基于 TDOA 等定位模型和粒子群优化算法的火箭残骸预测

摘要

本文针对火箭残骸的音爆位置确定建立了模型，并针对测量数据的对应问题和观测噪声的滤除分别进行了模型的改进，得到了一定的成效。

对于问题一，针对观测设备的个数确定，至少需要四个不共面的监测设备才能唯一确定一个信号源的位置时间信息。而针对残骸的求解，我们基恩首先通过音爆与设备距离的关系得到了关于同一音爆的 **TDOA** 方程组，并使用**粒子群优化算法**，在**时差误差**最小化的目的下，进行关于多个设备观测数据的超定方程解的计算。针对全部设备 TDOA 方程带入的优化结果不理想的情况，我们对所有合理的设备组合进行了优化运算，并得到了最佳观测设备组合 **1**、**3**、**5**、**7** 和对应的音爆点的三维坐标。对于未包含在 TDOA 方程内的音爆时间，我们以四个选定设备计算得到的音爆时间为出发点，对这四个设备和所有设备的**距离误差**进行关于音爆时间的二次优化，得到了误差最小的音爆时间。

对于问题二，对于多个残骸的音爆时间对应，我们需要增加最低观测设备的数量来降低解的模糊性，确定了五个测量设备作为观测的最低要求。同时针对第一问中部分设备数据不可信的问题和各种时间组合数量繁杂的问题，我们提出了一种**动态方程选择**策略，同时针对多个方程进行优化，但设立一系列限制条件，只使用其中最佳的数个方程作为优化目标，同时改用**多边测量模型**来减少优化运算规模。

对于问题三，在问题二的模型的基础上，经过多次优化运算，最终得到了所有数据结果均入选的优化结果，并得到了误差极小的四个残骸位置和音爆时间。

对于第四问，我们从 **RAN-SAC 算法**的去噪思想出发，提炼出“反演”和“舍弃”方法，将其带入粒子群优化的过程之中，在极少量数据的问题背景下，提出了一种全新的“**反演-改进-降权-舍弃**”的去噪思路。并在施加了标准差 0.5s 的高斯噪声下的时间进行了实验，最终结果取得了约 25% 的降噪效果。

最后，我们对模型进行了优缺点分析。

关键字：粒子群算法　TDOA 模型　多边测量模型　RAN-SAC 算法

目录

一、 问题重述

1.1 问题背景

多级火箭中的下级火箭在完成任务之后，会通过分离装置分离脱落，以我国载人航天所使用的长征二号 F（CZ-2F）运载火箭为例，在发射后三分钟内，火箭的逃逸塔、助推器、一级火箭、整流罩等重要组成部分会相继程序分离。如何快速精准地定位残骸并预测其落地点从而进行回收，成为了目前研究的主要问题。目前主要的回收方法是通过震动波检测设备检测火箭残骸在下落过程中产生的音爆，从而推算出音爆发生的位置，进而进行弹道外推预测出可能的落地点。[1]

1.2 需要解决的问题

我们通过分析相关数据，运用数学思想，建立数学模型来研究拍照软件定价中的下列问题：

问题 1. 建模分析至少需要多少台设备才能精确计算单个残骸的音爆点，并通过表中给出的实例进行计算具体坐标和音爆时间；

问题 2. 分析当存在四次音爆时，如何建模分析哪些震动波来自同一个残骸，以及至少需要多少台才能精确计算四个残骸各自的音爆位置和音爆时间；

问题 3. 利用表中给出的数据进行计算四个残骸的音爆位置和时间；

问题 4. 完善问题 2 给出的模型，以尽可能精确地确定当设备记录数据存在一定误差（0.5s）时各自的音爆位置和时间，分析结果误差并利用问题 3 给出的数据进行验证。

二、 问题分析

2.1 问题一分析

在三维空间中，理论上至少需要四个不共面的监测设备才能唯一确定一个信号源的位置。这是因为对于三维空间中的点，我们需要三个独立的距离方程（或时间差方程）来确定其在三维空间中的位置，再加上一个时间方程来同步时间。

至于确定空中单个残骸发生音爆时的位置坐标（经度、纬度、高程）和时间的问题，可以通过计算残骸到各个监测点的时间差，计算出残骸在产生音爆时到各个距离的位置，利用 TDOA 推算出残骸的位置。

2.2 问题二、三分析

问题一中已经指出，对于一个残骸，理论上至少需要四个不共面的监测设备才能唯一确定一个信号源的位置。考虑到残骸与数据的不对应性，需要再加入一个监测设备的

组数据消除解的模糊性。

本题依旧通过计算各个残骸到各个监测点的时间差，建立方程组来求解残骸的具体位置，考虑到不同残骸在监测设备中的数据不一样，在建立起时间差的方程后，采用动态选择方程数目的方法，利用多边测量法结合粒子群算法，优化推算出各个残骸的位置。

2.3 问题四分析

本题要求修正现有模型，以在设备记录时间存在 0.5 秒随机误差的情况下，更精确地确定四个残骸在空中发生音爆的位置和时间。我们根据 RANSAC 降噪算法，对基于少量数据的 RANSAC-PSO 算法进行改进，中提炼出“反演-改进-降权-舍弃”的去噪方法，通过在施加了 0.5 秒高斯噪声的数据对比看去噪效果。

三、 模型的假设

- 假设声速不随空气密度、温度或湿度的变化而变化，在一定的范围内空气中的声速始终为 340m/s。
- 假设在考虑的计算范围内，计算两点间距离时可忽略地面曲率，因此可以使用直角坐标系来表示地理位置。
- 假设传感器的位置是精确已知的，且不会受到测量误差的影响。
- 假设声源是点源，即声音在空间中的传播路径是直线。

四、 符号说明

符号	说明
t	如不特别标明，则指音爆时间
(x_i, y_i, z_i)	第 i 个监测设备的坐标
(x, y, z)	音爆点的坐标
c	声速
d	音爆点到监测设备的距离
obj	多变测量模型的优化目标函数

五、 模型的建立与求解

5.1 问题一模型的建立及求解

5.1.1 模型的建立

火箭残骸脱离火箭坠落的过程中，速度逐渐增大，当其速度达到声速的时候，会产生音爆的现象。音爆点的高度与残骸解体高度、空气阻力大小等因素有关。但总的来说，残骸一般在 100 千米高度就会与火箭主体分离，进入大气层时，残骸左复杂的变速运动，理论上当残骸接近地面 10-20km 时，速度达到 340m/s，产生音爆。为了能够及时捕捉到残骸信号，监测点位置会在残骸理论落点两侧或火箭发射场附近，总的来说在火箭发射场 20km 以内，以便及时接收到音爆。

设音爆源的三维坐标为 (x, y, z)，音爆发生的时间为 t_0，第 i 个监测设备的坐标为 (x_i, y_i, z_i)，观测到音爆的时间为 t_i,c 是音速。则有：对于每个监测设备，音爆源到监测设备的距离

$$d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{1}$$

$$d_i = c \cdot (t - t_i) \tag{2}$$

给定 7 台设备的三维坐标和音爆抵达时间，在本题中，我们利用到达时间差模型（TDOA）进行求解。TDOA 是基于各参考基站（即监测设备）与待定位对象（即音爆点）之间的距离之差通过求解非线性双曲方程组来推断待定位对象相对于各参考基站的相对位置的定位方法[2]。与常见的多变测量算法不同的是，TDOA 在方程的计算中省略了时间这一变量，搜索空间更小，更容易收敛。

在平面内，到两定点的距离差为定长的轨迹被称为双曲线。具体而言，对于两个监测设备，在待定位对象对这两个设备的距离差的情况下，待定位对象的坐标即分布在以这两个监测设备为焦点，距离差为左右焦半径之差的双曲线上。对于四个监测设备，我们共可以得到 C_4^2 即 6 个这样的双曲面，这六个双曲面的交点就是我们需要得到的解，如下图所示。

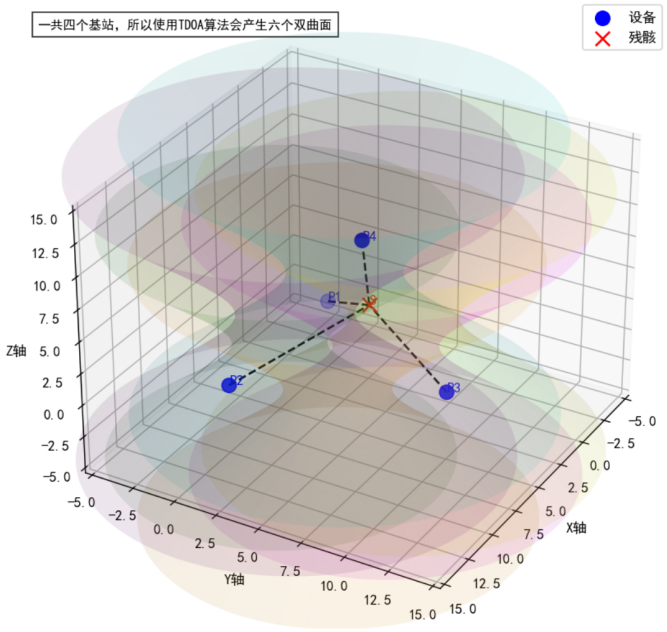

图 1　TDOA 基本原理图解

具体算法如下：

若已知两个监测设备的时间差 $\Delta t_{ij} = t_i - t_j$，则可以得到

$$\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} - c\Delta t_{ij} = \sqrt{(x-x_j)^2+(y-y_j)^2+(z-z_j)^2} \tag{3}$$

对于所有监测设备，音爆发生的时间满足 $t_i = t_0 + \frac{d_i}{c}$。因此，在方程（2）中，我们消去了时间的变量。因此，我们选取利用粒子群优化算法（PSO）进行求解。我们需要确认以下目标函数和约束条件：

1. 目标函数的确定

对于第 i, j 个监测设备，我们可以得到以下的方程。

$$\begin{aligned} &|\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} - c\Delta t_{ij} \\ &-\sqrt{(x-x_j)^2+(y-y_j)^2+(z-z_j)}| = 0 \end{aligned} \tag{4}$$

为了简化运算，我们取第一个设备为相对设备，即取 j = 1。对于 n-1 个其他的监测设备，我们可以得到 n-1 个这样的方程，对 n-1 个这样的方程，左式即我们想要进行的优化目标，也即时差误差的和。左式越接近 0，说明我们的解就越精确。即，我们的优化目标是：

$$\begin{aligned} &\min \sum_{i,j} (|\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} \\ &-c\Delta t_{ij} - \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2}|)^2 \end{aligned} \tag{5}$$

2. 约束条件的确定

为了使得得到的解符合实际情况，我们还需要约定以下约束条件：

$$x_{max} \geq x \geq x_{min}, \tag{6}$$

$$y_{max} \geq y \geq y_{min}, \tag{7}$$

$$z_{max} \geq z \geq 0, \tag{8}$$

$$t_0 \in [\min(t_i) - \frac{d_{\max}}{c}, \max(t_i)], \tag{9}$$

其中 (6)(7)(8) 式, 都表示对于三维坐标的约束，保证算法在合理的区间内进行求解，（9）表示时间先验，$d_{\max}$ 是音爆源到最远监测设备的最大可能距离。

