烟幕干扰弹最佳投放策略模型的研究与构建

摘　要

烟幕干扰弹是一种通过化学燃烧或爆炸分散形成烟幕或气溶胶云团，在目标前方特定空域形成遮蔽的干扰性武器。本论文旨在深入探讨并优化无人机投放烟幕干扰弹的策略，使得对真目标的有效遮蔽时间尽可能长。

针对问题一，本文首先基于**动力学公式**，建立起无人机、烟幕干扰弹、烟幕和导弹的**运动轨迹模型**。随后，基于导弹的观测等效为从导弹位置指向真目标中心的线段的假设，可以得到**有效遮蔽的判定标准**，即该线段与烟幕的空间分布范围发生相**交**。最后，将问题一的参数代入进行离散化计算，得到有效遮蔽时长约为 **1.40s**。

针对问题二，本文首先简化问题一的运动轨迹模型，得到烟幕中心的位置与无人机速度的**递推关系**。随后，基于问题一所给出的假设，将飞行方向、飞行速度、烟幕干扰弹投放点、烟幕干扰弹起爆点这**四个不确定因素**纳入建模考虑。接着，结合题目中的约束条件，建立了一个以**单枚**烟幕干扰弹对**单个**导弹遮蔽时间最大化为目标的**非线性优化模型**。最后，采用**遗传算法**进行求解，得到能够实现遮蔽时长最大化的最优飞行参数组合，其中最大有效遮蔽时长为 **4.450s**。

针对问题三，本文首先基于问题一的运动轨迹模型，得到**三枚烟幕干扰弹**的运动轨迹模型。随后，基于烟幕弹爆炸后不互相干扰的假设，将烟幕干扰弹生成的烟幕视为一个整体，推导出**烟幕整体**的范围公式。接着，结合问题二的模型，并引入连续投放间隔不超过 1s 的约束条件，建立了一个以**三枚**烟幕干扰弹对**单枚**导弹遮蔽时间最大化为目标的**非线性优化模型**。最后，采用**模拟退火算法**进行求解，得到能够实现遮蔽时长最大化的最优飞行参数组合和三枚导弹的投放策略，并将结果保存到文件 result1.xlsx 中。其中，最大有效遮蔽时长为 **6.300s**。

针对问题四，本文首先基于问题一的运动轨迹模型，得到**三架无人机**的运动轨迹模型。随后，推导出三架无人机投放的烟雾干扰弹的运动轨迹公式，以此推导出**烟幕整体**的范围公式。接着，结合问题二、问题三的模型，建立了一个以**三架**无人机投放的**单枚**烟幕干扰弹对**单枚**导弹遮蔽时间最大化为目标的**非线性优化模型**。最后，采用**遗传算法**进行求解，得到能够实现遮蔽时长最大化的各自飞行参数组合和各自的导弹的投放策略，并将结果保存到文件 result2.xlsx 中。其中，最大有效遮蔽时长为 **12.88s**。

针对问题五，本文首先基于问题一的运动轨迹模型，得到**三枚导弹**的运动轨迹模型。随后，将问题三、问题四与物体的运动轨迹公式相结合，即可得到**多个**无人机投放**多枚**烟雾干扰弹的烟雾范围模型。接着，将该模型与每一枚导弹进行有效遮蔽判定，将各自被干扰的时间相加，以此建立了一个以**多架**无人机投放的**多枚**烟幕干扰弹对**多枚**导弹遮蔽时间最大化为目标的**非线性优化模型**。最后，采用**遗传算法**进行求解，得到能够实现遮蔽时长最大化的各自飞行参数组合和各自的导弹的投放策略，并将结果保存到文件 result3.xlsx 中。其中，最大有效遮蔽时长为 **21.18s**。

关键词：运动轨迹 优化模型 遗传算法　模拟退火算法

一、 问题重述

1.1 问题背景

2025 年 9 月 3 日，是中国人民抗日战争暨世界反法西斯战争胜利 80 周年的重要时刻[1]，当天的阅兵式集中体现了我国强大的军队建设水平和新型作战能力。其中，有不少新旧型干扰性武器，烟幕干扰弹也是其中之一。烟幕干扰弹是通过释放气溶胶微粒改变电磁波传输特性，降低敌方光电探测、瞄准及制导系统效能的军事干扰技术。其核心原理是利用散射红外和激光、烟幕吸收等电磁波，使精确制导武器难以目标或偏离攻击轨道。[2]现已有多种投放方式使烟幕干扰弹精确抛撒至定点，即在抛撒前能精确操纵烟幕干扰弹到达指定位置，并通过定时引信时序技术操纵起爆时间。

1.2 问题提出

现有空地导弹来袭，其飞行速度为 300 m/s，以一个某半径 7 m、高 10 m 的圆柱形固定目标而专门为掩护设置的假目标为飞行方向。以假目标为原点，水平面为 xy 平面，真目标下底面的圆心为 (0,200,0)。现警戒雷达观测到来袭导弹时，3 枚导弹 M1、M2、M3 分别位于 (20000,0,2000)、(19000,600,2100)、(18000,−600,1900)；5 架无人机的名称、位置分别为 FY1(17800,0,1800)、FY2(12000,1400,1400)、FY3(6000,−3000,700)、FY4(11000,2000,1800)、FY5(13000,−2000,1300)。

问题 1：无人机 FY1 可投放 1 枚烟幕干扰弹打扰 M1，若 FY1 以 120 m/s 的速度飞向假目标方向，受领任务 1.5 s 后投放 1 枚烟幕干扰弹， 3.6 s 后起爆。请给出烟幕干扰弹对 M1 的有效遮蔽时长。

问题 2：无人机 FY1 可投放 1 枚烟幕干扰弹干扰 M1，现确定 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，使得遮蔽时间尽可能长。

问题 3：无人机 FY1 可投放 3 枚烟幕干扰弹，实施对 M1 的干扰。请给出烟幕干扰弹的投放策略，使得遮蔽时间尽可能长。

问题 4：若使 FY1、FY2、FY3 等 3 架无人机各投放 1 枚烟幕干扰弹，实施对 M1 的干扰。请给出烟幕干扰弹的投放策略，使得遮蔽时间尽可能长。

问题 5：现有 5 架无人机，每架无人机至多投放 3 枚烟幕干扰弹，实施对 M1、M2、M3 等 3 枚来袭导弹的干扰。请给出烟幕干扰弹的投放策略，使得遮蔽时间尽可能长。

二、 问题分析

2.1 问题一的分析

在问题一中，已知无人机 FY1 的飞行速度、飞行方向、投放时间及烟幕干扰弹的起爆时间等参数，需要求解烟幕干扰弹对导弹 M1 的有效遮蔽时长。首先，根据经典的距离—速度—加速度的动力学公式，可以分别建立无人机、烟幕干扰弹、烟幕和导弹的运动轨迹模型。无人机 FY1 的轨迹可由其初始位置、飞行速度向量及飞行时间确定；结合投放位置、飞行速度向量与起爆时间，可以推导出烟幕干扰弹的飞行轨迹，从而确定其爆炸位置；同时，导弹 M1 的运动轨迹可由其初始位置及速度向量构建；而烟幕在起爆后则按照给定的匀速下沉规律运动，其下沉轨迹亦可通过简单的动

力学公式得到。然后，假设导弹的观测过程可以等效为从导弹位置指向真目标圆柱体中心的一条视线。若该视线与烟幕的空间分布范围发生相交，则认为烟幕对导弹视线产生遮挡，进而判定为有效遮蔽。基于上述假设，可以形式化建立有效遮蔽的判定准则。最后，结合各物体的运动轨迹方程与遮蔽判定条件，以时间步长 0.01 s 进行离散化计算，对导弹与目标连线是否与烟幕范围相交进行逐步判断，从而得到有效遮蔽的时间区间，并进一步累加得到烟幕干扰弹对导弹的总有效遮蔽时长。

2.2 问题二的分析

在问题二中，题目给出了无人机的飞行方向、飞行速度、干扰弹的投放时间以及爆炸时间四个不确定因素，确定这四个因素并使烟幕干扰弹对导弹的总有效遮蔽时长最大化。首先，根据问题一所构建的无人机、导弹、烟幕干扰弹及烟幕的运动轨迹模型，可以清晰地了解到各项物体的运动轨迹。然后，根据四个不确定因素和各类约束因素，结合问题一的模型，建立了一个以烟幕干扰弹对导弹遮蔽时间最大化为目标的非线性优化模型。最后，在预设的飞行角度、速度范围、发射时间和问题时间内，采用遗传算法进行全局搜索与优化，以求得飞行方向、飞行速度、干扰弹的投放时间以及爆炸时间四个不确定因素和能够实现遮蔽时长最大化的最优飞行参数组合。

2.3 问题三的分析

在问题三中，现在利用 FY1 投放 3 枚烟幕干扰弹干扰 M1，分别确定它们的投放策略，并使得烟幕干扰弹对导弹的总有效遮蔽时长最大化。首先，根据问题一和问题二，可以清晰地了解到各项物体的运动轨迹和对于单次投放导弹的优化模型。假设烟幕弹爆炸后不互相干扰，并把烟幕干扰弹生成的烟幕视为一个整体，则可以根据 3 枚烟幕干扰弹的运动轨迹，结合问题一、二的模型，建立了一个以三个烟幕干扰弹对导弹遮蔽时间最大化为目标的非线性优化模型。最后，在预设的飞行角度、速度范围、发射时间、问题时间和发射间隔内，采用模拟退火算法进行全局搜索与优化，以求得三枚导弹的投放策略和能够实现遮蔽时长最大化的最优飞行参数组合。

2.4 问题四的分析

在问题四中，分别通过 FY1、FY2、FY3 各自投放 1 枚烟幕干扰弹，分别确定每一架的投放策略，使得烟幕干扰弹对导弹的总有效遮蔽时长最大化。首先，根据问题一可以清晰地了解到各项问题的运动轨迹，根据问题二可以理解单次投放烟雾干扰弹的优化模型，根据问题三可以得到单架无人机多次投放导弹的优化模型。根据问题一得到多个无人机的运动轨迹公式，以此推导烟幕中心的运动轨迹公式，并将烟幕视为一个整体，结合问题二、问题三的模型，就能建立一个以多架无人机分别投放一枚烟幕干扰弹对导弹遮蔽时间最大化为目标的非线性优化模型。最后，在预设的飞行角度、速度范围、发射时间、问题时间和发射间隔内，采用遗传算法进行全局搜索与优化，以求得多个无人机投放单枚烟雾干扰弹的最佳策略和能够实现遮蔽时长最大化的最优飞行参数组合。

2.5 问题五的分析

在问题五中，分别通过五架至多投放 3 枚烟幕干扰弹，实施对 3 枚来袭导弹的干扰，分别确定每一架的投放策略，使得烟幕干扰弹对导弹的总有效遮蔽时长最大化。首先，根据问题四可以得到多架无人机投放单枚烟雾干扰弹的模型，根据问题三可以得到单架无人机投放多枚烟雾干扰弹的模型，将问题三、问题四与物体的运动轨迹公式相结合，即可得到多架无人机投放多枚烟雾干扰弹的模型。然后将该模型分别与三个导弹进行有效遮蔽判定，并将结果进行合并，即可得到多架无人机投放多枚烟雾以对多个导弹遮蔽时间最大化为目标的优化模型。最后，在预设的飞行角度、速度范围、发射时间、问题时间和发射间隔内，采用遗传算法进行全局搜索与优化，以求得多架无人机投放单枚烟雾干扰弹的最佳策略和能够实现遮蔽时长最大化的最优飞行参数组合。

总的核心思路如下：

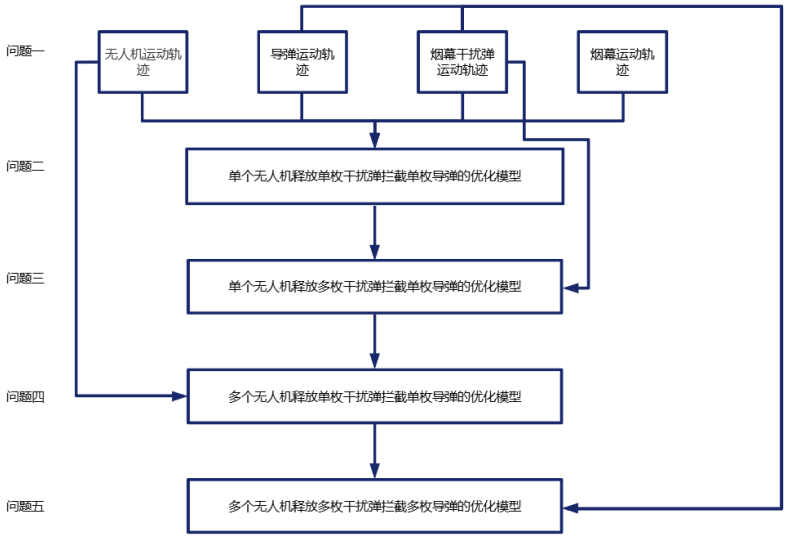

图 1.求解题目核心思路

三、 模型假设

1. **间隔时间：**每架无人机投放两枚烟幕干扰弹至少间隔 1 s。
2. **球体假设：**假设烟幕干扰弹爆炸之后形成一个球体，其半径为 10m。
3. **忽略空气阻力：**假设无人机、导弹、烟幕干扰弹和烟幕的运动不受空气阻力和其他微小力的干扰。
4. **忽略间隔时间：**忽略人为或程序受到命令后发射烟幕干扰弹的间隔时间
5. **正常起爆：**假设烟幕干扰弹正常起爆，烟幕正常对真目标进行遮蔽。
6. **云团独立：**假设每个云团的有效遮挡范围独立。

四、 符号说明

符号	说明	单位
Vm	导弹飞行速度	m/s

V_{fy}	无人机飞行速度	m/s
V_s	烟幕下沉速度	m/s
t	时间	s
g	重力加速度	m/s^2
R_c	烟幕遮蔽半径	m
P_m	导弹位置	m
P_{fy}	无人机位置	m
t_e	烟幕爆炸时间	s
$\vec{P}_t$	真目标的位置	m
t_{last}	飞机投放烟幕干扰弹的时间	s
$t_{explode}$	飞机投放烟幕干扰弹到烟幕干扰弹爆炸的间隔时间	s
$\vec{P}_c(t)$	t 时刻烟幕中心的具体位置	m
$\vec{P}_s(t)$	t 时刻烟幕干扰弹的具体位置	m
T_{all}	有效遮蔽时间	s
$t_{effective}$	烟幕有效时间	s
θ	无人机飞行方向与 x 轴产生的夹角	rad

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 建模思路

已知距离—速度—加速度的动力学公式[3]，由无人机 FY1 的初始位置、飞行速度向量和飞行时间，由烟幕干扰弹的投放位置、起爆时间和飞行速度向量，由导弹 M1 的飞行速度向量和初始位置和烟幕的匀速下沉速度，分别可以确定无人机、烟幕干扰弹、导弹和烟幕的运动轨迹。

