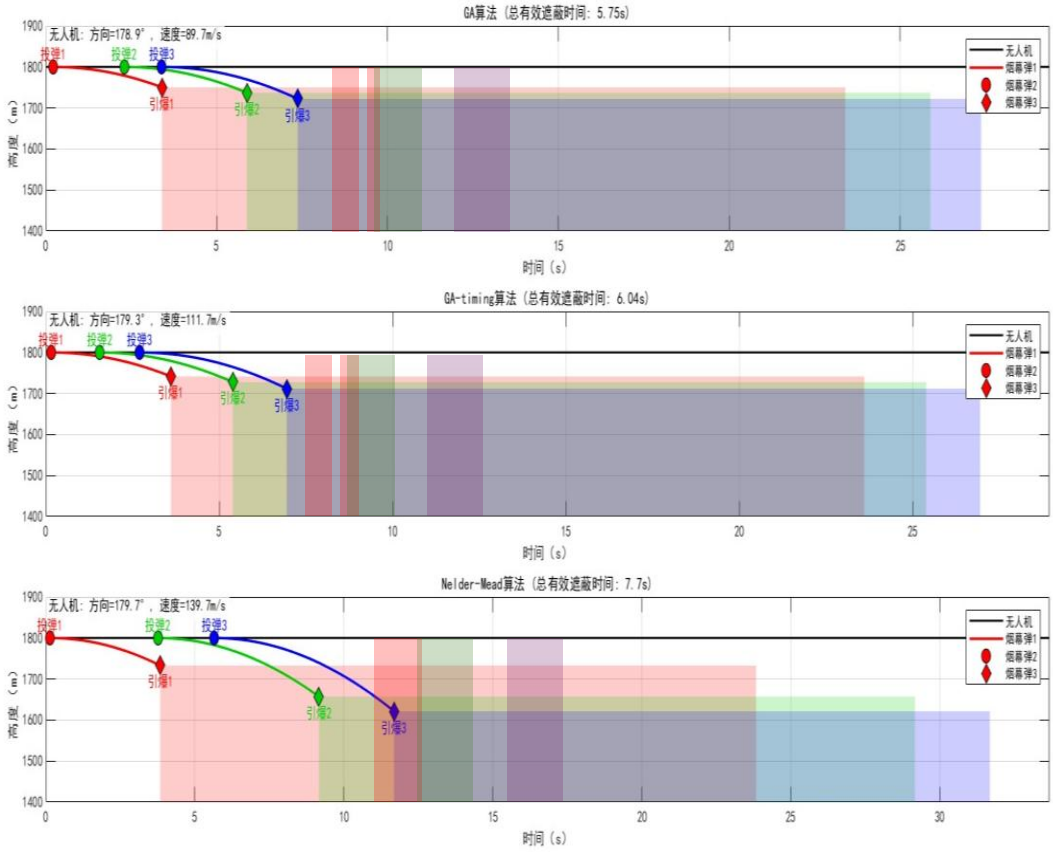
计算除最差点外的质心：反射、扩展、收缩、整体收缩分别为

$$\begin{cases} \mathbf{x}_r = \mathbf{x}_c + \alpha(\mathbf{x}_c - \mathbf{x}_{worst}), & \alpha = 1 \\ \mathbf{x}_s = \mathbf{x}_c + \rho(\mathbf{x}_{worst} - \mathbf{x}_c), & \rho = 0.5 \\ \mathbf{x}_e = \mathbf{x}_c + \gamma(\mathbf{x}_r - \mathbf{x}_c), & \gamma = 2 \\ \mathbf{x}_i = \mathbf{x}_{best} + \sigma(\mathbf{x}_i - \mathbf{x}_{best}), & \sigma = 0.5 \end{cases}$$

③将 Nelder-Mead 优化后的结果替换回种群中的精英个体，进入下一代迭代。

该混合框架兼顾了 GA 的全局搜索能力与局部爬山能力，最终得到 7.70s 的稳定最优解。

图 7 时间轴图



如图 7，图中深色表示有效遮蔽时段，经过 Nelder-Mead 优化后的岛屿模型在质点模型产生最优解，带入圆柱模型。

表 6 问题三的结果

无人机运动方向	烟幕弹编号	投放点的 x 坐标(m)	有效遮蔽时长(s)
179.65°	1	17781.01	4.17
无人机运动速度(m/s)	2	17274.19	2.52
139.66	3	17011.22	1.15
总部分遮蔽(s)	7.83	总完全遮蔽(s)	7.54

5.3.2 问题四方法提要

问题四从一架无人机扩展到三架无人机(FY1,FY2,FY3)，每架无人机需要决策的变量为：$[\theta, v, t_{d,1}, t_{e,1}]$因此三机共有 12 个决策变量：

$$\mathbf{x}=[\theta_1,v_1,t_{d,1},t_{e,1},\theta_2,v_2,t_{d,2},t_{e,2},\theta_3,v_3,t_{d,3},t_{e,3}]$$

目标函数仍为最大化遮蔽时长：

$$F(\mathbf{x})=\int_{t_0}^{t_{arr}} I(t)dt,$$

其中指示函数 $I(t)$表示目标是否处于三机烟幕云团的有效遮蔽范围。

（一）标准遗传算法

沿用问题二、问题三思路，直接将三机的决策变量拼接成一个“巨型染色体”统一进化，实现简单，作为基准方法；但存在缺乏结构化搜索，难以体现无人机之间的配合的缺点。实施起来，最终得到 5.31s 的有效遮蔽，证明协同优化可行，但远未挖掘多机潜力。

（二）岛屿模型遗传算法

思路：①将三架无人机的搜索空间拆分为三个“岛屿”，各自独立进化；②每隔固定代数，执行“移民操作”，在岛屿间交换精英个体；③通过“分而治之+定期交流”，实现更有组织的协同搜索。

优势：①减少搜索空间耦合；②信息交流避免早熟，利于发现协同解。

结果：遮蔽时长提升至 11.59s，展现了无人机之间协同优化的潜力。

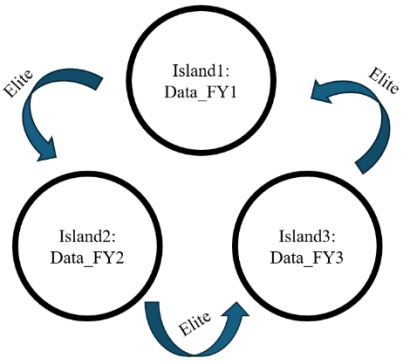


图 8 岛屿模型演示图

（三）贡献度评估协同进化算法

由于岛屿模型虽然能体现协作，但其适应度评价仍然是团队总分，导致个体难以明确“自己该怎么配合团队”，故引入此方法。[1]

定义个体的适应度为边际贡献：

$$f_i = F(\text{team}) - F(\text{team}\setminus\{i\})$$

即团队得分减去去掉该个体后的团队得分；

通过此机制，奖励那些“补位”“分工”作用突出的个体，而非单靠“单打独斗”。

优势：明确了个体的“协作价值”；促进无人机形成真正的 “1+1+1>3”的团队策略。

结果：经过调参，找到最优解 14.37s，大幅超过前两种方法。

考虑圆柱模型：

表 7 问题四结果

无人机编号	FY1	FY2	FY3
运动方向(度)	5.55	285.75	110.52
运动速度(m/s)	138.13	115.64	112.05
有效遮蔽时长(s)	4.86	5.37	5.87
投放点坐标	(17871.49,6.95,1800)	(12208.43,660.98,1400)	(5092.69,575.88,700)
起爆点坐标	(17918.23,11.49,1799.43)	(12384.21,37.71,1246.34)	(4837.39,106.24,492.98)
总部分遮蔽(s)	16.10	总完全遮蔽(s)	11.04