5.1.2 模型的求解

如 5.1.1 中所述，我们采用粒子群优化算法来进行求解。

粒子群优化算法（Particle Swarm Optimization，PSO）起源于对简单社会系统的模拟，在对粒子群算法数学化后，PSO 是一种很好的非线性方程的求解方式。

在粒子群算法（Particle Swarm Optimization，PSO）中，每个优化问题的潜在解都是搜索空间中的一个粒子。所有的粒子都是由一个被优化的函数决定的适值（fitness value），每个粒子还有一个速度决定他们飞翔的方向和距离[3]。粒子们就追随当前的最优粒子在解空间中搜寻最佳位置。

假设在一个群体中有 N 个粒子，不同的个体有不同的位置 X_j，他在空间中以某个速度 V_j 飞行，不同的位置对应于不同的与优化目标函数值相关的个体适应度函数值 $F(X_j)$。

POS 初始化为一群随机粒子（随机解），然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个极值来更新自己：第一个就是粒子本身所经历的最好位置，为 $P_j = [P_{j1}, P_{j2}...P_{j7}]^T$, 也称为 P_{best}；第二个就是群体中所有粒子经历的最好的位置，为 $P_g = [P_{g1}, P_{g2}...P_{g7}]^T$, 也称为 g_{best}。若已知第 k 代第 j 个粒子的速度 $V_j^{(k)}$ 及位置 $X_j^{(k)}$，则第（k+1）代第 j 个粒子的速度及位置为：

$$V_j^{(k+1)} = w^{(k)} V_j^{(k)} + c_1 r_{j1} \left(P_j - X_j^{(k)}\right) + c_2 r_{j2} \left(P_g - X_j^{(k)}\right) \tag{10}$$

$$X_j^{(k+1)} = X_j^{(k)} + V_j^{(k+1)} \tag{11}$$

$$w^{(k)} = w_{\max} - k\left(w_{\max} - w_{\min}\right)/k_{\max} \tag{12}$$

其中 $w^{(k)}$: 惯性权重系数，是迭代次数的函数，且随迭代次数线性减少；w_{max}：初始惯性权重；W_{min} 终止惯性权重；k_{max}：最大迭代次数。

c_1: 粒子自身加速度权重系数，一般在 0-2 之间取值；

c_2: 全局加速度权重系数，一般也在 0-2 之间取值；

r_1,r_2:[0,1] 范围内两个相互独立的、均匀分步的随机数。

基本粒子群算法流程如下：

1. 初始化粒子群，包括群体规模 N, 每个粒子的位置 X_j 和速度 V_j；
2. 计算每个粒子的适应度值 $F(X_j)$；
3. 对每个粒子，用他的适应度值 $F(X_j)$ 和个体极值 P_{best} 比较，如果 $F(X_j)$ 优于 P_{best}，$F(X_j)$ 替换掉 P_{best}；
4. 对每个粒子，用他的适应度值 $F(X_j)$ 和全局极值 g_{best} 比较，如果 $F(X_j)$ 优于 g_{best}，$F(X_j)$ 替换掉 g_{best}；
5. 根据公式（10）-（12）更新粒子速度 v_j 和位置 X_j;
6. 如果满足结束条件（误差足够好或达到最大循环次数）退出，否则返回第 2 步。

考虑到未知数的个数，理论上四个监测设备（即三个到达时差方程）就能解出音爆点的三维坐标，进而推算出音爆时间。利用全部七个设备的数据，理论上则可以加强约束，求解超定方程，得到更精准的数据。

但是事实上，在选取七个设备时，得到的优化函数误差偏大（接近五万米）。因此，我们开始怀疑各个设备记录的准确性和所有设备均代入组合的合理性，为了使得我们得到的数据误差最小，我们对于四个以上的各种组合均进行了测试。

前面提到，单纯的 TDOA 算法不包含对于音爆时间的求解，所以我们需要根据多个设备的信息倒推出对应的多个音爆时间。由第 n 个设备的数据和计算得到的音爆位置计算得到的音爆时间 $t_{n,o}$ 计算方式如下：

$$t_{n,o} = t_{n,arr} - \frac{\hat{d}}{c} \tag{13}$$

为了衡量通过这种方法计算得到的结果在时间上的准确性，我们使用各个计算得到的音爆时间的方差来衡量该组方程解在音爆时间上的表现。方差越小，说明计算得到的音爆时间越集中，结果可信度越高。反之，方差越大，方程在音爆时间上的表现越差，结果越不可信，计算公式如下：

$$\bar{t_o} = \frac{1}{n}\sum_{i=1}^{n} t_{i,o} \tag{14}$$

$$\mathrm{Var}(t_{1,o}, t_{2,o}, \ldots, t_{n,o}) = \frac{1}{n-1}\sum_{i=1}^{n}(t_{i,o} - \bar{t_o})^2 \tag{15}$$

以经纬度和高程的均值为初始猜测，在 x 和 y 方向（横纵坐标）以 x_{dis} 为扰动尺度，在高程方向以 z_{dis} 为扰动尺度，利用以目标函数与 0 之间的**优化函数误差**，还有对于一组选定的设备，选定每个设备作差时得到的各自的音爆时间的**时间方差**作为评价标准，如表 1 所示；

表 1　选取不同监测设备的收敛误差统计

设备编号	优化函数误差（单位：m）	音爆时间方差
1,2,3,4	5206.55	535.59
1,2,3,5	16236.77	938.35
1,2,3,6	6106.65	625.67
1,2,3,7	21623.02	2377.6
1,2,4,5	20487.32	1372.6
	.....	
1,2,5,7	约 $10^{-}9$	202.36
1,3,5,7	约 $10^{-}9$	67.01
	.....	
1,2,3,5,6,7	20604.24	518.33
1,2,4,5,6,7	43655.31	2836.6
1,3,4,5,6,7	33629.56	918.17
2,3,4,5,6,7	26577.78	1122.8
1,2,3,4,5,6,7	46899.76	2218.3

注：由于篇幅有限，这里仅展示部分数据。

最终，我们选取 1,3,5,7 的监测设备组合计算得到的音爆位置作为最佳位置。

关于音爆时间的选择，直接以四个基站计算得到的音爆时间取平均值并不一定在距离误差上表现的最好，所以我们考虑在一定的范围内搜寻最佳的音爆时间取值，对其进行二次优化，使得其对于四个亦或者全部设备都具有较好的适应性。

现在我们需要在得到的这个音爆时间区间中找到最优值，通过在 $[Rt_{o,min}, Rt_{o,max}]$ 这个扩大了的时间区间中以 δt 为步长，重新计算残骸和各设备间的距离误差来寻找使距离误差最小的音爆时间点。计算距离误差公式如下：

$$DistanceError = \sum_{i=0}^{k} \left| \sqrt{(\hat{x}-x_i)^2-(\hat{y}-y_i)^2-(\hat{z}-z_i)^2} - c\cdot(t-t_i) \right| \tag{16}$$

我们首先选取 1,3,5,7 这四个设备来计算距离误差，得到的结果如下图所示。

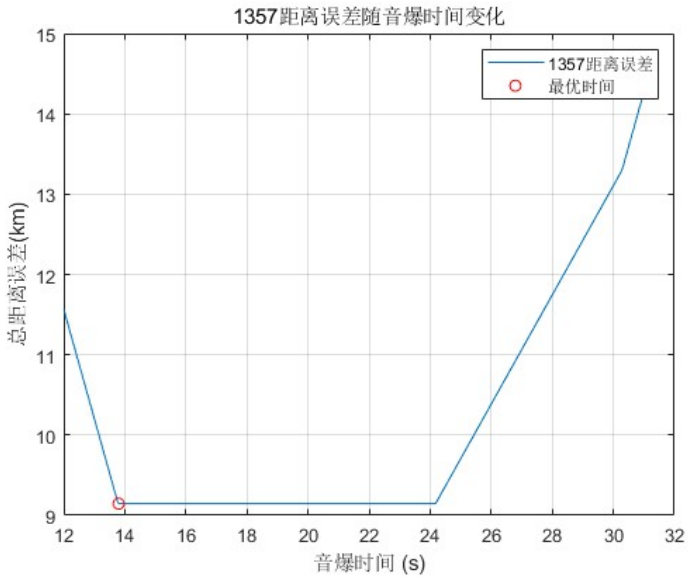

图 2　选取 1,3,5,7 四个设备时的误差计算

从上图可以看出，在某个区间内距离误差不随音爆时间的改变而变化。这是因为在这个区间内，改变音爆时间会使四个基站中的两个的到达时间与其产生相同的正偏差，另外两个与其产生相同的负偏差，从公式来看正负偏差的和一定等于 0，故总的时间误差不发生改变。

因此，我们再次选择针对所有设备的距离误差进行优化。同理，易得到七个设备的情况一定存在一个最小的误差取值时间，所以可以避免针对四个设备优化不能确定最终时间的问题，还能使得解对于全部设备的适应性比较好（尽管我们怀疑了数据的可信度，但这里我们仍然认为所有设备的数据仍然是有意义的）。最终得到图像如下：

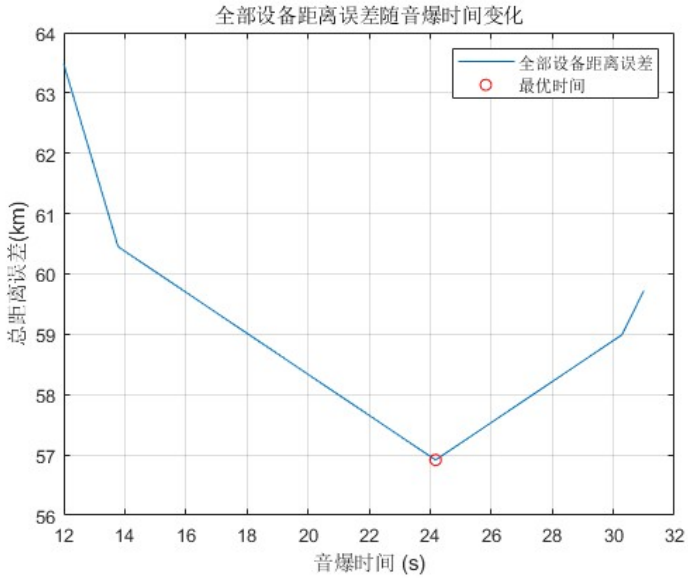

图 3　选取七个设备时的误差计算

由图三和上面数据可得，我们最后得到的相对于监测点开始监测的监测时间为**24.1800s**，相应的音爆点坐标为**经纬度 (110.41,27.33)，高程 14080.88m**，如下图所示。