假设导弹会向飞行方向形成视野，大面积向外探测，见下图：

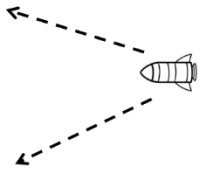

图 2.导弹向外探测

为简化模型，假设无论在何处，其观测视线等效为从导弹位置指向圆柱体中心的一条直线。

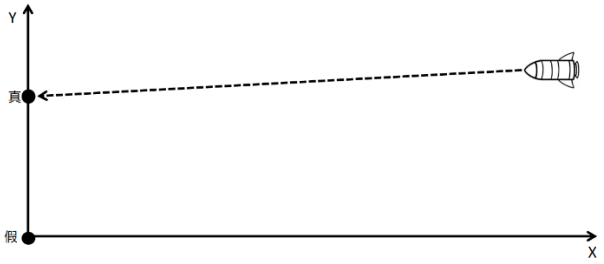

图 3.导弹视线（简化平面图）

当圆柱形的真目标被导弹发现之后，即可认为遮蔽失败，反之，即可认为遮蔽成功。

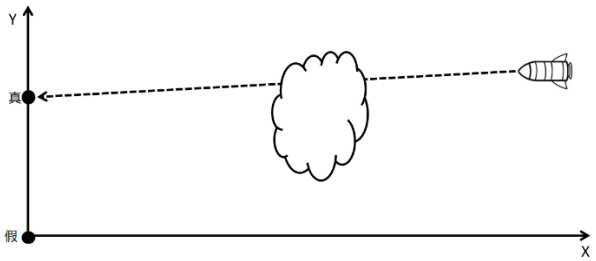

图 4.遮蔽成功（简化平面图）

若导弹与真目标连线在传播路径上与烟幕范围相交，则认为烟幕在几何上对导弹视线造成干扰，从而使导弹无法准确捕获真目标。此时判定为有效遮蔽。

根据以上内容，可获得有效遮蔽公式。代入各值，以时间步长 0.01 s 进行离散化计算，对导弹与真目标连线是否与烟幕范围相交进行逐步判断，从而得到有效遮蔽的时间区间。

5.1.2 各物体的运动公式

➢无人机的运动公式

根据动力学知识，易得以下的距离—速度—加速度公式。

$$\begin{cases} \vec{V}(t) = \overrightarrow{V_0} + \vec{a}t \\ \vec{x}(t) = \overrightarrow{x_0} + \vec{V}_0 t + \frac{1}{2}\vec{a}t^2 \end{cases} \tag{1}$$

将无人机 FY1 的速度向量、位置、加速度代入公式(1)，可得

$$\vec{P}_{fy1}(t) = \vec{P}_{fy1}(0) + \vec{V}_{fy1}t \tag{2}$$

其中，$\vec{P}_{fy1}(0)$为无人机 FY1 初始位置，$\vec{V}_{fy1}$为无人机 FY1 初始速度向量。由于无人机仅会等高度匀速直线飞行且 M1 向着假目标，即原点飞行，所以

$$\vec{V}_{fy1x} = v_{fy1}cos\theta_{fy1}, \vec{V}_{fy1y} = v_{fy1}sin\theta_{fy1} \tag{3}$$

$$tan\theta_{fy1} = \frac{P_{fy1y}}{P_{fy1x}} \tag{4}$$

θ_{fy1}为无人机 FY1 飞行方向与 x 轴产生的夹角，$\vec{V}_{fy1x}$为无人机 FY1 在 x 轴方向上的速度分量，$\vec{V}_{fy1y}$为无人机 FY1 在 y 轴方向上的速度分量，v_{fy1} = 120 m/s.

➢烟幕干扰弹的运动公式

由于惯性，烟幕干扰弹会有一个与发射飞机同样的速度向量，同时烟幕干扰弹受重力的影响，会有一个向下的重力加速度，形成斜抛运动，如下：

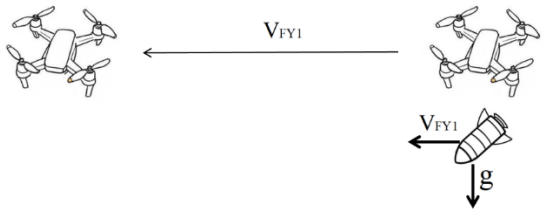

图 5.无人机发射烟幕干扰弹

由烟幕干扰弹的位置、加速度和飞行速度向量代入公式(1)，可得

$$\vec{P}_s(t) = \vec{P}_s(0) + \vec{V}_s(0)t + \frac{1}{2}\vec{g}t^2 \tag{5}$$

其中，$\vec{P}_s(t)$为 t 时刻烟幕干扰弹的具体位置，$\vec{P}_s(0)$为初始时刻烟幕干扰弹的具体位置，$\vec{V}_s(0)$为无人机 FY1 的速度向量，$\vec{g}$为重力加速度。部分量可由飞机的飞行轨迹求出，

$$\vec{V}_s(0) = \vec{V}_{fy1} \tag{6}$$

$$\vec{P}_s(0) = \vec{P}_{fy1}(t_{last}) \tag{7}$$

其中，t_{last}为飞机的投弹时间。

➢导弹的运动公式

由导弹 M1 的飞行速度向量和初始位置代入公式(1)，可得

$$\vec{P}_{m1}(t) = \vec{P}_{m1}(0) + \vec{V}_{m1}t \tag{8}$$

其中，$\vec{P}_{m1}(0)$为导弹 M1 初始位置，$\vec{V}_{m1}$为导弹 M1 初始速度向量。由于导弹向着假目标，即原点飞行，且和假目标在同一 Y 平面内，所以

$$\vec{V}_{m1x} = v_{m1}cos\mu_{m1}, \vec{V}_{m1z} = v_{m1}sin\mu_{m1} \tag{9}$$

$$tan\mu_{m1} = \frac{P_{m1z}}{P_{m1x}} \tag{10}$$

μ_{m1}为导弹飞行方向与 x 轴产生的夹角，$\vec{V}_{m1x}$为无人机在 x 轴方向上的速度分量，$\vec{V}_{m1z}$为无人机在 z 轴方向上的速度分量，v_{m1} = 300 m/s.

➢云团的运动公式

在现实中，爆炸后产生的云团并不会有一个与烟幕干扰弹同样的速度向量。爆炸后情况见下图。

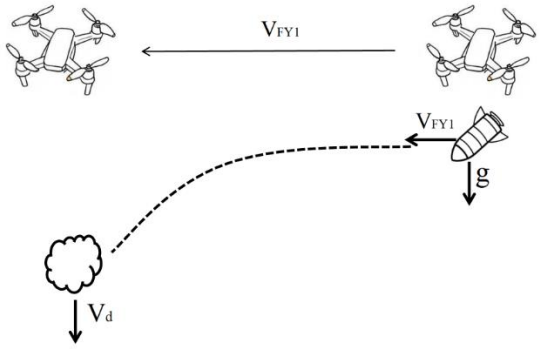

图 6.烟幕干扰弹爆炸

将烟幕的匀速下沉速度代入公式(1)，可得

$$\vec{P}_c(t) = \vec{P}_c(0) + \vec{V}_d t \tag{11}$$

其中，$\vec{P}_c(t)$为 t 时刻烟幕中心的具体位置，$\vec{P}_c(0)$为初始时刻烟幕中心的具体位置

$\vec{V}_c$为烟幕中心的速度向量。

5.1.3 有效遮蔽公式

若导弹与真目标连线在传播路径上与烟幕范围相交，则认为烟幕在几何上对导弹视线造成干扰，从而使导弹无法准确捕获真目标。

$$L(t) = \vec{P}_{m1}(\mathrm{t}) + \lambda\left(\vec{P}_t - \vec{P}_{m1}(t)\right) \quad \lambda \in [0,1] \tag{12}$$

其中，$\vec{P}_t$为真目标的位置。则有效遮蔽条件为

$$\exists\lambda \in [0,1], L(t) \in \Omega(t) \tag{13}$$

假设云团为球体，判定有效遮蔽公式为

$$C(t) = \begin{cases} 1, & \text{若 } L(t) \cap \Omega(t) \neq \varnothing \quad \text{（有效遮蔽）} \\ 0, & \text{若 } L(t) \cap \Omega(t) = \varnothing \quad \text{（遮蔽无效）} \end{cases} \tag{14}$$

$$\Omega(t) = \left\{\vec{x} \in R^3 : \left|\left|\vec{x} - \vec{P}_c(t)\right|\right|\right\} \leq R_c \tag{15}$$

R_c为烟幕遮蔽半径。

结合上面多个公式，即可进行求解。

5.1.4 有效遮蔽公式的求解

这里，我们采用离散化计算，对导弹与目标连线是否与烟幕范围相交进行逐步判断，得到有效遮蔽的时间区间，步骤如下：

先导入各值，并将时间步长 dt 设为 0.01s，在爆炸时间和起爆有效时间内进行多次循环，每次循环 t 加上一个时间步长，如果在该时间步长内，满足导弹与真目标的连线与烟幕范围相交，则记录在数组中，直到超过起爆有效时间。

以伪代码的形式呈现如下：

Algorithm 1 求解有效遮蔽公式的伪代码

初始化: $\mathrm{V_m}, g, \vec{P}_{m1}(0), \vec{P}_{fy1}(0), T_{all}, dt \to 0.01, answer = []$

$for\ t \in (t_{exp}, t_{exp} + 20)$

$\quad if\ L(t) \in \Omega(t)$

$\quad answer.append(t)$

$\quad t = t + dt$

end

最后求解出，烟幕干扰弹对 M1 的有效遮蔽时长约为 1.40s。

5.2 问题二模型的建立与求解

5.2.1 建模思路

从问题一可以得到飞机、导弹等多种物体的运动轨迹公式，再将无人机的飞行方向、飞行速度、干扰弹的投放时间以及爆炸时间四个不确定因素纳入建模考虑，结合题目所给的约束条件加以限定，以此建立一个以烟幕干扰弹对导弹遮蔽时间最大化为目标的非线性优化模型。随后，在预设的飞行角度范围与速度范围内，采用遗传算法进行全局搜索与优化，以求得能够实现遮蔽时长最大化的最优飞行参数组合。

5.2.2 模型的建立

由于无人机仅会等高度匀速直线飞行，为了减小算法量，可以将无人机视为在某个 Z 平面上的二维运动。

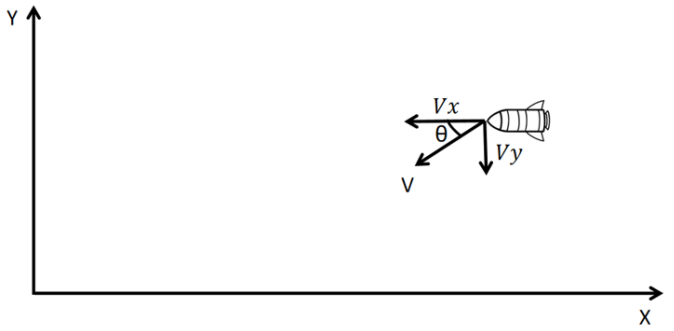

图 7.FY1 在 Z=2000 的平面内运动

设t_{last}为飞机投放烟幕干扰弹的时间，$t_{explode}$为飞机投放烟幕干扰弹到烟幕干扰弹爆炸的间隔时间。

由公式(7)和(2)，可得

$$\vec{P}_s(0) = \vec{P}_{fy1}(0) + \vec{V}_{fy1} t_{last} \tag{16}$$

由公式(5)、(6)和(16)，可得

$$\vec{P}_s(t) = \vec{P}_{fy1}(0) + \vec{V}_{fy1} t_{last} + \vec{V}_{fy1} t + \frac{1}{2}\vec{g}t^2 \tag{17}$$

代入$t_{explode}$

$$\vec{P}_s(t_{explode}) = \vec{P}_{fy1}(0) + \vec{V}_{fy1}t_{last} + \vec{V}_{fy1}t_{explode} + \frac{1}{2}\vec{g}{t_{explode}}^2 \tag{18}$$

由公式(11)和(18)，可得烟雾中心下沉的公式

$$\vec{P}_c(t) = \vec{P}_{fy1}(0) + \vec{V}_{fy1}t_{last} + \vec{V}_{fy1}t_{explode} + \frac{1}{2}\vec{g}{t_{explode}}^2 + \vec{V}_d(t - t_{explode}) \tag{19}$$

由公式(19)和(15)，可得烟雾的范围

$$\Omega(t) = \left\{\overrightarrow{x} \in R^3 : \left\|\vec{x} - [\vec{P}_{fy1}(0) + \vec{V}_{fy1}t_{last} + \vec{V}_{fy1}t_{explode} + \frac{1}{2}\vec{g}{t_{explode}}^2 + \vec{V}_d]\right\|\right\} \le R_c$$

可将公式(14)、(15)优化为

$$d_{seg}(\vec{P}_c(t),\ \vec{P}_{m1}(t),\ \vec{P}_t) = \vec{P}_{m1}(t) + \lambda^*\left(\vec{P}_t - \vec{P}_{m1}(t)\right) - \vec{P}_c(t) \tag{20}$$

此为云团中心点到导弹与真目标的线段的距离公式

$$\lambda^* = clip[0,1]\left(\frac{(\vec{P}_c(t) - \vec{P}_{m1}(t)) \cdot (\vec{P}_t - \vec{P}_{m1}(t))}{\left\|\left(\vec{P}_c(t) - \vec{P}_{m1}(t)\right)\right\|^2}\right) \tag{21}$$

则遮蔽条件等价为

$$C(t) = 1(d_{seg}(\vec{P}_c(t),\ \vec{P}_{m1}(t),\ \vec{P}_t) \le R_c) \tag{22}$$

总有效遮蔽时长即可表示为

$$T_{all} = \int_{t_{explode}}^{t_{explode}+t_{effective}} C(t)dt \tag{23}$$

其中，$t_{effective}$为烟雾有效时间。通过数值为Δt的步长和微分的定义可以近似为

$$T_{all} \approx \sum_{k=0}^{K} C(t_{explode} + k\Delta t)\Delta t,\ \ K = \left\lfloor\frac{t_{effective}}{\Delta t}\right\rfloor \tag{24}$$

其中，T_{all}为有效遮蔽时间。结合题目所给的约束条件，则有以下优化模型为

$$\max\left(\sum_{k=0}^{K} C(t_{explode} + k\Delta t)\Delta t\right)$$
$$s.t.\begin{cases} 0 \le \theta_{fy1} \le 2\pi \\ 70 < v_{fy1} < 140 \\ t_{last} > 0 \\ t_{explode} > 0 \end{cases} \tag{25}$$

关于约束条件的说明：

- 无人机 FY1 飞行与 x 轴产生的夹角θ_{fy1}，易知$\theta_{fy1} \in [0, 2\pi]$
- 无人机以 70~140 m/s 的速度大小等高度匀速直线飞行，故$v_{fy1} \in (70, 140)$

