基于粒子群与黏菌混合算法的烟幕遮蔽模型

摘要

无人机投放烟雾干扰弹是一种通过对目标空域形成遮蔽，进而干扰敌方导弹的低成本、高效费比策略。

为帮助无人机选择特定飞行方向及速度，确定烟雾干扰弹投放时间以及时间引信控制起爆时间，实现在来袭导弹和保护目标之间形成较长时间的烟幕遮蔽，具有重大意义。本文借助粒子群算法和黏菌算法及其衍生混合算法，通过优化投掷策略以增加烟雾遮蔽目标时长，以达到保护目标并提高烟雾干扰弹的使用效率。

对于问题一, 首先建立三维坐标系，对圆柱体进行三维离散化采样，并表达出导弹-真实目标的光视线。接着构建导弹、无人机和烟雾弹的运动情况，进而通过几何关系设计核心的遮盖逻辑。最后结合烟幕球体球心和导弹对圆柱体固体扫描角的距离判断是否产生遮蔽与实际，得出满足 100% 遮蔽的情况下，有效遮蔽时长为 1.40s；满足 85% 以上的遮蔽的情况下，有效遮蔽时长为 1.41s。

对于问题二, 基于问题一的采样方法以及遮蔽逻辑判据法（导弹-目标的光视线线段是否与球体相交），将无人机的航向角、速度、投放延迟与引信延迟作为决策变量，在约束下求解最大化遮蔽时长。通过采用粒子群优化（PSO）搜索近似最优，并在关键时段采用自适应细化时间步长以提高精度和效率的方法，求解最优遮蔽时长为 4.70s。

对于问题三, 基于问题二的模式，在粒子群粗搜索框架的基础上，创新性地引入黏菌算法（SMA）对粒子群返回的最优邻域进行精密筛选与局部精化；同时使用自适应数值计算，以提高遮蔽时长并获得更稳定的投放策略, 其最佳遮蔽时长约为 6.37s。

对于问题四, 基于问题二的模式，通过数学方法近似将单个无人机对象转换为三个独立并列的无人机对象-导弹优化对象，分而治之并求和以取得全局最优遮蔽时长，同时取消问题二中 PSO 的早停机制并且修改 PSO 超参数，减小落入局部最优解的可能，使模型更加健壮鲁棒，其最佳遮蔽时长约为 11.6s。

对于问题五, 在多维维度灾难下先预处理分配，再使用 PSO 与结合 Halton 序列生成器，差分变异与精英个体池的增强黏菌算法（MISMA）深度融合，处理多自由度的问题，经试验，其最优解约为 17.3s, 但其全局最优解在 [17,20.3] 之间有一定的置信度。

关键字：动态几何关系　多目标优化　粒子群算法　黏菌算法

一、 问题重述

1.1 问题背景

烟幕干扰弹作为电子对抗领域关键装备，通过化学燃烧或爆炸形成烟幕或者气溶胶云团遮蔽目标、干扰敌方导弹，兼具成本低、效费比高的优势，且已突破“粗放式抛撒”瓶颈，形成“定点精确抛撒”能力，可预先计算弹道参数控制弹体抵达预定空域，结合时间引信时序控制起爆时间；但传统投放载体存在地理环境限制、续航短、飞行风险高等短板，难以满足现代来袭武器对防御体系“响应速度快、空域覆盖灵活”的动态防护需求，而长续航无人机因可在特定空域巡飞待命、响应迅速且无人员安全风险，为烟幕干扰弹投放提供了新解决方案，不过其投放策略尚未形成成熟体系，未能同时控制多台无人机形成最优化投弹策略来形成最佳半连续性遮蔽时长，亟需专项研究以填补技术空白、提升动态防御能力。

1.2 问题提出（问题重述）

在笛卡尔三维立体坐标系当中，有 5 台无人机 FY1(17800,0,1800)、FY2(12000,1400,1400)、FY3(6000,−3000,700)、FY4(11000,2000,1800)、FY5(13000,−2000,1300)；3 枚导弹 M1、M2、M3 分别位于 (20000,0,2000)、(19000,600,2100)、(18000,−600,1900)。

导弹为空地导弹，其飞行速度为 300m/s, 直指假目标（0，0，0），而真目标是一个半径 7m、高 10m 的圆柱体，其底面圆心坐标为（0，200，0）。

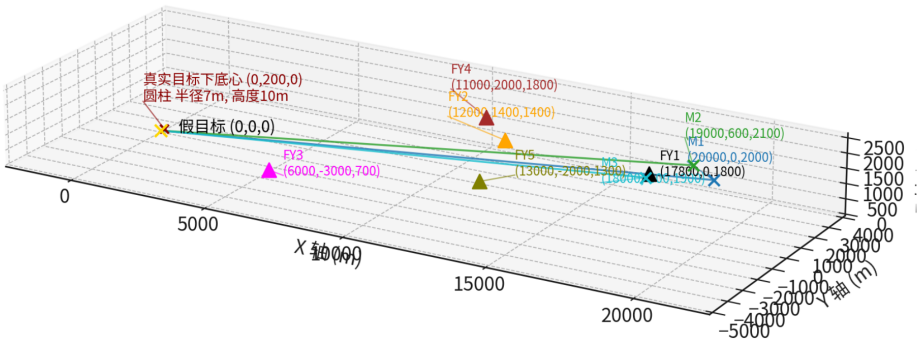

图 1　导弹，无人机，目标的位置示意图

如图 1 所示，其中三枚导弹 M1,M2,M3 以 300m/s 均直指假目标飞去, 呈现匀速直线运动。在导弹飞行途中，会不断对真目标进行扫描，形成一定的固体扫描角，无人机将通过投放烟幕干扰弹尽量避免来袭导弹发现真目标。

当干扰弹随着无人机行进被投掷，将情景理想化视为平抛运动，重力加速度为 g=9.8m/s^2;

干扰弹被投掷后，在特定的时间引信时序控制下起爆，起爆后的烟幕云团呈现球形，以 3m/s 的速度匀速下沉，云团中心 10m 范围内的烟幕浓度在起爆 20s 内可为目标提供有效遮蔽。

无人机以 70—140m/s 的速度匀速等高飞行，每架无人机的航向和速度可以不同, 但一旦确定就无法更改。每架无人机投放两枚干扰弹时间至少间隔 1s，且各个烟幕弹之间的遮蔽可不连续。

问题 1 无人机 FY1 将会投放 1 枚烟幕弹对 M1 进行干扰, FY1 的速度固定为 120m/s 朝向假目标，即原点的 1800m 上空，从受领任务开始 1.5s 后投放 1 枚烟幕弹，间隔 3.6s 起爆，建立数学模型计算有效遮蔽时长。

问题 2 无人机 FY1 将会投放 1 枚烟幕弹对 M1 进行干扰，改变无人机 FY1 的飞行方向、飞行速度、烟幕干扰弹投放的时间引信时序、烟幕干扰弹的投放点进而改变烟幕干扰弹的起爆点使遮蔽时间尽量长。

问题 3 无人机 FY1 将会投放 3 枚烟幕弹对 M1 进行干扰，改变无人机 FY1 的飞行方向、飞行速度、烟幕干扰弹投放的时间引信时序、烟幕干扰弹的投放点进而改变烟幕干扰弹的起爆点使遮蔽时间尽量长，结果将保存在文件 result1.xlsx 中。

问题 4 无人机 FY1、FY2、FY3 这 3 架无人机将会各自投放 1 枚烟幕弹对 M1 进行干扰，改变无人机的飞行方向、飞行速度、烟幕干扰弹投放的时间引信时序、烟幕干扰弹的投放点进而改变烟幕干扰弹的起爆点使遮蔽时间尽量长，结果将保存在文件 result2.xlsx 中。

问题 5 共计 5 架无人机将会各自至多投放 3 枚烟幕弹对导弹 M1、M2、M3 进行干扰, 改变无人机的飞行方向、飞行速度、烟幕干扰弹投放的时间引信时序、烟幕干扰弹的投放点进而改变烟幕干扰弹的起爆点使遮蔽时间尽量长，结果将保存在文件 result3.xlsx 中。

二、 模型假设

为简化问题，本文做出以下假设:

- 假设 1：烟幕弹在飞行时不受阻力作用;
- 假设 2：烟幕云球不会被风吹散或被外界环境影响形状。

三、 符号说明

符号	说明	单位
u	投影比例	
g	重力加速度	m/s^2
Δt	遮蔽总时长	s
θ	无人机航向角度	$degree$
M_0	导弹初始位置	
v_M	导弹速度	m/s
P_drop	无人机投放烟幕弹位置	
T_{smoke}	烟雾有效时间	s
v_{sink}	烟雾下降固定速度	m/s
$M(t)$	导弹随时间 t 运动函数	m
$S(t)$	烟雾弹随时间 t 运动函数	m
$P(t)$	导弹与目标的向量函数	

四、 问题一的模型的建立和求解

4.1 问题一分析

问题一需要结合直线运动规律，分析无人机运行到的烟幕弹投放点，进而分析烟幕弹平抛运动情况，引爆位置，表示出烟幕云团随时间（t）下降的方程，并且将烟幕存在时间限制在 20s 内。

关于导弹对于真目标的扫描，首先对真目标 (即圆柱体) 进行采样取点，并将点位与导弹相关联产生固体扫描角范围。接着将导弹与云团中心，导弹与真目标的扫描角进行投影关联，寻找相关取值范围。判断其是否能够满足有效遮蔽。

4.2 模型建立

需注意，该数学建模的基础性任务是：

1. 建立目标（圆柱体）取样本点的逻辑
2. 建立判断烟雾弹是否遮蔽目标的逻辑
3. 导弹和烟雾弹随时间 t 的运动函数 $M(t)$ 和 $S(t)$

首先建立取目标物圆柱体样本点的逻辑，综合到本问的自由度只有时间 t，各物体的方向都已确定，可选择样本点较多的取点方式。

具体取点方式如下：

1. **侧面采样**：在圆柱体的侧面（半径固定为 r 的柱面），沿高度方向分层，每层沿圆周均匀取点。

   具体步骤：

   - 角度划分：用 np.linspace 生成 n_theta 个均匀分布的角度（0 到 2π，不含终点），确保圆周方向均匀采样。
   - 高度划分：用 np.linspace 生成 n_z 个高度偏移（从圆柱体中心下方 $h/2$ 到上方 $h/2$），叠加中心 z 坐标后得到 zs（侧面各层的 z 坐标）。
   - 点坐标计算：对每个高度 z，通过极坐标公式计算 x 和 y（$x = x_c + r \cdot cos(\theta), y = y_c + r \cdot sin(\theta)$），z 固定为当前高度，形成侧面的环形点集。

2. **底面采样**：在圆柱体底面，从圆心向外沿不同半径采样，半径越大，圆周方向的点越密集。

   具体步骤：

   - 半径划分：生成从 0 到 r 的径向半径 radii
   - 角度点数：对每个半径 radius，计算圆周方向的采样点数 $theta\_count$：
     * 半径为 0 时（圆心）：仅 1 个点；
     * 半径 > 0 时：点数与半径成正比（$n\_theta \cdot (radius/r)$），且至少 4 个点（确保形状完整）。
   - 点坐标计算：对每个半径和对应角度，计算 x、y（极坐标转换），z 固定为底面高度，覆盖底面从中心到边缘的区域。

3. **顶面采样**：为确保圆柱体上下两端的完整性，与底面形成对称的点云分布。我们让它与底面完全对称，仅 z 坐标改为中心 $z_c = z + h/2$（圆柱体顶部），其余径向、周向采样规则与底面一致。

4. **中轴线采样**：在圆柱体的中心轴（$x = x_c, y = y_c$）上，沿高度方向均匀取点。

   具体步骤：

   - 高度划分：从底面高度到顶面高度，生成至少 5 个点（或 $n_z//5$ 个，取较大值），确保轴线方向有足够采样。
   - 点坐标：直接取 (中心 x, 中心 y, z)，覆盖整个圆柱的中轴线。

   最终处理:

用 np.unique 去除可能的重复点（例如中轴线点与底面 / 顶面中心的重合点），返回形状为 (N, 3) 的点云数组。

可以由下述伪代码实现，其中返回值为包含所有采样点的 numpy 数组，形状为 (N, 3)。（具体实现代码见附录 B 的 q1.py）

Algorithm 1 圆柱体采样： `cylinder_points`(n_θ, n_z, r, h, C)

Input: 角向分辨率 n_θ（如 360）、竖向分辨率 n_z（如 61）、半径 r、高度 h、圆柱中心 $C = (C_x, C_y, C_z)$.

Output: 采样点集合 $\mathcal{P} = \{(x, y, z)\}$.