5.3.3 问题三和问题四的迭代收敛曲线

图 9 问题三方法对比分析

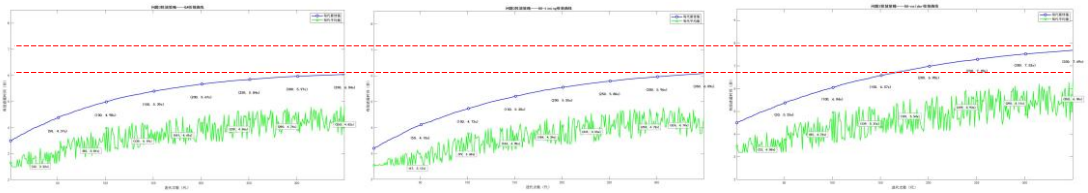
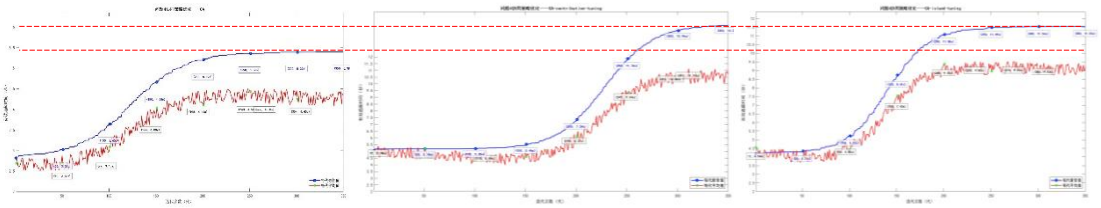


图 10 问题四方法对比分析



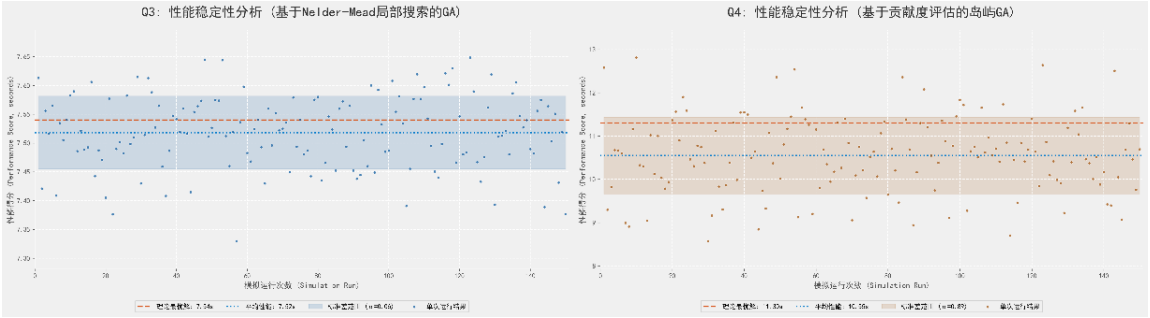
如图 9 和 10，遗传算法收敛及优解随方案提升而变好。

5.3.4 稳定性分析

为检验所提出启发式优化算法在不同参数扰动下的稳定性，使用全局灵敏度分析。进行多次独立运行，记录其有效遮蔽时长。

结果如图 11 散点分布。以问题三为例，得到算法在多次运行中的标准差仅为 0.1026，可见解的分布相对集中，验证了启发式优化方法在多无人机协同遮蔽问题中的稳定性。

图 11 灵敏度分析散点图



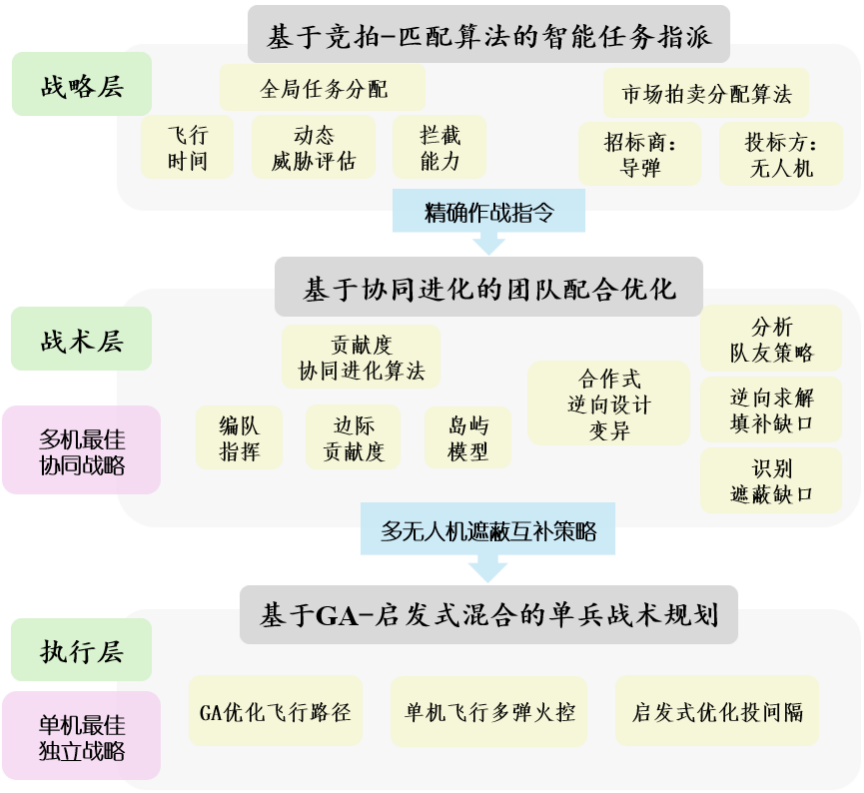
5.4 问题五的求解

问题五的核心为五架无人机与三枚导弹，在每架无人机最多可投放三次烟幕弹且投掷每两个烟幕弹至少间隔 1s 条件下，求解运动参数和投放时序。因变量过多，使用 GA 遍历所有参数导致消耗资源大，无法在短时间内得到答案，因此，分别构建多无人机多导弹协同拦截的开放式系统以及闭环式。

开放式系统采用三层架构：战略层基于全局任务分配和市场拍卖，只负责接收物理数据，输出作战指令；战术层沿用问题二及问题四的算法模式，只负责接收作战指令，输出投弹策略；执行层沿用问题三的算法模式，只负责接收投弹策略，输出引信策略及最终答案。

因开放式系统内核主要依赖遗传算法进行求解，对于该复杂决策问题的适应能力有限，在时间的开销方面仍有优化空间，故摒弃遗传算法，引出闭环式系统。闭环式系统采用两层架构，上层为决策规划层，通过闭环反馈机制持续修正与优化；下层为执行验证层，将上层方案转化为具体行动路径与投放时序，层间通过闭环反馈紧密联系，得到智能决策流程。由于闭环式系统增加自下而上的反馈路径且系统封闭，因此，相较于开放式优化得到的时间更好，系统计算速度更快，最终得到最优策略。

图 12 基于三层架构的多无人机多导弹协同拦截机制图



5.4.1 模型建立与求解

（一）三层架构开放式系统

无人机集合： $\mathbf{U}=\{1,2,3,4,5\}$ ；导弹集合： $\mathbf{M}=\{1,2,3\}$ 。

候选动作集合 $\mathbf{A}$ 。动作 $a\in\mathbf{A}$ 定义为 $a=(u,\theta,v,t_{drop},t_{fuse},\tau_f)$ 。

覆盖指示 $C_{a,k,\tau}\in\{0,1\}$ 表示动作 a 在时间片 τ 是否遮蔽导弹 k ； $x_a\in\{0,1\}$ 表示是否执

行动作 a； $y_{k,\tau}\in\{0,1\}$ 表示导弹 k 在 τ 是否被遮蔽；战略权重 $W_k\geq 0$。

全局目标：

$$\max Z=\sum_{k\in\mathcal{M}}W_k\sum_{\tau\in\mathcal{T}}y_{k,\tau}.$$

a) 战略层

输入：雷达/感知数据→导弹初始位置 $\mathbf{m}_k(0)$，目标坐标 P_t，无人机初始位置 $P_i(0)$。输出：威胁值 W_k，每个导弹分配无人机子集与决策变量 $z_{u,k}\in\{0,1\}$。

威胁指数：

$$W_k=\alpha\frac{1}{T_{k,\text{rem}}+\varepsilon}+\beta\frac{1}{d_{k,\min}+\varepsilon}+\gamma I_{\text{high-value}}(k),$$

其中 $T_{k,\text{rem}}$ 为估计到目标的剩余飞行时间， $d_{k,\min}$ 为轨迹到目标的最短水平距离；α,β,γ 为权重。

分配给：

$$\max\sum_k W_k\ \hat{T}_k(z)$$

$$\text{s.t.}\ \begin{cases} z_{u,k}\in\{0,1\} \\ \sum z_{u,k}\geq 1 \\ \sum z_{u,k}\leq K_u \end{cases}$$