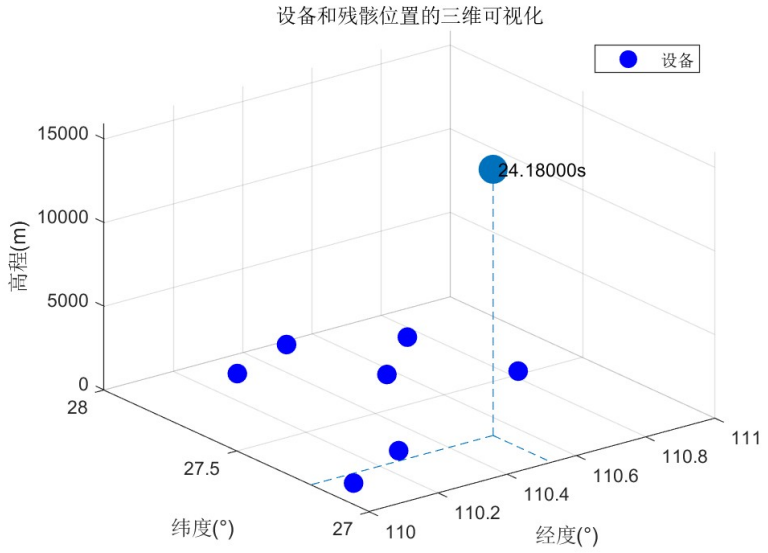

图 4　音爆点图示

5.2 问题二模型的建立及求解

5.2.1 模型的建立

鉴于 5.1 节中不同监测设备的数据存在误差或者数据的组合存在问题，对于第二问，我们决定针对方程的选择设计一种动态的选择策略。因为需要确定时间组合，并且由于抵达时间映射到空间距离的非线性特征，以抵达时间本身或抵达时间差构成的度量空间的距离属性不能直接用于分组，也就是说一般的聚类方法不是很适用于本题的时间组合的确定。在这种情况下，我们建立了一种动态选择时间组合策略，来进行各个抵达时间的匹配和结果的优化。同时对于本问题背景下 TDOA 候选时间差数量大的问题，本题我们更换使用了多边测量模型来减少优化的计算规模。

在二维空间中，如果知道某个待定位节点到已知位置的信标节点的距离，就可以通过求解已知半径和圆心坐标的圆的交点来确定该待定位节点的坐标。同样地，多边测量法是求对于各个已知位置和给定的距离，对应的球面的交点。

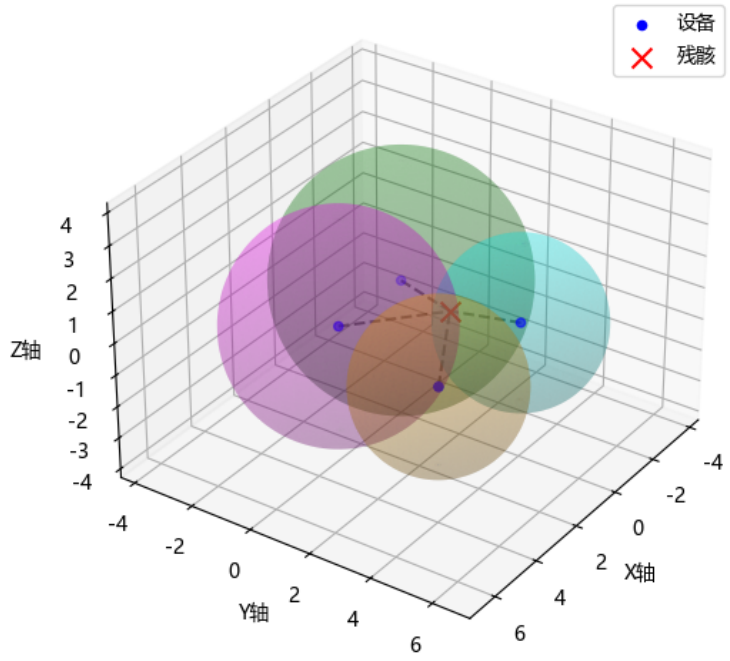

图 5　多边测量法基本原理图解

具体算法如下：

假设第 i 个监测设备监测到的第 k 个数据的初始时间为 $t_{i,k}$（i≤7，k≤4），则所有的时间形成一个 7*4 的矩阵。

对于音爆源，音爆发生时间 t_o，第 i 个监测设备的坐标 (x_i, y_i, z_i)，监测到的第 k 个音爆源 (x, y, z)，观测到的音爆时间 t_i, 音速 c, 满足以下关系：

$$\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2 + (z_k - z_i)^2} = c * t_{i,k} \tag{17}$$

对于第 i 个监测设备的所有 $t_{i,k}$，可以得到一个非线性方程组，将方程移项，右式中即可得到 $\Delta t_{i,ki-kj}$，这样可以构造一个 7*4 的方程残差矩阵。选取其中每行绝对值最小的值，即对于本次优化每个设备表现最好的抵达时间，记作 $\Delta t_{i,min}$。对于七个监测设备，我们可以得到七个这样的 $\Delta t_{i,min}$。

从这七个值中选取表现最好的前 n 个作为初始方程数量，对于 4 个残骸的情况，理论上至少需要 4 个方程才能求出解，因此 n≥4。

1. 目标函数的确定

和第一问相似，我们的优化目标同样是使得方程的残差平方和最小，得到以下的优化目标：

$$\begin{aligned} obj &= \sum_{i=1}^{n} \Delta ti, min \\ &= \sum_{i=1}^{n} t_o + \frac{d}{c} - t_{i,k} \end{aligned} \tag{18}$$

其中，d 是音爆点到监测设备距离，t_o 表示音爆时间。

2. 动态选择的策略

合群性检验

对于初始的方程组，n=4；当需要选取新的数据加入方程组时，对于某个粒子的某一次优化，当拟加入计算的数据 Δt_i 满足

$$\Delta t_i < \eta * \frac{\sum_{i=1}^{m} \Delta t_m}{m} \tag{19}$$

$$obj_{t_i} < obj_{inner1}, \tag{20}$$

$$\Delta t_i < M \tag{21}$$

时，便考虑将这一数据加入方程进行优化。

其中，obj_{inner1} 为开始进行动态选择的最大优化目标函数值。M 为容纳新方程的最低残差阈值。η 表示对于与引入新方程的容忍度，η 越大，表示对新方程加入的限制越低，反之则对新方程的限制越高。

（20）（21）式引入最低残差阈值和进行动态选择的最大优化目标函数值的意义是，当处于优化早期时，解的指向性不明确，各个方程都可能具有较大的偏差，在这种情况下只使用（19）式可能会容纳错误的方程进入优化过程，导致结果误差大或者方程不收敛，所以对解的残差和优化函数值做出最低限制，以避免这一情况。

5.2.2 模型的求解

结合公式（17）和公式（19），我们在多边测量法的基础上，使用结合了动态方程选择的粒子群优化进行求解，求解主要步骤如下：

1. 初始方程组的建立首先，我们根据初始构造的残差矩阵，选择出 4 个 $\Delta t_{i,min}$ 对应的 t_i 作为初始方程组的抵达时间。
2. 构建方程组利用公式（3）构建方程组。
3. 求解方程组我们利用粒子群优化算法对方程组进行求解。在求解过程中，考虑合群性检验的条件，即确保新加入的数据满足上述的合群性要求。
4. 迭代优化如果初始的 n 个方程不能给出足够精确的解，或者我们想要进一步提高解的精度，则进行迭代优化。在每次迭代中，根据合群性检验的条件选择新的数据加入方程组，并重新求解方程组。通过多次迭代，可以逐渐提高解的精度和稳定性。
5. 重复求解完成一个残骸的求解之后，对于其使用的时间进行舍弃。这里我们使用一个惩罚矩阵，为使用的时间对应的残差矩阵乘上一个很大的惩罚系数 γ 来确保后续的优化方程中不包含这个抵达时间。

6. 结果验证与输出最后，对求解得到的结果进行验证。这可以通过将求解得到的残骸位置与实际的监测数据进行比较来实现。如果结果满足一定的精度要求，则认为求解成功，并将结果输出。否则，需要重新检查数据或调整求解策略。

在此基础上，能够更进一步地确定哪些数据来自同一个残骸。这可以通过比较不同设备间音爆抵达时间的相对差异来实现。例如，如果两个设备对同一残骸的音爆抵达时间的差异与它们之间的距离和声速计算得出的传播时间差匹配，则可以假定它们是由同一音爆引起的。

5.2.1 中提到，对于一个残骸时，只需要四个设备三个方程就能确定所有的未知数。对于本问题，四个设备还不足以对应到各自的残骸，需要加入一个方程约束以消除解的模糊性；因此，至少需要五个监测设备。

5.3 问题三模型的求解

以所有设备经纬度均值、一万米高程和音爆时间 20s 作为初始猜测进行优化。和第一问结果不同的是，本题的所有观测数据都是可信，每次优化都会将所有的 7 个设备的最优抵达时间容纳进优化结果。结果表明，当 n=7 时，四个残骸的误差最小，此时得到的四个残骸的位置和相对音爆时间、误差如下。

表 2　四个残骸的求解结果

残骸编号	经度	纬度	高程（米）	相对音爆时间（秒）	音爆时间误差（秒）
1	110.70	27.65	13468.1	15.0	约 $10^{-}4$
2	110.50	27.95	11528.7	13.0	约 $10^{-}4$
3	110.50	27.31	12513.9	12.0	约 $10^{-}4$
4	110.30	27.95	11477.5	14.0	约 $10^{-}4$

值得一提的是，在我们没有对各个残骸音爆时间之间的关系进行约束的情况下，所有残骸的音爆时间相互差别不超过 5 秒。经过优化计算得到的答案呈现了完美的整洁性，经纬度均十分接近有限长的小数，而时间均为整数，这无疑令我们十分惊喜。

对于每个监测设备监测到的数据对应的残骸以及对应的距离误差，则如下图所示。

残骸\基站	A	B	C	D	E	F	G
残骸1	214.850	92.453	75.560	196.517	78.600	175.482	210.306
残骸2	270.065	196.583	110.696	94.653	126.669	67.274	206.789
残骸3	100.767	112.220	188.020	258.985	118.443	266.871	163.024
残骸4	164.229	169.362	156.936	141.409	86.216	166.270	103.738

图 6　每个残骸对应的设备记录音爆到达时间

最终的可视化结果如下图所示。

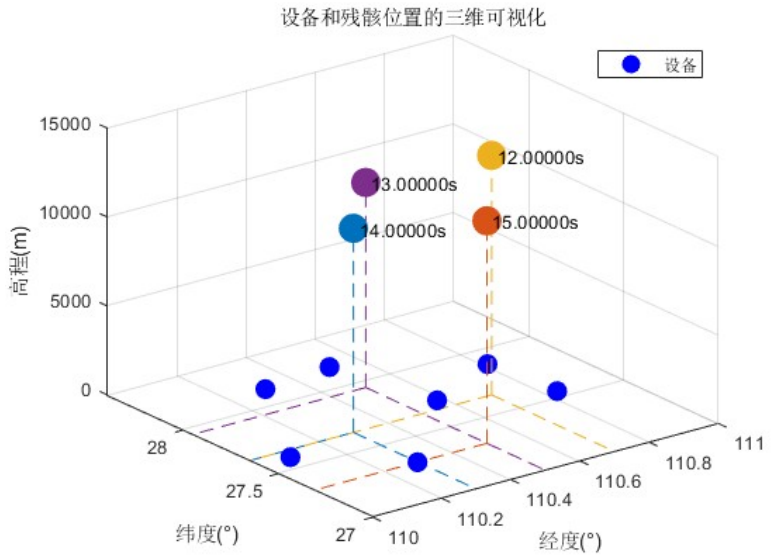

图 7　四个音爆点图示

5.4 问题四模型的建立及求解

5.4.1 模型的建立

对于 5.2 的模型，当加入标准差为 0.5s 的高斯噪声后，由于其样本数据量小，噪声对于其的影响较大，如下图所示。在 5.2 这样一个非线性模型中，更加地难以预测和控制误差。