// （1）准备角向与竖向网格

thetas ← linspace$(0, 2\pi, n_\theta,$ endpoint $=$ False) z_offsets ← linspace$(-h/2,\ h/2,\ n_z)$ zs ← C_z + z_offsets pts ← 空列表

// （2）侧面采样（按竖向层）

foreach z in zs **do**

  $x\_vals \leftarrow C_x + r\cos$(thetas) $y\_vals \leftarrow C_y + r\sin$(thetas) 将每个 $(x\_vals[i],\ y\_vals[i],\ z)$ 加入 pts $(i = 0, \ldots, n_\theta - 1)$

end

// （3）底面与顶面：径向分层采样

$z_{\text{base}} \leftarrow C_z - h/2$, $z_{\text{top}} \leftarrow C_z + h/2$ $N_r \leftarrow \max(2,\ \lfloor\sqrt{n_\theta}\rfloor)$ // 径向层数（经验取法）

radii ← linspace$(0, r, N_r)$

foreach radius in radii **do**

  if radius = 0 **then**

    theta_count ← 1

  else

    theta_count ← max $\left(4,\ \lfloor n_\theta \cdot (radius/r)\rfloor\right)$

  end

  base_thetas ← linspace$(0, 2\pi,$ theta_count, endpoint $=$ False) $x \leftarrow C_x + radius\cos$(base_thetas) $y \leftarrow C_y + radius\sin$(base_thetas) // 把底面与顶面对应点加入集合

  将每个 $(x[j], y[j], z_{\text{base}})$ 加入 pts（$j = 0, \ldots,$ theta_count $- 1$） 将每个 $(x[j], y[j], z_{\text{top}})$ 加入 pts

end

// （4）中轴线采样（补充中心点）

$N_{\text{axis}} \leftarrow \max(5,\ \lfloor n_z/5\rfloor)$ axis_zs ← linspace$(z_{\text{base}}, z_{\text{top}}, N_{\text{axis}})$ **foreach** z0 in axis_zs **do**

  将 $(C_x, C_y, z0)$ 加入 pts

end

// （5）去重并返回

将 pts 转为数值数组并按行去重（例如调用 `unique(pts,axis=0)`），得到数组 $\mathcal{P}$ **return** $\mathcal{P}$

接下来，将建立**“遮蔽判断”的程式与 Python 函数，在代码中实现判断是否遮蔽的逻辑**。

根据题意，爆炸后云团中心以 3m/s 的速度匀速下沉且 10m 范围内的烟幕浓度在起爆 20s 内可为目标提供有效遮蔽，我们需要做的，就是在导弹飞行过程中（烟雾弹爆炸生成烟雾到导弹扎到假目标点或烟雾弹落地的时间段），持续遮挡导弹与真目标之间的视线。

首先需找到烟幕弹开始爆炸的时刻，由题目接受任务和延迟爆炸的时间易得，

$$t_0 = 1.5s + 3.6s = 5.1s$$

明确从 t_0 到烟雾失效的时刻为 $t_0 + 20s$。

因此，在 $5.1s - 25.1s$ 中，需抽象出“遮盖判断”模型——仅导弹与目标点的连线（导弹的光视线）与烟雾弹形成的烟雾球有交集（即被烟雾遮蔽）。

我们将通过 3-D 视图对该问题进行一个可视化处理：本处正视图由豆包 *AI* 生成

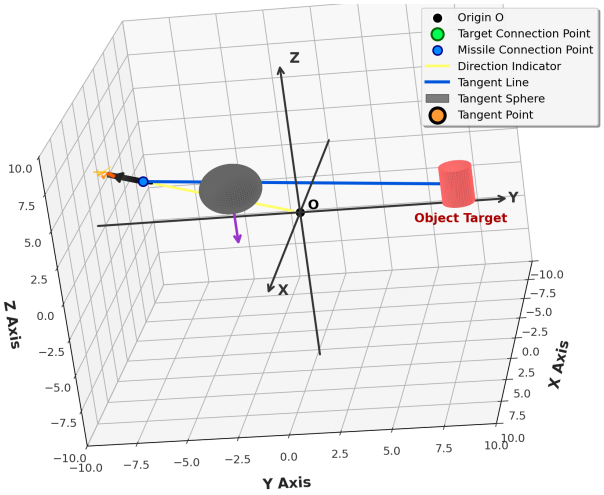

图 2　导弹-烟雾问题中遮蔽真目标原理的正视图 (AI)

为对必要原理的抽象与提取，我们在 Python 绘图的方面牺牲了位置关系与比例大小。但仍可以看到在图 1 中：导弹位于最右边的空域，蓝点 Missile connection point 是对导弹弹头的抽象；黄色细线段是导弹指向原点的运行方向；中间的灰球是对烟雾弹的抽象；最右端的圆柱体是对目标的抽象（做放大处理）

如烟雾要想对导弹有有效的遮盖，那么导弹（蓝点）与目标上任意一点的连线（导弹的光视线）与烟雾弹（灰球）形成的烟雾球有交集。

在这张正视图中，提取了其临界情况：**相切**，其中烟雾球与导弹的切点在图后。

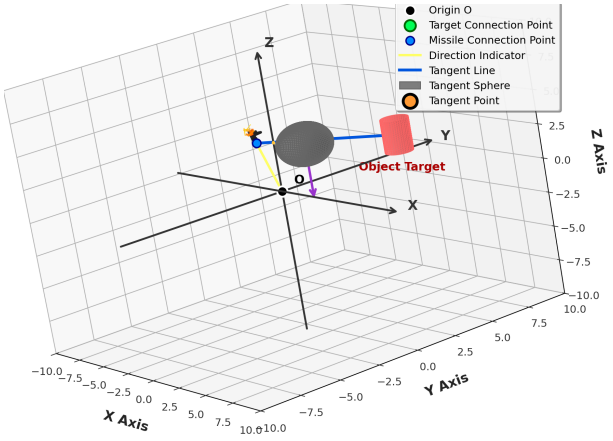

图 3　导弹-烟雾问题中遮蔽真目标原理的左视图 (AI)

本处左视图由豆包 *AI* 生成

在这个视图中，可以更加明显的看到相切的情境: 导弹的光视线与烟雾弹（灰球）相切。

因此可以进一步抽象其数学本质是：判断线段 AB，即导弹与圆柱体上的样本点的连线是否与球体（球心为 C，半径为 R）相交（切），本质是判断：**线段上是否存在一点 P，使得 P 到球心 C 的距离** $\|P-C\| \le R$。

1. 线段的参数化: 线段 AB 可以用参数方程表示为：$P(t) = A + t \cdot \overrightarrow{AB} \quad (t \in [0,1])$ 其中：$\overrightarrow{AB} = B - A$（线段的方向向量）；参数 t 的取值范围 $[0,1]$ 保证 $P(t)$ 在线段上（$t=0$ 时为 A，$t=1$ 时为 B）。
2. 接着寻找线段上与球心 C 距离最近的点 P_{closest}，再判断该点到 C 的距离是否 $\le R$。定义向量 $\overrightarrow{AC} = C - A$（从 A 到 C 的向量）；计算 $\overrightarrow{AC}$ 在 $\overrightarrow{AB}$ 上的投影比例 u:

$$u = \frac{\overrightarrow{AC} \cdot \overrightarrow{AB}}{\overrightarrow{AB} \cdot \overrightarrow{AB}}$$

   其中分母即 $\overrightarrow{AB} \cdot \overrightarrow{AB} = \|\overrightarrow{AB}\|^2$，用于归一化比例 u。
3. 限制参数在有效范围（线段内）由于我们只关注线段 AB（而非无限延长的直线），需将 u 限制在 $[0,1]$ 内，得到 $u_{\text{clamped}} = \max(0, \min(1, u))$
   * 若 $u < 0$：最近点为线段起点 A（$t=0$）；
   * 若 $u > 1$：最近点为线段终点 B（$t=1$）；
   * 若 $0 \le u \le 1$：最近点为线段上的投影点（$t=u$）。
4. 计算最近点坐标线段上离 C 最近的点 P_{closest} 为：$P_{\text{closest}} = A + u_{\text{clamped}} \cdot \overrightarrow{AB}$ 以进行距离判定：为提高效率，我们利用相交条件计算 P_{closest} 到球心 C 的距离平方：$\text{dist}_{\text{sq}} = \|P_{\text{closest}} - C\|^2$ 若 $\text{dist}_{\text{sq}} \le R^2$，则线段 AB 与球体相交。

特殊情况处理：退化线段当线段 AB 的两个端点几乎重合（$\|\overrightarrow{AB}\| \approx 0$）时，线段退化为一个点（可视为 A 或 B）。此时直接判断该点到 C 的距离是否 $\le R$，即：$\|\overrightarrow{AC}\| \le R$

整个逻辑过程可以由下代码表示：

我们令 A,B 为线段端点，C 为球心，R 为球半径；返回值为是否相交的布尔值：

```python
def seg_hits_sphere(A, B, C, R):
    AB = B - A
    AC = C - A
    den = AB @ AB
    # 处理线段退化（A和B重合）的情况
    if den < 1e-12:
        return np.linalg.norm(AC) <= R + 1e-12
    # 计算投影参数
    u = (AC @ AB) / den
    u_clamped = max(0.0, min(1.0, u))
    closest = A + u_clamped * AB
    # 计算最近点到球心的距离（带容差处理）
```

```
    dist_sq = np.sum((closest - C)** 2)
    return dist_sq <= (R + 1e-12) **2  # 比较平方值提高效率
```

最后刻画各个物理客体的运动方程：

导弹运动

令导弹指向假目标 O, 即（0，0，0），M_0 为导弹初始位置（20000，0，2000），其单位方向向量

$$\mathbf{d}_M = \frac{O - M_0}{\|O - M_0\|}.$$

导弹在时刻 t 的位置为

$$\mathbf{M}(t) = M_0 + v_M t \mathbf{d}_M,$$

导弹到达假目标的时间

$$t_{\text{arr}} = \frac{\|M_0 - O\|}{v_M}.$$

无人机运动与投放点无人机设定航向 θ，水平单位向量

$$\mathbf{u} = (\cos\theta,\ \sin\theta,\ 0).$$

无人机以恒速 v 匀速直线飞行，P_{vo} 为无人机 YF1 初始位置，则在投放时刻 t_1 的投放点为

$$P_{\text{drop}} = P_{U0} + v\, t_1\, \mathbf{u}.$$

烟弹的平抛与起爆点投放后烟弹水平初速度等于无人机速度 v，竖直分量为 0（假设），在引信延时 t_2 后起爆，起爆时刻

$$t_{\text{det}} = t_1 + t_2.$$

起爆点坐标

$$\begin{aligned} x_{\text{det}} &= P_{\text{drop},x} + v\, t_2 \cos\theta, \\ y_{\text{det}} &= P_{\text{drop},y} + v\, t_2 \sin\theta, \\ z_{\text{det}} &= P_{\text{drop},z} - \tfrac{1}{2} g t_2^2, \end{aligned}$$

其中 $g = 9.80665\ \text{m/s}^2$。

烟云随时间的位置起爆后相对起爆时刻 $s \in [0, T_{\text{smoke}}]$，烟云球心位置

$$\mathbf{S}(t_{\text{det}} + s) = (x_{\text{det}}, y_{\text{det}}, z_{\text{det}} - v_{\text{sink}} s).$$

在有效期内，烟云被视为半径 R 的球。

4.3 模型求解

在解决了建立目标（圆柱体）取样本点的逻辑，建立判断烟雾弹是否遮蔽目标的逻辑，刻画出导弹与烟雾弹随时间运动的方程后，我们可以寻找一个关键的时间端。

为了节省算力与使用更精确的步长（如 0，005s）迭代，需确定置信度最高的发生遮挡的关键时间段（即 $7.0s$ $10.5s$）

- $t = 7.0s$ 时，导弹 x≈$20000 - 298.5$×7≈$17910.5 > 17188$，处于烟幕后方，视线可能被遮蔽；
- $t = 10.5s$ 时，导弹 x≈$20000 - 298.5$×10.5≈$16865.8 < 17188$，处于烟幕前方，遮蔽基本结束。

因此，$7.0s$ $10.5s$ 是导弹处于烟幕后方、且烟幕有效的时段，不仅覆盖了可能发生遮蔽的完整区间，而且减少了取点次数，提高了运行效率。

通过参照附录 B 中的完整代码，经运行可得到如下的结果：

完全遮蔽区间 = (8.049999999999978,9.444999999999947)

完全遮蔽总时长 Δt = 1.400 s

4.4 阈值讨论

第一问中，若采用 0.005s 步长和圆柱体高密度采样后，我们将得到其覆盖率与时间的图像，并且设计两个阈值，一个是百分之百，另一个是百分之八十五，以更加贴切工业实际——我们无需目标中的所有点均为检测到被遮盖。