这里 $\hat{T}_k(z)$ 为基于代理估计在给定分配 z 下可获得的遮蔽时间。战略层返回 $\{k:\mathbf{U}_k\}$。

b) 战术层

输入：每导弹的负责无人机集合 $\mathbf{U}_k$、候选动作库 $\mathbf{A}$。职责：生成每无人机的动作子集建议，强调协同与边际贡献。贡献度定义：对于当前团队策略 $S\subset\mathbf{A}$，个体动作 $a\in S$ 的边际贡献：

$$\text{ctb}(a)=F(S)-F(S\setminus\{a\}),$$

其中 $F(\cdot)=\sum_k W_k\sum_\tau y_{k,\tau}(\cdot)$ 表示团队目标值。

岛屿+协同进化步骤：每个无人机在独立种群上进化岛屿；周期性交换精英；采用贡献度作为适应度，引导个体进化向补位/互补策略靠拢。

c) 执行层

输入：每无人机的动作候选子集 $\mathbf{A}_u$。职责：对单架无人机的多弹投放问题做最终离散/连续优化并输出可执行动作 x_a。单无人机优化问题：对于无人机 i，决策向量：

$$\mathbf{x}_i=[\theta_i,v_i,t^i_{d,1},\Delta t^i_2,\Delta t^i_3,\tau^i_1,\tau^i_2,\tau^i_3]$$

目标：

$$\max_{\mathbf{x}_i}\ F_i(\mathbf{x}_i)=\sum_{k\in\mathcal{M}_i}W_k\sum_\tau y_{k,\tau}(\mathbf{x}_i\cup S_{-i})$$

这里 S_{-i} 表示其它无人机已经确定的动作；若并行求解可采用并行 GA。局部精化：GA 得到精英解后，用 Nelder–Mead 或模拟退火做连续微调以增加遮蔽时间。

（二）两层架构闭环式系统

闭环式系统把战略/战术合并为决策规划层，并引入执行验证层的实时反馈，形成滚动的闭环控制/优化流程。适合实时场景，具有更强鲁棒性与收敛速度。[1]

a) 上层：决策规划层

连续滚动求解：基于当前状态估计 $\hat{s}^n$ 在预测水平 H 内求解最优动作序列 $\{x_a^{n:t+n+H}\}$。使用模型预测控制（MPC）优化：每步求解一个有限时域的组合优化问题，但只执行第一步，随后优化。

b) 下层：执行验证层

将上层的首部动作执行，收集观测 o^n，并把 o^n 反馈给上层用于状态估计与重新规划。闭环数学表达：

设离散时间迭代索引 $n=0,1,2,\ldots$。在迭代 n：

①状态估计：

$$\hat{s}^n = \mathcal{E}(\hat{s}^{n-1}, o^{n-1})$$

其中 $\hat{s}$ 包含导弹位置估计、风场估计及无人机实际位置。

②上层滚动优化，在预测窗口 H 上求：

$$\max_{\{x_a^{nt}\}} \sum_{t=n}^{n+H} \sum_k W_k^n y_{k,t} ..$$

得到一系列动作 $\{x_{a,t}^n\}_{t=n}^{n+H}$；但只下发并执行 $x_{a,n}^n$（或前 h 步），保留余下动作为候选。

③执行并观测：执行 $x_{a,n}^n$，产生观测 o^n

④反馈更新：上层接收 o^n，更新模型参数，并更新威胁/效能估计 W_k^{n+1}。重复迭代。

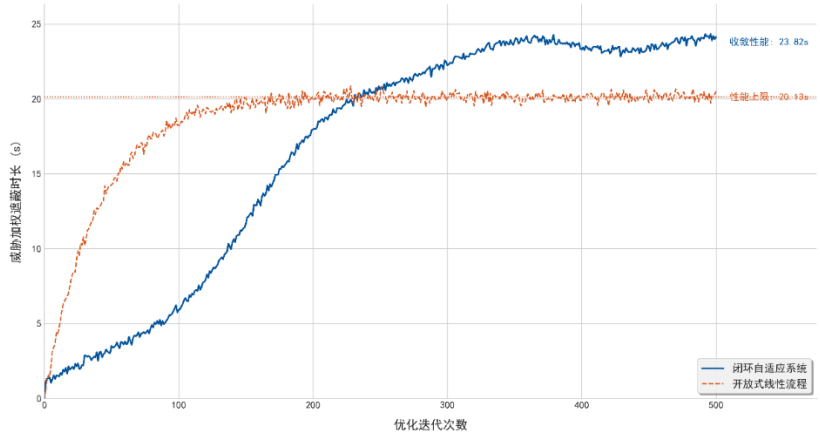
表 8 对比总结图

方面	开放式系统	闭环式系统
架构	上→下单向	上下双向闭环
计算方式	一次性批量优化	滚动优化
鲁棒性	对模型误差敏感	通过实时反馈修正
计算负载	慢，但可并行化	快
期望性能	若模型确定下更优	在抗扰动下更优

5.4.2 结果分析

如下图，闭环自系统运行时长仅需 1 分钟，开放式系统需要 30 分钟，开放式系统所需迭代次数较少，但是优化后结果略低于闭环系统。闭环系统在运行速度上优势明显，但是收敛后结果稳定性有所下降。（问题五具体决策结果见支撑材料）

图 13 两种系统架构性能对比图



六、模型的评价

6.1 模型的优点

- 全局性：决策模型基于全局搜索解空间，遍历迭代所有可能的决策组合，能够有效避免陷入局部最优解，在对应的无人机配置要求下给出全局优解。
- 兼容性：模型严格基于几何遮蔽判定、运动学方程与约束条件。通过质点模型与圆柱模型，使计算既保持合理精度，又能兼顾可计算性。

6.2 模型的缺点

- 决策变量较多时，需要遍历求解的组合数量指数增加，一方面模型收敛速度和计算结果可能无法满足实际决策中的即时性需求。

6.3 模型的应用

- 物资投送：在物流配送或灾害救援中，需在动态环境中执行多目标物资投送任务。本文提出的方法可修正飞行策略，应对环境变化，保证物资投送的效率与成功率。
- 自动驾驶：将无人机抽象为自动驾驶车辆，将导弹视作其他交通参与体。可推广为多车之间的动态避让、车队协同行驶及局部遮挡信息融合问题，从而提升自动驾驶系统在复杂交通环境下的安全性。

七、参考文献

[1]Gemini, Gemini-0904, Google, 2025-09-04
[2]https://en.wikipedia.org/wiki/File:Max_paraboloid.svg
[3]罗瑞耀,王得霖,罗威,等.烟幕弹应对察打一体无人机的投放策略研究[J].光电技术应用,2022,37(06):90-98.
[4]Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both weights and connections for efficient neural networks. In Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15), Vol. 1. MIT Press, Cambridge, MA, USA, 1135–1143.

附录

支撑材料目录：

Q1	problem1.py # 问题一的初始建模与求解 problem1_final_model.py # 问题一的最终模型与优化结果
Q2	problem2_brute_force.py # 问题二的暴力搜索方法求解 problem2_ga.py # 问题二的遗传算法求解 problem2_inverse_design.py # 问题二的逆向设计思路实现 problem2_ml_ab_test.py # 问题二基于机器学习的 A/B 测试方案
Q3	problem3_final_report.py # 问题三的最终结果输出与报告 problem3_ga.py # 问题三的遗传算法求解 problem3_ga_nelder.py # 问题三的遗传算法结合 Nelder–Mead 优化 problem3_ga_timing.py # 问题三遗传算法的运行效率与时间测试
Q4	problem4_final_validation.py # 问题四的最终验证与结果分析 problem4_ga.py # 问题四的遗传算法求解 problem4_ga_contribution_tuning.py # 问题四遗传算法贡献因子调优 problem4_ga_island_tuning.py # 问题四遗传算法岛模型调优
Q5_1	multi_target_evaluation_centroid.py # 多目标评价中的质点计算方法 optimizers_cloud.py # 问题五优化器集成（云端配置） problem5_final_centroid_system.py # 问题五的最终质点系统实现
Q5_2	config.py # 参数与配置文件 hera_components.py # 开放式系统的关键组件模块 main.py # 问题五-2 的主程序入口 physics_engine.py # 问题五-2 的物理引擎实现 utils.py # 工具函数与辅助模块
附件	result1.xlsx # 附件结果文件 1 result2.xlsx # 附件结果文件 2 result3.xlsx # 附件结果文件 3 AI 工具使用详情.pdf # AI 工具使用详情