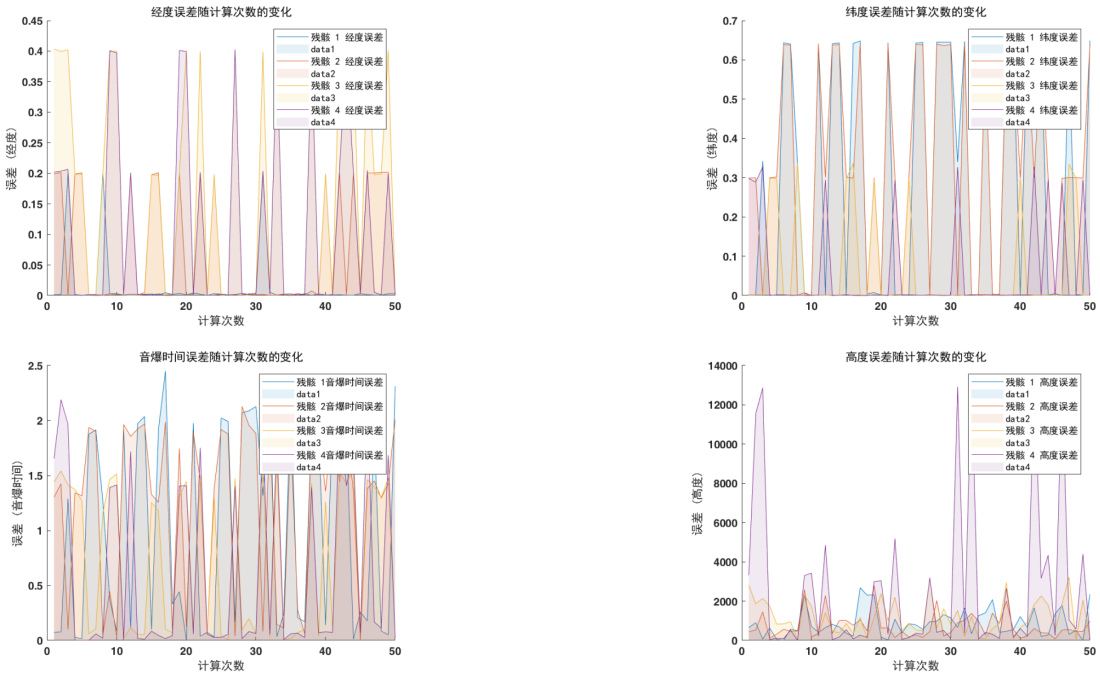

图 8　存在噪声时各监测设备数据误差比较

对于噪声的处理，我们考虑用 RANSAC 算法剔除异常值。RANSAC，即 Random Sample Consensus，是随机样本一致算法，它是根据一组包含异常数据的样本数据集，计算出数据的数学模型参数，得到有效样本数据的算法。其基本流程如下：

1. 确定反演所需的最小样本数量，并随机抽取这些样本。使用模型对这些最小样本进行优化，以**反演**所给模型的滑动参数。反演就是根据观测数据恢复模型参数的过程。
2. 假设得到的参数是最优的，并用它们来计算观测值的预测。
3. 计算预测值与实际观测值之间的残差，并与设定的误差阈值比较，小于阈值的数据点被定义为内部值，大于阈值的数据被视为异常值剔除。
4. 通过迭代，最终确定最优模型参数。

但是，对于本题而言，在数据量本身就非常少的情况下，再去除一部分数据会使结果的不确定性大大增加。并且在少量数据的情况下去除异常值可能会比使用异常值本身带来更大的误差。因此我们需要对于原本的 RANSAC 算进行改进。

因此，本题基于 RANSAC 模型的思想进行误差处理[5]，提出了一种 RANSAC-PSO 算法。这一模型将 PSO 算法和 RANSAC 算法相结合，对监测设备监测数据进行粗差定位和剔除，从而提高参数的精度。结合了 RANSAC 和 PSO 的 RANSAC-PSO 算法，旨在提高反演参数的精度，特别是在存在粗差的情况下。

5.4.2 模型的求解

考虑到监测设备数据只有七个，本题基于 5.2 的多边测量和方程的动态调整，对于 RANSAC-PSO 算法进行改进，使其能够符合本题情况的求解。具体算法如下；

0. 基本前提

若最小残差阈值为 obj_{inner2}，则 obj 需要满足

$$obj < obj_{inner2} \tag{22}$$

1. 反演

先选取进行 PSO 的最小样本, 开始执行粒子群的优化算法时，不舍弃原样本参数。当某粒子的 obj_1 值小于优化后的新值 obj_{re} 时，开始反演：对前五个方程进行参数的优化，直到优化到 obj_1< 阈值 obj_{l_0}

2. 改进

当某粒子的 obj_2 值小于 obj_re 时，对于第 m 个粒子的第 i,j 个监测设备的时间差 $t_{i,j}^m$，对这一时间差进行适当修正，调整方法如下:

对于给定的参数 ϵ，

$$(new)t_{i,j}^m = (old)t_{i,j}^m \pm \epsilon\sigma_{gauss} \tag{23}$$

调整的方向为 obj($t_{i,j}$) 减小的方向。

为提高修正速度，在区间 $[-3\sigma_{gauss}, 0)$ 和 $(0, 3\sigma_{gauss}]$ 两个区间进行二分查找最优 ϵ 的取值（根据正态分布，所求值有 99.6% 的概率落在这一区间），直到找到使 obj_2 最小的值，同时设置查找次数上限 N_{2ser} 来保证优化算法的速度。

3. 降权

对于经过反演、改进的七个候选的数据，计算对应的 Δt_n，然后进行归一化处理：

$$\Delta\widetilde{t_i} = \frac{\Delta t_i - \Delta t_{n,min}}{\Delta t_{n,max} - \Delta t_{n,min}} \tag{24}$$

得到 $\vec{\beta} = [\Delta\widetilde{t_1}, \Delta\widetilde{t_2}, ..., \Delta\widetilde{t_7}]$;

为了保持优化函数的基本规模,对前四个值进行保留,更新 $\vec{\beta}$ 为 $\vec{\beta_1} = [\Delta 1, 1, 1, 1, \Delta\widetilde{t_5}, ..., \Delta\widetilde{t_7}]$

如果优化后的结果仍然误差较大，则按照给定的参数 $\vec{\beta_1}$ 对原来的 obj 进行降权处理：

$$obj_{new} = ||\beta \cdot obj_{old}|| \tag{25}$$

参数 β_1 是逐渐降低的，$\Delta\widetilde{t_1}$ 最接近 1，$\Delta\widetilde{t_1}$ 最接近 1，使用这种方法可以降低这一参数对于目标 obj 的影响。

4. 舍弃

假定 Δt_{thre} 是给定的时间差阈值，当 $\Delta\widetilde{t_n} < \Delta t_{thre}$ 时，数据已明显不可信并无挽救价值，舍去这一数据，即令 $\Delta\widetilde{t_n} = 0$。

按照这一步骤，降噪得到的图如下图所示。

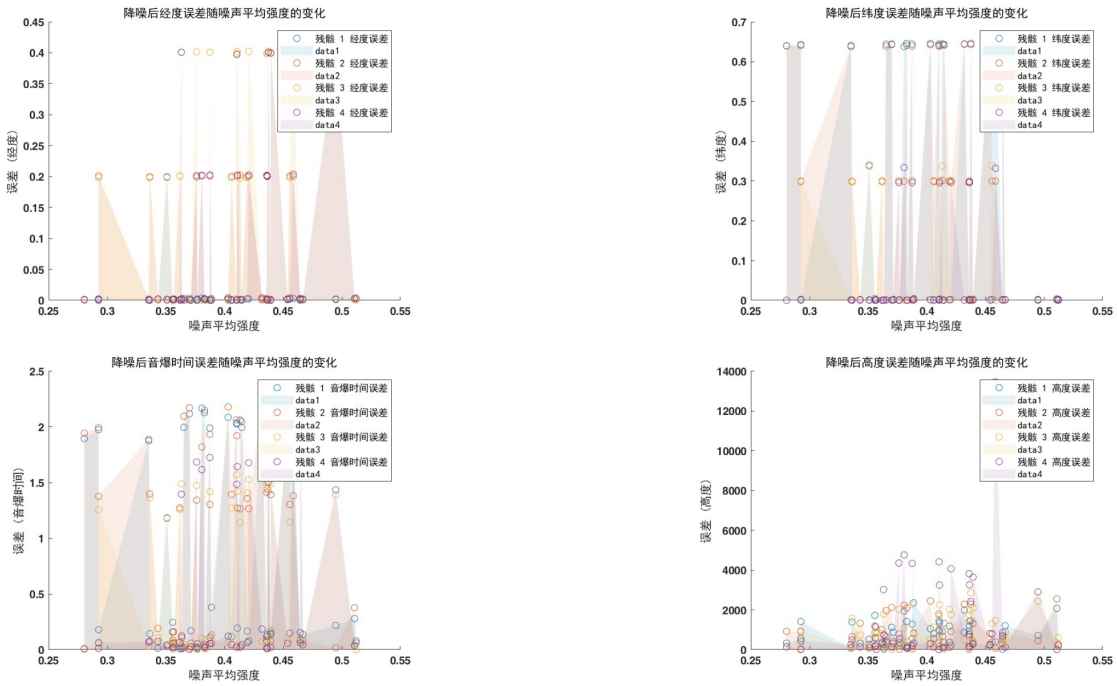

图 9　降噪后音爆时间和位置误差比较

降噪前后的平均误差如下表所示。

表 3　降噪前后误差对比

降噪前后	平均经度误差	平均纬度误差	平均高程误差	平均音爆时间误差
噪声施加后	0.089148	0.193433	1352.989085	0.926040
降噪后	0.067086	0.158714	992.531351	0.742608
降幅	24.74%	17.94%	26.64%	16.94%

5.4.3 误差分析

观察到在弱噪声环境下模型表现良好，能够有效地减少误差，提高数据的准确性。然而，在高强度噪声环境下，模型的表现并不理想，甚至出现了误差增大的情况。

经过深入分析，我们认为造成这一现象的原因主要有数据量不足、模型复杂度和泛化能力不足、参数和算法有待优化等。

六、 模型的评价

6.1 模型的优点

1. 精确性：通过粒子群优化算法和 TDOA（Time Difference of Arrival）方程组，模型能够较准确地确定音爆源的位置和时间。在多个残骸的音爆定位中，模型通过动态方程选择策略和多边测量模型，进一步提高了定位的精确性。

2. 灵活性：模型在处理数据时展现了高度的灵活性。例如，在面临部分设备数据不可信或各种时间组合数量繁杂的问题时，模型能够动态地选择最佳的数个方程作为优化目标，从而减少优化运算规模。

3. 鲁棒性：模型在处理带有噪声的数据时表现出良好的鲁棒性。即使在设备记录时间存在随机误差的情况下，模型仍能通过引入时间误差修正项来提高定位精度。

6.2 模型的缺点

1. 计算复杂性：粒子群优化算法和多边测量模型等高级算法虽然提高了定位精度，但也增加了计算的复杂性。特别是在处理大量数据时，可能需要更多的计算资源和时间。

2. 数据依赖性：模型对于输入数据的准确性和完整性有较高的依赖。如果输入数据存在较大的误差或缺失，可能会影响定位结果的准确性。

3. 参数敏感性：模型的性能可能受到参数设置的影响。例如，粒子群优化算法中的参数（如粒子数量、迭代次数等）需要根据具体问题进行调整，以达到最佳效果。这可能需要一定的经验和试错过程。

参考文献

[1] 王强, 李伟, 龚建泽等. 基于火箭残骸实时定位信息的落点计算模型 [J]. 计算机测量与控制,2021,29,(5):154-158.DOI:10.16526/j.cnki.11-4762/tp.2021.05.031.