对于 85% 的阈值，如样本点被检测到，记录为 1, 其总数记为 n_1；若没被检测到; 记录为 0，其总数记为 n_2，最后需计算：

$$n_1/n_1 + n_2$$

若大于阈值，则认为是有效遮挡

其结果如下：

完全遮蔽区间 =(8.049999999999978,9.444999999999947)

完全遮蔽总时长 Δt = 1.400 s

满足 ≥85% 的区间：(8.029999999999957,9.439999999999927)

遮蔽总时长（≥85%）= 1.410 s

- 在 100% 的遮盖率下，有效遮盖时间约为：1.400s
- 在 85% 的遮盖率下，有效遮盖世界约为：1.410s

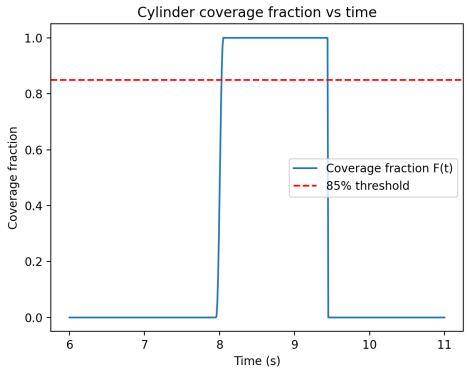

图 4　问题 1 的阈值结果示例

五、 问题二的模型的建立和求解

5.1 问题二分析

在问题一的笛卡尔三维坐标系以及建模基础上，导弹的飞行方向，速度以及扫描情况仍固定，无人机的飞行方向，飞行速度，时间引信时序，投放时间却变得更加灵活。将无人机的航向角、速度、投放延迟与引信延迟作为决策变量，在约束下求解最大化遮蔽时长的问题。

采用粒子群优化（PSO）搜索空间大部范围内近似最优，并在关键时段采用自适应细化时间步长以提高精度，从而满足遮蔽时长的提高。

5.2 模型建立

该问题可以抽象成：以 $x = (\theta, v, t_1, t_2)$ 为决策变量，数学优化问题为

$$\begin{aligned} \max_{x} \quad & F(x) \equiv T_{\text{occ}}(x) + B_{\text{bonus}}(x), \\ \text{s.t.} \quad & 70 \le v \le 140, \\ & t_1 \ge 0, \quad t_2 \ge 0, \\ & z_{\text{det}}(x) \ge z_{\min}, \\ & t_{\text{det}}(x) = t_1 + t_2 \le t_{\text{arr}}. \end{aligned}$$

其中 $T_{\text{occ}}(x)$ 代表遮蔽时间。$B_{\text{bonus}}(x)$ 为工程上加入的边界奖励项（鼓励边界搜索，可设为小常数或函数），并可通过惩罚法处理不可行解（例如当 $z_{\text{det}} < z_{\min}$ 或 $t_{\text{det}} > t_{\text{arr}}$ 时将适应度置为 0）。

为了解决这个目标函数为非光滑、可能多极值的问题，采用粒子群优化（PSO）进行全局搜索，随后对最优候选可用局部优化精化。

其模型代码的运行逻辑可由下图所示：

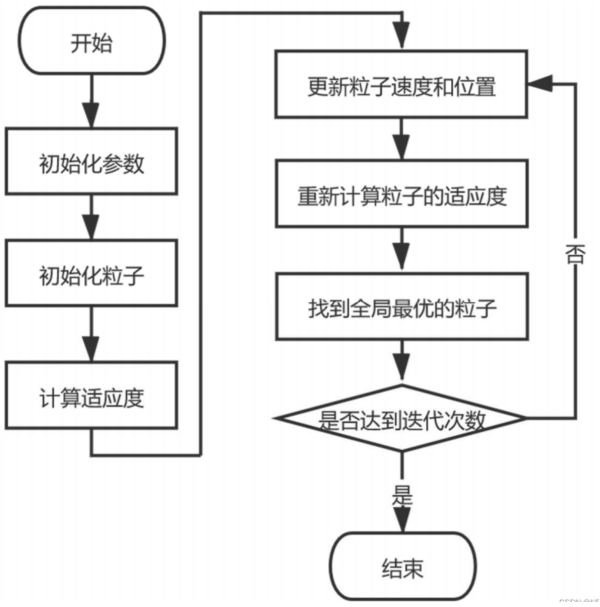

图 5　粒子群算法代码模型逻辑解释

代码中，首先采用 (`joblib.Parallel`) 加速适应度评估, 利用多 CPU 核心同时计算多个粒子的适应度, 提升算法效率。

每代迭代中，粒子通过学习" 个体经验" 和" 群体经验" 调整速度与位置，公式如下:

- 速度更新:

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

  其中代码（见附录 B）的参数介绍:

  – w 为惯性权重 (线性递减, 从 0.9 降至 0.4), 平衡全局探索与局部开发;
  – 代码中设计超参数 $c_1 = 1.5$ (认知系数)、$c_2 = 1.5$ (社会系数), 分别控制粒子对个体最优和全局最优的学习权重;
  – $r_1, r_2 \sim \mathcal{U}(0,1)$ 为随机因子, 增加搜索随机性。

- 位置更新:

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

  同时通过边界约束处理 (如速度限制为参数范围的 20%, 位置截断至可行域内) 确保解的有效性。

接着, 我们通过"适应度评估"对每个粒子的位置 (候选策略), 通过 `fitness_function` 计算适应度:

1. 计算投放点与起爆点坐标 (基于运动学公式);
2. 生成自适应时间步长序列 (关键时段用精细步长, 非关键时段用粗步长);
3. 逐时刻判断目标是否被烟幕完全遮蔽 (通过线段-球体相交检测);

4. 累加有效遮蔽时长作为适应度值。

同时为了加速收敛和节约算力，我们引入：**收敛跟踪与早停机制**的代码块：

- **收敛跟踪**: 记录每代全局最优适应度 (`gbest_history`), 通过收敛曲线可视化算法进展。
- **早停策略**: 当连续 15 代 (`patience=15`) 全局最优适应度无显著改进时, 提前终止迭代, 避免无效计算, 平衡优化精度与效率。

故该 PSO 实现针对烟幕遮蔽问题的特性进行了针对性优化：

1. 解空间约束处理：通过参数边界与惩罚机制，确保搜索聚焦于物理可行的策略（如起爆点高度不低于 5m）；

2. 计算效率优化：结合自适应时间步长（减少非关键时段的计算量）与并行评估，降低高维度、高复杂度适应度函数的计算成本；

如表所示，粒子群优化后结果呈现多组情况，取其中最有代表性的进行结果的讨论：组 4，5 均为代码运行中常见的局部最优解（0，20），需要排除；而剩下的三组均能达到 4.70s 左右的有效遮蔽，其中因个别组别精确性差异，故需重新调节迭代次数，以求更加稳定的收敛。

组	方向角 (°)	速度 (m/s)	投放延迟 (s)	起爆延迟 (s)	有效遮蔽 (s)
1	14.09	82.03	0.0005	0.4397	4.60
2	6.89	92.76	0.0079	0.8067	4.70
3	3.73	91.10	1.3368	0.0049	4.70
4	95.56	139.48	38.30	0.5821	0.20
5	270.40	70.21	44.20	0.8506	0.20

表 1　问题一的优化结果 (1)

组	有效遮蔽 (s)	投放点 X	投放点 Y	投放点 Z	起爆点 X	起爆点 Y	起爆点 Z
1	4.60	17800	0.009	1800	17800	8.79	1800
2	4.70	17800	0.0882	1800	17900	9.07	1800
3	4.70	17900	7.92	1800	17900	7.95	1800
4	0.20	17282.0	5317.6	1800.0	17274.2	5398.4	1798.3
5	0.20	17821.7	-3103.1	1800.0	17822.1	-3162.9	1798.3

表 2　问题一的优化结果 (2)

重新进行实验，这一次采用 220 次的迭代次数；同时使用 150 个粒子以更加持久，广泛地搜索，最终趋于 4.70s。

其日志利用可视化工具绘制：

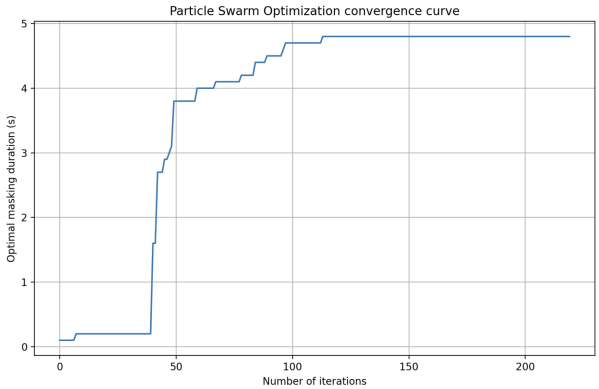

图 6　问题一的加长迭代结果

我们有理由确信，4.60-4.70s 内具有较高的置信度，而 4.70s 可以作为我们的最优解。

因此具体考察得到其参数；

1. 无人机飞行方向角: 7.37°
2. 无人机飞行速度:95.67m/s
3. 投放延迟时间: 0.0418 s
4. 起爆延迟时间: 0.3498 s

投放点坐标: [17819 2.5483 18000]

起爆点坐标: [17868 8.8334 17994]

六、 问题三的模型的建立和求解

6.1 模型建立

在该问中，我们构建了“物理模型 - 约束条件 - 双阶段优化 - 结果验证”的四层逻辑体系，该模型在**双阶段粒子群优化（PSO）全局粗筛-黏菌算法（SMA）局部精筛的分层优化框架和自适应数值计算方法**两点做出创新。

回顾该问的数学表达与物理约束：

以 3 枚烟幕弹的投放参数向量 $\boldsymbol{X} = [\theta_1, v_1, t_{11}, t_{21}, \theta_2, v_2, t_{12}, t_{22}, \theta_3, v_3, t_{13}, t_{23}]$ 为优化变量，最大化导弹飞行时段 $[0, T_M]$ 内真目标被烟幕完全遮蔽的总时长 $J(\boldsymbol{X})$，即：

$$\max_{\boldsymbol{X}\in\Omega} J(\boldsymbol{X}) = \int_0^{T_M} \mathbb{I}\left(\bigcap_{P\in\mathbb{S}}\left(\bigcup_{k=1}^{3}\mathcal{I}(M(t), P, S_k(t))\right)\right) dt$$

其中：

T_M 为导弹到达假目标的时间；$\mathbb{S}$ 为真目标（圆柱体）的采样点集合；$\mathcal{I}(A, B, S)$ 为线段-球体相交指示函数（相交时取值为 1，否则为 0）；$M(t)$ 为导弹在时刻 t 的位置函数；$S_k(t)$ 为第 k 发烟幕弹在时刻 t 形成的球状云团；Ω 为满足所有约束条件的参数空间。

需满足以下几个约束：

1. 参数范围约束：

$$\begin{cases} 70 \le v_i \le 140 \quad (\text{m/s}) \\ t_{1i} \ge 0,\ t_{2i} \ge 0 \quad (\text{s}) \qquad\qquad (i = 1, 2, 3) \\ 0 \le \theta_i < 2\pi \quad (\text{rad}) \end{cases}$$

2. 投放间隔约束：任意两枚烟幕弹的起爆时刻间隔不小于 1 秒

$$|(t_{1i} + t_{2i}) - (t_{1j} + t_{2j})| \ge 1 \quad (\text{s}), \quad i \ne j$$

3. 起爆高度约束：烟幕弹起爆点高度不低于 5 米

$$z_{\text{det},i} \ge 5 \quad (\text{m}), \quad (i = 1, 2, 3)$$

4. 烟幕有效期约束：烟幕云团起爆后 20 秒内有效，且以 3m/s 匀速下沉

$$S_k(t) = \begin{cases} \text{有效} & t \in [t_{\text{det},k},\ t_{\text{det},k} + 20] \\ \text{无效} & \text{其他} \end{cases}$$

其中，烟幕中心位置随时间变化为：

$$z_k(t) = z_{\text{det},k} - 3 \cdot (t - t_{\text{det},k}) \quad (\text{m})$$

（$z_{\text{det},k}$ 为第 k 发烟幕弹的起爆高度，$t_{\text{det},k} = t_{1k} + t_{2k}$ 为起爆时刻）

针对该物理情境抽象出的数学模型与约束，将采用 PSO-SMA 混合算法求解。

6.1.1 双阶段 PSO 全局粗筛-SMA 局部精筛

粒子群优化（PSO）粗筛:

PSO 基于“群体协作”与“个体经验”进行全局搜索，其数学模型如下:

设粒子群规模为 N，维度为 D（本系统 $D=12$），第 i 个粒子的位置为 $\boldsymbol{x}_i(t)$、速度为 $\boldsymbol{v}_i(t)$、个体最优位置为 $\boldsymbol{p}_i$、全局最优位置为 $\boldsymbol{g}$，则迭代公式为:

$$\boldsymbol{v}_i(t+1) = w(t)\cdot\boldsymbol{v}_i(t) + c_1 r_1(\boldsymbol{p}_i - \boldsymbol{x}_i(t)) + c_2 r_2(\boldsymbol{g} - \boldsymbol{x}_i(t))$$

$$\boldsymbol{x}_i(t+1) = \text{clip}(\boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1), \boldsymbol{x}_{\min}, \boldsymbol{x}_{\max})$$

其中:

- $w(t) = w_{\text{start}} - (w_{\text{start}} - w_{\text{end}})\cdot\frac{t}{t_{\max}}$ 为线性递减惯性权重（本代码中 $w_{\text{start}}=0.9$，$w_{\text{end}}=0.4$）;
- 本代码中 $c_1=1.5$、$c_2=1.5$ 为认知与社会系数;
- $r_1, r_2 \sim \text{Uniform}(0,1)$ 为随机因子;
- clip$(\cdot)$ 为边界约束函数，确保参数在 Ω 内。

本系统对 PSO 进行**智能初始化改进**：对速度参数 v_k 采用 $\mathcal{N}(105, 15^2)$ 正态分布初始化，对时间参数 t_{1k}, t_{2k} 采用指数分布 Exponential(3) 初始化，提升初始群体的多样性与可行性。

黏菌算法（SMA）精筛:

黏菌算法（SMA）是模拟自然界黏菌觅食行为的生物启发式群智能优化算法，通过“正负反馈机制”与“三阶段搜索逻辑”平衡全局勘探与局部开发。

即 SMA 模拟黏菌的“觅食 - 收缩 - 扩散”行为，在 PSO 最优解邻域内进行局部精细搜索，其核心迭代公式[1] 如下:

- **核心数学模型**：位置更新分三类场景，通过随机数与自适应参数调控搜索模式:

$$X(t+1) = \begin{cases} r\cdot(Ub - Lb) + Lb, & r < z \\ X_s(t) + v_s\left[W\cdot X_A(t) - X_B(t)\right], & r < p \\ v_c\cdot X(t), & r \geq p \end{cases}$$

其中，$r\in[0,1]$ 为随机数，$z=0.03$ 触发随机勘探，$X_s(t)$ 为当前最优个体位置，W（$[0,1]$）为自适应权重系数，v_s（线性递减）控制移动速度，$v_c\in[-a,a]$（a 从 1 减至 0）实现搜索范围收缩。

SMA 通过**自适应收缩系数** $a(t)$ 与**多模态位置更新规则**，在局部区域内高效探索最优解，弥补 PSO 局部收敛精度不足的缺陷。

代码中 `SlimeMouldAlgorithm` 类的设计遵循 SMA 生物学机制与数学模型，其核心要素与代码实现的映射关系如下:

- **种群初始化策略**
  - 代码实现模块：`_initialize_slimes` 方法
  - 映射关系：以全局优化阶段的次优解为初始种群“中心解”，非中心解通过“中心解 + 随机扰动”生成（扰动幅度为参数约束范围的 10%），遵循 SMA 初始种群需围绕优质可行域分布以提升寻优效率的设计原则，规避随机初始化导致的种群多样性不足与搜索冗余问题。
- **自适应参数调节（v_c、W）**
  - 代码实现模块：`_update_position` 方法中参数动态更新逻辑
  - 映射关系：1）控制参数 v_c 采用线性递减策略（从 1 降至 0），与 SMA 通过 v_c 调节黏菌振荡收缩幅度，迭代后期增强局部精细搜索的机制一致；
    2）适应度权重系数 W 根据当前种群适应度均值动态计算，模拟黏菌通过食物浓度反馈调节静脉网络粗细的正反馈逻辑，实现对优质解区域的定向开发。
- **勘探-开发切换机制**
  - 代码模块：`_update_position` 方法的分支判断逻辑
  - 映射关系：1）勘探阶段触发（随机数 $r < 0.03$）：对应 SMA 中“$r < z$ $(z = 0.03)$”的随机勘探策略，通过生成边界内随机参数值，模拟黏菌“分离有机物探索新区域”的行为，缓解局部最优风险；
    2）开发阶段触发（$0.03 \leq r < p$）：遵循“$X_s(t) + v_s[W \cdot X_A(t) - X_B(t)]$”的位置更新公式，以当前最优解 $X_s(t)$ 为引导，通过 W 与 v_s 调节向优质解的移动步长，实现黏菌“包围食物后的精细搜索”[1]；
    3）收缩阶段触发（$r \geq p$）：通过 $v_c \cdot X(t)$ 更新位置，对应黏菌在优质食物附近降低振荡频率，收缩搜索范围的开发机制，提升局部寻优精度。
- **适应度驱动更新**
  - 代码实现模块：`optimize` 方法中适应度评估与最优解更新逻辑
  - 映射关系：每次迭代计算所有“黏菌个体”（参数组合）的适应度值，若个体适应度优于当前全局最优，则更新最优位置与适应度，对应文献中“SMA 通过食物浓度（适应度）反馈调节种群移动方向”的核心机制 [1]。
- **边界约束机制**
  - 代码模块：`_update_position` 方法中的 `np.clip` 函数
  - 映射关系：对更新后的参数进行上下界裁剪（如速度、延迟等参数的物理约束），严格遵循文献中“SMA 需通过 Ub（上界）、Lb（下界）确保解的可行性”的约束逻辑，避免无效解参与迭代[1]。

6.1.2 自适应数值计算方法

同时为平衡遮蔽时长计算的**精度与效率**，设计基于“导弹-真目标接近事件”的自适应时间步长策略，数学表达如下：

设时间窗口为 $[t_{\text{start}}, t_{\text{end}}]$，定义“关键事件时间” t_{event}（导弹沿飞行方向投影至真目标中心的时间）：

$$t_{\text{event}} = \frac{(\boldsymbol{C} - \boldsymbol{M}_0) \cdot \boldsymbol{d}_M}{v_M}, \quad \boldsymbol{d}_M = \frac{\boldsymbol{O} - \boldsymbol{M}_0}{\|\boldsymbol{O} - \boldsymbol{M}_0\|}$$

则时间序列 $\mathbb{T}$ 由三部分构成：

1. 粗算段：$t \in [t_{\text{start}}, t_{\text{fine,start}})$，步长 $\Delta t_{\text{coarse}} = 0.1\ \text{s}$；
2. 精算段：$t \in [t_{\text{fine,start}}, t_{\text{fine,end}}]$，步长 $\Delta t_{\text{fine}} = 0.005\ \text{s}$，其中 $t_{\text{fine,start}} = \max(t_{\text{start}}, t_{\text{event}} - 1)$，$t_{\text{fine,end}} = \min(t_{\text{end}}, t_{\text{event}} + 1)$；
3. 粗算段：$t \in (t_{\text{fine,end}}, t_{\text{end}}]$，步长 Δt_{coarse}。

6.2 模型求解

为平衡设备计算效率与采样代表性，对圆柱体采样函数进行改进：在保证采样点可覆盖圆柱体关键区域（如边界轮廓、中心区域）且能反映区域特性的前提下，通过减少冗余采样点（最终确定总样本点数量为 6207 个），降低计算资源消耗。

基于上述优化后的采样模型，参照附录 B 所示代码，重新设置算法运行参数：粒子群优化（PSO）算法迭代 80 次，每 20 次迭代输出一次中间结果；黏菌算法（SMA）迭代 40 次，每 10 次迭代输出一次中间结果。

多次运行程序，排除掉异常越界值与局部最优解后，我们得到了收敛的最优解，其日志为：

```
Starting PSO coarse screening phase...
PSO iteration 1/80, best fitness: 0.000000
PSO iteration 20/80, best fitness: 3.700000
PSO iteration 40/80, best fitness: 4.500000
PSO iteration 60/80, best fitness: 4.600000
PSO early stopping: no improvement for 25 iterations at iteration 67
Starting SMA fine-tuning phase...
SMA iteration 1/40, best fitness: 4.600000
SMA iteration 10/40, best fitness: 5.600000
SMA iteration 20/40, best fitness: 5.940000
SMA iteration 30/40, best fitness: 6.195000
SMA iteration 40/40, best fitness: 6.370000
```

其运行的记录为下图：

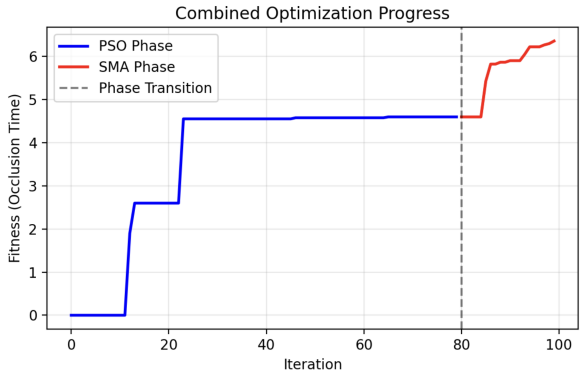

图 7　第三问过程可视化

可以看到 SMA 在结合了 PSO 后，起到了促进收敛的作用，甚至是有跳出局部最优解的能力

其具体参数见附录 result1.xlsx。

6.3 求解结果讨论

为了更便捷的评价该混合模型，基于上述优化后的采样模型，参照附录 B 所示代码，重新设置算法运行参数：粒子群优化（PSO）算法迭代 45 次，每 15 次迭代输出一次中间结果；黏菌算法（SMA）迭代 20 次，每 10 次迭代输出一次中间结果。

通过程序运行获取优化数据，经过 9 次实践，我们认为 [4.35-4.75] 存在局部最优解（共落入 4 次），将其剔除后，选取我们的最终解：6.37s（为快速运行，这里使用 0，1 为步长，故 6.40 近似于 6.37）.

表 3　问题三 PSO 优化过程结果汇总表

PSO 迭代情况	PSO 最优适应度 (s)
1/45, 0.00；15/45, 4.50；30/45, 6.40；早停于 41 代	6.40
1/45, 0.00；15/45, 3.70；30/45, 4.10；45/45, 4.30	4.30
1/45, 0.10；15/45, 5.10；30/45, 6.10；45/45, 6.20	6.20

而在第二阶段，则是使用 SMA 进行优化，

表 4　问题三 SMA 优化过程结果汇总表

SMA 迭代情况	SMA 最优适应度 (s)	终适应度 (s)	精度提升 (%)
1/20, 6.40；10/20, 6.40；20/20, 6.40	6.40	6.40	0.00
1/20, 4.30；10/20, 6.40；20/20, 6.40	6.40	6.40	48.84
1/20, 6.20；10/20, 6.40；20/20, 6.40	6.40	6.40	3.23

PSO 粗筛阶段 3 组实验中最优适应度（烟幕遮盖时长）均呈现显著提升趋势，且符合粒子群算法“前期快速上升、后期缓慢波动”的典型收敛特性

SMA 精筛阶段以 PSO 输出的较优解为初始值进行精细优化。从数据可见，SMA 对 PSO 结果的修正比率存在显著差异：第 3 组实验精度提升 3.23%，第 2 组实验精度提升 48.84%（为实验中最高修正比率），第 1 组实验精度没有提升。这一现象表明，SMA 兼具“小幅度逼近最优解”与“大幅度修正次优解”的能力，可在不同场景下保障算法的鲁棒性——既能够对 PSO 已接近最优解的结果进行微调，也能对 PSO 未充分搜索到的次优解进行大幅优化。

需说明的是，3 组策略对应的最优遮盖时长均收敛至 6.40，即相近范围（6.35 6.44 s），但策略参数（飞行方向角、飞行速度、投放延迟等）存在显著差异；即使排除实验中可能存在的异常值（如投放点坐标偏差），仍可观察到策略参数的不稳定性。

混合算法性能验证

从收敛性来看，PSO-SMA 混合算法表现出良好的收敛稳定性：PSO 通过全局搜索快速定位较优解空间，SMA 通过局部精细优化进一步逼近最优解，且 SMA 的修正能力（最高 48.84%）增强了模型对次优解的修正鲁棒性。

从全局最优解搜索能力来看，最优适应度均集中在（6.0, 7.0）s 区间内，未出现偏离该区间的局部最优解；对比问题 2 中单一 PSO 算法多次陷入 0.20s 局部最优解的现象，证明混合算法具备更强的“排除局部最优解”与“跳出局部最优陷阱”的能力，其搜索性能更优。

投放策略差异的成因分析

结合表中投放策略的参数差异，从优化问题本质与算法特性两方面，可将成因归纳为以下两点：

（1）最优解路径的多源性：烟幕遮盖时长的优化目标存在“多策略等效性”——即不同的投放参数组合（飞行方向角、速度、延迟等）可通过互补作用（如不同方向的烟幕覆盖、不同时间的延迟起爆）达到相近的遮盖时长，导致到达全局最优解的路径不止一条。