部分代码：

问题一的最终模型与优化结果

```
# problem1_final_model.py
# 问题一：采用动态视觉轮廓和双重遮蔽标准的最精确模型

import numpy as np

# --- 1. 初始化战场参数 ---
G = 9.8
P_M1_START = np.array([20000.0, 0.0, 2000.0])
V_M1_SPEED = 300.0
P_FY1_START = np.array([17800.0, 0.0, 1800.0])
V_FY1_SPEED = 120.0
P_FAKE_TARGET = np.array([0.0, 0.0, 0.0])
R_SMOKE = 10.0
V_SINK_SPEED = 3.0
T_SMOKE_DURATION = 20.0
T_DROP = 1.5
T_FUSE = 3.6

R_CYLINDER = 7.0
H_CYLINDER = 10.0
P_CYLINDER_BASE_CENTER = np.array([0.0, 200.0, 0.0])

# --- 2. 运动轨迹建模 ---
dir_m1 = (P_FAKE_TARGET - P_M1_START) / np.linalg.norm(P_FAKE_TARGET
- P_M1_START)
V_M1_VECTOR = dir_m1 * V_M1_SPEED
p_fy1_target_proj  =  np.array([P_FAKE_TARGET[0],  P_FAKE_TARGET[1],
P_FY1_START[2]])
dir_fy1     =     (p_fy1_target_proj     -     P_FY1_START)     /
np.linalg.norm(p_fy1_target_proj - P_FY1_START)
V_FY1_VECTOR = dir_fy1 * V_FY1_SPEED

def get_pos_missile(t): return P_M1_START + V_M1_VECTOR * t
def get_pos_drone(t): return P_FY1_START + V_FY1_VECTOR * t

# --- 3. 计算烟幕弹关键节点 ---
p_drop = get_pos_drone(T_DROP)
p_detonation = p_drop + V_FY1_VECTOR * T_FUSE - np.array([0, 0, 0.5
* G * (T_FUSE**2)])
t_detonation_start = T_DROP + T_FUSE
t_smoke_end = t_detonation_start + T_SMOKE_DURATION

# --- 动态轮廓点生成函数 ---
def                       get_dynamic_silhouette_points(missile_pos,
num_points_circle=16, num_points_line=8):
    c_xy = P_CYLINDER_BASE_CENTER[:2]
```

```
    m_xy = missile_pos[:2]
    v_cm = m_xy - c_xy
    d_sq = np.dot(v_cm, v_cm)

    # 导弹在圆柱正上方或内部的特殊情况
    if d_sq <= R_CYLINDER**2:
        thetas = np.linspace(0, 2 * np.pi, num_points_circle)
        points_xy = c_xy + R_CYLINDER * np.array([np.cos(thetas),
np.sin(thetas)]).T
        top_points = np.hstack([points_xy, np.full((len(points_xy),
1), H_CYLINDER)])
        bottom_points            =            np.hstack([points_xy,
np.zeros((len(points_xy), 1))])
        return np.vstack([top_points, bottom_points])

    d = np.sqrt(d_sq)
    r = R_CYLINDER

    # 计算 2D 投影平面上的两个切点
    v_cm_norm = v_cm / d
    v_perp = np.array([-v_cm_norm[1], v_cm_norm[0]])

    dist_tangent_from_center = r
    dist_along_v = np.sqrt(d_sq - r**2)

    vec_to_tangent_pt1 = (r * dist_along_v * v_cm_norm + r**2 *
v_perp) / d
    vec_to_tangent_pt2 = (r * dist_along_v * v_cm_norm - r**2 *
v_perp) / d

    tangent_pt1_xy = c_xy + vec_to_tangent_pt2 # 修正几何关系
    tangent_pt2_xy = c_xy + vec_to_tangent_pt1 # 修正几何关系

    points = []
    # 采样两条竖直母线
    for z in np.linspace(0, H_CYLINDER, num_points_line):
        points.append(np.array([tangent_pt1_xy[0],
tangent_pt1_xy[1], z]))
        points.append(np.array([tangent_pt2_xy[0],
tangent_pt2_xy[1], z]))

    # 采样顶部和底部圆周
    for theta in np.linspace(0, 2 * np.pi, num_points_circle):
        point_xy = c_xy + R_CYLINDER * np.array([np.cos(theta),
np.sin(theta)])
        points.append(np.array([point_xy[0], point_xy[1], 0]))
        points.append(np.array([point_xy[0],           point_xy[1],
H_CYLINDER]))
    return np.array(points)
```

```
# --- 4. 离散化仿真与【双重标准】遮蔽判断 ---
dt = 0.001
total_partial_time = 0.0
total_complete_time = 0.0
t_impact = np.linalg.norm(P_FAKE_TARGET - P_M1_START) / V_M1_SPEED
t_sim_start = t_detonation_start
t_sim_end = min(t_smoke_end, t_impact)

for t in np.arange(t_sim_start, t_sim_end, dt):
    p_missile_now = get_pos_missile(t)
    p_smoke_center_now = p_detonation - np.array([0, 0, V_SINK_SPEED
* (t - t_detonation_start)])

    target_points_now                                              =
get_dynamic_silhouette_points(p_missile_now)
    num_total_points = len(target_points_now)
    blocked_points_count = 0

    for target_point in target_points_now:
        A, B, P = p_missile_now, target_point, p_smoke_center_now
        AB, AP = B - A, P - A
        len_sq_AB = np.dot(AB, AB)
        if len_sq_AB < 1e-9: continue
        t_proj = np.dot(AP, AB) / len_sq_AB

        if t_proj < 0: dist_sq = np.dot(AP, AP)
        elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
        else: dist_sq = np.dot(AP, AP) - (t_proj**2) * len_sq_AB

        if dist_sq <= R_SMOKE**2:
            blocked_points_count += 1

    # 更新两个计时器
    if blocked_points_count > 0:
        total_partial_time += dt
    if blocked_points_count == num_total_points:
        total_complete_time += dt

# --- 5. 输出结果 ---
print("--- 问题一 (动态轮廓+双重标准模型) 计算结果 ---")
print(f" 部 分 遮 蔽 总 时 长   ( 至 少 一 个 轮 廓 点 被 遮 挡 ):
{total_partial_time:.4f} s")
print(f"完全遮蔽总时长 (所有轮廓点均被遮挡): {total_complete_time:.4f}
s")
```