[2] 王佩, 徐珊, 熊伟等. 基于 TDOA 的弹箭残骸轨迹定位监测技术研究 [J]. 火力与指挥控制,2023,48,(11):139-144,151.DOI:10.3969/j.issn.1002-0640.2023.11.021.

[3] 王万良, 金雅文, 陈嘉诚等. 多角色多策略多目标粒子群优化算法 [J]. 浙江大学学报（工学版）,2022,56,(3):531-541.DOI:10.3785/j.issn.1008-973X.2022.03.012.

[4] 樊胜利. 一种基于 MATLAB 的节点分布仿真研究 [J]. 信息工程期刊（中英文版）,2014,4(3):57-63

[5] 段虎荣, 李闰, 陈胜雷等.RANSAC-PSO 算法的抗差反演——以芦山地震为例 [J]. 大地测量与地球动力学,2018,38,(5):454-458.DOI:10.14075/j.jgg.2018.05.003.

附录 A　单个残骸位置及音爆时间计算

```
clc;
clear;
close all;
format long;

% 初始化数据
data = [
110.241, 27.204, 824, 100.767;
110.780, 27.456, 727, 112.220;
110.712, 27.785, 742, 188.020;
110.251, 27.825, 850, 258.985;
110.524, 27.617, 786, 118.443;
110.467, 27.921, 678, 266.871;
110.047, 27.121, 575, 163.024
];

% 距离换算因子（米）
lon_dist_per_deg = 111263; % 经度每度的距离（m）
lat_dist_per_deg = 97304; % 纬度每度的距离（m）

% 初始化最小误差和最佳组合
min_error = Inf;
best_combination = [];
best_result = [];
best_v_tdoa = [];
boom_info = {};

% 音速（米/秒）
c = 340;
for j = 4:4
    % 所有可能的组合
    combinations = nchoosek(1:7, j);

    for i = 1:size(combinations, 1)
        % 初始化最小误差和最佳组合


        comb = combinations(i, :);
        if ~isequal(comb, [1,2,3,5,7])
            %continue;
        end
        disp(comb);

        selected_data = data(comb, :);
```

```
% 转换经纬度到米
x = (selected_data(:,1) - mean(selected_data(:,1))) * lon_dist_per_deg;
y = (selected_data(:,2) - mean(selected_data(:,2))) * lat_dist_per_deg;
z = selected_data(:,3);
t = selected_data(:,4);

% 计算时间差（TDOA），使用第一个基站作为相对基站
tdoa = zeros(size(selected_data, 1) - 1, 1);
for j = 2:size(selected_data, 1)
    tdoa(j-1) = t(j) - t(1);
end

% 将初始猜测设置为基站的平均值和最高的设备高程
x0 = mean(x);
y0 = mean(y);
z0 = max(z);

% 优化目标函数（只优化位置）
objective_tdoa = @(v) sum(abs(sqrt((x(2:end)-v(1)).^2 + (y(2:end)-v(2)).^2 +
    (z(2:end)-v(3)).^2) - sqrt((x(1)-v(1)).^2 + (y(1)-v(2)).^2 + (z(1)-v(3)).^2) - c *
    tdoa));

% 添加高程非负约束
lb = [-Inf, -Inf, 0];
ub = [Inf, Inf, Inf];

% 定义扰动尺度
x_scale = 10000000; % X 方向的扰动尺度
y_scale = 10000000; % Y 方向的扰动尺度
z_scale = 1000000; % Z 方向的扰动尺度（假设Z的量级较小）
number = 1000;%粒子数目

% 生成初始粒子群
initialSwarm = repmat([x0, y0, z0], number, 1) + ...
    [randn(number, 1) * x_scale, ...
    randn(number, 1) * y_scale, ...
    randn(number, 1) * z_scale];

% 粒子群优化算法设置
options = optimoptions('particleswarm', ...
    'SwarmSize', number, ...
    'InertiaRange', [0.8, 1.2], ...
    'SelfAdjustmentWeight', 0.7, ...
    'SocialAdjustmentWeight', 0.8, ...
    'MaxIterations', 1000, ...
    'MaxStallIterations', 100, ...
```

```
            'FunctionTolerance', 1e-4, ...
            'InitialSwarmMatrix', initialSwarm, ...
            'Display', 'iter');

        % 执行优化算法
        [v_tdoa, fval_tdoa, exitflag_tdoa, output_tdoa] = particleswarm(objective_tdoa, 3, lb,
            ub, options);
        error = sum(abs(sqrt((x(2:end)-v_tdoa(1)).^2 + (y(2:end)-v_tdoa(2)).^2 +
            (z(2:end)-v_tdoa(3)).^2) - sqrt((x(1)-v_tdoa(1)).^2 + (y(1)-v_tdoa(2)).^2 +
            (z(1)-v_tdoa(3)).^2) - c * tdoa));

        % 将结果转换回经纬度和高程
        final_lon_tdoa = v_tdoa(1) / lon_dist_per_deg + mean(selected_data(:,1));
        final_lat_tdoa = v_tdoa(2) / lat_dist_per_deg + mean(selected_data(:,2));
        final_z_tdoa = v_tdoa(3);

        result = computeTimeDifferences(selected_data, [final_lon_tdoa, final_lat_tdoa,
            final_z_tdoa], false)

        % 使用第二个函数计算并显示误差
        [boom_time_mean, boom_time_range, boom_time_var, boom_times] =
            calculate_boom_times(selected_data, [final_lon_tdoa, final_lat_tdoa, final_z_tdoa]);
        %error = result(end, end);

        disp(best_combination);
        fprintf('音爆发生的经度: %f°\n', final_lon_tdoa);
        fprintf('音爆发生的纬度: %f°\n', final_lat_tdoa);
        fprintf('音爆发生的高程: %f m\n', final_z_tdoa);
        fprintf('爆炸时间平均值 (boom_time_mean): %.2f\n', boom_time_mean);
        fprintf('爆炸时间极差 (boom_time_range): %.2f\n', boom_time_range);
        fprintf('爆炸时间方差 (boom_time_var): %.2f\n', boom_time_var);
        fprintf('爆炸时间数组 (boom_times):\n');

        % 更新最小误差和最佳组合
        if error < min_error
            min_error = error;
            best_combination = comb;
            best_result = result;
            best_v_tdoa = [final_lon_tdoa, final_lat_tdoa, final_z_tdoa];
            boom_info = {boom_time_mean, boom_time_range, boom_time_var, boom_times};
        end
    end
end
% 绘制最佳组合的结果
selected_data = data(best_combination, :);
x = (selected_data(:,1) - mean(selected_data(:,1))) * lon_dist_per_deg;
y = (selected_data(:,2) - mean(selected_data(:,2))) * lat_dist_per_deg;
```

```
z = selected_data(:,3);
t = selected_data(:,4);

figure;
hold on;
scatter3(x, y, z, 'filled');
plot3(best_v_tdoa(1), best_v_tdoa(2), best_v_tdoa(3), 'ro', 'MarkerSize', 15,
    'MarkerFaceColor', 'r');

xlabel('X (meters)');
ylabel('Y (meters)');
zlabel('Z (meters)');
title('监测设备和音爆位置 (TDOA)');
legend('监测设备位置', 'TDOA计算得到的音爆位置');
grid on;
view(3);
hold off;

% 显示最佳结果的相关信息
fprintf('最小误差的组合为: ');
disp(best_combination);
fprintf('音爆发生的经度: %f°\n', best_v_tdoa(1));
fprintf('音爆发生的纬度: %f°\n', best_v_tdoa(2));
fprintf('音爆发生的高程: %f m\n', best_v_tdoa(3));
fprintf('误差: %f m\n', min_error);
% 打印输出内容
fprintf('爆炸时间平均值 (boom_time_mean): %.2f\n', boom_info{1});
fprintf('爆炸时间范围 (boom_time_range): %.2f\n', boom_info{2});
fprintf('爆炸时间方差 (boom_time_var): %.2f\n', boom_info{3});
fprintf('爆炸时间数组 (boom_times):\n');
disp(boom_info{4});

result = computeTimeDifferences(selected_data, best_v_tdoa(1:3), "print")
%result = computeTimeDifferences(data, best_v_tdoa(1:3), "print")
resultg = computeDistanceErrors(data, best_v_tdoa(1:3), boom_info{1}, "print")

function stop = pswplotbestf(optimValues, state)
% 粒子群优化过程收敛情况绘制和打印最终收敛值
    stop = false;
    switch state
        case 'init'
            hold on;
            plot(optimValues.iteration, optimValues.bestfval, '.');
            title('Optimization Progress');
            xlabel('Iteration');
            ylabel('Objective Function Value');
        case 'iter'
```

```
            plot(optimValues.iteration, optimValues.bestfval, '.');
        case 'done'
            plot(optimValues.iteration, optimValues.bestfval, '.');
            hold off;
            % 打印最终收敛值
            fprintf('Optimization converged to best value: %f\n', optimValues.bestfval);
    end
end

function obj = calculate_objective(v, data)
    % 提取观测点的坐标和时间
    x = data(:, 1);
    y = data(:, 2);
    z = data(:, 3);
    tdoa = data(:, 4);


    % 声速（可以根据实际情况调整）
    c = 340; % 单位：m/s


    %%%%%%%
    boom_times = compute_boom_times(data, v);
    result = computeDistanceErrors(data, v, boom_times);

    % 计算音爆到达时间差
    time_diffs = sqrt((x(2:end) - v(1)).^2 + (y(2:end) - v(2)).^2 + (z(2:end) - v(3)).^2) - ...
                 sqrt((x(1) - v(1)).^2 + (y(1) - v(2)).^2 + (z(1) - v(3)).^2) - c * tdoa(2:end);

    % 计算音爆源时间
    source_times = sqrt((x - v(1)).^2 + (y - v(2)).^2 + (z - v(3)).^2) / c - tdoa;

    % 计算音爆源时间的方差
    variance = var(source_times);

    % 计算最终的优化目标值
    %obj = sum(abs(time_diffs)) + variance * 0;
    obj = result(end,end);
end

function boom_times = compute_boom_times(data, positions)
    % calculate_boom_times 根据数据和音爆点的位置计算音爆时间
    % data: n x 4 矩阵，前三列为测量站的xyz坐标，最后一列为测量的时间
    % positions: 1 x 3 矩阵，音爆源的xyz坐标
    % 返回音爆时间的数值向量

    % 音速，单位为m/s
```

```
    speed_of_sound = 340;

    % 从data中提取测量站坐标和测量时间
    station_coords = data(:, 1:3);
    arrival_times = data(:, 4);

    % 判断输入数据类型
    if max(abs(station_coords(:,1))) > 180 || max(abs(station_coords(:,2))) > 90
        % 输入坐标已经是米，不需要转换
        coords_in_meters = station_coords;
        pos_in_meters = positions;
    else
        % 输入坐标为经纬度，需要转换为米
        % 经纬度转换为米
        lat_to_meter = 111.263 * 1000; % 每度纬度对应的距离（米）
        lon_to_meter = 97.304 * 1000; % 每度经度对应的距离（米）

        coords_in_meters = [station_coords(:,1) * lon_to_meter, ...
                            station_coords(:,2) * lat_to_meter, ...
                            station_coords(:,3)];
        pos_in_meters = [positions(1) * lon_to_meter, ...
                         positions(2) * lat_to_meter, ...
                         positions(3)];
    end

    % 计算测量站到音爆源的距离
    distances = sqrt(sum((coords_in_meters - pos_in_meters).^2, 2));

    % 计算每个基站的音爆时间
    boom_times = arrival_times - distances / speed_of_sound;
end
```