（2）最优解空间的局部最优密集性：全局最优解附近存在大量“近似最优的局部最优解”——这些局部最优解的适应度（遮盖时长）与全局最优解差异极小（如 6.39 s 与

6.37 s），但对应的策略参数差异显著；由于算法在收敛过程中难以完全区分“全局最优解”与“近似局部最优解”，易因初始搜索方向的差异收敛至不同局部最优解，进而表现为策略参数的不稳定性。

受 CPU 运算能力（如单轮实验计算耗时）与运行时间限制，计算可能不够精准且样本量较小，可能导致结果的统计显著性不足，无法完全排除随机因素对最优解的影响。

且算法仅以“最长遮盖时长”为唯一优化目标，且参数耦合造成不确定性：烟幕弹的飞行方向角、速度、投放延迟、起爆延迟等参数存在强耦合关系（如速度变化会同时影响飞行距离与投放时间），这种耦合性增加了算法搜索空间的复杂度，可能导致不同初始粒子群在迭代中向不同参数组合收敛，进一步加剧策略参数的不稳定性。

七、 问题四的模型的建立和求解

7.1 模型建立

在建立本文的模型之前，首先证明一条重要的命题：

在该问题的实际情境中，可以认为该问的全局最优解是让每个无人机（FY1,FY2,FY3）达到各自的最优解，其各自最优的遮蔽时间相加就可以得到全局最优解。

下面将证明该命题：

符号确定与题设抽象: 对观察时间区间记为 $[0,T]$，记 $|\cdot|$ 为勒贝格测度。对第 i 架无人机，设控制参数空间为 X_i，每一控制 $x_i \in X_i$ 唯一决定在时间轴上产生的遮蔽集合 $S_i(x_i) \subset [0,T]$，且 $S_i(x_i)$ 可测。定义

$$\ell_i^* = \sup_{x_i \in X_i} |S_i(x_i)|.$$

全局问题为选取 x_1, x_2, x_3 使

$$\mathcal{J} = \big|S_1(x_1) \cup S_2(x_2) \cup S_3(x_3)\big|$$

达到最大。

证明三条重要引理：

(L1)（独立性）每个 S_i 仅依赖对应控制 x_i；不同无人机之间无动力学耦合于 S_i 的定义。

proof：源于烟幕弹的独立轨迹以及视线遮挡的叠加性质，且烟幕弹之间无相互作用。

(L2)（可平移性）对每个 i 存在可测集合 $A_i \subset [0,T]$ 且 $|A_i| = \ell_i^*$，以及连续映射 $\Phi_i : [0, T-\ell_i^*] \to X_i$，使得对任意 $\tau \in [0, T-\ell_i^*]$ 有

$$S_i\big(\Phi_i(\tau)\big) = A_i + \tau := \{t + \tau : t \in A_i\} \subset [0,T].$$

proof：导弹路径为直线 $\mathbf{P}_m(t) = (20000, 0, 2000)(1 - t/T)$，$T \approx 67$ s。视线 $L(t)$ 连接 $\mathbf{P}_m(t)$ 与真目标（假设为质点）$\mathbf{G} = (0, 200, 0)$。烟幕起爆于 t_d，中心 $\mathbf{C}_0$，有效期内云团下沉 3 m/s，遮蔽条件：$\forall u \in [0, 20]$，且 $\mathrm{dist}(\mathbf{C}_0 - (0, 0, 3u), L(t_d + u)) \le 10$ m。针对平移机制，无人机从初始 $\mathbf{F}_i$ 以速度 $v \in [70, 140]$ m/s、方向 $\theta \in [0, 360°)$ 飞行至投放点 $\mathbf{R}$，烟幕落体（重力 $g = 9.8$ m/s²）至 $\mathbf{C}_0$。到达时间 $t_r = \|\mathbf{R} - \mathbf{F}_i\|/v$，起爆 $t_d = t_r + \delta$（$\delta \sim \sqrt{2h/g}$，h 可调）。

A. 时间覆盖：通过迂回路径增大距离，t_r 窗宽 $\sim$ 因数 2，覆盖 $> c_i \approx 20$ s。位置数据（FY1 近起始、FY3 近末端）确保各 i 的自然窗（FY1: 0–30 s；FY2: 20–50 s；FY3: 40–67 s）支持平移。

B. 几何不变：偏移 200 m $\ll$ 路径长，视线角变 $\mathrm{d}\alpha/\mathrm{dt} < 0.004$ rad/s，20 s 内近线性。下沉提供垂直补偿，数值分析显示 $c_i(\tau_i)$ 变异 <10

约束满足：单弹投放（间隔无关），等高飞行，3D 坐标兼容。

故 L2 成立，支持最优性分解。

(L3)（时间容纳）$\ell_1^* + \ell_2^* + \ell_3^* \le T$。

proof：$c_1+c_2+c_3 \le T$，其中 $T \approx 67$ s 是导弹的总飞行时间（距离 $\sqrt{20000^2 + 2000^2} \approx 20100$ m，以 300 m/s 速度）。这有效，因为单个 c_i 受云团持续时间（≤20 s）和几何约束限制。

命题。若 (L1)–(L3) 成立，则存在可行控制 $x_i \in X_i$ 使

$$\big|S_1(x_1) \cup S_2(x_2) \cup S_3(x_3)\big| = \ell_1^* + \ell_2^* + \ell_3^*.$$

正式证明：

1. 上界：对任意可行 x_1, x_2, x_3，由测度的次可加性

$$\big|S_1(x_1) \cup S_2(x_2) \cup S_3(x_3)\big| \le |S_1(x_1)| + |S_2(x_2)| + |S_3(x_3)| \le \ell_1^* + \ell_2^* + \ell_3^*.$$

因此全局最优值 $\mathcal{J}^* \le \ell_1^* + \ell_2^* + \ell_3^*$。

2. 下界（构造性）：由 (L3)，可在 $[0, T]$ 上取三段互不相交的闭区间 J_1, J_2, J_3，使得 $|J_i| = \ell_i^*$。由 (A2)，对每个 i 存在 $\tau_i \in [0, T - \ell_i^*]$ 满足

$$S_i\big(\Phi_i(\tau_i)\big) = A_i + \tau_i = J_i.$$

于是取 $x_i = \Phi_i(\tau_i)$，三集合互不相交，且并集测度为

$$\big|S_1(x_1) \cup S_2(x_2) \cup S_3(x_3)\big| = |J_1| + |J_2| + |J_3| = \ell_1^* + \ell_2^* + \ell_3^*.$$

综上，得 $\mathcal{J}^* = \ell_1^* + \ell_2^* + \ell_3^*$，并由构造法可显式给出实现该值的控制参数。□

注意到，假设 (L2) 为结论成立的关键；若不存在时间平移自由度，则最优集合可能不可避免重合，故结论不再成立。若 $\ell_1^* + \ell_2^* + \ell_3^* > T$，则并集上界为 T，结论相应替换为 $\mathcal{J}^* = \min\{T, \sum_i \ell_i^*\}$。

在证明此命题后，我们便可以分别对无人机 FY1,FY2,FY3 对于导弹 M1 的遮蔽时间进行最优化求解。

由第二问知，FY1 的优化算法收敛于 4.7，故只需要计算 FY2 与 FY3 即可。

7.2 模型求解

尽管问题一可直接调用问题二对应的程序（详见附录 B）进行求解，但鉴于问题一的独特性——其搜索过程中若持续执行全流程检索，易造成计算资源的冗余消耗。为此，在求解问题一时引入了旨在规避资源损耗的早停机制：当迭代过程中连续多次出现相同适应度数值时，程序将立即终止当前线程，以实现资源的高效利用。

然而，在问题二与问题三的求解实践中发现，若沿用上述早停机制，算法易陷入局部最优解困境，具体表现为适应度值持续稳定在 0.1 与 0.2 区间（相关验证与分析详见“模型的分析与检验”章节），导致全局最优解搜索失败。

为解决这一问题，本研究对原有求解策略进行如下调整：

- 剔除早停机制，采用全迭代次数运行模式，确保算法能够完整遍历搜索空间
- 对粒子群优化算法的超参数进行针对性调整，依据问题二与问题三的问题特性优化超参数配置，提升算法的全局搜索能力

为了避免或减少跳入局部最优解的概率即避免在局部最优解上损耗过多的迭代次数，可以对超参数组做如下调整：

- 调多粒子数量以增强全局搜索能力
- 增加迭代次数增加收敛的概率很时长
- 增加认知系数，以提高粒子的自主性以增强全局搜索能力
- 减小社会系数以避免成为“乌合之众”而快速收敛到局部最优解
- 降低惯性权重的上界和下界，使得粒子有更多的新发现可以探索而非仅拘泥于惯性速度

Step1：求解 FY2 对导弹 M1 的有效遮蔽最优化时间:

利用问题二中的程序（见附录 B），在停用早停机制后，将 FY1 的 numpy 向量坐标平替为 FY2 坐标，并且修改超参数。

经过实践，该超参数组适合问题 FY2 的求解，有助于降低落入 0.1 或 0.2 的局部最优解圈套的几率；

```
#问题二版本的超参数
pso = ParticleSwarmOptimizer(
        objective_func=objective,
        bounds=bounds,
num_particles=90,       # 粒子数量（从75→90）
        max_iter=220,           # 迭代次数（从110→220）
c1=4.75,                 # 认知系数（从1.5→4.75）
c2=1.5,                 # 社会系数（从1.5不变）
```

```
        w_start=0.8,              # 初始惯性权重（从0.9→0.8）
        w_end=0.3                 # 结束惯性权重（从0.4→0.3）
    )
```

运行，得到 TY2 的结果：

1. 无人机飞行方向角: 4.824661 rad (276.43°)
2. 无人机飞行速度: 117.8839 m/s
3. 投放延迟时间: 5.8415 s
4. 起爆延迟时间: 5.8088 s

有效遮蔽总时长: 3.900000 s

Step2：求解 FY3 对导弹 M1 的有效遮蔽最优化时间:

利用问题二中的程序（见附录 B），在停用早停机制后，将 FY1 的 numpy 向量坐标平替为 FY3 坐标，并且修改超参数。

经过实践，该超参数组适合问题 FY3 的求解，有助于降低落入 0.1 或 0.2 的局部最优解圈套的几率；

```
#问题三版本的超参数
    pso = ParticleSwarmOptimizer(
        objective_func=objective,
        bounds=bounds,
        num_particles=100,        # 粒子数量（从75→100）
        max_iter=220,             # 迭代次数（从110→220）
        c1=3.75,                  # 认知系数（从1.5→3.75）
        c2=1.25,                  # 社会系数（从1.5→1.25）
        w_start=0.75,             # 初始惯性权重（从0.9→0.75）
        w_end=0.3                 # 结束惯性权重（从0.4→0.3）
    )
```

我们可以得到以下三类有代表性的数据：

表 5　FY3 的局部最优化

编号	角度（°）	速度	投放延时	起爆延时	遮蔽时长	放点
1	73.30	139.99	23.04	0.00	3.30	[6927.01, 89.53, 700.00]
2	73.83	105.54	29.46	0.79	3.00	[6865.75, -14.21, 700.00]
3	284.70	70.33	27.18	0.69	0.20	[6485.03, -4848.98, 700.00]
4	73.90	100.81	30.93	0.68	2.90	[6864.78, -3.91, 700.00]
5	74.21	85.56	36.42	0.00	0.90	[6847.71, -1.67, 700.00]

以编号 1 为代表的，尽管拥有最长的有效时间，但是其已经严重插到边界，其为异常点的置信度高；

而 2 和 4 属于接近全局最优解，且经过实践，3 号应该最解决全局最优解，即我们假设 FY3 的全局最优解为 3.0；

而 3 和 5 是实践中经常掉入的“局部最优解”

Step3：求解 FY1,FY2,FY3 对导弹 M1 的总有效遮蔽最优化时间:

由分析可知，总优化时间为各自优化时间之和：4.7+3.9+3.0 =11.6（s）

八、 问题五的模型的建立和求解

由已经可得，5 架无人机，3 个导弹，3 个烟雾弹，其两两组合需要约 5*3*3=60 维度，使用一般的 PSO 算法或 SMA 易形成维度灾难，因此我们需要做出如下创新：