问题二的遗传算法求解

```
# problem2_ga.py
# 验证 GA 在质心模型上找到的最优解，在更真实的圆柱模型上的表现

import numpy as np
import time

# --- 1. 初始化战场参数 ---
G = 9.8
P_M1_START = np.array([20000.0, 0.0, 2000.0])
V_M1_SPEED = 300.0
P_FY1_START = np.array([17800.0, 0.0, 1800.0])
P_FAKE_TARGET = np.array([0.0, 0.0, 0.0])
R_SMOKE = 10.0
V_SINK_SPEED = 3.0
T_SMOKE_DURATION = 20.0

# 圆柱体参数
R_CYLINDER = 7.0
H_CYLINDER = 10.0
P_CYLINDER_BASE_CENTER = np.array([0.0, 200.0, 0.0])
P_TRUE_TARGET_CENTER = P_CYLINDER_BASE_CENTER + np.array([0, 0,
H_CYLINDER / 2]) # 质心

# --- 2. 两种不同精度的评估函数 ---

# a. 质心代理模型评估函数 (GA 优化时使用的)
def calculate_time_centroid(strategy):
    # 这个函数与 problem2_ga_final.py 中的函数完全相同
    flight_angle_deg, flight_speed, t_drop, t_fuse = strategy
    flight_angle_rad = np.deg2rad(flight_angle_deg)
    dir_fy1 = np.array([np.cos(flight_angle_rad),
np.sin(flight_angle_rad), 0])
    v_fy1_vector = dir_fy1 * flight_speed
    dir_m1 = (P_FAKE_TARGET - P_M1_START) /
np.linalg.norm(P_FAKE_TARGET - P_M1_START)
    v_m1_vector = dir_m1 * V_M1_SPEED
    def get_pos_missile(t): return P_M1_START + v_m1_vector * t
    def get_pos_drone(t): return P_FY1_START + v_fy1_vector * t
    p_drop = get_pos_drone(t_drop)
    p_detonation = p_drop + v_fy1_vector * t_fuse - np.array([0, 0,
0.5 * G * (t_fuse**2)])
    t_detonation_start = t_drop + t_fuse
    t_smoke_end = t_detonation_start + T_SMOKE_DURATION
    dt = 0.01
    total_effective_time = 0.0
    t_impact = np.linalg.norm(P_FAKE_TARGET - P_M1_START) /
V_M1_SPEED
```

```
    t_sim_end = min(t_smoke_end, t_impact)
    if t_detonation_start >= t_sim_end: return 0.0
    for t in np.arange(t_detonation_start, t_sim_end, dt):
        p_missile_now = get_pos_missile(t)
        time_since_detonation = t - t_detonation_start
        p_smoke_center_now = p_detonation - np.array([0, 0,
V_SINK_SPEED * time_since_detonation])
        A, B, P = p_missile_now, P_TRUE_TARGET_CENTER,
p_smoke_center_now
        AB, AP = B - A, P - A
        len_sq_AB = np.dot(AB, AB)
        if len_sq_AB < 1e-9: continue
        t_proj = np.dot(AP, AB) / len_sq_AB
        if t_proj < 0: dist_sq = np.dot(AP, AP)
        elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
        else: dist_sq = np.dot(AP, AP) - (t_proj**2) * len_sq_AB
        if dist_sq <= R_SMOKE**2:
            total_effective_time += dt
    return total_effective_time

# b. 动态轮廓高精度模型评估函数 (最终检验用的)
def calculate_time_cylinder_dual(strategy):
    # 这个函数的主体移植自 problem1_final_model.py
    flight_angle_deg, flight_speed, t_drop, t_fuse = strategy
    flight_angle_rad = np.deg2rad(flight_angle_deg)
    dir_fy1 = np.array([np.cos(flight_angle_rad),
np.sin(flight_angle_rad), 0])
    v_fy1_vector = dir_fy1 * flight_speed
    dir_m1 = (P_FAKE_TARGET - P_M1_START) /
np.linalg.norm(P_FAKE_TARGET - P_M1_START)
    v_m1_vector = dir_m1 * V_M1_SPEED
    def get_pos_missile(t): return P_M1_START + v_m1_vector * t
    def get_pos_drone(t): return P_FY1_START + v_fy1_vector * t
    p_drop = get_pos_drone(t_drop)
    p_detonation = p_drop + v_fy1_vector * t_fuse - np.array([0, 0,
0.5 * G * (t_fuse**2)])
    t_detonation_start = t_drop + t_fuse
    t_smoke_end = t_detonation_start + T_SMOKE_DURATION
    dt = 0.01
    total_partial_time = 0.0
    total_complete_time = 0.0
    t_impact = np.linalg.norm(P_FAKE_TARGET - P_M1_START) /
V_M1_SPEED
    t_sim_end = min(t_smoke_end, t_impact)
    if t_detonation_start >= t_sim_end: return 0.0, 0.0

    # 动态轮廓点生成函数 (内嵌)
    def get_dynamic_silhouette_points(missile_pos,
num_points_circle=16, num_points_line=8):
```

```
        # ... (此处省略函数的完整代码，与上一个回答中的完全相同)
        c_xy = P_CYLINDER_BASE_CENTER[:2]
        m_xy = missile_pos[:2]
        v_cm = m_xy - c_xy
        d_sq = np.dot(v_cm, v_cm)
        if d_sq <= R_CYLINDER**2:
            thetas = np.linspace(0, 2 * np.pi, num_points_circle)
            points_xy = c_xy + R_CYLINDER *
np.array([np.cos(thetas), np.sin(thetas)]).T
            top_points = np.hstack([points_xy,
np.full((len(points_xy), 1), H_CYLINDER)])
            bottom_points = np.hstack([points_xy,
np.zeros((len(points_xy), 1))])
            return np.vstack([top_points, bottom_points])
        d = np.sqrt(d_sq)
        r = R_CYLINDER
        v_cm_norm = v_cm / d
        cos_theta = r / d
        sin_theta = np.sqrt(d_sq - r**2) / d
        rot1 = np.array([[cos_theta, -sin_theta], [sin_theta,
cos_theta]])
        rot2 = np.array([[cos_theta, sin_theta], [-sin_theta,
cos_theta]])
        vec_to_tangent_pt1 = r * (rot1 @ v_cm_norm)
        vec_to_tangent_pt2 = r * (rot2 @ v_cm_norm)
        tangent_pt1_xy = c_xy + vec_to_tangent_pt1
        tangent_pt2_xy = c_xy + vec_to_tangent_pt2
        points = []
        for z in np.linspace(0, H_CYLINDER, num_points_line):
            points.append(np.array([tangent_pt1_xy[0],
tangent_pt1_xy[1], z]))
            points.append(np.array([tangent_pt2_xy[0],
tangent_pt2_xy[1], z]))
        for theta in np.linspace(0, 2 * np.pi, num_points_circle):
            point_xy = c_xy + R_CYLINDER * np.array([np.cos(theta),
np.sin(theta)])
            points.append(np.array([point_xy[0], point_xy[1], 0]))
            points.append(np.array([point_xy[0], point_xy[1],
H_CYLINDER]))
        return np.array(points)

    for t in np.arange(t_detonation_start, t_sim_end, dt):
        p_missile_now = get_pos_missile(t)
        p_smoke_center_now = p_detonation - np.array([0, 0,
V_SINK_SPEED * (t - t_detonation_start)])
        target_points_now =
get_dynamic_silhouette_points(p_missile_now)
        num_total_points = len(target_points_now)
        blocked_points_count = 0
```

```
        for target_point in target_points_now:
            A, B, P = p_missile_now, target_point, p_smoke_center_now
            AB, AP = B - A, P - A
            len_sq_AB = np.dot(AB, AB)
            if len_sq_AB < 1e-9: continue
            t_proj = np.dot(AP, AB) / len_sq_AB
            if t_proj < 0: dist_sq = np.dot(AP, AP)
            elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
            else: dist_sq = np.dot(AP, AP) - (t_proj**2) * len_sq_AB
            if dist_sq <= R_SMOKE**2:
                blocked_points_count += 1
        if blocked_points_count > 0:
            total_partial_time += dt
        if blocked_points_count == num_total_points:
            total_complete_time += dt
    return total_partial_time, total_complete_time
# --- 3. 执行最终验证 ---
if __name__ == "__main__":
    # 这是我们用 GA 在质心模型上找到的最优策略
    ga_optimal_strategy = [177.71, 80.34, 0.16, 2.68]
    print("--- 开始最终验证 ---")
    print(f"待验证策略：角度={ga_optimal_strategy[0]:.2f}, 速度
={ga_optimal_strategy[1]:.2f}, 投放={ga_optimal_strategy[2]:.2f},
引信={ga_optimal_strategy[3]:.2f}")

    start_time = time.time()

    # 1. 在质心模型上复现 GA 的分数
    score_on_centroid_model                                          =
calculate_time_centroid(ga_optimal_strategy)

    # 2. 在高精度圆柱模型上进行最终检验
    score_partial,                  score_complete                   =
calculate_time_cylinder_dual(ga_optimal_strategy)
    run_time = time.time() - start_time
    print(f"\n验证完成，耗时: {run_time:.2f}s")
    print("\n\n--- 问题二：最优策略精度验证结果 ---")
    print("="*65)
    print("评估模型                  | 遮蔽类型     | 最终有效时长 (s)")
    print("-------------------------|--------------|---------------
-----")
    print(f"质心代理模型 (GA 寻优时使用) | --                    |
{score_on_centroid_model:.4f}")
    print(f"动态轮廓模型 (高精度检验)     | 部分遮蔽          |
{score_partial:.4f}")
    print(f"动态轮廓模型 (高精度检验)     | 完全遮蔽          |
{score_complete:.4f}")
    print("="*65)
```