附录 B　寻找最小距离误差

```
function result = computeDistanceErrors(data, sonicBoomPosition, boomArrivalTime, printMode)
    % Constants
    speedOfSound = 340; % m/s
    latDistance = 111.263 * 1000; % m per degree latitude
    lonDistance = 97.304 * 1000; % m per degree longitude
    mToKm = 1 / 1000; % Conversion factor from meters to kilometers

    % Extract base station data
    baseStation = data(1, :);
    latBase = baseStation(2);
    lonBase = baseStation(1);
    altBase = baseStation(3);

    % Initialize result arrays
    n = size(data, 1);
    actualDistances = zeros(1, n);
    estimatedDistances = zeros(1, n);
    distanceErrors = zeros(1, n);

    % Compute actual and estimated distances
    for i = 1:n
        % Distance from current station to sonic boom position
        latCurrent = data(i, 2);
        lonCurrent = data(i, 1);
        altCurrent = data(i, 3);
        dCurrent = sqrt(((latCurrent - sonicBoomPosition(2)) * latDistance)^2 + ...
                    ((lonCurrent - sonicBoomPosition(1)) * lonDistance)^2 + ...
                    (altCurrent - sonicBoomPosition(3))^2) * mToKm; % Convert to km
        dCurrent = abs(dCurrent);
        actualDistances(i) = abs(dCurrent);

        % Estimated distance based on arrival time
        timeDifference = boomArrivalTime - data(i, 4);
        estimatedDistance = speedOfSound * timeDifference * mToKm; % Convert to km
        estimatedDistance = abs(estimatedDistance);
        estimatedDistances(i) = estimatedDistance;

        % Calculate error
        distanceErrors(i) = abs(estimatedDistance - dCurrent);
    end

    % Sum of distances
    sumActual = sum(actualDistances);
    sumEstimated = sum(estimatedDistances);
```

```
    sumError = sum(distanceErrors);

    % Construct result matrix
    result = [actualDistances, sumActual;
              estimatedDistances, sumEstimated;
              distanceErrors, sumError];

    % Print mode
    if nargin > 3 && strcmp(printMode, 'print')
        % Display results
        disp('Actual Distances (km):');
        disp(actualDistances);
        disp(['Sum of Actual Distances (km): ', num2str(sumActual)]);

        disp('Estimated Distances (km):');
        disp(estimatedDistances);
        disp(['Sum of Estimated Distances (km): ', num2str(sumEstimated)]);

        disp('Distance Errors (km):');
        disp(distanceErrors);
        disp(['Sum of Distance Errors (km): ', num2str(sumError)]);

        % Plot results
        figure;
        bar([actualDistances', estimatedDistances', distanceErrors']);
        legend('Actual Distances', 'Estimated Distances', 'Distance Errors');
        xlabel('Observation Station Index');
        ylabel('Distance (km)');
        title('Comparison of Actual and Estimated Distances');
    end
end




% 定义数据矩阵，包含设备的经度、纬度、高程和音爆抵达时间
data = [
    110.241, 27.204, 824, 100.767;
    110.712, 27.785, 742, 188.020;
    110.524, 27.617, 786, 118.443;
    110.047, 27.121, 575, 163.024
];
data1 = [
    110.241, 27.204, 824, 100.767;
110.780, 27.456, 727, 112.220;
110.712, 27.785, 742, 188.020;
110.251, 27.825, 850, 258.985;
```

```
110.524, 27.617, 786, 118.443;
110.467, 27.921, 678, 266.871;
110.047, 27.121, 575, 163.024
];
% 音爆位置
sonicBoomPosition = [110.415, 27.336, 14080.888944];

% 时间区间 [12, 31] 以 0.01 为步长
timeRange = 12:0.000001:31;

% 初始化最小误差和对应的时间
minError = inf;
optimalTime = 0;

% 初始化数组保存每个时间点的误差总和
errors = zeros(length(timeRange), 1);

% 遍历时间区间，找到误差最小的时间
for idx = 1:length(timeRange)
    boomArrivalTime = timeRange(idx);

    % 计算当前时间下的误差
    result = computeDistanceErrors(data, sonicBoomPosition, boomArrivalTime);

    % 获取误差总和
    totalError = result(3, end);

    % 保存当前误差总和
    errors(idx) = totalError;

    % 如果当前误差小于最小误差，则更新最小误差和最优时间
    if totalError < minError
        minError = totalError;
        optimalTime = boomArrivalTime;
    end
end

% 输出结果
fprintf('最小误差: %f\n', minError);
fprintf('最优音爆到达时间: %f s\n', optimalTime);

% 绘制误差总和随时间变化的图像
figure;
plot(timeRange, errors);
xlabel('音爆时间 (s)');
ylabel('总距离误差(km)');
title('1357距离误差随音爆时间变化');
```

```
grid on;

% 显示最优时间点
hold on;
plot(optimalTime, minError, 'ro');
legend('1357距离误差', '最优时间');
hold off;
```

附录 C　多个残骸时的音爆位置及对应时间

```
clc,clear;
format long;


data = [
    110.241, 27.204, 824, 100.767, 164.229, 214.850, 270.065;
    110.783, 27.456, 727, 92.453, 112.220, 169.362, 196.583;
    110.762, 27.785, 742, 75.560, 110.696, 156.936, 188.020;
    110.251, 28.025, 850, 94.653, 141.409, 196.517, 258.985;
    110.524, 27.617, 786, 78.600, 86.216, 118.443, 126.669;
    110.467, 28.081, 678, 67.274, 166.270, 175.482, 266.871;
    110.047, 27.521, 575, 103.738, 163.024, 206.789, 210.306
];

lon_dist_per_deg = 97.304 * 1000; % 每度经度对应的距离（米）
lat_dist_per_deg = 111.263 * 1000; % 每度纬度对应的距离（米）
speed_of_sound = 340; % 音速（米/秒）

% 初始猜测
x0 = mean(data(:, 1)) * lon_dist_per_deg;
y0 = mean(data(:, 2)) * lat_dist_per_deg;
z0 = 10000;
t0 = 20;

lb = [100 * lon_dist_per_deg, 10 * lat_dist_per_deg, 0, -40];
ub = [130 * lon_dist_per_deg, 30 * lat_dist_per_deg, 20000, 40];

%lb = [-Inf, -Inf, 0, -Inf];
%ub = [Inf, Inf, Inf, Inf];

x_scale = 10000000;
y_scale = 10000000;
z_scale = 10000000;
t_scale = 10000;
number = 500;
```

```
t_range_all = {};

initialSwarm = repmat([x0, y0, z0, t0], number, 1) + ...
    [randn(number, 1) * x_scale, ...
    randn(number, 1) * y_scale, ...
    randn(number, 1) * z_scale,...
    randn(number, 1) * t_scale];

options = optimoptions('particleswarm', ...
    'SwarmSize', number, ...
    'InertiaRange', [0.8, 1.2], ...
    'SelfAdjustmentWeight', 0.7, ...
    'SocialAdjustmentWeight', 0.8, ...
    'MaxIterations', 1000, ...
    'MaxStallIterations', 100, ...
    'FunctionTolerance', 1e-4, ...
    'Display', 'iter');

for m= 1:10
    fprintf("第%d组",m);
    n = 7
    used_t = ones(7, 4);
    uesd_mar = ones(7, 4);
    last_uesd_mar = ones(7, 4);
    boom_positions = [];
    residual = [];

    for i = 1:4

        last_uesd_mar = last_uesd_mar.*uesd_mar;

        objective_tdoa = @(v) calculate_objective(v, data, last_uesd_mar, n);

        [v_tdoa, fval_tdoa, exitflag_tdoa, output_tdoa] = particleswarm(objective_tdoa, 4, lb,
            ub, options);

        dis_diff = compute_dis_diff(v_tdoa, data, speed_of_sound);
        [min_vals, min_indices] = min(dis_diff, [], 2);
        [~, sorted_indices] = sort(min_vals);
        best_indices = sorted_indices(1:n);

        used_times = false(size(dis_diff));
        used_times(sub2ind(size(dis_diff), best_indices, min_indices(best_indices))) = true;
        uesd_mar(used_times) = 100000000;
        uesd_mar(~used_times) = 1;
```

```
        boom_positions = [boom_positions; [v_tdoa, find_non_one_positions(uesd_mar)]];
        residual = [residual; fval_tdoa];
    end

    fprintf('n = %d\n', n);
    for k = 1:size(boom_positions, 1)
        final_lon = boom_positions(k, 1) / lon_dist_per_deg;
        final_lat = boom_positions(k, 2) / lat_dist_per_deg;
        final_alt = boom_positions(k, 3);
        final_time_indices = boom_positions(k, 4:end);
        fprintf('残骸 %d: 经度 %f, 纬度 %f, 高程 %f, 音爆时间索引 %s\n', ...
            k, final_lon, final_lat, final_alt, mat2str(final_time_indices));
    end
    fprintf('优化目标值 (residual):\n');
    disp(residual);
    t_range = max(boom_positions(:, 4)) - min(boom_positions(:, 4))
    t_range_all{end+1} = t_range;
end
disp(t_range_all);

function [obj, used_t] = calculate_objective(v, data, uesd_mar, n)

    % 计算距离差异矩阵

    dis_diff = compute_dis_diff(v, data, 340);

    % 应用使用矩阵惩罚
    dis_diff = dis_diff .* uesd_mar;


    % 计算每行最小的n个值之和作为优化目标
    min_vals = min(dis_diff, [], 2);
    sorted_min_vals = sort(min_vals);

    result = computeDistanceErrors(data,v(1:3),v(4));
    %obj = result(end,end);
    obj = sum(sorted_min_vals(1:n));



end

function dis_diff = compute_dis_diff(v, data, speed_of_sound)
    % 提取观测点的坐标和时间
    x = data(:, 1) * 97.304 * 1000;
    y = data(:, 2) * 111.263 * 1000;
    z = data(:, 3);
```

```
    t_all = data(:, 4:7);

    % 计算每个基站的预测音爆抵达时间
    predicted_times =v(4) + sqrt((x - v(1)).^2 + (y - v(2)).^2 + (z - v(3)).^2) /
        speed_of_sound;

    % 计算距离差异矩阵
    dis_diff = abs(predicted_times - t_all);
end

function positions = find_non_one_positions(uesd_mar)
    % 检查输入矩阵大小是否为7x4
    if size(uesd_mar, 1) ~= 7 || size(uesd_mar, 2) ~= 4
        error('输入矩阵必须是7x4大小');
    end

    % 预分配一个向量来存储每行值不为1的位置
    positions = zeros(1, 7);

    % 遍历每一行，找到值不为1的位置
    for i = 1:size(uesd_mar, 1)
        % 找到值不为1的索引
        idx = find(uesd_mar(i, :) ~= 1);

        % 假设每行有且仅有一个值不为1的元素
        if length(idx) == 1
            positions(i) = idx;
        else
            error('每行应有且仅有一个值不为1的元素');
        end
    end
end
```