• 无人机从受领任务到投放烟幕干扰弹的时间大于 0s，故$t_{last} > 0$

• 烟幕干扰弹从被投放到爆炸的时间大于 0s，故$t_{explode} > 0$

在实际求解过程中，由于$\vec{V}_{m1} > \vec{V}_{fy1}$，故当导弹的位置比无人机或烟幕的位置更接近于真目标的时候，此时不可能被阻挡，可以进一步对t_{last}和$t_{explode}$进行限制。

5.2.3 遗传算法与模型的求解

遗传算法属于一种通用的算法。其基本的思想借鉴了达尔文进化论和遗传学中的思想，通过模拟生物进化过程来搜索最优解，通过数学的方式，利用计算机仿真运算，将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异和自然选择等过程，目标是让种群不断进化，最终逐步接近问题的最优解。[4]本模型的遗传算法伪代码如下：

Algorithm 2 遗传算法的伪代码

初始化:随机生成 population, best_score → 0,
设定 Gens, Pop_size, Mutation_rate
for gen = 1 → GENS do
    计算每个个体的适应度 fitness
    记录当前代最大适应度 max_fitness 及对应个体 best_solution
    选择操作：根据适应度保留优秀个体 selected
    交叉操作：对 selected 中的个体两两交叉生成新个体
    变异操作：根据 Mutation_rate 对部分个体进行随机变异
    population ← 新一代个体
end for

种群数量，迭代代数和变异概率这三个值的大小，对于遗传算法至关重要，这决定了算法是否能够避免在求解过程中陷入局部最优解的情况，根据多次调试、模拟和测试，最终选定种群数量为 40, 迭代次数为 100，变异概率为 20%。

代入控制参数、关键参数与约束条件，将上述算法运行 10 次，取其中最常见偏差最小的解，可以得到问题二最大遮蔽时间的收敛曲线图如下

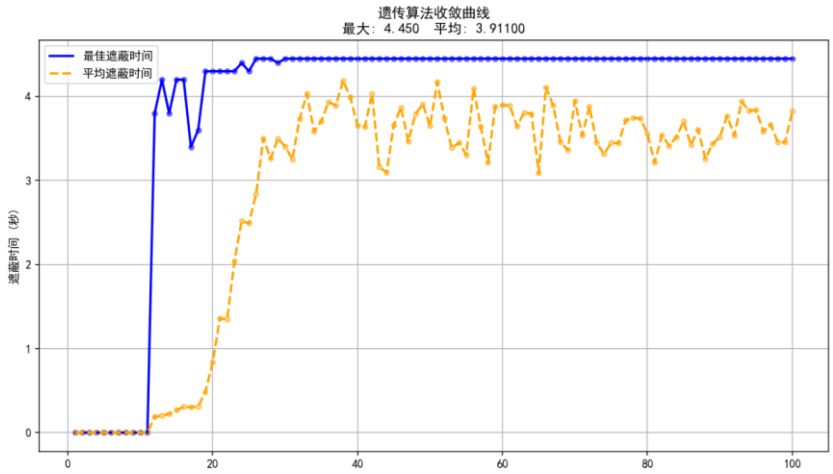

图 8.问题二遗传算法收敛曲线[5]

由此，我们可以得到在所求得的最优方案中，烟幕干扰弹对 M1 的有效遮蔽时长最大为 4.450s。

5.3 问题三模型的建立与求解

5.3.1 建模思路

从问题一可以得到飞机、导弹等多种物体的运动轨迹公式，从问题二可以得到单个无人机投放单枚烟幕干扰弹的优化模型，问题三需要无人机 FY1 投放 3 枚烟幕干扰弹，使得遮蔽时间尽可能长。已知，每架无人机投放两枚烟幕干扰弹至少间隔 1 s。由问题一，可以得到 3 枚烟幕干扰弹分别的运动公式。假设烟幕之间相互独立，如下图

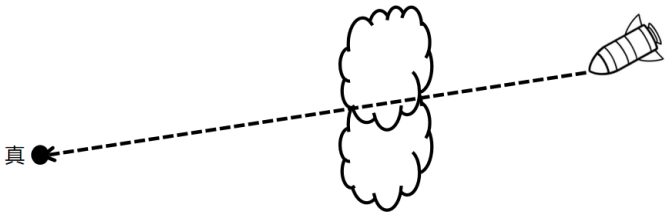

图 9.烟幕相互独立（平面简化图）

通过 3 枚烟雾干扰弹分别的运动公式和问题二的优化模型，可以分别得到 3 枚烟幕干扰弹的优化模型，合并相加即可得到问题三的优化模型。

但在实际情况中，烟幕可能会重叠一起遮蔽烟幕干扰弹，所以将 3 枚烟幕干扰弹释放出的烟幕视为一个整体，真目标中心与该导弹的连线和该整体相交，即可认为有效遮蔽，以此建立优化模型。最后，在预设的飞行角度范围与速度范围，投放间隔时长等约束条件内，采用模拟退火算法进行全局搜索与优化，以求得能够实现遮蔽时长最大化的最优飞行参数组合和最佳投放策略。

5.3.2 模型的建立

由公式(19)，可得到多个烟雾的运行轨迹公式

$$\vec{P}_{ci}(t) = \vec{P}_{fy1}(0) + \vec{V}_{fy1}t_{lasti} + \vec{V}_{fy1}t_{explodei} + \frac{1}{2}\vec{g}{t_{explodei}}^2 + \vec{V}_d(t - t_{explodei}) \quad (26)$$

其中，$\vec{P}_{ci}(t)$为第 i 枚投下的烟雾干扰弹运动轨迹，t_{lasti}分别为无人机从受领任务后到投下第 i 枚的时间，$t_{explodei}$分别为无人机投下第 i 枚后到引爆各自的时间。

由公式(26)和公式(15)，可得多个烟雾的范围公式

$$\Omega(t) = \bigcup_{i=1}^{3} \Omega_i(t) \quad (27)$$

$$\Omega_i(t) = \left\{\vec{x} \in R^3 : \left|\left|\vec{x} - \vec{P}_{ci}(t)\right|\right|\right\} \leq R_c$$

其中$\Omega_i(t)$为第 i 枚烟雾干扰弹形成的烟雾范围。

公式(20)和公式(21)，也可以改为适用第 i 个烟雾干扰弹的公式。

$$d_{seg}^{(i)}\left(\vec{P}_{ci}(t),\ \vec{P}_{m1}(t),\ \vec{P}_t\right) = \left|\left|\vec{P}_{m1}(t) + \lambda_i^*\left(\vec{P}_t - \vec{P}_{m1}(t)\right) - \vec{P}_{ci}(t)\right|\right| \tag{28}$$

$$\lambda_i^* = clip[0,1]\left(\frac{(\vec{P}_{ci}(t) - \vec{P}_{m1}(t)) \cdot (\vec{P}_t - \vec{P}_{m1}(t))}{\left\|\left(\vec{P}_c(t) - \vec{P}_{m1}(t)\right)\right\|^2}\right) \tag{29}$$

由遮蔽判据公式(22)可得到新的遮蔽判据公式，

$$C(t) = 1\left(\min_i d_{seg}^{(i)}\left(\vec{P}_{ci}(t),\ \vec{P}_{m1}(t),\ \vec{P}_t\right) \leq R_c\right) \tag{30}$$

由总有效遮蔽时长公式(23)可得到新的总有效遮蔽时长

$$T_{all} = \int C(t)dt \approx \sum_k C(t_k)\Delta t \quad t_k \in \bigcup_i [t_{explodei}, t_{explodei} + t_{effective}] \tag{31}$$

结合题目所给的约束条件，则有以下优化模型为

$$\max\left(\sum_k C(t_k)\Delta t\right) \ t_k \in \bigcup_i [t_{explodei}, t_{explodei} + t_{effective}]$$

$$s.t. \begin{cases} 0 \leq \theta_{fy1} \leq 2\pi \\ 70 < v_{fy1} < 140 \\ 0 < t_{lasti} \\ 0 < t_{explodei} \\ \forall\, i \in [1,2,3], t_{explodei+1} - t_{explodei} \geq 1 \\ \forall\, i \in [1,2,3], t_{lasti+1} - t_{lasti} \geq 1 \end{cases} \tag{32}$$

关于约束条件的说明：

- 由无人机 FY1 的飞行方向与 x 轴产生的夹角θ_{fy1}，易知$\theta_{fy1} \in [0, 2\pi]$
- 无人机 FY1 以 70~140 m/s 的速度等高度匀速直线飞行，故$v_{fy1} \in (70, 140)$
- 无人机从受领任务到投放烟幕干扰弹的时间大于 0s，故$t_{last} > 0$
- 烟幕干扰弹从被投放到爆炸的时间大于 0s，故$t_{explode} > 0$
- 每架无人机投放两枚烟幕干扰弹至少间隔 1 s，故两枚导弹被投放的时间差 $t_{lasti+1} - t_{lasti} \geq 1$
- 每架无人机投放两枚烟幕干扰弹至少间隔 1 s，故两枚导弹投放后到爆炸前的时间间隔差 $t_{explodei+1} - t_{explodei} \geq 1$

5.3.3 模拟退火算法和模型的求解

模拟退火算法（SA）属于一种通用的随机搜索算法。其基本的思想来源于固体的退火过程，一种基于概率的算法：将固体加温至充分高，再让其冷却。[6]加温时，固体内部粒子随温升变为无序状态，而徐徐冷却时粒子渐渐有序。在每个温度都达到平衡态，最后在常温通过达到稳定态，内能逐渐最小。以优化问题建模为物理退火过程的模拟性为基础，适当的控制温度的下降过程，便可模拟退火从而达到全局优化的目的。本模型的模拟退火算法的基本流程如下：

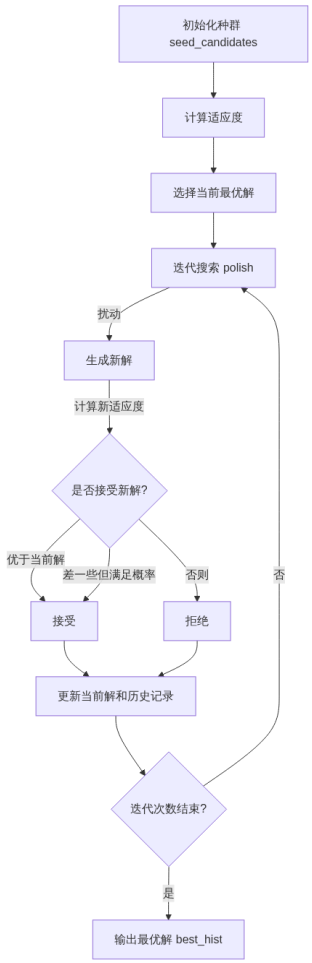

图 10.模拟退火算法的基本流程[5]

在 polish 步骤中集中使用了模拟退火算法。

模拟退火算法的关键在于在每一次退火循环时，能够在初始可行解的领域内产生一个新的可行解并依照相应的规则选择接受或拒绝这个新解。借此思想，我们首先通过人为构造的方式提出一个可行解 X_0作为算法开始搜索的一个初始值。在之后进行算法迭代的时候则每次取之前求解所得的最优解作为初始的可行解。

代入控制参数、关键参数与约束条件，将上述算法运行 10 次，取其中最常见偏差最小的解，可以得到最大遮蔽时间的收敛曲线图如下

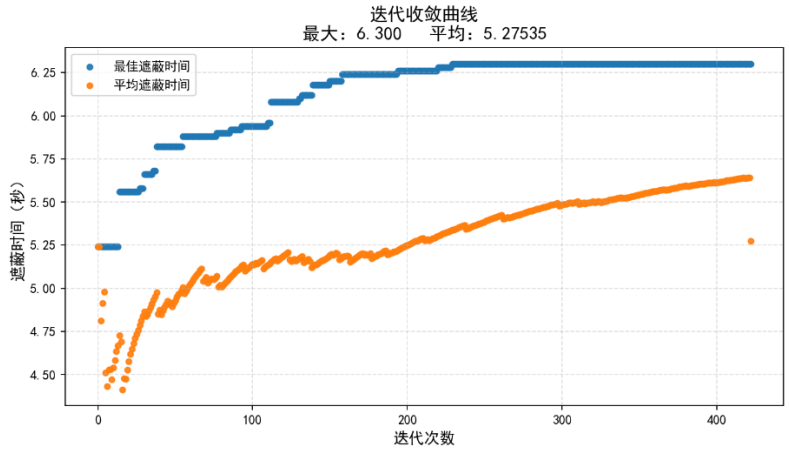

图 11. 模拟退火算法收敛曲线

由此，我们可以得到在所求得的最优方案中，烟幕干扰弹对 M1 的有效遮蔽时长最大为 6.300s。其中每一个烟幕干扰弹的投放策略展示如下

表 1 问题三最优策略中的烟幕干扰弹的投放策略

编号	1	2	3
无人机运动方向	179.41	179.41	179.41
无人机运动速度	120.12	120.12	120.12
烟幕干扰弹投放点的 x 坐标	17800	17800	17800
烟幕干扰弹投放点的 y 坐标	0	4.12	8.76
烟幕干扰弹投放点的 z 坐标	1800	1800	1800
烟幕干扰弹起爆点的 x 坐标	17392.78	16833.58	16411.78
烟幕干扰弹起爆点的 y 坐标	4.19	9.93	14.27
烟幕干扰弹起爆点的 z 坐标	1743.67	1691.44	1702.36
有效干扰时长	3.8	2.58	0

5.4 问题四模型的建立与求解

5.4.1 建模思路

从问题一可以得到飞机、导弹等多种物体的运动轨迹公式，从问题二可以得到单个无人机投放单枚烟幕干扰弹的最大化遮蔽时间优化模型，从问题三可以得到单个无人机投放大化遮蔽时间多枚烟幕干扰弹的优化模型。问题四给出了 3 架无人机 FY1、FY2、FY3，它们分别投放一枚烟幕干扰弹，使得遮蔽时间尽可能长。题目已给 3 架无人机的坐标，建立空间直角坐标系，有以下图

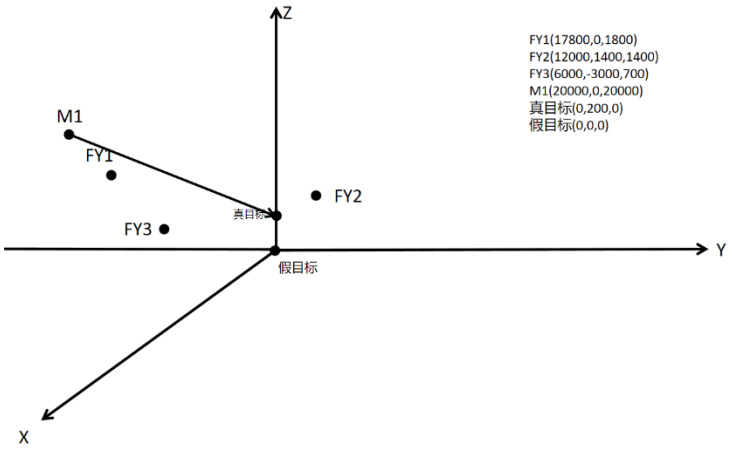

图 12.三维空间内问题四各物体的分布

由问题一，可以得到 3 架无人机分别的运动公式，根据运动公式可以推导出 3 枚烟雾干扰弹分别的运动公式，结合问题二和问题三的优化模型，将 3 架无人机分别投放烟幕干扰弹释放出的烟幕视为一个整体，真目标中心与该导弹的连线和该整体相交，即可认为有效遮蔽，以此建立优化模型。最后，在预设的飞行角度范围与速度范围等约束条件内，采用遗传算法进行全局搜索与优化，以求得能够实现遮蔽时长最大化的 3 架无人机各自的最优飞行参数组合。