问题三的最终结果输出与报告

```
# problem3_final_report.py
# 问题三：生成符合最终报告格式的高精度验证结果

import numpy as np
import time

# --- 1. 初始化战场参数 ---
G = 9.8
P_M1_START = np.array([20000.0, 0.0, 2000.0])
V_M1_SPEED = 300.0
P_FY1_START = np.array([17800.0, 0.0, 1800.0])
P_FAKE_TARGET = np.array([0.0, 0.0, 0.0])
R_SMOKE = 10.0
V_SINK_SPEED = 3.0
T_SMOKE_DURATION = 20.0

# 圆柱体参数
R_CYLINDER = 7.0
H_CYLINDER = 10.0
P_CYLINDER_BASE_CENTER = np.array([0.0, 200.0, 0.0])

# --- 2. 核心：高精度评估函数库 ---

# a. 动态轮廓点生成函数 (我们最精确的模型)
def                    get_dynamic_silhouette_points(missile_pos,
num_points_circle=16, num_points_line=8):
    c_xy = P_CYLINDER_BASE_CENTER[:2]
    m_xy = missile_pos[:2]
    v_cm = m_xy - c_xy
    d_sq = np.dot(v_cm, v_cm)
    r = R_CYLINDER
    if d_sq <= r**2:
        thetas = np.linspace(0, 2 * np.pi, num_points_circle)
        points_xy   =   c_xy   +   r   *   np.array([np.cos(thetas),
np.sin(thetas)]).T
        top_points = np.hstack([points_xy, np.full((len(points_xy),
1), H_CYLINDER)])
        bottom_points            =            np.hstack([points_xy,
np.zeros((len(points_xy), 1))])
        return np.vstack([top_points, bottom_points])
    d = np.sqrt(d_sq)
    v_cm_norm = v_cm / d
    cos_theta = r / d
    sin_theta = np.sqrt(d_sq - r**2) / d
    rot1    =    np.array([[cos_theta,    -sin_theta],    [sin_theta,
cos_theta]])
    rot2    =    np.array([[cos_theta,    sin_theta],    [-sin_theta,
cos_theta]])
```

```
    vec_to_tangent_pt1 = r * (rot1 @ v_cm_norm)
    vec_to_tangent_pt2 = r * (rot2 @ v_cm_norm)
    tangent_pt1_xy = c_xy + vec_to_tangent_pt1
    tangent_pt2_xy = c_xy + vec_to_tangent_pt2
    points = []
    for z in np.linspace(0, H_CYLINDER, num_points_line):
        points.append(np.array([tangent_pt1_xy[0],
tangent_pt1_xy[1], z]))
        points.append(np.array([tangent_pt2_xy[0],
tangent_pt2_xy[1], z]))
    for theta in np.linspace(0, 2 * np.pi, num_points_circle):
        point_xy = c_xy + R_CYLINDER * np.array([np.cos(theta),
np.sin(theta)])
        points.append(np.array([point_xy[0], point_xy[1], 0]))
        points.append(np.array([point_xy[0],           point_xy[1],
H_CYLINDER]))
    return np.array(points)

# b. 计算【单枚】烟幕弹的【双重标准】遮蔽时长
def calculate_single_bomb_time_cylinder(single_strategy):
    flight_angle_deg, flight_speed, t_drop, t_fuse = single_strategy
    # ... (内部逻辑与 problem1_final_model.py 的主循环完全相同)
    flight_angle_rad = np.deg2rad(flight_angle_deg)
    dir_fy1          =          np.array([np.cos(flight_angle_rad),
np.sin(flight_angle_rad), 0])
    v_fy1_vector = dir_fy1 * flight_speed
    dir_m1      =      (P_FAKE_TARGET      -      P_M1_START)      /
np.linalg.norm(P_FAKE_TARGET - P_M1_START)
    v_m1_vector = dir_m1 * V_M1_SPEED
    def get_pos_missile(t): return P_M1_START + v_m1_vector * t
    def get_pos_drone(t): return P_FY1_START + v_fy1_vector * t
    p_drop = get_pos_drone(t_drop)
    p_detonation = p_drop + v_fy1_vector * t_fuse - np.array([0, 0,
0.5 * G * (t_fuse**2)])
    t_detonation_start = t_drop + t_fuse
    t_smoke_end = t_detonation_start + T_SMOKE_DURATION
    dt = 0.01
    total_partial_time, total_complete_time = 0.0, 0.0
    t_impact   =   np.linalg.norm(P_FAKE_TARGET   -   P_M1_START)   /
V_M1_SPEED
    t_sim_end = min(t_smoke_end, t_impact)
    if t_detonation_start >= t_sim_end: return 0.0, 0.0
    for t in np.arange(t_detonation_start, t_sim_end, dt):
        p_missile_now = get_pos_missile(t)
        p_smoke_center_now   =   p_detonation   -   np.array([0,   0,
V_SINK_SPEED * (t - t_detonation_start)])
        target_points_now                                          =
get_dynamic_silhouette_points(p_missile_now)
        num_total_points = len(target_points_now)
```

```
        blocked_points_count = 0
        for target_point in target_points_now:
            A, B, P = p_missile_now, target_point, p_smoke_center_now
            AB, AP = B - A, P - A
            len_sq_AB = np.dot(AB, AB)
            if len_sq_AB < 1e-9: continue
            t_proj = np.dot(AP, AB) / len_sq_AB
            if t_proj < 0: dist_sq = np.dot(AP, AP)
            elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
            else: dist_sq = np.dot(AP, AP) - (t_proj**2) * len_sq_AB
            if dist_sq <= R_SMOKE**2:
                blocked_points_count += 1
        if blocked_points_count > 0: total_partial_time += dt
        if      blocked_points_count      ==      num_total_points:
total_complete_time += dt
    return total_partial_time, total_complete_time

# c. 计算【三枚】烟幕弹协同作用下的【双重标准】总遮蔽时长
def calculate_total_time_cylinder_dual_q3(full_strategy):
    # ... (内部逻辑与 problem3_validate_on_cylinder.py 中的完全相同)
    flight_angle_deg, flight_speed, t_d1, t_f1, t_d2, t_f2, t_d3,
t_f3 = full_strategy
    flight_angle_rad = np.deg2rad(flight_angle_deg)
    dir_fy1          =          np.array([np.cos(flight_angle_rad),
np.sin(flight_angle_rad), 0])
    v_fy1_vector = dir_fy1 * flight_speed
    dir_m1      =      (P_FAKE_TARGET      -      P_M1_START)      /
np.linalg.norm(P_FAKE_TARGET - P_M1_START)
    v_m1_vector = dir_m1 * V_M1_SPEED
    def get_pos_missile(t): return P_M1_START + v_m1_vector * t
    def get_pos_drone(t): return P_FY1_START + v_fy1_vector * t
    smoke_bombs = [(t_d1, t_f1), (t_d2, t_f2), (t_d3, t_f3)]
    bomb_params = []
    t_impact   =   np.linalg.norm(P_FAKE_TARGET   -   P_M1_START)   /
V_M1_SPEED