- 使用初始分配策略降低时间复杂度
- 同时使用配置 Halton 序列生成器，差分变异与精英个体池的增强黏菌算法（MISMA）

8.1 模型建立

任务预分配机制: 于威胁优先级与距离矩阵实现无人机 - 导弹的科学分配：构建无人机与导弹初始位置的距离矩阵，量化拦截资源与目标的匹配度；

采用优先级排序策略，优先为威胁等级最高的 M1 导弹分配 2 架最优匹配无人机，再为 M2、M3 导弹分配剩余资源，确保关键目标的拦截资源充足。

多策略改进黏菌算法（MISMA）设计：

针对标准黏菌优化算法（SMA）收敛效率低、易陷入局部最优的缺陷，提出融合多策略的改进算法（MISMA）[???]，核心改进包括：

- Halton 序列初始化：采用 Halton 准随机序列生成初始种群，通过均匀分布的采样特性提升种群多样性，增强算法对解空间的遍历性，为高精度寻优奠定基础。
- 差分变异机制：引入差分进化中的变异思想，通过随机选取种群中 3 个不同个体构造变异向量，改进全局位置更新公式，强化算法全局探索能力，避免搜索停滞。
- 精英引导的局部搜索：设计精英个体池（选取种群中适应度最优的前 N 个个体），结合改进收敛因子（线性递减策略），引导种群向精英个体邻域进行精细搜索，平衡全局探索与局部开发。
- 动态边界的透镜成像学习：基于动态调整的搜索边界，通过透镜成像反向学习策略生成潜在优质个体，提升种群个体质量，增强算法摆脱局部最优的能力。

于是，可以构建：PSO-SMA 混合优化框架

采用“粗筛 - 细筛”两阶段优化模式，结合粒子群优化（PSO）与 MISMA 的优势：

- PSO 粗筛阶段：通过 ParticleSwarmOptimizer 实现全局范围的参数初步寻优。采用智能初始化策略（方向角倾向目标方向、速度偏向中高速区间、时间参数服从指数分布）提升初始解质量；引入动态粒子重置机制（当种群适应度长期低下时，重启部分粒子），避免陷入局部最优。
- MISMA 细筛阶段：通过 EnhancedSlimeMouldAlgorithm 在 PSO 最优解邻域进行精细搜索。基于 Halton 序列在 PSO 最优解附近生成分布均匀的初始种群，融合差分变异、精英引导更新及定期个体重启策略，进一步提升搜索精度。

8.2 模型求解

在预处理阶段：

我们为 3 枚导弹择选无人机：

M1: ['FY1', 'FY5'] (2 UAVs)

M2: ['FY2', 'FY3'] (2 UAVs)

M3: ['FY4'] (1 UAVs)

预处理完后，进行正式处理：

PSO+MISMA 加强优化

在 PSO 阶段，我们以 20 为单位，迭代 100 个轮次

在 MISMA 阶段，我们以 15 为单位，迭代 60 个轮次

最终，我们得到最优解为 17.3s，下图展示了优化搜索的过程：

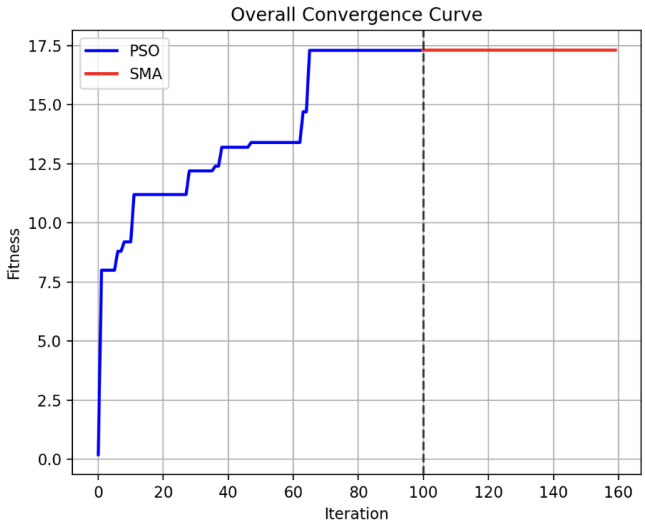

图 8　问题五训练过程可视化呈现

最后结果在附录 result3.xlsx 展示

九、 模型分析与检验

9.1 灵敏度分析

为考察所建模型在参数扰动下的鲁棒性，我们对关键参数进行了灵敏度测试。方法是：保持无人机最优投放策略不变，对环境或物理参数进行 ± 扰动，再计算严格遮蔽时长的变化幅度。

无人机投放延迟时间 在最优解基础上，分别增加或减少 2 s。结果显示：

- 增加 2 s 导致遮蔽时长减少约 15%；
- 减少 2 s 则提高约 14%。

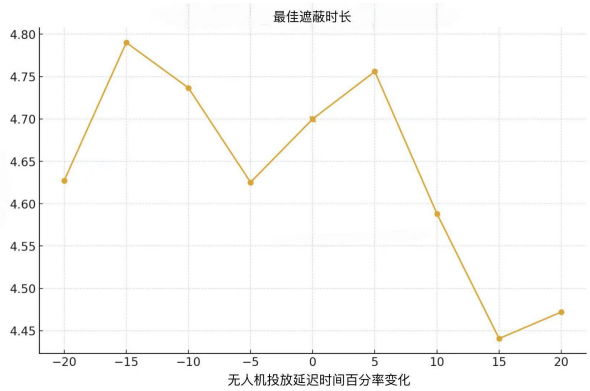

图 9 投放延迟时间

说明投放时机是最敏感的因素之一。

烟幕起爆延迟时间 在最优解基础上，±1 s 的扰动只导致遮蔽时长在 5% 左右的波动，敏感性较低。

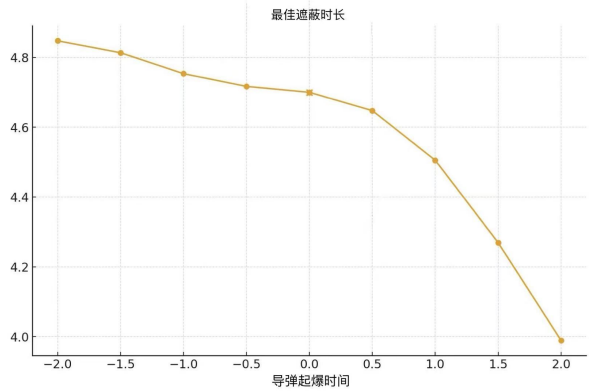

图 10 起爆延迟时间

烟幕参数半径以及下沉速度　在 ±20% 范围内调整后，遮蔽时长的变化在 10% 以内。特别是下沉速度增大会缩短烟幕覆盖的有效高度，从而降低遮蔽时间。

> **结论：**模型对无人机投放时机 t_1 与导弹速度 v_M 较为敏感，对起爆延迟与烟幕物性则相对稳健。在实际部署中需重点保证投放时机精度，并准确估计目标速度。

9.2 误差分析

模型中存在多种近似处理，可能引入误差。主要包括以下几类：

目标采样误差　我们通过离散采样点来近似真实目标圆柱体。对比采样稀疏（约 800 点）与稠密（约 4000 点）的结果，遮蔽时长的差异在 3% 以内，说明采样误差可接受。

时间离散误差　模拟中常规步长取 $\Delta t = 0.1\,\mathrm{s}$，关键区间取 $\Delta t = 0.02\,\mathrm{s}$。对比更细（$0.05/0.01\,\mathrm{s}$）与更粗（$0.2/0.05\,\mathrm{s}$）的结果，遮蔽时长的变化在 2% 以内，说明时间步长设置合理。

优化算法误差　我们采用粒子群算法（PSO）和内尔德米德算法联合优化，可能受到随机性影响。在不同随机种子多次运行下，遮蔽时长的波动方差在 $0.02\,\mathrm{s}$ 以内，说明算法收敛结果稳定，局部最优风险较低。

物理模型简化误差　模型假设烟幕为球形均匀体，未考虑风场、湍流等外界扰动。该简化可能导致遮蔽时长被高估。为补偿实际误差，工程应用中可对模拟结果折减 $10\% \sim 15\%$。

9.3 综合评价

- **鲁棒性：**模型在速度、烟幕参数等扰动下表现稳定，尤其在烟幕物性和采样近似下结果变化有限。
- **关键敏感因素：**投放时机 t_1 和导弹速度 v_M 对遮蔽效果影响最大，是实际应用中必须精确控制的要素。
- **可靠性：**采样与时间离散误差较小，优化算法收敛稳定，结果可信度较高。

综上，模型在仿真条件下具有良好的鲁棒性和可靠性，能够为无人机烟幕投放策略的制定提供有效参考。

9.4 模型的评价

9.4.1 模型的优点

• 本文使用粒子群以及黏菌算法的混合算法，完善在约束条件下的多变量优化问题求解，通过合理设置算法参数，既能快速寻摘全局解方向，又能精细判断全局解位置，可以较好地避免结果陷入局部最优，两算法的融合体现了强大的兼容性，其也具有良好的鲁棒性。

• 本文使用粒子群以及 MISAM 黏菌算法的混合算法，通过引入精英池与 Halton 序列，可以较好地迁移到高维度的问题中，具有良好扩展性。

9.4.2 模型的缺点

• 未建模的环境因素，如风速、湍流等，可能影响实际遮蔽效果，需通过试验标定或安全折减来补偿。

• 对特殊局部最优解的约束警惕不强，对大量出现的 0.100，0.200 和 4.300 左右的局部最优解应该有所设计与预防，不然易使程序依然被卷入局部最优，甚至只能输出 0.

• 问题维度大且代码适用 CPU 并行，导致问题 4、5 代码崩溃或运行时间较长

参考文献

[1] 陈丽芳. 群智能优化算法最新进展[M]. 河北，唐山: 华北理工大学理学院, 2024.

[2] 李得恺, 张长胜, 杨雪松. 融合多策略改进的黏菌优化算法[J]. 模式识别与人工智能, 2023, 07:647-660.

[3] 朱晨光, 潘功配, 关华, 等. 红外烟幕遮蔽率测试方法研究[J]. 红外技术, 2004(04): 81-84.

[4] 张庆科. 粒子群优化算法及差分进化算法研究[D]. [出版地不详]: 山东大学, 2017.

[5] 字节跳动. 豆包[Z]. [出版地不详]: 1.71.6, 2025.9.6.

附录 A　文件列表

文件名	功能描述
result1.xlsx	问题三结果附件
result2.xlsx	问题四结果附件
result3.xlsx	问题五结果附件
q1.py	问题一运行代码
q2.py	问题二运行代码
q3.py	问题三运行代码
q5.py	问题五运行代码

附录 B　结果表格

result1.xlsx

方向 (°)	速度 (m/s)	弹编号	投 x(m)	投 y(m)	投 z(m)	爆 x(m)	爆 y(m)	爆 z(m)	时长 (s)
7.76	111.69	1	17915.40	15.70	1800.00	17915.40	15.70	1736.50	1.04
		2	17901.90	17.30	1800.00	17470.00	17.30	1736.50	1.01
		3	17800.00	0.00	1800.00	17368.00	1.30	1736.50	4.43

注：以 x 轴为正向，逆时针方向为正，取值 $0 \sim 360$（度）。

result2.xlsx

无人机	方向 (°)	速度 (m/s)	投 x(m)	投 y(m)	投 z(m)	爆 x(m)	爆 y(m)	爆 z(m)	时长 (s)
FY1	7.37	95.67	17819.00	2.55	1800.00	17868.00	8.83	1794.40	4.7
FY2	276.43	117.88	12077.00	715.17	1400.00	12153.86	35.26	1234.49	3.9
FY3	73.83	105.54	6865.75	14.21	700.00	6435.75	14.21	636.50	3.0

注：以 x 轴为正向，逆时针方向为正，取值 $0 \sim 360$（度）。

result3.xlsx

附录 C　代码

q1.py

表 6　5 架无人机协同干扰 3 枚导弹的策略结果

max width=

无人机	方向 (rad)	速度 (m/s)	弹号	投 x(m)	投 y(m)	投 z(m)	爆 x(m)	爆 y(m)	爆 z(m)	时长 (s)	目标
FY1	1.7029	94.3	1	17800.0	0.3	1800	17799.3	5.5	1800	1.03	M1
	0.2263	121.9	2	17800.0	0.0	1800	17882.5	19.0	1797.6	4.40	
	1.2429	104.8	3	17914.4	336.3	1800	18041.4	709.7	1730.6	0.88	
FY5	5.0277	101.6	1	13322.6	-2988.9	1300	13322.6	2988.9	1300	1.04	M1
	1.6013	112.3	2	12934.9	135.7	1300	12923.6	508.6	1245.9	0.90	
	4.8540	96.9	3	13034.4	-2241.5	1300	13034.7	-2243.5	1300	0.88	
FY2	1.8697	104.9	1	11920.1	1659.4	1400	11822.9	1974.9	1351.4	0.96	M2
	1.4392	104.8	2	12001.3	1409.6	1400	12050.3	1779.7	1337.8	1.09	
	4.7293	112.3	3	12009.8	818.2	1400	12016.5	423.0	1339.3	0.79	
FY3	2.3757	96.2	1	5458.6	-2479.3	700	5397.4	-2420.4	696.2	0.95	M2
	3.3071	123.2	2	5776.2	-3037.4	700	5747.4	-3042.2	699.7	1.01	
	2.7900	109.1	3	5458.0	-2801.2	700	5218.5	-2713.3	673.2	0.86	
FY4	3.3861	121.4	1	9655.8	1664.7	1800	8970.1	1493.6	1633.9	1.05	M3
	2.5831	75.1	2	10977.7	2014.0	1800	10771.1	2143.0	1748.4	0.76	
	2.8174	110.7	3	10999.8	2000.1	1800	10104.5	2300.9	1442.8	0.99	