5. 4. 2 模型的建立

由公式(26)，可得到 j 号无人机分别投出的烟幕干扰弹的运行轨迹公式

$$\vec{P}_{c_{i,j}}(t) = \vec{P}_{fyj}(0) + \vec{V}_{fyj} t_{last_{i,j}} + \vec{V}_{fy} t_{explode_{i,j}} + \frac{1}{2}\vec{g} t_{explode_{i,j}}{}^{2} + \vec{V}_d (t - t_{explode_{i,j}}) \tag{33}$$

其中，$\vec{P}_{ci}(t)$为第 i 枚投下的烟雾干扰弹运动轨迹，t_{lasti}分别为无人机从受领任务后到投下第 i 枚的时间，$t_{explodei}$分别为无人机投下第 i 枚后到引爆各自的时间。

由公式(33)和公式(15)，可得多个烟雾的范围公式

$$\Omega(t) = \bigcup_{i=1}^{1} \bigcup_{j=1}^{3} \Omega_{i,j}(t) = \left\{ \vec{x} \in R^3 : \left\| \vec{x} - \vec{P}_{c_{i,j}}(t) \right\| \right\} \leq R_c \tag{34}$$

公式(28)和公式(29)，也可以改为适用第 i 枚烟雾干扰弹的公式。

$$d_{seg}^{(i,j)}\left(\vec{P}_{c_{i,j}}(t),\ \vec{P}_{m1}(t),\ \vec{P}_t\right) = \left| \left| \vec{P}_{m1}(t) + \lambda_{i,j}^{*}\left(\vec{P}_t - \vec{P}_{m1}(t)\right) - \vec{P}_{c_{i,j}}(t) \right| \right| \tag{35}$$

$$\lambda_{i,j}^{*} = clip[0,1]\left(\frac{(\vec{P}_{c_{i,j}}(t) - \vec{P}_{m1}(t)) \cdot (\vec{P}_t - \vec{P}_{m1}(t))}{\left\| \left(\vec{P}_{c_{i,j}}(t) - \vec{P}_{m1}(t)\right) \right\|^{2}}\right) \tag{36}$$

由遮蔽判据公式(30)可得到新的遮蔽判据公式，

$$C(t) = 1\left(\min_i \min_j d_{seg}^{(i,j)}\left(\vec{P}_{c_{i,j}}(t),\ \vec{P}_{m1}(t),\ \vec{P}_t\right) \leq R_c\right) \tag{37}$$

由总有效遮蔽时长公式(23)可得到新的总有效遮蔽时长

$$T_{all} = \int C(t)dt \approx \sum_k C(t_k)\Delta t \quad t_k \in \bigcup_i \bigcup_j [t_{explode_{ij}}, t_{explode_{ij}} + t_{effective}] \tag{38}$$

结合题目所给的约束条件，则有以下优化模型为

$$\max\left(\sum_k C(t_h)\Delta t\right) \; t_h \in \bigcup_i [t_{explode_{ij}}, t_{explode_{ij}} + t_{effective}]$$

$$s.t.\begin{cases} 0 \le \theta_j \le 2\pi \\ 70 < v_{fyj} < 140 \\ t_{last_{i,j}} > 0 \\ t_{explode_{i,j}} > 0 \\ j \in [1,2,3] \end{cases} \tag{39}$$

关于约束条件的说明：

- 由第 j 架无人机飞行方向与 x 轴产生的夹角θ_j，易知$\theta_j \in [0, 2\pi]$
- 每架无人机以 70~140 m/s 的速度等高度匀速直线飞行，故$v_{fyj} \in (70, 140)$
- 无人机从受领任务到投放烟幕干扰弹的时间大于 0s，故$t_{last_{i,j}} > 0$
- 烟幕干扰弹从被投放到爆炸的时间大于 0s，故$t_{explode_{i,j}} > 0$
- 每架无人机投放两枚烟幕干扰弹至少间隔 1 s，故两枚导弹被投放的时间差 $t_{explode_{i+1,j}} - t_{explode_{i,j}} \ge 1$
- 每架无人机投放两枚烟幕干扰弹至少间隔 1 s，故两枚导弹投放后到爆炸前的时间间隔差$t_{t_{last_{i+1,j}}} - t_{last_{i,j}} \ge 1$

5.4.3 模型的求解

遗传算法最大的优点是通用性与全局搜索能力，它不需要目标函数可导，也不需要明确的数学模型，只需设计合适的编码和适应度函数，就可以高效探索复杂解空间，由于它能在解空间的多个区域进行探索，采用选择、交叉、变异等操作跳出局部最优，减少陷入局部最优解的风险，特别适合解决搜索空间大、目标函数复杂、局部极值多的问题。因此，对于问题四这种有三个主体变量，每个主体变量还有四个不确定因素的多变量高维度复杂性问题，我们采用了遗传算法。遗传算法的具体操作详见 Algorithm 2。

种群数量，迭代代数和变异概率这三个值的大小，对于遗传算法至关重要，根据多次调试、模拟和测试，最终选定种群数量为 120, 迭代次数为 60，变异概率为 20%。

代入控制参数、关键参数与约束条件，将上述算法运行数十次，取其中最常见偏差最小的解，可以得到问题四最大遮蔽时间的收敛曲线图如下

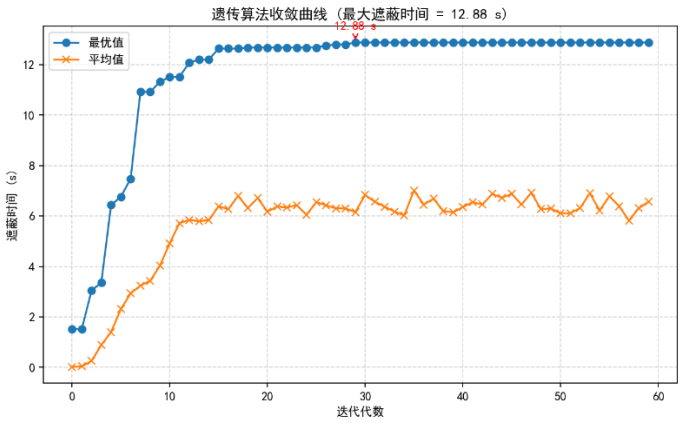

图 13.问题四遗传算法收敛曲线[5]

由此，我们可以得到在所求得的最优方案中，烟幕干扰弹对 M1 的有效遮蔽时长最大为 12.88s。

其中每一个无人机的投放策略展示如下

表 2 问题四最优策略中的烟幕干扰弹的投放策略

无人机编号	FY1	FY2	FY3
无人机 运动方向	179.41	277.66	86.02
无人机 运动速度	126.67	124.32	133.33
烟幕干扰弹投放点的 x 坐标	17544.75	12084.18	6184.27
烟幕干扰弹投放点的 y 坐标	1.99	773.73	-349.22
烟幕干扰弹投放点的 z 坐标	1800	1400	700
烟幕干扰弹起爆点的 x 坐标	16991.81	12182.27	6214.02
烟幕干扰弹起爆点的 y 坐标	6.29	44.01	78.68
烟幕干扰弹起爆点的 z 坐标	1706.63	1228.13	649.29
有效 干扰时长	3.73	4.76	4.34

5.5 问题五模型的建立与求解

5.5.1 建模思路

从问题一可以得到飞机、导弹等多种物体的运动轨迹公式，从问题二可以得到单个无人机投放单枚烟幕干扰弹的最大化遮蔽时间优化模型，从问题三可以得到单个无人机投放大化遮蔽时间多枚烟幕干扰弹的优化模型，从问题四可以得到多个无人机投放单枚烟幕干扰弹的最大化遮蔽时间优化模型。问题五要求使用 5 架无人机，每架无

人机至多投放 3 枚烟幕干扰弹，实施对 M1、M2、M3 等 3 枚来袭导弹的干扰，使得对 M1、M2、M3 等 3 枚来袭导弹遮蔽时间尽可能长。题目已给 5 架无人机的坐标，建立空间直角坐标系，有以下图

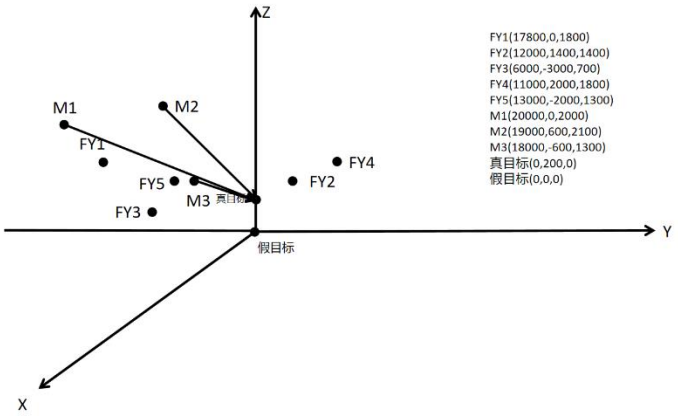

图 14.三维空间内问题五各物体的分布

由问题一，可以得到 5 架无人机分别的运动公式，根据运动公式可以推导出 15 枚烟雾干扰弹分别的运动公式，结合问题二、问题三和问题四的优化模型，将 5 架无人机分别投放的烟幕干扰弹释放出的烟幕视为一个整体。分别判定真目标中心与 M1、M2、M3 导弹的连线是否和该整体相交，如果相交，即可认为有效遮蔽，以此建立优化模型。最后，在预设的飞行角度范围与速度范围等约束条件内，采用遗传算法进行全局搜索与优化，以求得能够实现遮蔽时长最大化的 3 架无人机各自的最优飞行参数组合。

5. 5. 2 模型的建立

由公式(33)可知 15 枚烟雾干扰弹分别的运动公式

$$\vec{P}_{c_{i,j}}(t)=\vec{P}_{fyj}(0)+\vec{V}_{fyj}t_{last_{i,j}}+\vec{V}_{fy}t_{explode_{i,j}}+\frac{1}{2}\vec{g}{t_{explode}}_{i,j}^{\ 2}+\vec{V}_d(t-t_{explode_{i,j}}) \tag{40}$$

由公式(40)和公式(35)，可得多个烟雾的范围公式

$$\Omega(t)=\bigcup_{i=1}^{3}\bigcup_{j=1}^{5}\Omega_{i,j}(t)=\left\{\vec{x}\in R^3:\left\|\vec{x}-\vec{P}_{c_{i,j}}(t)\right\|\right\}\le R_c \tag{41}$$

由公式(8)可得

$$\vec{P}_{mk}(t)=\vec{P}_{mk}(0)+\vec{V}_{mk}t \tag{42}$$

其中，$\vec{P}_{mk}(0)$为导弹 MK 初始位置，$\vec{V}_{mk}$为导弹 MK 初始速度向量。由于导弹向着假目标，即原点飞行，所以由公式(9)(10)扩展到三维空间有

$$\begin{gathered}\vec{V}_{mk}=v_{mk}\hat{u}_{mk}=(cos\mu_k cos\varphi_k,cos\mu_k sin\varphi_k,sin\mu_k)\\ \hat{u}_{mk}=(cos\mu_k cos\varphi_k,cos\mu_k sin\varphi_k,sin\mu_k)\end{gathered} \tag{43}$$

其中φ_k为第 K 枚导弹的方位角，即在xy平面内，第 K 枚导弹的速度向量与从$+x$轴逆时针量形成的夹角，μ_k为第 K 枚导弹的仰角，即为相对xy平面，向上为正，第 K 枚导弹的速度向量与速度向量在xy平面上的投影所形成的夹角。

因此

$$\vec{V}_{mkx} = v_{mk}cos\mu cos\varphi$$

$$\vec{V}_{mky} = v_{mk}cos\mu sin\varphi \tag{44}$$

$$\vec{V}_{mkz} = v_{mk}sin\mu$$

导弹朝着原点飞行意味着$\hat{u}_{mk}$取向量$-\vec{P}_{mk}(0)$。由此有

$$sin\mu_k = \frac{P_{mkz}}{\sqrt{P_{mkx}^2 + P_{mky}^2 + P_{mkz}^2}} \quad cos\mu_k = \frac{\sqrt{P_{mkx}^2 + P_{mkz}^2}}{\sqrt{P_{mkx}^2 + P_{mky}^2 + P_{mkz}^2}} \tag{45}$$

$$tan\varphi = \frac{P_{mky}}{P_{mkx}} \tag{46}$$

根据公式(35)、公式(36)和公式(42)，扩展烟雾中心与各个导弹和真目标分别连线形成所的线段的距离公式

$$d_{seg}^{(i,j,k)}\left(\vec{P}_{c_{i,j}}(t),\ \vec{P}_{mk}(t),\ \vec{P}_t\right) = \left|\left|\vec{P}_{mk}(t) + \lambda_{i,j,k}^{*}\left(\vec{P}_t - \vec{P}_{mk}(t)\right) - \vec{P}_{c_{i,j}}(t)\right|\right| \tag{47}$$

$$\lambda_{i,j,k}^{*} = clip[0,1]\left(\frac{(\vec{P}_{c_{i,j}}(t) - \vec{P}_{mk}(t)) \cdot (\vec{P}_t - \vec{P}_{mk}(t))}{\left\|\left(\vec{P}_{c_{i,j}}(t) - \vec{P}_{mk}(t)\right)\right\|^2}\right) \tag{48}$$

由遮蔽判据公式(37)可得到新的遮蔽判据公式，

$$C(t) = 1(\min_i \min_j \min_k d_{seg}^{(i,j,k)}\left(\vec{P}_{c_{i,j}}(t),\ \vec{P}_{mk}(t),\ \vec{P}_t\right) \le R_c) \tag{49}$$

由总有效遮蔽时长公式(38)可得到新的总有效遮蔽时长

$$T_{all} = \int C(t)dt \approx \sum_h C(t_h)\Delta t \quad t_h \in \bigcup_i \bigcup_j [t_{explode_{ij}}, t_{explode_{ij}} + t_{effective}] \tag{50}$$

结合题目所给的约束条件，则有以下优化模型为

$$\max\left(\sum_k C(t_h)\Delta t\right) \ t_h \in \bigcup_i [t_{explode_{ij}}, t_{explode_{ij}} + t_{effective}]$$

$$s.t.\begin{cases} 0 \le \theta_j \le 2\pi \\ 70 < v_{fyj} < 140 \\ t_{last_{i,j}} > 0 \\ t_{explode_{i,j}} > 0 \\ \forall\, i \in [1,2,3,4,5], t_{explode_{i+1,j}} - t_{explode_{i,j}} \ge 1 \\ \forall\, i \in [1,2,3,4,5], t_{t_{last_{i+1,j}}} - t_{last_{i,j}} \ge 1 \\ j \in [1,2,3] \\ k \in [1,2,3] \end{cases} \tag{51}$$