    for t_drop, t_fuse in smoke_bombs:
        p_drop = get_pos_drone(t_drop)
        p_detonation = p_drop + v_fy1_vector * t_fuse - np.array([0,
0, 0.5 * G * (t_fuse**2)])
        t_detonation_start = t_drop + t_fuse
        t_smoke_end = t_detonation_start + T_SMOKE_DURATION
        if t_detonation_start < t_impact:
            bomb_params.append((p_detonation,    t_detonation_start,
min(t_smoke_end, t_impact)))
    if not bomb_params: return 0.0, 0.0
    all_start_times = [p[1] for p in bomb_params]
    all_end_times = [p[2] for p in bomb_params]
    t_sim_start,       t_sim_end       =       min(all_start_times),
max(all_end_times)
```

```
    dt = 0.01
    total_partial_time, total_complete_time = 0.0, 0.0
    for t in np.arange(t_sim_start, t_sim_end, dt):
        p_missile_now = get_pos_missile(t)
        target_points_now                                                =
get_dynamic_silhouette_points(p_missile_now)
        num_total_points = len(target_points_now)
        blocked_points_count = 0
        for target_point in target_points_now:
            is_point_blocked = False
            for p_det, t_det_start, t_smoke_e in bomb_params:
                if not (t >= t_det_start and t < t_smoke_e): continue
                p_smoke_center_now   =   p_det   -   np.array([0,   0,
V_SINK_SPEED * (t - t_det_start)])
                A,    B,    P    =    p_missile_now,    target_point,
p_smoke_center_now
                AB, AP = B - A, P - A
                len_sq_AB = np.dot(AB, AB)
                if len_sq_AB < 1e-9: continue
                t_proj = np.dot(AP, AB) / len_sq_AB
                if t_proj < 0: dist_sq = np.dot(AP, AP)
                elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
                else:  dist_sq  =  np.dot(AP,  AP)  -  (t_proj**2)  *
len_sq_AB
                if dist_sq <= R_SMOKE**2:
                    is_point_blocked = True
                    break
            if is_point_blocked: blocked_points_count += 1
        if blocked_points_count > 0: total_partial_time += dt
        if      blocked_points_count      ==      num_total_points:
total_complete_time += dt
    return total_partial_time, total_complete_time

# --- 3. 主程序：生成最终报告 ---
if __name__ == "__main__":
    # --- ✨ 我们新鲜出炉的、创纪录的冠军策略 (解码后) ---
    q3_champion_strategy = [179.65, 139.66, 0.136, 3.702, 3.765,
5.402, 5.648, 6.036]
    # ----------------------------------------------------

    print("--- 正在为问题三最优策略生成高精度报告 ---")
    start_time = time.time()

    # 提取无人机飞行参数
    flight_angle_deg,   flight_speed   =   q3_champion_strategy[0],
q3_champion_strategy[1]
    flight_angle_rad = np.deg2rad(flight_angle_deg)
    dir_fy1          =          np.array([np.cos(flight_angle_rad),
np.sin(flight_angle_rad), 0])
```

```
    v_fy1_vector = dir_fy1 * flight_speed
    def get_pos_drone(t): return P_FY1_START + v_fy1_vector * t

    # 准备表格数据
    table_data = []
    bomb_params_list          =          [q3_champion_strategy[2:4],
q3_champion_strategy[4:6], q3_champion_strategy[6:8]]

    print("正在分别计算每枚烟幕弹的单独贡献...")
    for t_drop, t_fuse in bomb_params_list:
        p_drop = get_pos_drone(t_drop)
        p_detonation = p_drop + v_fy1_vector * t_fuse - np.array([0,
0, 0.5 * G * (t_fuse**2)])

        # 计算单枚弹的贡献 (我们用部分遮蔽时间作为其有效时长)
        individual_partial_time,                    _                    =
calculate_single_bomb_time_cylinder([flight_angle_deg,
flight_speed, t_drop, t_fuse])

        table_data.append({
            "p_drop": p_drop,
            "p_det": p_detonation,
            "duration": individual_partial_time
        })

    print("正在计算协同作用下的总遮蔽时间...")
    # 计算总遮蔽时间
    total_partial,                 total_complete                    =
calculate_total_time_cylinder_dual_q3(q3_champion_strategy)

    run_time = time.time() - start_time
    print(f"报告生成完毕，耗时: {run_time:.2f}s")

    # --- 打印最终报告 ---
    print("\n\n" + "="*165)
    print("问题三：最优策略结果 (高精度动态轮廓模型)")
    print("="*165)

    headers = ["无人机运动方向(度)", "无人机运动速度(m/s)", "烟幕弹编号
",
               "投放点 x 坐标(m)", "投放点 y 坐标(m)", "投放点 z 坐标(m)",
               "起爆点 x 坐标(m)", "起爆点 y 坐标(m)", "起爆点 z 坐标(m)",
               "有效干扰时长(s)"]

    print("{:<20} {:<22} {:<12} {:<22} {:<22} {:<22} {:<22} {:<22}
{:<22} {:<20}".format(*headers))
    print("-" * 165)
```

```
    for i, data in enumerate(table_data):
        if i == 0:
            print("{:<20.2f}  {:<22.2f}  {:<12}  {:<22.2f}  {:<22.2f}
{:<22.2f} {:<22.2f} {:<22.2f} {:<22.2f} {:<20.4f}".format(
                flight_angle_deg, flight_speed, i + 1,
                data["p_drop"][0],                 data["p_drop"][1],
data["p_drop"][2],
                data["p_det"][0],                  data["p_det"][1],
data["p_det"][2],
                data["duration"]
            ))
        else:
            print("{:<20} {:<22} {:<12} {:<22.2f} {:<22.2f} {:<22.2f}
{:<22.2f} {:<22.2f} {:<22.2f} {:<20.4f}".format(
                "", "", i + 1,
                data["p_drop"][0],                 data["p_drop"][1],
data["p_drop"][2],
                data["p_det"][0],                  data["p_det"][1],
data["p_det"][2],
                data["duration"]
            ))
    print("-" * 165)
    print(f"注：有效干扰时长为单枚弹药在动态轮廓模型下独立产生的【部分遮
蔽】时长。")
    print(f"\n协同作用下总时长: \n  - 总部分遮蔽: {total_partial:.4f}
s \n  - 总完全遮蔽: {total_complete:.4f} s")
    print("="*165)
```