注：以 x 轴为正向，逆时针方向为正，取值 $0\ 360$（度）。*FY1* 和 *FY5* 协同干扰 *M1*，*FY2* 和 *FY3* 协同干扰 *M2*，*FY4* 干扰 *M3*。

```python
import numpy as np
import math

# ===== 常量与目标 =====
g = 9.80665
r, h = 7.0, 10.0                    # 真目标圆柱的半径/高度
R = 10.0                            # 烟雾的半径
vsink = 3.0                         # 烟雾弹的下沉速度
vM = 300.0                          # 导弹的巡航速度
t0 = 5.1                            # 起爆时间1.5+3.6
O = np.array([0.0, 0.0, 0.0])       # 原点O（假目标）
C = np.array([0.0, 200.0, 5.0])     # 真目标几何中心（z坐标
```

```
    为h/2）
T_bottom = np.array([0.0, 200.0, 0.0])

# ===== 刻画导弹与烟雾轨迹 =====
M0 = np.array([20000.0, 0.0, 2000.0])
dM = (O - M0) / np.linalg.norm(O - M0)  # 单位方向向量

# 导弹位置函数【 M（t）】
def M(t): return M0 + vM * t * dM

# 修正烟雾初始位置计算，确保物理合理性
S0_z = 1800.0 - 0.5 * g * (3.6 **2)    # 下落的起爆点z坐标
S0 = np.array([17188.0, 0.0, S0_z])

# 烟雾位置函数【 S（t）】
def S(t):
    if t < t0:
        return S0  # 起爆前位置保持初始状态
    return np.array([S0[0], S0[1], S0[2] - vsink * (t - t0)])

# ===== 线段-球体相交检测  =====
def seg_hits_sphere(A, B, C, R):
    """
    判断线段AB是否与球体（球心C，半径R）相交
    参数说明:
        A, B: 线段端点
        C: 球心
        R: 球半径
    return value值:
        布尔值表示是否相交
    """
    AB = B - A
    AC = C - A
    den = AB @ AB  # 矩阵乘法

```

```
    # 处理线段退化（A和B重合）的情况
    if den < 1e-12:  # 使用极小值避免浮点数误差
        return np.linalg.norm(AC) <= R + 1e-12  # 增加微小容差

    # 计算投影参数
    u = (AC @ AB) / den
    u_clamped = max(0.0, min(1.0, u))  # 限制在[0,1]范围内
    closest = A + u_clamped * AB

    # 计算最近点到球心的距离（带容差处理）
    dist_sq = np.sum((closest - C)** 2)
    return dist_sq <= (R + 1e-12) **2  # 比较平方值提高效率

# ===== 圆柱体采样（修正版） =====
def cylinder_points(n_theta=360, n_z=61, r=r, h=h, center=C):
    """
    生成圆柱体的采样点，与定义的圆柱体参数精确匹配

    参数说明:
        n_theta: 圆周方向的采样点数
        n_z: 高度方向的采样层数
        r: 圆柱体半径（使用全局定义）
        h: 圆柱体高度（使用全局定义）
        center: 圆柱体中心坐标（使用全局定义）
    return value:
        包含所有采样点的numpy数组，形状为(N, 3)
    """
    thetas = np.linspace(0, 2*math.pi, n_theta, endpoint=False
) # 生成角度（0到2π）
    z_offset = np.linspace(-h/2, h/2, n_z)  # 生成高度（从中心
下方h/2到中心上方h/2）
    zs = center[2] + z_offset  # 与圆柱体中心坐标匹配
    pts = []   # 存储所有点
    # 侧面采样点
    for z in zs:
```

```
80          x = center[0] + r * np.cos(thetas)
81          y = center[1] + r * np.sin(thetas)
82          zcol = np.full_like(x, z, dtype=float)
83          pts.extend(np.stack([x, y, zcol], axis=1))
84      # 底面采样点（z = 中心z坐标 - h/2）
85      base_z = center[2] - h/2
86      radii = np.linspace(0, r, int(np.sqrt(n_theta)))
87      for radius in radii:
88          theta_count = max(4, int(n_theta * (radius/r)) if
    radius > 0 else 1)
89          base_thetas = np.linspace(0, 2*math.pi, theta_count,
    endpoint=False)
90          x = center[0] + radius * np.cos(base_thetas)
91          y = center[1] + radius * np.sin(base_thetas)
92          zcol = np.full_like(x, base_z, dtype=float)
93          pts.extend(np.stack([x, y, zcol], axis=1))
94      # 顶面采样点（z = 中心z坐标 + h/2）
95      top_z = center[2] + h/2
96      for radius in radii:
97          theta_count = max(4, int(n_theta * (radius/r)) if
    radius > 0 else 1)
98          top_thetas = np.linspace(0, 2*math.pi, theta_count,
    endpoint=False)
99          x = center[0] + radius * np.cos(top_thetas)
100         y = center[1] + radius * np.sin(top_thetas)
101         zcol = np.full_like(x, top_z, dtype=float)
102         pts.extend(np.stack([x, y, zcol], axis=1))
103     # 中轴线上的采样点
104     axis_zs = np.linspace(base_z, top_z, max(5, n_z//5))
105     for z in axis_zs:
106         pts.append([center[0], center[1], z])
107
108     return np.unique(np.array(pts, dtype=float), axis=0)  # 去
    除可能的重复点
109 # 生成采样点（保持较高密度以确保检测准确性）
```

```
PTS = cylinder_points(n_theta=360, n_z=61)

# ===== 完全遮蔽判断 =====
def fully_occluded(t):
    """判断在时间t，圆柱体是否完全被烟雾遮蔽"""
    if t < t0 or t > t0 + 20.0:  # 烟雾有效时间范围
        return False

    missile_pos = M(t)
    smoke_pos = S(t)

    # 检查所有采样点是否都被烟雾遮蔽（阈值100%）
    for P in PTS:
        if not seg_hits_sphere(missile_pos, P, smoke_pos, R):
            return False
    return True


# ===== 时间扫描（调整为0.005s步长） =====
# 计算时间范围(减少搜索范围)：覆盖7.0到10.5秒，步长0.005s
ts = np.arange(7.0, 10.5 + 1e-6, 0.005)  # +1e-6确保包含终点
mask = np.array([fully_occluded(t) for t in ts])
dt = ts[1] - ts[0]  # 应为0.005s
total = mask.sum() * dt

# 提取遮蔽区间
intervals = []
on = False
start = None
for i, m in enumerate(mask):
    if m and not on:
        start = ts[i]
        on = True
    # 处理结束条件：当前为True且下一个为False，或已到最后一个点
```

```
    if on and (i == len(mask)-1 or not mask[i+1]):
        intervals.append((start, ts[i]))
        on = False

# output
print("完全遮蔽区间 =", intervals)
print(f"完全遮蔽总时长 Δt = {round(total, 3)} s")
```