5.5.3 模型的求解

问题五有多个主体变量，每个主体变量还有四个不确定因素，且还需要对多个目标进行检验的多变量高维度复杂性问题，借由前几个问题的经验，我们采用了遗传算法。遗传算法的具体操作详见 Algorithm 2。

种群数量，迭代代数和变异概率这三个值的大小，对于遗传算法至关重要，根据多次调试、模拟和测试，最终选定种群数量为 220，迭代次数为 100，变异概率为 12%。

代入控制参数、关键参数与约束条件，将上述算法运行数十次，取其中最常见偏差最小的解，可以得到问题五最大遮蔽时间的收敛曲线图如下

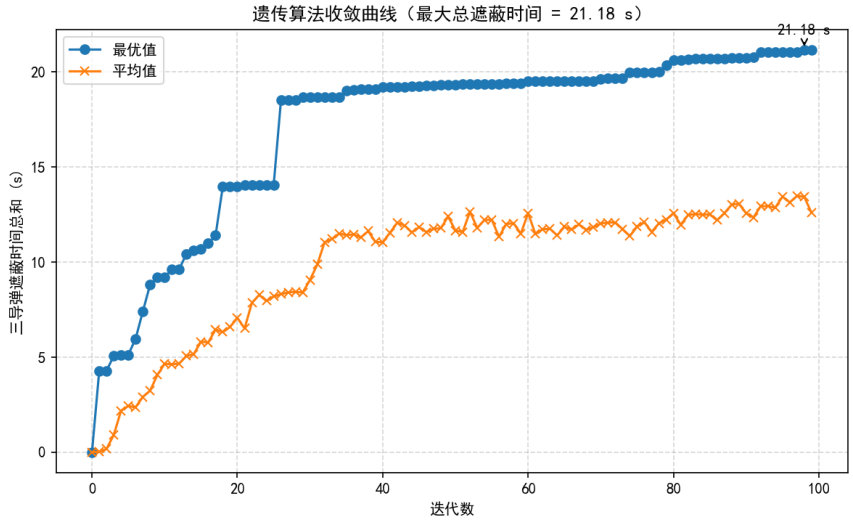

图 15. 问题五遗传算法收敛曲线[5]

由此，我们可以得到在所求得的最优方案中，烟幕干扰弹对三枚导弹的有效遮蔽时长最大为 21.88s。

其中每一个无人机的投放策略展示如下

表 3 问题五最优策略中的烟幕干扰弹的投放策略（以 FY1 为例，详见附录）

无人机	FY1	FY1	FY1
无人机运动方向	179.21	179.21	179.21
无人机运动速度	136.05	136.05	136.05
烟幕干扰弹投放点的 x 坐标	17753.56	17470.69	17175.87
烟幕干扰弹投放点的 y 坐标	0.64	4.51	8.55
烟幕干扰弹投放点的 z 坐标	1800	1800	1800
烟幕干扰弹起爆点的 x 坐标	17251.23	17058.51	16743.47
烟幕干扰弹起爆点的 y 坐标	7.52	10.16	14.48
烟幕干扰弹起爆点的 z 坐标	1733.19	1755.02	1750.49

有效 干扰时长	4.58	0	0
烟幕干扰弹编号	1	2	3
干扰的导弹编号	1	1	2

六、 模型的分析与检验

6.1 基于动态调参的灵敏度分析

为验证非线性优化模型的正确性，本节针对问题二建立的非线性优化模型进行灵敏度分析。

在烟幕干扰弹的投放策略中存在大量环境变量，包括烟幕的半径R_s、云团的下沉速度大小v_s、导弹的飞行速度大小v_m等等，它们的初值为$R_s = 10m$，v_s=3m/s，$v_m = 300m/s$。令这些量上下连续波动 5%，利用 python 软件绘制者三个参数分别与有效遮蔽时间之间的示意图，详见如下。

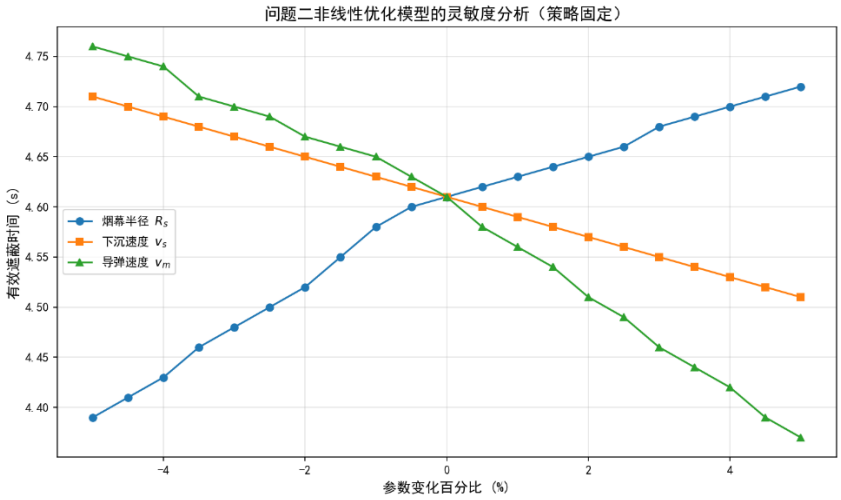

图 16.问题二的灵敏度分析[5]

图 16 中，问题二的有效遮蔽时间对烟幕的半径R_s、云团的下沉速度大小v_s、导弹的飞行速度大小v_m这三个参数的变化都较为敏感，其中导弹的速度会对有效遮蔽时间造成较大影响。因此在实际项目中应该额外注意它们的取值情况。

七、 模型的评价、改进与推广

7.1 模型的优点

1) 模型统一，通用性强

五个问题使用的都是统一的基础模型，并且逐步扩展模型复杂度。从单架无人机单枚烟幕干扰弹到多架无人机、多导弹、多次投放。[5]

2) 考虑了实际物理约束

模型中考虑了导弹、无人机、烟幕干扰弹和烟幕的动力学特性，保证了模型与真实情况的切合度。[5]

3) 优化合理，结果可靠
模型中假设导弹具有视野，并以导弹与真目标连成的线段是否在烟幕范围内作为判定为有效遮蔽的条件，减小了求解成本，同时与实际紧密联系，能得到全局最优解，结果可靠。

4) 模型简单易懂，方法灵活，具有较强的推广性

7.2 模型的缺点

1) 忽略空气阻力和风场影响
在实际环境中风速变化会显著影响烟幕扩散和位置，会导致模型结果与真实情况产生偏差。[5]

2) 缺乏多导弹协同攻击情况下的适应性
模型假设导弹独立飞行且无规避策略，未考虑导弹在受干扰后可能采取的机动调整行为。[5]

7.3 模型的改进

1) 引入空气动力学模型
在烟幕干扰弹及云团运动中加入空气阻力和风场模型，例如流体力学方程，使计算结果更加符合实际。[5]
2) 引入对抗博弈模型
考虑导弹具备智能规避与路径调整功能，将问题扩展为无人机与导弹之间的博弈优化模型，从单向遮蔽问题转化为双向对抗优化问题。[5]

7.4 模型的推广

该模型适用性强，应用范围广，不仅可以扩展到舰船、装甲车辆等多目标防御作战，还可以应用于民用领域的无人机群体协同控制，如农业喷洒作业中的路径规划与药剂覆盖优化。[5]

八、 参考文献

[1]本报记者.见证强军路共谱新篇章[N].福建日报,2025-09-06(005).
[2]吴鸿超,梁增友,邓德志,等.烟幕干扰弹新型战斗部结构研究[J].科学技术与工程,2016,16(01):183-187.
[3]王学建.牛顿第二定律的基本特性及应用[J].科技信息,2010,(28):554+556.
[4]马永杰,云文霞.遗传算法研究进展[J].计算机应用研究,2012,29(04):1201-1206+1210.
[5] ChatGPT, ChatGPT-5, OpenAI, 2025-09-06 2025-09-07.
[6]肖晓伟,肖迪,林锦国,等.多目标优化问题的研究概述[J].计算机应用研究,2011,28(03):805-808+827.

附录

附录 1

介绍：支撑材料的文件列表

文件名	文件内容
AI 工具使用详情.pdf	介绍本题使用 AI 工具的详细情况
problem1.py	问题一的代码解答
problem2.py	问题二的代码解答
problem2 anlysis.py	问题二模型的灵敏度分析
problem3.py	问题三的代码解答
problem4.py	问题四的代码解答
problem5.py	问题五的代码解答
result1.xlsx	结果表一
result2.xlsx	结果表二
result3.xlsx	结果表三

附录 2

表 3 问题五最优策略中的烟幕干扰弹的投放策略

表 3 问题五最优策略中的烟幕干扰弹的投放策略

无人机	FY1	FY1	FY1
无人机 运动方向	179.21	179.21	179.21
无人机 运动速度	136.05	136.05	136.05
烟幕干扰弹投放点 的 x 坐标	17753.56	17470.69	17175.87
烟幕干扰弹投放点 的 y 坐标	0.64	4.51	8.55
烟幕干扰弹投放点 的 z 坐标	1800	1800	1800
烟幕干扰弹起爆点 的 x 坐标	17251.23	17058.51	16743.47
烟幕干扰弹起爆点 的 y 坐标	7.52	10.16	14.48
烟幕干扰弹起爆点 的 z 坐标	1733.19	1755.02	1750.49
有效 干扰时长	4.58	0	0
烟幕干扰弹编号	1	2	3
干扰的导弹编号	1	1	2

无人机编号	FY2	FY2	FY2
无人机 运动方向	268.8	268.8	268.8
无人机 运动速度	140	140	140
烟幕干扰弹投放点 的 x 坐标	11991.1	11988.16	11982.18
烟幕干扰弹投放点 的 y 坐标	976.15	836.18	551.61
烟幕干扰弹投放点 的 z 坐标	1400	1400	1400
烟幕干扰弹起爆点 的 x 坐标	11979.6	11971.14	11964.54
烟幕干扰弹起爆点 的 y 坐标	428.62	25.89	-288.2
烟幕干扰弹起爆点 的 z 坐标	1325.02	1235.79	1223.6
有效	3.64	3.68	0

干扰时长			
烟幕干扰弹编号	1	2	3
干扰的导弹编号	2	1	1

无人机编号	FY3	FY3	FY3
无人机 运动方向	87.31	87.31	87.31
无人机 运动速度	140	140	140
烟幕干扰弹投放点 的 x 坐标	6101.43	6114.61	6121.17
烟幕干扰弹投放点 的 y 坐标	-838.52	-557.71	-417.87
烟幕干扰弹投放点 的 z 坐标	700	700	700
烟幕干扰弹起爆点 的 x 坐标	6130.86	6135.29	6134.3
烟幕干扰弹起爆点 的 y 坐标	-211.49	-116.96	-138.17
烟幕干扰弹起爆点 的 z 坐标	601.49	651.33	680.4
有效 干扰时长	0	4.35	3.26
烟幕干扰弹编号	1	2	3
干扰的导弹编号	3	3	3

无人机编号	FY4	FY4	FY5
无人机 运动方向	235.23	235.23	105.6
无人机 运动速度	140	140	127.35
烟幕干扰弹投放点 的 x 坐标	9408.51	9049.56	12518.25
烟幕干扰弹投放点 的 y 坐标	-292.73	-809.85	-274.37
烟幕干扰弹投放点 的 z 坐标	1800	1800	1300
烟幕干扰弹起爆点 的 x 坐标	8929.52	8570.57	12439.39
烟幕干扰弹起爆点 的 y 坐标	-982.78	-1499.9	8.08

烟幕干扰弹起爆点的 z 坐标	1623.6	1623.6	1274.02
有效干扰时长	0	0	2.22
烟幕干扰弹编号	1	2	1
干扰的导弹编号	1	3	1

附录 3

这段代码是用 Python 编写，作用是求解问题一烟幕干扰弹对 M1 的有效遮蔽时长。

```
import numpy as np

g = 9.8                          #重力加速度
sink_v = 3.0                     #烟幕下沉速度
radius = 10.0                    #有效遮蔽半径
effective_time = 20.0          #有效遮蔽时长

M0 = np.array([20000.0, 0.0, 2000.0])     #导弹  M1  初始位置
F0 = np.array([17800.0, 0.0, 1800.0])     #无人机  FY1  初始位置
T    = np.array([0.0, 200.0, 0.0])          #真实目标位置

u_m = -M0/np.linalg.norm(M0)     #单位方向向量
v_m = 300.0                              #导弹速

def M(t):
"""导弹在时间 t 的位置"""
return M0+v_m*t*u_m

u_f = np.array([-1.0,0.0,0.0])   #无人机朝向
v_f = 120.0                      #无人机速度
t_drop = 1.5                     #投放时间
fuse = 3.6                      #起爆延时

P_drop = F0+v_f*t_drop*u_f
C_exp = P_drop+v_f*fuse*u_f+np.array([0.0,0.0,-0.5*g*fuse**2])

def C(t):
"""烟幕中心随时间下沉"""
return C_exp + np.array([0.0,0.0,-sink_v*(t-(t_drop+fuse))])

def dist_to_segment(P,A,B):
AB = B- A
AP = P- A
t = np.dot(AP,AB)/np.dot(AB,AB)
```

```
t = max(0,min(1, t))
Q = A+t*AB
return np.linalg.norm(P-Q)

t_exp = t_drop + fuse
dt = 0.01
times = np.arange(t_exp,t_exp+effective_time,dt)
inside = []

for t in times:
if dist_to_segment(C(t),M(t),T)<=radius:
inside.append(t)
cover_time = inside[-1]-inside[0] if inside else 0
print("有效遮蔽时间: ",cover_time,"秒")
```