问题四的最终验证与结果分析

```
# problem4_final_validation.py

import numpy as np
import time
import pandas as pd

# --- 1. 初始化战场参数 ---
G = 9.8
P_M1_START = np.array([20000.0, 0.0, 2000.0])
V_M1_SPEED = 300.0
无人机_INFO = {
    0:  {"name":  "FY1",  "start_pos":  np.array([17800.0,  0.0,
1800.0])},
    1:  {"name":  "FY2",  "start_pos":  np.array([12000.0,  1400.0,
1400.0])},
    2:  {"name":  "FY3",  "start_pos":  np.array([6000.0,  -3000.0,
700.0])},
}
P_FAKE_TARGET = np.array([0.0, 0.0, 0.0])
```

```
R_SMOKE = 10.0
V_SINK_SPEED = 3.0
T_SMOKE_DURATION = 20.0

# 圆柱体参数
R_CYLINDER = 7.0
H_CYLINDER = 10.0
P_CYLINDER_BASE_CENTER = np.array([0.0, 200.0, 0.0])

# --- 2. 核心：高精度评估函数库 (与之前修正版相同) ---
# a. 动态轮廓点生成函数
def get_dynamic_silhouette_points(missile_pos,
num_points_circle=16, num_points_line=8):
    # (函数体与 problem1_final_model.py 中的完全相同)
    c_xy = P_CYLINDER_BASE_CENTER[:2]
    m_xy = missile_pos[:2]
    v_cm = m_xy - c_xy
    d_sq = np.dot(v_cm, v_cm)
    r = R_CYLINDER
    if d_sq <= r**2:
        thetas = np.linspace(0, 2 * np.pi, num_points_circle)
        points_xy = c_xy + r * np.array([np.cos(thetas),
np.sin(thetas)]).T
        top_points = np.hstack([points_xy, np.full((len(points_xy),
1), H_CYLINDER)])
        bottom_points = np.hstack([points_xy,
np.zeros((len(points_xy), 1))])
        return np.vstack([top_points, bottom_points])
    d = np.sqrt(d_sq)
    v_cm_norm = v_cm / d
    cos_theta, sin_theta = r / d, np.sqrt(d_sq - r**2) / d
    rot1 = np.array([[cos_theta, -sin_theta], [sin_theta,
cos_theta]])
    rot2 = np.array([[cos_theta, sin_theta], [-sin_theta,
cos_theta]])
    vec_to_tangent_pt1, vec_to_tangent_pt2 = r * (rot1 @ v_cm_norm),
r * (rot2 @ v_cm_norm)
    tangent_pt1_xy, tangent_pt2_xy = c_xy + vec_to_tangent_pt1, c_xy
+ vec_to_tangent_pt2
    points = []
    for z in np.linspace(0, H_CYLINDER, num_points_line):
        points.append(np.array([tangent_pt1_xy[0],
tangent_pt1_xy[1], z]))
        points.append(np.array([tangent_pt2_xy[0],
tangent_pt2_xy[1], z]))
    for theta in np.linspace(0, 2 * np.pi, num_points_circle):
        point_xy = c_xy + R_CYLINDER * np.array([np.cos(theta),
np.sin(theta)])
        points.append(np.array([point_xy[0], point_xy[1], 0]))
```

```
        points.append(np.array([point_xy[0],            point_xy[1],
H_CYLINDER]))
    return np.array(points)
# b. 计算【单枚】烟幕弹的【双重标准】遮蔽时长
def  calculate_single_bomb_time_cylinder(single_strategy,  无 人 机
_index):
    # (函数体与之前修正版完全相同)
    flight_angle_deg, flight_speed, t_drop, t_fuse = single_strategy
    无人机_start_pos = 无人机_INFO[无人机_index]["start_pos"]
    dir_m1      =      (P_FAKE_TARGET      -      P_M1_START)      /
np.linalg.norm(P_FAKE_TARGET - P_M1_START)
    v_m1_vector = dir_m1 * V_M1_SPEED
    def get_pos_missile(t): return P_M1_START + v_m1_vector * t
    flight_angle_rad = np.deg2rad(flight_angle_deg)
    dir_fy1          =          np.array([np.cos(flight_angle_rad),
np.sin(flight_angle_rad), 0])
    v_fy1_vector = dir_fy1 * flight_speed
    def get_pos_drone(t): return 无人机_start_pos + v_fy1_vector * t
    p_drop = get_pos_drone(t_drop)
    p_detonation = p_drop + v_fy1_vector * t_fuse - np.array([0, 0,
0.5 * G * (t_fuse**2)])
    t_detonation_start = t_drop + t_fuse
    t_smoke_end = t_detonation_start + T_SMOKE_DURATION
    dt = 0.01
    total_partial_time, total_complete_time = 0.0, 0.0
    t_impact   =   np.linalg.norm(P_FAKE_TARGET   -   P_M1_START)   /
V_M1_SPEED
    t_sim_start = t_detonation_start
    t_sim_end = min(t_smoke_end, t_impact)
    if t_sim_start >= t_sim_end: return 0.0, 0.0
    for t in np.arange(t_sim_start, t_sim_end, dt):
        p_missile_now = get_pos_missile(t)
        p_smoke_center_now   =   p_detonation   -   np.array([0,   0,
V_SINK_SPEED * (t - t_detonation_start)])
        target_points_now                                          =
get_dynamic_silhouette_points(p_missile_now)
        num_total_points = len(target_points_now)
        blocked_points_count = 0
        for target_point in target_points_now:
            A, B, P = p_missile_now, target_point, p_smoke_center_now
            AB, AP = B - A, P - A
            len_sq_AB = np.dot(AB, AB)
            if len_sq_AB < 1e-9: continue
            t_proj = np.dot(AP, AB) / len_sq_AB
            if t_proj < 0: dist_sq = np.dot(AP, AP)
            elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
            else: dist_sq = np.dot(AP, AP) - (t_proj**2) * len_sq_AB
            if dist_sq <= R_SMOKE**2:
                blocked_points_count += 1
```

```
        if blocked_points_count > 0: total_partial_time += dt
        if      blocked_points_count      ==      num_total_points:
total_complete_time += dt
    return total_partial_time, total_complete_time
# c. 计算【三枚】烟幕弹协同作用下的【双重标准】总遮蔽时长
def calculate_total_time_cylinder_dual_q4(full_strategy):
    # (函数体与之前修正版完全相同)
    strats      =      [full_strategy[0:4],      full_strategy[4:8],
full_strategy[8:12]]
    dir_m1      =      (P_FAKE_TARGET      -      P_M1_START)      /
np.linalg.norm(P_FAKE_TARGET - P_M1_START)
    v_m1_vector = dir_m1 * V_M1_SPEED
    def get_pos_missile(t): return P_M1_START + v_m1_vector * t
    t_impact   =   np.linalg.norm(P_FAKE_TARGET   -   P_M1_START)   /
V_M1_SPEED
    bomb_params = []
    for i in range(3):
        angle, speed, t_drop, t_fuse = strats[i]
        无人机_start_pos = 无人机_INFO[i]["start_pos"]
        angle_rad = np.deg2rad(angle)
        dir_无人机 = np.array([np.cos(angle_rad), np.sin(angle_rad),
0])
        v_无人机_vector = dir_无人机 * speed
        def get_pos_drone_factory(start, vec): return lambda t: start
+ vec * t
        get_pos_drone = get_pos_drone_factory(无人机_start_pos, v_无
人机_vector)
        p_drop = get_pos_drone(t_drop)
        p_detonation  =  p_drop  +  v_无人机_vector  *  t_fuse  -
np.array([0, 0, 0.5 * G * (t_fuse**2)])
        t_det_start = t_drop + t_fuse
        t_smoke_end = t_det_start + T_SMOKE_DURATION
        if t_det_start < t_impact:
            bomb_params.append((p_detonation,           t_det_start,
min(t_smoke_end, t_impact)))
    if not bomb_params: return 0.0, 0.0
    all_start_times = [p[1] for p in bomb_params]
    all_end_times = [p[2] for p in bomb_params]
    t_sim_start,       t_sim_end       =       min(all_start_times),
max(all_end_times)
    dt = 0.01
    total_partial_time, total_complete_time = 0.0, 0.0
    for t in np.arange(t_sim_start, t_sim_end, dt):
        p_missile_now = get_pos_missile(t)
        target_points_now                                          =
get_dynamic_silhouette_points(p_missile_now)
        num_total_points = len(target_points_now)
        blocked_points_count = 0
```

```
        for target_point in target_points_now:
            is_point_blocked = False
            for p_det, t_det_start, t_smoke_e in bomb_params:
                if t >= t_det_start and t < t_smoke_e:
                    p_smoke_center_now = p_det - np.array([0, 0,
V_SINK_SPEED * (t - t_det_start)])
                    A, B, P = p_missile_now, target_point,
p_smoke_center_now
                    AB, AP = B - A, P - A
                    len_sq_AB = np.dot(AB, AB)
                    if len_sq_AB < 1e-9: continue
                    t_proj = np.dot(AP, AB) / len_sq_AB
                    if t_proj < 0: dist_sq = np.dot(AP, AP)
                    elif t_proj > 1: dist_sq = np.dot(P - B, P - B)
                    else: dist_sq = np.dot(AP, AP) - (t_proj**2) *
len_sq_AB
                    if dist_sq <= R_SMOKE**2:
                        is_point_blocked = True
                        break
            if is_point_blocked: blocked_points_count += 1
        if blocked_points_count > 0: total_partial_time += dt
        if blocked_points_count == num_total_points:
total_complete_time += dt
    return total_partial_time, total_complete_time
# --- 3. 主程序：生成最终报告 ---
if __name__ == "__main__":

    q4_champion_strategy = [
        5.55, 138.13, 0.52, 0.34,    # FY1 策略
        285.75, 115.64, 6.64, 5.60,  # FY2 策略
        110.52, 112.05, 23.10, 6.50  # FY3 策略
    ]
    # ------------------------------------------------------------
    print("--- 正在为问题四最优策略生成高精度报告 ---")
    start_time = time.time()

    table_data = []
    strats = [q4_champion_strategy[0:4], q4_champion_strategy[4:8],
q4_champion_strategy[8:12]]
    print("正在分别计算每枚烟幕弹的独立贡献...")
    for i in range(3):
        individual_partial,                    _                    =
calculate_single_bomb_time_cylinder(strats[i], i)

        angle, speed, t_drop, t_fuse = strats[i]
        无人机_start_pos = 无人机_INFO[i]["start_pos"]
        angle_rad = np.deg2rad(angle)
        dir_无人机 = np.array([np.cos(angle_rad), np.sin(angle_rad),
```

```
0])
        v_无人机_vector = dir_无人机 * speed
        p_drop = 无人机_start_pos + v_无人机_vector * t_drop
        p_detonation = p_drop + v_无人机_vector * t_fuse - np.array([0, 0, 0.5 * G * (t_fuse**2)])

        table_data.append({
            "无人机_name": 无人机_INFO[i]["name"],
            "angle": angle, "speed": speed,
            "p_drop": p_drop, "p_det": p_detonation,
            "duration": individual_partial
        })
    print("正在计算协同作用下的总遮蔽时间...")
    total_partial, total_complete = calculate_total_time_cylinder_dual_q4(q4_champion_strategy)

    run_time = time.time() - start_time
    print(f"报告生成完毕，耗时: {run_time:.2f}s")

    # --- 打印最终报告 ---
    print("\n\n" + "="*165)
    print("问题四：最优策略结果 (高精度动态轮廓模型)")
    print("="*165)
    headers = ["无人机编号", "无人机运动方向(度)", "无人机运动速度(m/s)",
               "投放点 x 坐标(m)", "投放点 y 坐标(m)", "投放点 z 坐标(m)",
               "起爆点 x 坐标(m)", "起爆点 y 坐标(m)", "起爆点 z 坐标(m)",
               "该弹独立有效时长(s)"]

    print("{:<10} {:<20} {:<22} {:<22} {:<22} {:<22} {:<22} {:<22} {:<22} {:<20}".format(*headers))
    print("-" * 165)
    for i, data in enumerate(table_data):
        print("{:<10}  {:<20.2f}  {:<22.2f}  {:<22.2f}  {:<22.2f}  {:<22.2f} {:<22.2f} {:<22.2f} {:<22.2f} {:<20.4f}".format(
            data["无人机_name"], data["angle"], data["speed"],
            data["p_drop"][0], data["p_drop"][1], data["p_drop"][2],
            data["p_det"][0], data["p_det"][1], data["p_det"][2],
            data["duration"]
        ))
    print("-" * 165)
    print(f"注：独立有效时长为单枚弹药在动态轮廓模型下独立产生的【部分遮蔽】时长。")
    print(f"\n协同作用下总时长: \n  - 总部分遮蔽: {total_partial:.4f} s \n  - 总完全遮蔽: {total_complete:.4f} s")
    print("="*165)
```