附录 D　降低时间噪声

```
clc,clear;
format long;
FontName="SimSun";

data = [
    110.241, 27.204, 824, 100.767, 164.229, 214.850, 270.065;
    110.783, 27.456, 727, 92.453, 112.220, 169.362, 196.583;
    110.762, 27.785, 742, 75.560, 110.696, 156.936, 188.020;
    110.251, 28.025, 850, 94.653, 141.409, 196.517, 258.985;
```

```
    110.524, 27.617, 786, 78.600, 86.216, 118.443, 126.669;
    110.467, 28.081, 678, 67.274, 166.270, 175.482, 266.871;
    110.047, 27.521, 575, 103.738, 163.024, 206.789, 210.306
];
org_data = [
    110.241, 27.204, 824, 100.767, 164.229, 214.850, 270.065;
    110.783, 27.456, 727, 92.453, 112.220, 169.362, 196.583;
    110.762, 27.785, 742, 75.560, 110.696, 156.936, 188.020;
    110.251, 28.025, 850, 94.653, 141.409, 196.517, 258.985;
    110.524, 27.617, 786, 78.600, 86.216, 118.443, 126.669;
    110.467, 28.081, 678, 67.274, 166.270, 175.482, 266.871;
    110.047, 27.521, 575, 103.738, 163.024, 206.789, 210.306
];


lon_dist_per_deg = 97.304 * 1000; % 每度经度对应的距离（米）
lat_dist_per_deg = 111.263 * 1000; % 每度纬度对应的距离（米）
speed_of_sound = 340; % 音速（米/秒）

% 初始猜测
x0 = mean(data(:, 1)) * lon_dist_per_deg;
y0 = mean(data(:, 2)) * lat_dist_per_deg;
z0 = 10000;
t0 = 20;

lb = [100 * lon_dist_per_deg, 10 * lat_dist_per_deg, 0, -40];
ub = [130 * lon_dist_per_deg, 30 * lat_dist_per_deg, 20000, 40];

%lb = [-Inf, -Inf, 0, -Inf];
%ub = [Inf, Inf, Inf, Inf];

x_scale = 10000000;
y_scale = 10000000;
z_scale = 10000000;
t_scale = 10000;
number = 700;

t_range_all = {};

initialSwarm = repmat([x0, y0, z0, t0], number, 1) + ...
    [randn(number, 1) * x_scale, ...
    randn(number, 1) * y_scale, ...
    randn(number, 1) * z_scale,...
    randn(number, 1) * t_scale];

options = optimoptions('particleswarm', ...
    'SwarmSize', number, ...
```

```
    'InertiaRange', [0.8, 1.2], ...
    'SelfAdjustmentWeight', 0.7, ...
    'SocialAdjustmentWeight', 0.8, ...
    'MaxIterations', 1000, ...
    'MaxStallIterations', 100, ...
    'FunctionTolerance', 1e-4);

%-------------------------------------------------------------

% 标准答案
standard_boom_positions = [
    110.3, 27.65, 11477.580795, 14.0, 2, 3, 3, 2, 2, 2, 1;
    110.5, 27.31, 12513.929829, 12.0, 1, 2, 4, 4, 3, 4, 2;
    110.5, 27.95, 11528.505634, 13.0, 4, 4, 2, 1, 4, 1, 3;
    110.7, 27.65, 13468.130704, 15.0, 3, 1, 1, 3, 1, 3, 4
];
standard_boom_positions = sortrows(standard_boom_positions, 4);

% 初始化
num_iterations = 50;
results = cell(num_iterations, 1);
t_range_all = cell(num_iterations, 1);
noise_effects = cell(num_iterations, 1);


valid_results_counter = 0; % 初始化有效结果计数器
total_iterations = 0;   % 初始化总迭代计数器
results = {};           % 初始化结果存储

% 初始化存储 mean_noise 的数组
all_mean_noises = zeros(num_iterations, 1);

while valid_results_counter < num_iterations
    total_iterations = total_iterations + 1; % 增加总迭代计数器
    fprintf("第%d组\n", total_iterations);

    % 生成高斯噪声，均值为0，标准差为0.5
    noise = 0.5 * randn(size(data, 1), 4);
    abs_noise = abs(noise);
    %mean_noise = mean(mean(abs_noise));
    mean_noise = max(abs_noise(:));


    % 将高斯噪声加到后四列
    data(:, 4:7) = org_data(:, 4:7) + noise;

    n = 7;
```

```
used_t = ones(7, 4);
uesd_mar = ones(7, 4);
last_uesd_mar = ones(7, 4);
boom_positions = [];
residual = [];

for i = 1:4
    last_uesd_mar = last_uesd_mar.*uesd_mar;
    objective_tdoa = @(v) calculate_objective(v, data, last_uesd_mar, n);
    [v_tdoa, fval_tdoa, exitflag_tdoa, output_tdoa] = particleswarm(objective_tdoa, 4, lb,
        ub, options);

    dis_diff = compute_dis_diff(v_tdoa, data, speed_of_sound);
    [min_vals, min_indices] = min(dis_diff, [], 2);
    [~, sorted_indices] = sort(min_vals);
    best_indices = sorted_indices(1:n);

    used_times = false(size(dis_diff));
    used_times(sub2ind(size(dis_diff), best_indices, min_indices(best_indices))) = true;
    uesd_mar(used_times) = 100000000;
    uesd_mar(~used_times) = 1;

    boom_positions = [boom_positions; [v_tdoa, find_non_one_positions(uesd_mar)]];
    residual = [residual; fval_tdoa];
end

% 按音爆时间排序
boom_positions = sortrows(boom_positions, 4);

% 检查是否有效结果
isEqual = isequal(sortrows(boom_positions(:,5:end), 5),
    sortrows(standard_boom_positions(:,5:end), 5));
if isEqual
    disp("OOOOOOOOOOOOHHHHHHHHHHHHHHHHHHHHHHHHH!!!!!!!!!!!!!!!!!!!!!!!")
    valid_results_counter = valid_results_counter + 1; % 增加有效结果计数器
    all_mean_noises(valid_results_counter) = mean_noise; % 保存 mean_noise 的值

    % 保留计算结果
    results{valid_results_counter} = boom_positions;

    % 计算时间范围
    t_range = max(boom_positions(:, 4)) - min(boom_positions(:, 4));
    t_range_all{valid_results_counter} = t_range;

    % 计算噪声带来的影响
    noise_effects{valid_results_counter} = struct();
    for k = 1:size(boom_positions, 1)
```

```
            final_lon = boom_positions(k, 1) / lon_dist_per_deg;
            final_lat = boom_positions(k, 2) / lat_dist_per_deg;
            final_alt = boom_positions(k, 3);
            final_time_indices = boom_positions(k, 4:end);

            noise_effects{valid_results_counter}.(['residual_' num2str(k)]) =
                abs(standard_boom_positions(k, 1:3) - [final_lon, final_lat, final_alt]);
            noise_effects{valid_results_counter}.(['time_diff_' num2str(k)]) =
                abs(standard_boom_positions(k, 4:end) - final_time_indices);
        end

        fprintf('n = %d\n', n);
        for k = 1:size(boom_positions, 1)
            final_lon = boom_positions(k, 1) / lon_dist_per_deg;
            final_lat = boom_positions(k, 2) / lat_dist_per_deg;
            final_alt = boom_positions(k, 3);
            final_time_indices = boom_positions(k, 4:end);
            fprintf('残骸 %d: 经度 %f, 纬度 %f, 高程 %f, 音爆时间索引 %s\n', ...
               k, final_lon, final_lat, final_alt, mat2str(final_time_indices));
        end
        fprintf('优化目标值 (residual):\n');
        disp(residual);
    end
end

% 计算总误差的平均值并保存到txt文件中
total_errors_lon = zeros(num_iterations, 4);
total_errors_lat = zeros(num_iterations, 4);
total_errors_alt = zeros(num_iterations, 4);
total_errors_time = zeros(num_iterations, 4);

for m = 1:num_iterations
    for k = 1:4
        total_errors_lon(m, k) = noise_effects{m}.(['residual_' num2str(k)])(1);
        total_errors_lat(m, k) = noise_effects{m}.(['residual_' num2str(k)])(2);
        total_errors_alt(m, k) = noise_effects{m}.(['residual_' num2str(k)])(3);
        total_errors_time(m, k) = mean(noise_effects{m}.(['time_diff_' num2str(k)]));
    end
end

avg_error_lon = mean(total_errors_lon, 'all');
avg_error_lat = mean(total_errors_lat, 'all');
avg_error_alt = mean(total_errors_alt, 'all');
avg_error_time = mean(total_errors_time, 'all');

fileID = fopen('average_errors.txt','w');
fprintf(fileID, '平均经度误差: %f\n', avg_error_lon);
```

```
fprintf(fileID, '平均纬度误差: %f\n', avg_error_lat);
fprintf(fileID, '平均高度误差: %f\n', avg_error_alt);
fprintf(fileID, '平均时间误差: %f\n', avg_error_time);
fclose(fileID);

% 对 mean_noises 进行排序，获取排序索引
[sorted_mean_noises, sorted_indices] = sort(all_mean_noises);

% 按排序索引重新排序误差数据
sorted_total_errors_lon = total_errors_lon(sorted_indices, :);
sorted_total_errors_lat = total_errors_lat(sorted_indices, :);
sorted_total_errors_alt = total_errors_alt(sorted_indices, :);
sorted_total_errors_time = total_errors_time(sorted_indices, :);

% 计算总误差的平均值并保存到txt文件中
avg_error_lon = mean(sorted_total_errors_lon, 'all');
avg_error_lat = mean(sorted_total_errors_lat, 'all');
avg_error_alt = mean(sorted_total_errors_alt, 'all');
avg_error_time = mean(sorted_total_errors_time, 'all');

fileID = fopen('average_errors.txt','w');
fprintf(fileID, '噪声施加后平均经度误差: %f\n', avg_error_lon);
fprintf(fileID, '噪声施加后平均纬度误差: %f\n', avg_error_lat);
fprintf(fileID, '噪声施加后平均高度误差: %f\n', avg_error_alt);
fprintf(fileID, '噪声施加后平均时间误差: %f\n', avg_error_time);
fclose(fileID);

% 生成并保存图表
figure;
hold on;
colors = lines(4);
for k = 1:4
    scatter(sorted_mean_noises, sorted_total_errors_lon(:, k), [], colors(k, :), 'DisplayName',
        ['残骸 ' num2str(k) ' 经度误差']);
    fill([sorted_mean_noises', fliplr(sorted_mean_noises')], ...
        [sorted_total_errors_lon(:, k)', zeros(1, num_iterations)], ...
        colors(k, :), 'FaceAlpha', 0.1, 'EdgeColor', 'none');
end
xlabel('噪声平均强度', 'FontName', 'SimHei');
ylabel('误差 (经度)', 'FontName', 'SimHei');
title('噪声施加后经度误差随噪声平均强度的变化', 'FontName', 'SimHei');
legend('FontName', 'SimHei');
saveas(gcf, 'longitude_errors.png');

figure;
hold on;
for k = 1:4
```

```
    scatter(sorted_mean_noises, sorted_total_errors_lat(:, k), [], colors(k, :), 'DisplayName',
        ['残骸 ' num2str(k) ' 纬度误差']);
    fill([sorted_mean_noises', fliplr(sorted_mean_noises')], ...
        [sorted_total_errors_lat(:, k)', zeros(1, num_iterations)], ...
        colors(k, :), 'FaceAlpha', 0.1, 'EdgeColor', 'none');
end
xlabel('噪声平均强度', 'FontName', 'SimHei');
ylabel('误差 (纬度)', 'FontName', 'SimHei');
title('噪声施加后纬度误差随噪声平均强度的变化', 'FontName', 'SimHei');
legend('FontName', 'SimHei');
saveas(gcf, 'latitude_errors.png');

figure;
hold on;
for k = 1:4
    scatter(sorted_mean_noises, sorted_total_errors_alt(:, k), [], colors(k, :), 'DisplayName',
        ['残骸 ' num2str(k) ' 高度误差']);
    fill([sorted_mean_noises', fliplr(sorted_mean_noises')], ...
        [sorted_total_errors_alt(:, k)', zeros(1, num_iterations)], ...
        colors(k, :), 'FaceAlpha', 0.1, 'EdgeColor', 'none');
end
xlabel('噪声平均强度', 'FontName', 'SimHei');
ylabel('误差 (高度)', 'FontName', 'SimHei');
title('噪声施加后高度误差随噪声平均强度的变化', 'FontName', 'SimHei');
legend('FontName', 'SimHei');
saveas(gcf, 'altitude_errors.png');

figure;
hold on;
for k = 1:4
    scatter(sorted_mean_noises, sorted_total_errors_time(:, k), [], colors(k, :),
        'DisplayName', ['残骸 ' num2str(k) ' 音爆时间误差']);
    fill([sorted_mean_noises', fliplr(sorted_mean_noises')], ...
        [sorted_total_errors_time(:, k)', zeros(1, num_iterations)], ...
        colors(k, :), 'FaceAlpha', 0.1, 'EdgeColor', 'none');
end
xlabel('噪声平均强度', 'FontName', 'SimHei');
ylabel('误差 (音爆时间)', 'FontName', 'SimHei');
title('噪声施加后音爆时间误差随噪声平均强度的变化', 'FontName', 'SimHei');
legend('FontName', 'SimHei');
saveas(gcf, 'time_errors.png');

% 保存图片生成信息到txt文件
fileID = fopen('image_generation_info.txt','w');
fprintf(fileID, 'longitude_errors.png: 噪声施加后经度误差随噪声平均强度的变化\n');
fprintf(fileID, 'latitude_errors.png: 噪声施加后纬度误差随噪声平均强度的变化\n');
fprintf(fileID, 'altitude_errors.png: 噪声施加后高度误差随噪声平均强度的变化\n');
```

```
fprintf(fileID, 'time_errors.png: 噪声施加后音爆时间误差随噪声平均强度的变化\n');
fclose(fileID);

function [obj, used_t] = calculate_objective(v, data, uesd_mar, n)

    % 计算距离差异矩阵

    dis_diff = compute_dis_diff(v, data, 340);

    % 应用使用矩阵惩罚
    dis_diff = dis_diff .* uesd_mar;


    % 计算每行最小的n个值之和作为优化目标
    min_vals = min(dis_diff, [], 2);
    sorted_min_vals = sort(min_vals);

    result = computeDistanceErrors(data,v(1:3),v(4));
    %obj = result(end,end);
    obj = sum(sorted_min_vals(1:n).^2);



end

function dis_diff = compute_dis_diff(v, data, speed_of_sound)
    % 提取观测点的坐标和时间
    x = data(:, 1) * 97.304 * 1000;
    y = data(:, 2) * 111.263 * 1000;
    z = data(:, 3);
    t_all = data(:, 4:7);

    % 计算每个基站的预测音爆抵达时间
    predicted_times =v(4) + sqrt((x - v(1)).^2 + (y - v(2)).^2 + (z - v(3)).^2) /
        speed_of_sound;

    % 计算距离差异矩阵
    dis_diff = abs(predicted_times - t_all);
end

function positions = find_non_one_positions(uesd_mar)
    % 检查输入矩阵大小是否为7x4
    if size(uesd_mar, 1) ~= 7 || size(uesd_mar, 2) ~= 4
        error('输入矩阵必须是7x4大小');
    end

    % 预分配一个向量来存储每行值不为1的位置
```

```
    positions = zeros(1, 7);

    % 遍历每一行，找到值不为1的位置
    for i = 1:size(uesd_mar, 1)
        % 找到值不为1的索引
        idx = find(uesd_mar(i, :) ~= 1);

        % 假设每行有且仅有一个值不为1的元素
        if length(idx) == 1
            positions(i) = idx;
        else
            error('每行应有且仅有一个值不为1的元素');
        end
    end
end
```