附录 4

这段代码是用 Python 编写，作用是求解问题二确定 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，使得遮蔽时间尽可能长。

```
import numpy as np
import random
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

g = 9.8                                  #重力加速度
sink_v = 3.0                            #烟幕下沉速度
radius = 10.0                            #烟幕有效遮蔽半径
effective_time = 20.0                  #烟幕有效时间

M0 = np.array([20000.0,0.0,2000.0])       #导弹初始位置
F0 = np.array([17800.0,0.0,1800.0])       #无人机初始位置
TARGET = np.array([0.0,200.0,0.0])        #真目标位置

v_m = 300.0
u_m = -M0/np.linalg.norm(M0)       #导弹飞向目标的单位方向向量

def M(t):
```

```
    """导弹在时间 t 的位置"""
    return M0+v_m*t*u_m

def dist_to_segment(P, A, B):
    """计算点 P 到线段 AB 的最短距离"""
    AB = B-A
    AP = P-A
    t = np.dot(AP,AB)/np.dot(AB,AB)
    t = max(0, min(1,t))                          #限制在[0,1]区间
    Q = A+t*AB
    return np.linalg.norm(P-Q)

def compute_cover(v_f,theta,t_drop,fuse):
    """
    给定四个策略参数，计算遮蔽时间
    v_f    : 无人机飞行速度
    theta : 无人机飞行方向（平面角度）
    t_drop: 投放时间
    fuse   : 烟幕弹引信延时
    """

    u_f = np.array([np.cos(theta),np.sin(theta),0.0])

    P_drop = F0+v_f*t_drop*u_f

    C_exp = P_drop+v_f*fuse*u_f+np.array([0.0,0.0,-0.5*g*fuse**2])

    t_exp = t_drop+fuse
    dt = 0.05
    times = np.arange(t_exp,t_exp+effective_time,dt)
    inside = []

    for t in times:

        C_t = C_exp + np.array([0.0,0.0,-sink_v*(t-t_exp)])

        if dist_to_segment(C_t, M(t), TARGET) <= radius:
            inside.append(t)

    return inside[-1] - inside[0] if inside else 0.0

POP_SIZE = 40                  #种群数量
GENS = 100                     #迭代代数
MUTATION_RATE = 0.2            #变异概率
```

```
V_RANGE = (70.0, 140.0)
THETA_RANGE = (0, 2 * np.pi)
DROP_RANGE = (0, 20.0)
FUSE_RANGE = (2.0, 6.0)

def random_individual():
    return [
        random.uniform(*V_RANGE),
        random.uniform(*THETA_RANGE),
        random.uniform(*DROP_RANGE),
        random.uniform(*FUSE_RANGE)
    ]

def mutate(ind):
    i = random.randint(0,3)
    if i == 0:
        ind[0] = random.uniform(*V_RANGE)
    elif i == 1:
        ind[1] = random.uniform(*THETA_RANGE)
    elif i == 2:
        ind[2] = random.uniform(*DROP_RANGE)
    else:
        ind[3] = random.uniform(*FUSE_RANGE)
    return ind

def crossover(p1,p2):
    point = random.randint(1,3)
    return p1[:point]+p2[point:],p2[:point]+p1[point:]

population = [random_individual() for _ in range(POP_SIZE)]

best_score = -1
best_solution = None

best_scores = []
avg_scores = []

for gen in range(GENS):

    fitness = [compute_cover(*ind) for ind in population]

    max_f = max(fitness)
    avg_f = sum(fitness)/len(fitness)
```

```
    best_scores.append(max_f)
    avg_scores.append(avg_f)

    if max_f > best_score:
        best_score = max_f
        best_solution = population[fitness.index(max_f)]

    print(f"第{gen+1}代 最佳遮蔽时间: {max_f:.2f} 秒")

    selected = []
    for _ in range(POP_SIZE):
        i, j = random.sample(range(POP_SIZE),2)
        selected.append(population[i] if fitness[i] > fitness[j] else population[j])

    next_gen = []
    for i in range(0, POP_SIZE,2):
        c1, c2 = crossover(selected[i], selected[i+1])
        if random.random() < MUTATION_RATE:
            c1 = mutate(c1)
        if random.random() < MUTATION_RATE:
            c2 = mutate(c2)
        next_gen.extend([c1, c2])
    population = next_gen

if best_solution is not None:
    print("\n 最优解:")
    print(f"1. 无人机飞行速度 v_f = {best_solution[0]:.2f} m/s")
    print(f"2. 无人机航向角 theta = {np.degrees(best_solution[1]):.2f} 度")
    print(f"3. 烟幕投放时间 t_drop = {best_solution[2]:.2f} s")
    print(f"4. 烟幕引信延时 fuse = {best_solution[3]:.2f} s")
    print(f"最大遮蔽时间 = {best_score:.2f} 秒")
else:
    print("未找到有效解，请检查模型或参数设置。")

plt.figure(figsize=(10, 6))
plt.scatter(range(1, GENS + 1),best_scores,color='blue',s=20,label='最佳遮蔽时间')
plt.scatter(range(1, GENS + 1),avg_scores,color='orange',s=20,label='平均遮蔽时间')

plt.xlabel("迭代次数", fontsize=12)
plt.ylabel("遮蔽时间 (秒)", fontsize=12)
plt.title(f" 遗 传 算 法 收 敛 曲 线 \n 最 大 : {max(best_scores):.3f}  平 均 : {np.mean(best_scores):.5f}",fontsize=14)
plt.legend()
```

```
plt.grid(True)
plt.show()
```

附录 5

这段代码是用 Python 编写，作用是求解问题三烟幕干扰弹的投放策略，并将结果保存到文件 result1.xlsx 中。

```
import numpy as np
import random
import pandas as pd
from pathlib import Path
import time

g = 9.8
sink_v = 3.0
radius = 10.0
effective_time = 20.0

M0 = np.array([20000.0,0.0,2000.0])       # 导弹初始
F0 = np.array([17800.0,0.0,1800.0])       # 无人机初始
TARGET = np.array([0.0,200.0,0.0])        # 真目标

v_m = 300.0
u_m = -M0 / np.linalg.norm(M0)
t_hit_fake = np.linalg.norm(M0)/v_m

THETA_RANGE = (np.pi-0.55, np.pi+0.25)
V_RANGE = (70.0,140.0)
T_RANGE = (0.0,20.0)
F_RANGE = (2.0,6.0)

def missile_pos_vec(t):
    t = np.asarray(t)
    return M0 + (v_m*t)[:,None]*u_m

def dist_point_to_segment_vec(P,A,B):
    AB = B[None,:]-A
    AP = P-A
    denom = np.einsum('ij,ij->i',AB,AB)
```

```
    denom = np.where(denom < 1e-12,1e-12,denom)
    tau = np.einsum('ij,ij->i', AP,AB)/denom
    tau = np.clip(tau,0.0,1.0)
    Q = A + tau[:,None]*AB
    return np.linalg.norm(P-Q,axis=1)

def one_cloud_mask(C_exp,t_exp,t_grid):
    dz = -sink_v * np.maximum(0.0,t_grid-t_exp)
    C_t = C_exp[None, :] + np.stack([np.zeros_like(t_grid),
                                     np.zeros_like(t_grid),
                                     dz], axis=1)
    valid = (t_grid >= t_exp)&(t_grid <= t_exp+effective_time)
    A = missile_pos_vec(t_grid)
    d = dist_point_to_segment_vec(C_t,A,TARGET)
    return valid & (d <= radius)

def drop_and_explode(F0,v_f, theta,t_drop,fuse):
    u_f = np.array([np.cos(theta),np.sin(theta),0.0])
    P_drop = F0+v_f*t_drop*u_f
    C_exp   = P_drop+v_f*fuse*u_f+np.array([0.0,0.0,-0.5*g*fuse**2])
    return P_drop, C_exp, (t_drop+fuse)

def evaluate_union(plan,dt=0.02):
    """
    返回：union_time（并集遮蔽时长）、per_times(三枚各自时长)、drops(明细)
    """
    theta, v_f, t1, f1, t2, f2, t3, f3 = plan

    # 硬约束
    if not (THETA_RANGE[0] <= theta <= THETA_RANGE[1]): return 0.0, [0,0,0],
None
    if not (V_RANGE[0] <= v_f <= V_RANGE[1]):              return 0.0, [0,0,0], None
    if t1 < 0.0 or t2 < 1.0 or t3 < 1.0:                  return 0.0, [0,0,0], None
     if not (F_RANGE[0] <= f1 <= F_RANGE[1] and F_RANGE[0] <= f2 <=
F_RANGE[1] and F_RANGE[0] <= f3 <= F_RANGE[1]):
          return 0.0, [0,0,0], None

    P1, C1, Te1 = drop_and_explode(F0,v_f,theta,t1,f1)
    P2, C2, Te2 = drop_and_explode(F0,v_f,theta,t1+t2,f2)
    P3, C3, Te3 = drop_and_explode(F0,v_f,theta,t1+t2+t3,f3)

    drops = [
        {"i":1,"P":P1,"C":C1,"t_drop":t1,"t_exp":Te1,"fuse":f1},
        {"i":2,"P":P2,"C":C2,"t_drop":t1+t2,"t_exp":Te2,"fuse":f2},
```

```
            {"i":3,"P":P3,"C":C3,"t_drop":t1+t2+t3,"t_exp":Te3,"fuse":f3},
        ]

        t0 = min(Te1, Te2, Te3)
        t_end = min(t_hit_fake, max(Te1, Te2, Te3) + effective_time)
        if t_end <= t0: return 0.0, [0,0,0], drops

        t_grid = np.arange(t0,t_end+1e-9, dt)
        m1 = one_cloud_mask(C1,Te1,t_grid)
        m2 = one_cloud_mask(C2,Te2,t_grid)
        m3 = one_cloud_mask(C3,Te3,t_grid)

        per = [float(m1.sum()*dt), float(m2.sum()*dt), float(m3.sum()*dt)]
        union_time = float(( m1 | m2 | m3 ).sum() * dt)
        return union_time, per, drops

def rand_plan():
        return [
            random.uniform(*THETA_RANGE),
            random.uniform(*V_RANGE),
            random.uniform(*T_RANGE),
            random.uniform(*F_RANGE),
            1.0 + random.uniform(0.0,T_RANGE[1]),
            random.uniform(*F_RANGE),
            1.0 + random.uniform(0.0,T_RANGE[1]),
            random.uniform(*F_RANGE),
        ]

def seed_candidates(n=260):
        seeds=[]
        for _ in range(n):
            theta = np.random.normal(loc=np.pi-0.12,scale=0.18)
            theta = float(np.clip(theta, *THETA_RANGE))
            v_f   = float(np.clip(np.random.normal(118,16), *V_RANGE))
            t1    = random.uniform(0.10,6.0)
            f1    = random.uniform(2.3,4.9)
            t2    = 1.0 + random.uniform(0.8,9.0)
            f2    = random.uniform(2.3,4.9)
            t3    = 1.0 + random.uniform(0.8,9.0)
            f3    = random.uniform(2.3,4.9)
            seeds.append([theta, v_f,t1,f1,t2,f2,t3,f3])
        return seeds

def polish(base_plan,iters=420,dt=0.02):
```

```
"""
局部随机坐标扰动 + 接受改进（只要并集更大就接受）
退火缩步：每 80 步缩 0.6 倍
记录：best_hist（到目前为止的最佳并集）、avg_hist（累计平均并集）
"""
best = np.array(base_plan,dtype=float)
best_union,best_per,best_drops = evaluate_union(best, dt=dt)

best_hist = [best_union]
cum_values = [best_union]
avg_hist   = [np.mean(cum_values)]

step = np.array([0.12,4.0,0.8,0.45,0.9,0.45,0.9,0.45])
for k in range(iters):
    cand = best.copy()
    j = np.random.randint(0, 8)
    cand[j] += np.random.uniform(-1, 1) * step[j]
    # 边界
    cand[0] = float(np.clip(cand[0],*THETA_RANGE))
    cand[1] = float(np.clip(cand[1],*V_RANGE))
    cand[2] = float(np.clip(cand[2],*T_RANGE))
    cand[3] = float(np.clip(cand[3],*F_RANGE))
    cand[4] = float(max(1.0, min(cand[4],T_RANGE[1]+1.0)))
    cand[5] = float(np.clip(cand[5],*F_RANGE))
    cand[6] = float(max(1.0, min(cand[6], T_RANGE[1]+1.0)))
    cand[7] = float(np.clip(cand[7],*F_RANGE))

    union_c, per_c, drops_c = evaluate_union(cand, dt=dt)
    cum_values.append(union_c)
    avg_hist.append(float(np.mean(cum_values)))

    if union_c > best_union:
        best, best_union, best_per, best_drops = cand, union_c, per_c, drops_c
    best_hist.append(best_union)

    if (k+1) % 80 == 0:
        step *= 0.6

union_c, per_c, drops_c = evaluate_union(best,dt=0.01)
cum_values.append(union_c)
avg_hist.append(float(np.mean(cum_values)))
if union_c > best_union:
    best_union, best_per, best_drops = union_c,per_c,drops_c
best_hist.append(best_union)
```

```
    return best.tolist(),best_union,best_per,best_drops,best_hist,avg_hist

def plot_convergence(best_hist,avg_hist):
    import matplotlib.pyplot as plt
    import matplotlib
     matplotlib.rcParams['font.sans-serif'] = ['SimHei','Microsoft YaHei','Arial Unicode
MS']
    matplotlib.rcParams['axes.unicode_minus'] = False

    x = np.arange(len(best_hist))
    plt.figure(figsize=(8.6, 5.0),dpi=110)
    plt.scatter(x,best_hist, s=18,label="最佳遮蔽时间",alpha=0.9)
    plt.scatter(x,avg_hist,   s=18,label="平均遮蔽时间",alpha=0.9)

    title = f"迭代收敛曲线\n 最大：{max(best_hist):.3f}     平均：{avg_hist[-1]:.5f}"
    plt.title(title,fontsize=14)
    plt.xlabel("迭代次数",fontsize=12)
    plt.ylabel("遮蔽时间（秒）",fontsize=12)
    plt.grid(True, linestyle="--",alpha=0.4)
    plt.legend()
    plt.tight_layout()
    plt.show()

def main():
    seeds = seed_candidates(260)
    scored = []
    for s in seeds:
        u, _, _ = evaluate_union(s,dt=0.04)
        scored.append((u, s))
    scored.sort(key=lambda x: x[0],reverse=True)
    pool = [x[1] for x in scored[:36]]

    best_plan, best_union, best_per, best_drops = None,-1.0,None,None
    best_hist_for_plot, avg_hist_for_plot = None,None
    for p in pool:
        cand, uni, per, dr, bh, ah = polish(p,iters=420,dt=0.02)
        if uni > best_union:
            best_plan, best_union, best_per, best_drops = cand,uni,per,dr
            best_hist_for_plot, avg_hist_for_plot = bh,ah

    final_u, final_per, final_drops = evaluate_union(best_plan,dt=0.01)
    if final_u > best_union:
        best_union, best_per, best_drops = final_u, final_per, final_drops
```

```
    if best_hist_for_plot is not None and avg_hist_for_plot is not None:
        best_hist_for_plot.append(best_union)
        avg_hist_for_plot.append(float(np.mean(avg_hist_for_plot + [best_union])))

    theta, v_f, *_ = best_plan
    deg = float(np.degrees(theta) % 360.0)
    print("\n【最终最优策略（MaxUnion）】")
    print(f"无人机运动方向 = {deg:.2f}°")
    print(f"速度 v_f = {v_f:.2f} m/s")
    print(f"单枚遮蔽时间：{[round(x,2) for x in best_per]} s")
    print(f"三枚烟幕弹总遮蔽时间（并集） = {best_union:.2f} s")

    cols = [
        "无人机运动方向",
        "无人机运动速度(m/s)",
        "烟幕干扰弹编号",
        "烟幕干扰弹投放点的 x 坐标(m)",
        "烟幕干扰弹投放点的 y 坐标(m)",
        "烟幕干扰弹投放点的 z 坐标(m)",
        "烟幕干扰弹起爆点的 x 坐标(m)",
        "烟幕干扰弹起爆点的 y 坐标(m)",
        "烟幕干扰弹起爆点的 z 坐标(m)",
        "有效干扰时长(s)"
    ]

    rows = []
    for d, per_t in zip(best_drops,best_per):
        rows.append({
            "无人机运动方向": round(deg,2),
            "无人机运动速度(m/s)": round(v_f,2),
            "烟幕干扰弹编号": d["i"],
            "烟幕干扰弹投放点的 x 坐标(m)": round(d["P"][0],2),
            "烟幕干扰弹投放点的 y 坐标(m)": round(d["P"][1],2),
            "烟幕干扰弹投放点的 z 坐标(m)": round(d["P"][2],2),
            "烟幕干扰弹起爆点的 x 坐标(m)": round(d["C"][0],2),
            "烟幕干扰弹起爆点的 y 坐标(m)": round(d["C"][1],2),
            "烟幕干扰弹起爆点的 z 坐标(m)": round(d["C"][2],2),
            "有效干扰时长(s)": round(per_t,2)
        })
    df_out = pd.DataFrame(rows,columns=cols)

    note = "注：以 x 轴为正向，逆时针方向为正，取值 0~360（度）。"
    note_row = {c: "" for c in cols}
    note_row[cols[0]] = note
```

```
    df_out = pd.concat([df_out, pd.DataFrame([note_row])],ignore_index=True)

    save_template_only(df_out,"result1.xlsx")

    if best_hist_for_plot is not None and avg_hist_for_plot is not None:
        plot_convergence(best_hist_for_plot, avg_hist_for_plot)

def save_template_only(df_out, out_path="result1.xlsx"):
    out_path = Path(out_path)
    try:
        with pd.ExcelWriter(out_path,engine="openpyxl",mode="w") as writer:
            df_out.to_excel(writer,sheet_name="结果",index=False)
        print(f"\n 已保存到文件：{out_path.resolve()}")
    except PermissionError:
        ts = time.strftime("%Y%m%d_%H%M%S")
        alt_path = out_path.with_name(f"{out_path.stem}_{ts}{out_path.suffix}")
        with pd.ExcelWriter(alt_path, engine="openpyxl", mode="w") as writer:
            df_out.to_excel(writer, sheet_name="结果", index=False)
        print(f"\n 原文件被占用，改名保存到：{alt_path.resolve()}")

if __name__ == "__main__":
    main()
```