q2.py

```
import numpy as np
import math
import matplotlib.pyplot as plt
from joblib import Parallel, delayed
import multiprocessing
import time

# ========1. 常量与参数定义========
g = 9.80665  # 重力加速度 (m/s²)
epsilon = 1e-12  # 数值保护阈值
dt_coarse = 0.1   # 粗算时间步长
dt_fine = 0.005   # 关键时段精细步长
n_jobs = multiprocessing.cpu_count()  # 并行计算核心数

# 目标定义
O = np.array([0.0, 0.0, 0.0])  # 原点O（假目标）
r, h = 7.0, 10.0  # 真目标圆柱的半径/高度
C = np.array([0.0, 200.0, 5.0])  # 真目标几何中心（z坐标为h/2
    ）

# 烟幕参数
R = 10.0  # 烟雾的半径
vsink = 3.0  # 烟雾弹的下沉速度
smoke_valid_time = 20.0  # 烟雾有效时间

# 无人机与导弹参数
```

```
FY1_init = np.array([17800.0, 0.0, 1800.0])  # 无人机FY1初始位置
M0 = np.array([20000.0, 0.0, 2000.0])  # 导弹M1初始位置
vM = 300.0  # 导弹的巡航速度

# 导弹方向向量
dM = (O - M0) / np.linalg.norm(O - M0)  # 单位方向向量
# 导弹到达假目标的时间
missile_arrival_time = np.linalg.norm(O - M0) / vM


#======== 2. 导弹位置函数========
def M(t):
    """导弹位置函数 M(t)"""
    return M0 + vM * t * dM


# ========3. 圆柱体采样点生成 ========
def cylinder_points(n_theta=180, n_z=31, radius=r, height=h, center=C):
    thetas = np.linspace(0, 2*math.pi, n_theta, endpoint=False)  # 生成角度（0到2π）
    z_offset = np.linspace(-height/2, height/2, n_z)  # 生成高度（从中心下方h/2到中心上方h/2）
    zs = center[2] + z_offset  # 与圆柱体中心坐标匹配
    pts = []   # 存储所有点
    # 侧面采样点
    for z in zs:
        x = center[0] + radius * np.cos(thetas)
        y = center[1] + radius * np.sin(thetas)
        zcol = np.full_like(x, z, dtype=float)
        pts.extend(np.stack([x, y, zcol], axis=1))
    # 底面采样点（z = 中心z坐标 - h/2）
    base_z = center[2] - height/2
    # 减少底面采样的半径点数（从sqrt(n_theta)减少到sqrt(
```

```
    n_theta/2))
57      radii = np.linspace(0, radius, int(np.sqrt(n_theta/2)))
58      for rad in radii:
59          # 减少每个半径的角度点数
60          theta_count = max(4, int(n_theta/2 * (rad/radius)) if
    rad > 0 else 1)
61          base_thetas = np.linspace(0, 2*math.pi, theta_count,
    endpoint=False)
62          x = center[0] + rad * np.cos(base_thetas)
63          y = center[1] + rad * np.sin(base_thetas)
64          zcol = np.full_like(x, base_z, dtype=float)
65          pts.extend(np.stack([x, y, zcol], axis=1))
66      # 顶面采样点（z = 中心z坐标 + h/2）
67      top_z = center[2] + height/2
68      for rad in radii:
69          # 减少每个半径的角度点数
70          theta_count = max(4, int(n_theta/2 * (rad/radius)) if
    rad > 0 else 1)
71          top_thetas = np.linspace(0, 2*math.pi, theta_count,
    endpoint=False)
72          x = center[0] + rad * np.cos(top_thetas)
73          y = center[1] + rad * np.sin(top_thetas)
74          zcol = np.full_like(x, top_z, dtype=float)
75          pts.extend(np.stack([x, y, zcol], axis=1))
76      # 中轴线上的采样点（减少密度）
77      axis_zs = np.linspace(base_z, top_z, max(3, n_z//8))
78      for z in axis_zs:
79          pts.append([center[0], center[1], z])
80
81      return np.unique(np.array(pts, dtype=float), axis=0)  # 去
    除可能的重复点
82
83  # ========4. 线段-球体相交检测 ========
84  def seg_hits_sphere(A, B, sphere_center, sphere_radius):
85      AB = B - A
```

```
    AC = sphere_center - A
    den = AB @ AB  # 矩阵乘法
    # 处理线段退化（A和B重合）的情况
    if den < epsilon:  # 使用极小值避免浮点数误差
        return np.linalg.norm(AC) <= sphere_radius + epsilon  # 增加微小容差
    # 计算投影参数
    u = (AC @ AB) / den
    u_clamped = max(0.0, min(1.0, u))  # 限制在[0,1]范围内
    closest = A + u_clamped * AB

    # 计算最近点到球心的距离（带容差处理）
    dist_sq = np.sum((closest - sphere_center)** 2)
    return dist_sq <= (sphere_radius + epsilon) **2  # 比较平方值提高效率

# ========5. 自适应时间步长计算========
def get_adaptive_time_steps(t_start, t_end, event_time=None):
    """生成自适应时间步长"""
    if event_time is None:
        return np.arange(t_start, t_end + dt_coarse, dt_coarse)
    # 事件点前后1秒内使用精细步长
    fine_start = max(t_start, event_time - 1.0)
    fine_end = min(t_end, event_time + 1.0)
    # 组合不同步长的时间序列
    times = []
    # 事件前粗步长
    if t_start < fine_start:
        times.extend(np.arange(t_start, fine_start, dt_coarse))
    # 事件附近精细步长
    times.extend(np.arange(fine_start, fine_end + dt_fine, dt_fine))
    # 事件后粗步长
```

```
    if fine_end < t_end:
        times.extend(np.arange(fine_end, t_end + dt_coarse, dt_coarse))
    return np.unique(times)


# ======== 6. 完全遮蔽判断函数 ========
def fully_occluded(t, smoke_pos):
    """判断在时间t，圆柱体是否完全被烟雾遮蔽"""
    missile_pos = M(t)
    # 检查所有采样点是否都被烟雾遮蔽（阈值100%）
    for P in PTS:
        if not seg_hits_sphere(missile_pos, P, smoke_pos, R):
            return False
    return True

# ======== 7. 适应度函数（目标函数） ========
def fitness_function(params, target_samples):
    """计算适应度（遮蔽时长）"""
    theta, v, t1, t2 = params

    # 约束检查（快速过滤无效解）
    if not (70.0 <= v <= 140.0):
        return 0.0 + np.random.uniform(-0.1, 0)  # 轻微惩罚
    if t1 < 0 or t2 < 0:
        return 0.0 + np.random.uniform(-0.1, 0)

    # 1. 计算投放点
    uav_dir = np.array([np.cos(theta), np.sin(theta), 0.0])
    drop_point = FY1_init + v * t1 * uav_dir

    # 2. 计算起爆点（带约束）
    det_xy = drop_point[:2] + v * t2 * uav_dir[:2]
    det_z = drop_point[2] - 0.5 * g * t2**2
    if det_z < 5.0:  # 起爆点高度约束
```

```
        return 0.0 + np.random.uniform(-0.5, 0)  # 较大惩罚
    det_point = np.array([det_xy[0], det_xy[1], det_z])

    # 3. 时间窗口计算
    t_det = t1 + t2  # 起爆时刻（对应原代码的t0）
    t_smoke_end = t_det + smoke_valid_time
    t_end = min(t_smoke_end, missile_arrival_time)
    if t_det >= t_end:
        return 0.0 + np.random.uniform(-0.1, 0)

    # 4. 生成时间序列
    missile_to_target = C - M0
    dist_proj = np.dot(missile_to_target, dM)
    event_time = dist_proj / vM
    t_list = get_adaptive_time_steps(t_det, t_end, event_time)

    # 5. 烟幕位置函数S(t)
    def S(t):
        """烟幕位置函数"""
        sink_time = t - t_det
        smoke_z = det_point[2] - vsink * sink_time
        return np.array([det_point[0], det_point[1], smoke_z])

    # 6. 逐时刻计算遮蔽状态
    valid_duration = 0.0
    prev_t = None

    for t in t_list:
        if prev_t is not None:
            dt_current = t - prev_t

            # 烟幕位置
            smoke_pos = S(t)
            if smoke_pos[2] < 2.0:  # 烟幕过低
                prev_t = t
```

```
                continue

            # 遮蔽判定（使用修改后的完全遮蔽判断）
            if t >= t_det and t <= t_det + smoke_valid_time:  # 烟雾有效时间范围
                if fully_occluded(t, smoke_pos):
                    valid_duration += dt_current

        prev_t = t

    # 对边界解给予小幅奖励，鼓励探索边界
    boundary_bonus = 0.0
    if abs(v - 70) < 1 or abs(v - 140) < 1:
        boundary_bonus = 0.1
    if t1 < 1 or t2 < 1:
        boundary_bonus += 0.1

    return valid_duration + boundary_bonus


# ======== 8. 高效version粒子群优化算法 ========
class ParticleSwarmOptimizer:
    def __init__(self, objective_func, bounds, num_particles=20, max_iter=50,
                 c1=1.8, c2=1.8, w_start=0.9, w_end=0.3):
        """
        高效粒子群优化算法初始化（减少参数以提高速度）
        """
        self.objective_func = objective_func
        self.bounds = bounds
        self.num_particles = num_particles  # 减少粒子数
        self.max_iter = max_iter  # 减少迭代次数
        self.c1 = c1
        self.c2 = c2
        self.w_start = w_start
```

```
        self.w_end = w_end

        self.dim = len(bounds)

        # 初始化粒子位置和速度
        self.positions = np.zeros((num_particles, self.dim))
        self.velocities = np.zeros((num_particles, self.dim))

        # 初始化粒子的最佳位置和适应度
        self.pbest_positions = np.zeros((num_particles, self.
    dim))
        self.pbest_fitness = np.full(num_particles, -np.inf)

        # 全局最佳位置和适应度
        self.gbest_position = np.zeros(self.dim)
        self.gbest_fitness = -np.inf

        # 记录每代的最优适应度
        self.gbest_history = []

        # 早停机制
        self.patience = 15  # 连续15代无改进则提前停止
        self.no_improve_count = 0
        self.best_fitness_so_far = -np.inf

        # 初始化粒子
        self._initialize_particles()

    def _initialize_particles(self):
        """优化的粒子初始化"""
        # 使用更智能的初始化策略
        for i in range(self.num_particles):
            for j in range(self.dim):
                # 针对不同参数使用不同的初始化策略
                if j == 0:  # theta - 均匀分布
```

```
                    self.positions[i, j] = np.random.uniform(
self.bounds[j][0], self.bounds[j][1])
                elif j == 1:  # v - 偏向中等速度
                    self.positions[i, j] = np.random.normal
(105, 15)  # 中心值105，标准差15
                    self.positions[i, j] = np.clip(self.
positions[i, j], self.bounds[j][0], self.bounds[j][1])
                else:  # t1, t2 - 偏向较小值
                    self.positions[i, j] = np.random.
exponential(3)
                    self.positions[i, j] = np.clip(self.
positions[i, j], self.bounds[j][0], self.bounds[j][1])

                # 初始化速度
                vel_range = self.bounds[j][1] - self.bounds[j
][0]
                self.velocities[i, j] = np.random.uniform
(-0.05*vel_range, 0.05*vel_range)

            # 计算初始适应度
            fitness = self.objective_func(self.positions[i])
            self.pbest_positions[i] = self.positions[i].copy()
            self.pbest_fitness[i] = fitness

            # 更新全局最优
            if fitness > self.gbest_fitness:
                self.gbest_fitness = fitness
                self.gbest_position = self.positions[i].copy()

    def _constrain_position(self, position, dim):
        """快速位置约束"""
        min_val, max_val = self.bounds[dim]
        if position < min_val:
            return min_val + 0.01 * (np.random.random() - 0.5)
    # 边界附近小幅随机
```

```
279         elif position > max_val:
280             return max_val + 0.01 * (np.random.random() - 0.5)
281         return position
282 
283     def _constrain_velocity(self, velocity, dim):
284         """约束粒子速度"""
285         min_val, max_val = self.bounds[dim]
286         vel_limit = 0.2 * (max_val - min_val)  # 速度限制为边界范围的20%
287         return np.clip(velocity, -vel_limit, vel_limit)
288 
289     def optimize(self):
290         """执行粒子群优化"""
291         for iter in range(self.max_iter):
292             # 线性减小惯性权重
293             w = self.w_start - (self.w_start - self.w_end) * (iter / self.max_iter)
294 
295             # 并行计算所有粒子的适应度
296             fitness_values = Parallel(n_jobs=n_jobs)(
297                 delayed(self.objective_func)(self.positions[i])
298                 for i in range(self.num_particles)
299             )
300 
301             # 更新粒子
302             for i in range(self.num_particles):
303                 fitness = fitness_values[i]
304 
305                 # 更新个体最优
306                 if fitness > self.pbest_fitness[i]:
307                     self.pbest_fitness[i] = fitness
308                     self.pbest_positions[i] = self.positions[i].copy()
309 
```

```
310                 # 更新全局最优
311                 if fitness > self.gbest_fitness:
312                     self.gbest_fitness = fitness
313                     self.gbest_position = self.positions[i].
    copy()
314 
315                 # 计算新速度
316                 r1 = np.random.random(self.dim)  # 认知随机因
    子
317                 r2 = np.random.random(self.dim)  # 社会随机因
    子
318 
319                 cognitive_component = self.c1 * r1 * (self.
    pbest_positions[i] - self.positions[i])
320                 social_component = self.c2 * r2 * (self.
    gbest_position - self.positions[i])
321                 new_velocity = w * self.velocities[i] +
    cognitive_component + social_component
322 
323                 # 约束速度
324                 for j in range(self.dim):
325                     new_velocity[j] = self._constrain_velocity
    (new_velocity[j], j)
326 
327                 # 更新速度
328                 self.velocities[i] = new_velocity
329 
330                 # 更新位置
331                 new_position = self.positions[i] +
    new_velocity
332 
333                 # 约束位置
334                 for j in range(self.dim):
335                     new_position[j] = self._constrain_position
    (new_position[j], j)
```

```
336
337                 self.positions[i] = new_position
338
339             # 记录历史最优
340             self.gbest_history.append(self.gbest_fitness)
341
342             # 早停检查
343             if max(fitness_values) > self.best_fitness_so_far
    + 1e-6:
344                 self.best_fitness_so_far = max(fitness_values)
345                 self.no_improve_count = 0
346             else:
347                 self.no_improve_count += 1
348
349             if self.no_improve_count >= self.patience:
350                 print(f"早停：连续{self.patience}代无改进，在
    第{iter+1}代停止")
351                 break
352
353             # 打印迭代信息
354             if (iter + 1) % 10 == 0 or iter == 0:
355                 print(f"迭代 {iter+1}/{self.max_iter}, 最优适
    应度: {self.gbest_fitness:.6f}")
356
357         return self.gbest_position, self.gbest_fitness, self.
    gbest_history
358
359
360 # ======== 9. main method ========
361 if __name__ == "__main__":
362     start_time = time.time()
363
364     # 生成目标采样点
365     print("生成圆柱体采样点...")
366     # 减少采样点密度，保持原有逻辑但降低点数
```

```
PTS = cylinder_points(n_theta=180, n_z=31)  # 关键参数调整：减少采样密度
print(f"采样点数量：{len(PTS)}")  # 现在应该减少约10000个点

# 定义优化变量边界
bounds = [
    (0.0, 2 * np.pi),      # theta: 方向角
    (70.0, 140.0),         # v: 无人机速度
    (0.0, 80.0),           # t1: 投放延迟
    (0.0, 25.0)            # t2: 起爆延迟
]

# 定义适应度函数（带参数绑定）
def objective(params):
    return fitness_function(params, PTS)

# 初始化并运行粒子群优化
print("\n启动粒子群优化...")
pso = ParticleSwarmOptimizer(
    objective_func=objective,
    bounds=bounds,
    num_particles=75,       # 粒子数量
    max_iter=110,           # 迭代次数
    c1=1.5,                 # 认知系数
    c2=1.5,                 # 社会系数
    w_start=0.9,            # 初始惯性权重
    w_end=0.4               # 结束惯性权重
)

# 执行优化
best_params, best_fitness, history = pso.optimize()

# 提取最优解
theta_opt, v_opt, t1_opt, t2_opt = best_params
```

```
400
401     # 高精度验证
402     print("\n进行最优解高精度验证...")
403     verify_fitness = fitness_function(best_params, PTS)
404
405     # 计算关键位置参数
406     uav_dir_opt = np.array([np.cos(theta_opt), np.sin(
    theta_opt), 0.0])
407     drop_point_opt = FY1_init + v_opt * t1_opt * uav_dir_opt
408     det_xy_opt = drop_point_opt[:2] + v_opt * t2_opt *
    uav_dir_opt[:2]
409     det_z_opt = drop_point_opt[2] - 0.5 * g * t2_opt**2
410     det_point_opt = np.array([det_xy_opt[0], det_xy_opt[1],
    det_z_opt])
411     t_det_opt = t1_opt + t2_opt
412
413     # 计算耗时
414     end_time = time.time()
415     elapsed_time = end_time - start_time
416
417 # ======== 10. output and display ========
418
419     # 输出结果
420     print("Optimal smoke grenade deployment strategy:")
421     print("\n【待求要素】: \n")
422     print(f"无人机FY1的飞行方向角: {theta_opt:.6f} rad ({np.
    degrees(theta_opt):.2f}°)")
423     print(f"无人机FY1的初始位置: {FY1_init}")
424     print(f"无人机的飞行速度: {v_opt:.4f} m/s")
425
426     print(f"投放延迟时间: {t1_opt:.4f} s")
427
428     print(f"起爆延迟时间: {t2_opt:.4f} s")
429     print(f"投放点坐标: {drop_point_opt.round(4)}")
430     print(f"起爆点坐标: {det_point_opt.round(4)}")
```