附录 E　TDOA 原理可视化图

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用SimHei字体
plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题

# 设置三维图形
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# 定义传感器和信号源的坐标
sensors = np.array([[0, 0, 0], [10, 0, 0], [5, 10, 0], [5, 5, 10]])
source = np.array([6, 6, 6])

# 缩小范围
ax.set_xlim(-5, 15)
ax.set_ylim(-5, 15)
ax.set_zlim(-5, 15)

# 绘制传感器位置
ax.scatter(sensors[:, 0], sensors[:, 1], sensors[:, 2], c='b', marker='o', s=100, label='设备')

# 标注传感器
for i, sensor in enumerate(sensors):
    ax.text(sensor[0], sensor[1], sensor[2], f'P{i+1}', color='blue')
```

```
# 绘制信号源位置
ax.scatter(source[0], source[1], source[2], c='r', marker='x', s=100, label='残骸')

# 标注信号源
ax.text(source[0], source[1], source[2], 'S', color='red')

# 绘制传感器到信号源的连线
for sensor in sensors:
    ax.plot([sensor[0], source[0]], [sensor[1], source[1]], [sensor[2], source[2]], 'k--')

# 设置坐标轴标签
ax.set_xlabel('X轴')
ax.set_ylabel('Y轴')
ax.set_zlabel('Z轴')

# 添加图例
ax.legend()

# 设置图形标题
ax.set_title('TDOA算法示意图')

# 定义双曲面函数
def hyperboloid(a, b, c, x0, y0, z0, u_range, v_range):
    u = np.linspace(u_range[0], u_range[1], 100)
    v = np.linspace(v_range[0], v_range[1], 100)
    u, v = np.meshgrid(u, v)
    x = x0 + a * np.cosh(u) * np.cos(v)
    y = y0 + b * np.cosh(u) * np.sin(v)
    z = z0 + c * np.sinh(u)
    return x, y, z

# 定义颜色列表
colors = ['green', 'magenta', 'cyan', 'orange', 'purple', 'yellow']

# 绘制双曲面
color_index = 0
for i in range(len(sensors)):
    for j in range(i + 1, len(sensors)):
        sensor1 = sensors[i]
        sensor2 = sensors[j]
        mid_point = (sensor1 + sensor2) / 2
        distance = np.linalg.norm(sensor1 - sensor2) / 2
        x, y, z = hyperboloid(distance/2, distance/2, distance/2, mid_point[0], mid_point[1],
            mid_point[2], (-2, 2), (0, 2 * np.pi))
        ax.plot_surface(x, y, z, alpha=0.1, color=colors[color_index % len(colors)])
        color_index += 1
```

```
# 调整视角
ax.view_init(elev=30, azim=30)

# 添加文本注释，调整位置
ax.text2D(0.0, 0.95, "一共四个基站，所以使用TDOA算法会产生六个双曲面", transform=ax.transAxes,
    fontsize=9, color='black', bbox=dict(facecolor='white', alpha=0.8))

plt.show()
```

附录 F　多边测量法可视化

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# 设置中文字体
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei'] # 使用Microsoft YaHei字体
plt.rcParams['axes.unicode_minus'] = False # 解决负号显示问题

# 定义锚点和目标位置
anchors = np.array([[0, 0, 0], [3, 0, 0], [0, 4, 0], [4, 4, 0]])
target = np.array([2, 3, 1])

# 创建一个三维图形
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

# 绘制锚点和目标位置
ax.scatter(anchors[:, 0], anchors[:, 1], anchors[:, 2], c='b', marker='o', label='设备')
ax.scatter(target[0], target[1], target[2], c='r', marker='x', s=100, label='残骸')

# 绘制锚点到目标位置的连线
for anchor in anchors:
    ax.plot([anchor[0], target[0]], [anchor[1], target[1]], [anchor[2], target[2]], 'k--',
        alpha=0.5)

# 绘制球面表示测量的范围
colors = ['green', 'magenta', 'cyan', 'orange']
for i, anchor in enumerate(anchors):
    distance = np.linalg.norm(anchor - target)
    u = np.linspace(0, 2 * np.pi, 100)
    v = np.linspace(0, np.pi, 100)
    x = distance * np.outer(np.cos(u), np.sin(v)) + anchor[0]
    y = distance * np.outer(np.sin(u), np.sin(v)) + anchor[1]
```

```
    z = distance * np.outer(np.ones(np.size(u)), np.cos(v)) + anchor[2]
    ax.plot_surface(x, y, z, color=colors[i], alpha=0.2)

# 设置坐标轴标签
ax.set_xlabel('X轴')
ax.set_ylabel('Y轴')
ax.set_zlabel('Z轴')

# 添加图例和标题
ax.legend()
ax.set_title('多边测量法原理示意图')

# 调整视角
ax.view_init(elev=30, azim=30)

plt.show()
```