附录 6

这段代码是用 Python 编写，作用是求解问题四——FY1、FY2 和 FY33 架无人机各投放 1 枚烟幕干扰弹的投放策略，并将结果保存到文件 result2.xlsx 中。

```
mport numpy as np
import random
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

g = 9.8
sink_v = 3.0
radius = 10.0
effective_time = 20.0
```

```
M0 = np.array([20000.0,0.0,2000.0])          #导弹初始
TARGET = np.array([0.0,200.0,0.0])             #真目标
FAKE_TARGET = np.array([0.0,0.0,0.0])      #假目标

FY = {
    "FY1": np.array([17800.0,0.0,1800.0]),
    "FY2": np.array([12000.0,1400.0,1400.0]),
    "FY3": np.array([6000.0,-3000.0,700.0]),
}

v_m = 300.0
u_m = (FAKE_TARGET-M0)/np.linalg.norm(FAKE_TARGET-M0)
t_hit_fake = np.linalg.norm(M0-FAKE_TARGET)/v_m

THETA_RANGE = (0.0,2*np.pi)
V_RANGE = (70.0,140.0)
T_RANGE = (0.0,20.0)
F_RANGE = (2.0,6.0)

def missile_pos_vec(t):
    t = np.asarray(t)
    return M0 + (v_m*t)[:, None]*u_m

def dist_point_to_segment_vec(P,A,B):
    AB = B[None, :]-A
    AP = P-A
    denom = np.einsum("ij,ij->i",AB,AB)
    denom = np.where(denom < 1e-12,1e-12,denom)
    tau = np.einsum("ij,ij->i",AP,AB)/denom
    tau = np.clip(tau,0.0,1.0)
    Q = A+tau[:, None]*AB
    return np.linalg.norm(P-Q,axis=1)

def one_cloud_mask(C_exp,t_exp,t_grid):
    dz = -sink_v*np.maximum(0.0,t_grid-t_exp)
    C_t = C_exp[None, :]+np.stack([
        np.zeros_like(t_grid),
        np.zeros_like(t_grid),
        dz
    ], axis=1)
    valid = (t_grid >= t_exp)&(t_grid <= t_exp + effective_time)
    A = missile_pos_vec(t_grid)
    d = dist_point_to_segment_vec(C_t, A, TARGET)
    return valid & (d <= radius)
```

```
def drop_and_explode(F0, v_f, theta, t_drop, fuse):
    u_f = np.array([np.cos(theta), np.sin(theta), 0.0])
    P_drop = F0 + v_f * t_drop * u_f
    C_exp    = P_drop+v_f*fuse*u_f+np.array([0.0,0.0,-0.5*g*fuse**2])
    return P_drop,C_exp, (t_drop+fuse)

def evaluate_union_3(plan12,dt=0.02):
    th1,v1,t1,f1, th2,v2,t2,f2, th3,v3,t3,f3 = plan12
    for v in (v1,v2,v3):
        if not (V_RANGE[0] <= v <= V_RANGE[1]): return 0,[0,0,0],None

    P1,C1,Te1 = drop_and_explode(FY["FY1"], v1,th1,t1,f1)
    P2,C2,Te2 = drop_and_explode(FY["FY2"], v2,th2,t2,f2)
    P3,C3,Te3 = drop_and_explode(FY["FY3"], v3,th3,t3,f3)

    drops = [P1,C1,Te1],[P2,C2,Te2],[P3,C3,Te3]
    t0 = min(Te1,Te2,Te3)
    tend = min(t_hit_fake, max(Te1,Te2,Te3)+effective_time)
    if tend <= t0: return 0,[0,0,0],drops

    t_grid = np.arange(t0, tend+1e-9, dt)
    m1 = one_cloud_mask(C1,Te1,t_grid)
    m2 = one_cloud_mask(C2,Te2,t_grid)
    m3 = one_cloud_mask(C3,Te3,t_grid)
    per=[float(m1.sum()*dt), float(m2.sum()*dt), float(m3.sum()*dt)]
    union=float((m1|m2|m3).sum()*dt)
    return union,per,drops

def backsolve_seed(F0):
    t_star = random.uniform(5.0,25.0)
    A = missile_pos_vec(np.array([t_star]))[0]
    B = TARGET
    AB = B-A
    tau = np.dot(F0-A,AB)/np.dot(AB,AB)
    tau = np.clip(tau,0,1)
    Q = A + tau*AB
    base_theta = np.arctan2(Q[1]-F0[1],Q[0]-F0[0])
    theta = float((base_theta + np.random.normal(0,0.2))%(2*np.pi))
    v = random.uniform(*V_RANGE)
    fuse = random.uniform(*F_RANGE)
    t_drop = max(0.0,t_star-fuse)
    return [theta,v,t_drop,fuse]
```

```
def seed_candidates(n=100):
    seeds=[]
    for _ in range(n):
        vals=[]
        for name in ("FY1","FY2","FY3"):
            vals+=backsolve_seed(FY[name])
        seeds.append(vals)
    return seeds

def crossover(p1,p2):
    point=random.randint(1,len(p1)-2)
    return p1[:point]+p2[point:],p2[:point]+p1[point:]

def mutate(ind,rate=0.2):
    for j in range(len(ind)):
        if random.random()<rate:
            if j%4==0: ind[j]=(ind[j]+random.uniform(-0.3,0.3))%(2*np.pi)
            elif j%4==1: ind[j]=random.uniform(*V_RANGE)
            elif j%4==2: ind[j]=random.uniform(*T_RANGE)
            else: ind[j]=random.uniform(*F_RANGE)
    return ind

def genetic(n_gen=40,pop_size=80):
    pop=seed_candidates(pop_size)
    best=None; best_val=-1
    best_hist=[]; avg_hist=[]
    for g in range(n_gen):
        scored=[(evaluate_union_3(ind,dt=0.04)[0],ind) for ind in pop]
        scored.sort(key=lambda x:x[0],reverse=True)
        vals=[s for s,_ in scored]
        avg_hist.append(np.mean(vals))
        best_hist.append(vals[0])
        if scored[0][0]>best_val:
            best_val,best=scored[0]
        elites=[ind for _,ind in scored[:10]]
        new_pop=elites.copy()
        while len(new_pop)<pop_size:
            p1,p2=random.sample(elites,2)
            c1,c2=crossover(p1.copy(),p2.copy())
            new_pop.append(mutate(c1))
            if len(new_pop)<pop_size: new_pop.append(mutate(c2))
        pop=new_pop
    return best,best_hist,avg_hist
```

```
def plot_convergence(best_hist, avg_hist):
    plt.figure(figsize=(8,5),dpi=110)
    plt.plot(best_hist,label="最优值",marker="o")
    plt.plot(avg_hist,label="平均值",marker="x")
    plt.xlabel("迭代代数")
    plt.ylabel("遮蔽时间 (s)")
    max_val = max(best_hist)
    max_gen = int(np.argmax(best_hist))
    plt.title(f"遗传算法收敛曲线 (最大遮蔽时间 = {max_val:.2f} s)")
    plt.grid(True,ls="--",alpha=0.5)
    plt.legend()

    plt.annotate(f"{max_val:.2f} s",
                 xy=(max_gen, max_val),
                 xytext=(max_gen, max_val+0.5),
                 arrowprops=dict(arrowstyle="->",color="red"),
                 ha="center",color="red")
    plt.tight_layout()
    plt.show()

def main():
    best_plan,best_hist,avg_hist=genetic(n_gen=60,pop_size=120)
    best_union,best_per,best_drops=evaluate_union_3(best_plan,dt=0.01)
    print("\n【问题四 最优策略】")
    for name,(P,C,Te),per in zip(("FY1","FY2","FY3"),best_drops,best_per):
        print(f"{name}: 遮蔽={per:.2f}s 起爆={Te:.2f}")
    print("并集遮蔽时间=",best_union)

    th1,v1,t1,f1, th2,v2,t2,f2, th3,v3,t3,f3 = best_plan
    items = [
        ("FY1",th1,v1,best_drops[0],best_per[0]),
        ("FY2",th2,v2,best_drops[1],best_per[1]),
        ("FY3",th3,v3,best_drops[2],best_per[2]),
    ]

    cols = [
        "无人机编号",
        "无人机运动方向",
        "无人机运动速度(m/s)",
        "烟幕干扰弹投放点的 x 坐标(m)",
        "烟幕干扰弹投放点的 y 坐标(m)",
        "烟幕干扰弹投放点的 z 坐标(m)",
        "烟幕干扰弹起爆点的 x 坐标(m)",
        "烟幕干扰弹起爆点的 y 坐标(m)",
```

```
        "烟幕干扰弹起爆点的 z 坐标(m)",
        "有效干扰时长(s)"
    ]

    out_rows = []
    for name, th, v, (P, C, Te), per in items:
        deg = float(np.degrees(th) % 360.0)
        out_rows.append({
            "无人机编号": name,
            "无人机运动方向": round(deg, 2),
            "无人机运动速度(m/s)": round(v, 2),
            "烟幕干扰弹投放点的 x 坐标(m)": round(P[0],2),
            "烟幕干扰弹投放点的 y 坐标(m)": round(P[1],2),
            "烟幕干扰弹投放点的 z 坐标(m)": round(P[2],2),
            "烟幕干扰弹起爆点的 x 坐标(m)": round(C[0],2),
            "烟幕干扰弹起爆点的 y 坐标(m)": round(C[1],2),
            "烟幕干扰弹起爆点的 z 坐标(m)": round(C[2],2),
            "有效干扰时长(s)": round(per,2)
        })

    df_out = pd.DataFrame(out_rows,columns=cols)
    with pd.ExcelWriter("result2.xlsx",engine="openpyxl",mode="w") as writer:
        df_out.to_excel(writer, sheet_name="结果",index=False)
    print("已保存为 result2.xlsx（格式与截图一致，仅此一张表）")

    plot_convergence(best_hist,avg_hist)

if __name__=="__main__":
    main()
```

附录 7

这段代码是用 Python 编写，作用是求解问题四——5 架无人机，每架无人机至多投放 3 枚烟幕干扰弹，实施对 M1、M2、M3 等 3 枚来袭导弹的干扰，请给出烟幕干扰弹的投放策略，并将结果保存到文件 result3.xlsx 中。

```
import numpy as np
import random
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
```

```
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

g = 9.8
SINK_V = 3.0
R_EFFECT = 10.0
EFFECTIVE_T = 20.0

M_INIT = {
    0: np.array([20000.0,         0.0, 2000.0]),     #M1
    1: np.array([19000.0,      600.0, 2100.0]),     #M2
    2: np.array([18000.0,    -600.0, 1900.0])       #M3
}
FAKE_TARGET = np.array([0.0,0.0,0.0])
TARGET_TRUE = np.array([0.0,200.0,0.0])
V_M = 300.0

FY = {
    "FY1": np.array([17800.0,0.0,1800.0]),
    "FY2": np.array([12000.0,1400.0,1400.0]),
    "FY3": np.array([ 6000.0,-3000.0, 700.0]),
    "FY4": np.array([11000.0,2000.0,1800.0]),
    "FY5": np.array([13000.0,-2000.0,1300.0]),
}

THETA_RANGE    = (0.0,2*np.pi)
V_RANGE = (70.0,140.0)
T_DROP_RANGE = (0.0,20.0)
FUSE_RANGE = (2.0,6.0)
MIN_GAP = 1.0

UAV_LIST = ["FY1","FY2","FY3","FY4","FY5"]
MISS_IDS = [0,1,2]
NUM_SMOKES_PER_UAV = 3

def missile_dir(m_id):
    u = (FAKE_TARGET-M_INIT[m_id])
    return u/np.linalg.norm(u)

def missile_pos_vec(m_id,t):
    t = np.asarray(t)
    return M_INIT[m_id]+(V_M*t)[:,None]*missile_dir(m_id)

def t_hit_fake(m_id):
```

```
    return np.linalg.norm(M_INIT[m_id]-FAKE_TARGET)/V_M

def dist_point_to_segment_vec(P,A,B):
    AB = B-A
    denom = np.einsum('ij,ij->i',AB,AB)
    denom = np.where(denom < 1e-12,1e-12,denom)
    tau = np.einsum('ij,ij->i', (P - A),AB)/denom
    tau = np.clip(tau,0.0,1.0)
    Q = A + tau[:,None]*AB
    return np.linalg.norm(P-Q,axis=1)

def one_cloud_mask(m_id,C_exp,t_exp,t_grid):
    dz = -SINK_V*np.maximum(0.0,t_grid-t_exp)
    C_t = C_exp[None,:]+np.stack([np.zeros_like(t_grid),
                                  np.zeros_like(t_grid),
                                  dz],axis=1)
    valid = (t_grid >= t_exp)&(t_grid <= t_exp+EFFECTIVE_T)
    A = missile_pos_vec(m_id,t_grid)
    d = dist_point_to_segment_vec(C_t,A,TARGET_TRUE)
    return valid & (d <= R_EFFECT)

def drop_and_explode(F0,v_uav,theta,t_drop,fuse):
    u = np.array([np.cos(theta),np.sin(theta),0.0])
    P_drop = F0+v_uav*t_drop*u
    C_exp    = P_drop+v_uav*fuse*u+np.array([0.0,0.0,-0.5*g*fuse**2])
    return P_drop,C_exp,t_drop+fuse

GENE_LEN_PER_UAV = 2+NUM_SMOKES_PER_UAV*4
CHROM_LEN = len(UAV_LIST)*GENE_LEN_PER_UAV

def clip_gene_vec(ch):
    ch = list(ch)
    for i in range(len(UAV_LIST)):
        base = i*GENE_LEN_PER_UAV
        ch[base+0] %= 2*np.pi
        ch[base+1] = float(np.clip(ch[base+1],*V_RANGE))
        for k in range(NUM_SMOKES_PER_UAV):
            o = base + 2 + k*4
            ch[o+0] = 1 if ch[o+0] >= 0.5 else 0
            if k == 0:
                ch[o+1] = float(np.clip(ch[o+1],*T_DROP_RANGE))   # 绝对投放
时刻
            else:
                ch[o+1] = float(max(MIN_GAP,ch[o+1]))                # 相对间隔
```

```
≥1s
                ch[o+2] = float(np.clip(ch[o+2],*FUSE_RANGE))
                ch[o+3] = int(np.clip(round(ch[o+3]),0,2))
    return ch

def evaluate(chrom, dt=0.02):
    chrom = clip_gene_vec(chrom)

    smokes = []
    for i,uav in enumerate(UAV_LIST):
        F0 = FY[uav]
        base = i*GENE_LEN_PER_UAV
        theta, v = chrom[base+0],chrom[base+1]
        last_drop_time = None
        for k in range(NUM_SMOKES_PER_UAV):
            o = base + 2 + k*4
            active, t_or_gap, fuse,mid = chrom[o:o+4]
            if active != 1: continue
            if k == 0 or last_drop_time is None:
                t_drop = float(np.clip(t_or_gap,*T_DROP_RANGE))
            else:
                t_drop = last_drop_time+float(max(MIN_GAP,t_or_gap))
            P_drop, C_exp, t_exp = drop_and_explode(F0,v,theta,t_drop,fuse)
            smokes.append((mid, t_exp,C_exp,P_drop,uav, theta,v,t_drop,fuse))
            last_drop_time = t_drop

    union_sum = 0.0
    per_missile = {m:0.0 for m in MISS_IDS}
    per_smoke_time = []

    for m in MISS_IDS:
        group = [(t_exp,C_exp,P_drop,uav,theta,v,t_drop,fuse)
                 for (mid,t_exp,C_exp,P_drop,uav,theta,v,t_drop,fuse) in smokes if
mid==m]
        if not group: continue
        t0 = min(t for (t,_,_,_,_,_,_,_) in group)
        tend = min(t_hit_fake(m), max(t for (t,_,_,_,_,_,_,_) in group) + EFFECTIVE_T)
        if tend <= t0: continue
        t_grid = np.arange(t0,tend+1e-9,dt)
        masks = []
        for (t_exp,C_exp,P_drop,uav,theta,v,t_drop,fuse) in group:
            msk = one_cloud_mask(m, C_exp, t_exp, t_grid)
            per_smoke_time.append((uav, m, float(msk.sum()*dt), t_exp, P_drop,
C_exp, theta, v, t_drop, fuse))
```

```
                masks.append(msk)
            M = np.zeros_like(masks[0],dtype=bool)
            for msk in masks: M |= msk
            per = float(M.sum()*dt)
            per_missile[m] = per
            union_sum += per

    return union_sum, per_missile, per_smoke_time, chrom

def closest_Q_on_LOS(m_id,F0,t_min=6.0,t_max=35.0):
    ts = np.linspace(t_min,min(t_max, t_hit_fake(m_id)),120)
    A = missile_pos_vec(m_id, ts)
    B = TARGET_TRUE
    AB = B-A
    denom = np.einsum('ij,ij->i',AB,AB)
    denom = np.where(denom < 1e-12,1e-12,denom)
    tau = np.einsum('ij,ij->i',(F0[None,:]-A),AB)/denom
    tau = np.clip(tau,0.0,1.0)
    Q = A + tau[:,None]*AB
    d = np.linalg.norm(Q-F0[None,:],axis=1)
    k = int(np.argmin(d))
    return float(ts[k]),Q[k]

def ballistic_seed_one_smoke(F0,m_id):
    t_star, Q = closest_Q_on_LOS(m_id,F0)
    if F0[2] >= Q[2]:
        fuse = np.sqrt(max(2.0*(F0[2]-Q[2])/g,FUSE_RANGE[0]**2))
    else:
        fuse = FUSE_RANGE[0]
    fuse = float(np.clip(fuse,*FUSE_RANGE))

    Dxy = Q[:2] - F0[:2]
    dist_xy = float(np.linalg.norm(Dxy))
    if dist_xy < 1e-6:
        theta = 0.0; v = V_RANGE[0]
    else:
        theta = float(np.arctan2(Dxy[1],Dxy[0]))
        v = dist_xy / fuse
        v = float(np.clip(v,*V_RANGE))
    t_drop = t_star - fuse
    t_drop = float(np.clip(t_drop,*T_DROP_RANGE))
    return theta % (2*np.pi),v,t_drop,fuse,m_id

def ballistic_aim_seeds_for_uav(uav_name):
```

```
    F0 = FY[uav_name]
    seeds=[]
    mids = random.sample(MISS_IDS,k=len(MISS_IDS))
    for mid in mids:
        seeds.append(ballistic_seed_one_smoke(F0, mid))
    return seeds

def make_chrom_from_ballistic():
    chrom=[]
    for uav in UAV_LIST:
        shoots = ballistic_aim_seeds_for_uav(uav)
        if len(shoots)==0:
            theta = 0.0; v = np.mean(V_RANGE)
        else:
            theta, v = shoots[0][0], shoots[0][1]
        chrom += [theta, float(np.clip(v,*V_RANGE))]
        for k in range(NUM_SMOKES_PER_UAV):
            if k < len(shoots):
                th, vv, t_drop, fuse, mid = shoots[k]
                chrom += [1, t_drop if k==0 else max(MIN_GAP, 0.8),fuse,mid]
            else:
                chrom += [0,T_DROP_RANGE[0],np.mean(FUSE_RANGE),0]
    return clip_gene_vec(chrom)

def seed_population(pop_size=180):
    pop=[]
    for i in range(pop_size):
        ch = make_chrom_from_ballistic()
        if i >= pop_size//2:
            for j in range(CHROM_LEN):
                if random.random() < 0.25:
                    r = j % GENE_LEN_PER_UAV
                    if r==0: ch[j] = (ch[j]+np.random.normal(0,0.25))%(2*np.pi)
                                                                elif r==1: ch[j] =
float(np.clip(ch[j]+np.random.normal(0,6.0),*V_RANGE))
                    elif (r-2)%4==0: ch[j] = 1 if random.random()<0.75 else 0
                    elif (r-2)%4==1:
                                                                        ch[j] =
float(np.clip(ch[j]+np.random.normal(0,1.2),*T_DROP_RANGE)) if (r-2)//4==0 else
max(MIN_GAP, ch[j]+np.random.normal(0,0.8))
                                                            elif (r-2)%4==2: ch[j] =
float(np.clip(ch[j]+np.random.normal(0,0.4),*FUSE_RANGE))
                    else: ch[j] = random.randint(0,2)
            ch = clip_gene_vec(ch)
```

```
            pop.append(ch)
    return pop

def crossover(p1,p2):
    i = random.randint(1, CHROM_LEN-2)
    return p1[:i]+p2[i:], p2[:i]+p1[i:]

def mutate(ind, rate=0.12):
    ind = list(ind)
    for j in range(CHROM_LEN):
        if random.random() < rate:
            r = j % GENE_LEN_PER_UAV
            if r==0: ind[j]=(ind[j]+random.uniform(-0.35,0.35))%(2*np.pi)
                    elif r==1: ind[j]=float(np.clip(ind[j]+np.random.normal(0,5.0),
*V_RANGE))
            elif (r-2)%4==0: ind[j]=1 if random.random()<0.65 else 0
                elif (r-2)%4==1: ind[j]=random.uniform(*T_DROP_RANGE) if (r-
2)//4==0 else max(MIN_GAP, ind[j]+np.random.normal(0,0.9))
                  elif (r-2)%4==2: ind[j]=float(np.clip(ind[j]+np.random.normal(0,0.35),
*FUSE_RANGE))
            else: ind[j]=random.randint(0,2)
    return clip_gene_vec(ind)

def genetic(n_gen=100,pop_size=220):
    pop = seed_population(pop_size)
    best=None; best_val=-1.0
    best_hist=[]; avg_hist=[]
    for g in range(n_gen):
        scored=[]; vals=[]
        for ind in pop:
            val,_,_,_ = evaluate(ind,dt=0.03)
            scored.append((val,ind)); vals.append(val)
        scored.sort(key=lambda x:x[0], reverse=True)
        best_hist.append(scored[0][0]); avg_hist.append(float(np.mean(vals)))
        if scored[0][0] > best_val: best_val, best = scored[0]
        elites=[ind for _,ind in scored[:max(12,pop_size//10)]]
        new_pop=elites.copy()
        while len(new_pop) < pop_size:
            p1,p2=random.sample(elites,2)
            c1,c2=crossover(p1,p2)
            new_pop.append(mutate(c1))
            if len(new_pop)<pop_size: new_pop.append(mutate(c2))
        pop=new_pop
    return best, best_hist, avg_hist
```

```
def polish(plan, iters=600, dt=0.02):
    best = np.array(plan, dtype=float)
    best_val, _, _, _ = evaluate(best, dt=dt)
    step = np.full(CHROM_LEN, 0.0)
    for i in range(len(UAV_LIST)):
        base = i*GENE_LEN_PER_UAV
        step[base+0]=0.15; step[base+1]=3.5
        for k in range(NUM_SMOKES_PER_UAV):
            o=base+2+k*4
            step[o+0]=0.0;step[o+1]=0.8 if k>0 else 0.6;step[o+2]=0.25; step[o+3]=0.0
    for t in range(iters):
        cand=best.copy()
        j = np.random.randint(0,CHROM_LEN)
        r = j % GENE_LEN_PER_UAV
        delta = (np.random.uniform(-1,1)*step[j])
        if r==0: cand[j]=(cand[j]+delta)%(2*np.pi)
        elif r==1: cand[j]=float(np.clip(cand[j]+delta,*V_RANGE))
        elif (r-2)%4==1:
            if (r-2)//4==0: cand[j]=float(np.clip(cand[j]+delta,*T_DROP_RANGE))
            else: cand[j]=max(MIN_GAP, cand[j]+delta)
        elif (r-2)%4==2: cand[j]=float(np.clip(cand[j]+delta,*FUSE_RANGE))
        cand = clip_gene_vec(cand)
        val,_,_,_ = evaluate(cand, dt=dt)
        if val > best_val:
            best, best_val = cand, val
        if (t+1)%120==0:
            step *= 0.7
    final_val,_,_,_ = evaluate(best,dt=0.01)
    if final_val > best_val:
        best_val = final_val
    return best, best_val

def plot_convergence(best_hist, avg_hist):
    plt.figure(figsize=(8,5))
    plt.plot(best_hist, marker="o", label="最优值")
    plt.plot(avg_hist,   marker="x", label="平均值")
    mv=max(best_hist); mg=int(np.argmax(best_hist))
    plt.title(f"遗传算法收敛曲线（最大总遮蔽时间 = {mv:.2f} s）")
    plt.xlabel("迭代数"); plt.ylabel("三导弹遮蔽时间总和 (s)")
    plt.grid(ls="--", alpha=0.5); plt.legend()
    plt.annotate(f"{mv:.2f} s", xy=(mg,mv), xytext=(mg, mv+0.8),
                 arrowprops=dict(arrowstyle='->'),
                 ha='center')
```

```
    plt.tight_layout();plt.show()

def main():
    random.seed(42); np.random.seed(42)
    base, bh, ah = genetic(n_gen=100, pop_size=220)
    best_plan, best_val = polish(base,iters=800,dt=0.02)
    total, per_miss, per_smokes, _ = evaluate(best_plan,dt=0.01)

    print("\n【问题五 最优策略（强化版）】")
    print(f"M1={per_miss[0]:.2f}s  M2={per_miss[1]:.2f}s  M3={per_miss[2]:.2f}s
总和={total:.2f}s")

    uav_rank = {u:i for i,u in enumerate(UAV_LIST)}

    per_smokes_sorted = sorted(
        per_smokes,
        key=lambda x: (uav_rank[x[0]], x[8], x[3])
    )

    smoke_no_counter = {u: 0 for u in UAV_LIST}

    cols = [
        "无人机编号",
        "无人机运动方向",
        "无人机运动速度(m/s)",
        "烟幕干扰弹编号",
        "烟幕干扰弹投放点的 x 坐标(m)",
        "烟幕干扰弹投放点的 y 坐标(m)",
        "烟幕干扰弹投放点的 z 坐标(m)",
        "烟幕干扰弹起爆点的 x 坐标(m)",
        "烟幕干扰弹起爆点的 y 坐标(m)",
        "烟幕干扰弹起爆点的 z 坐标(m)",
        "有效干扰时长(s)",
        "干扰的导弹编号",
    ]

    out_rows = []
        for (uav, m_id,t_single,t_exp,P_drop,C_exp,theta,v,t_drop,fuse) in
per_smokes_sorted:
        smoke_no_counter[uav] += 1
        smoke_no = smoke_no_counter[uav]
        deg = float(np.degrees(theta) % 360.0)

        out_rows.append({
```

```
                "无人机编号": uav,
                "无人机运动方向": round(deg, 2),
                "无人机运动速度(m/s)": round(v, 2),
                "烟幕干扰弹编号": smoke_no,
                "烟幕干扰弹投放点的 x 坐标(m)": round(P_drop[0], 2),
                "烟幕干扰弹投放点的 y 坐标(m)": round(P_drop[1], 2),
                "烟幕干扰弹投放点的 z 坐标(m)": round(P_drop[2], 2),
                "烟幕干扰弹起爆点的 x 坐标(m)": round(C_exp[0], 2),
                "烟幕干扰弹起爆点的 y 坐标(m)": round(C_exp[1], 2),
                "烟幕干扰弹起爆点的 z 坐标(m)": round(C_exp[2], 2),
                "有效干扰时长(s)": round(float(t_single), 2),
                "干扰的导弹编号": int(m_id + 1),
            })

    df_out = pd.DataFrame(out_rows, columns=cols)
    with pd.ExcelWriter("result3.xlsx", engine="openpyxl", mode="w") as writer:
        df_out.to_excel(writer, sheet_name="结果", index=False)
    print("已保存到 result3.xlsx（按 UAV 序号且单机内按发射时间排序）")

    plot_convergence(bh, ah)

if __name__=="__main__":
    main()
```