基于信赖域反射算法的火箭残骸精准定位模型

摘　要

21 世纪以来，随着科技的高速发展，我国至 2023 年就已成功实施了 67 次发射任务，共发射了 221 个航天器，航天器发射随着时代发展已经逐渐成为一种普遍性活动，特别是在航天探索、通信卫星部署、科学研究以及军事应用等领域更为明显。火箭产生的残骸会对人类活动产生其他影响，所以我们需要对残骸进行定位求解。

问题一中，我们利用距离与时间差公式，可得到残骸坐标的关系式，七台设备便提供了七个非线性方程，由于仅有四个未知数，我们仅需四台设备便能实现对残骸爆炸点的求解。我们建立的目标函数为距离差减去时间差乘以声速的平方和，能很好的转化为求最小化优化问题，本题中我们利用了梯度下降法来对目标函数进行寻优求解，取各监测设备经纬度、高程的均值作为其初始，时间的均值取为零。得到了第一问残骸的坐标为**(110.615929°,27.144525°，1472.202286m，0)**分别对应经度，维度，高程以及爆炸时间。误差均值为 **4.485428310363489km**。

问题二中，我们通过第一问可知，**四台**设备便可求解一个残骸，多个残骸也是相同的原理，所以**四台**检测设备便可以进行对多残骸爆炸定位。四台设备四个残骸可以得出 16 个非线性方程，随后构建数学模型实现对多个设备进行分类处理。通过分析分析，如果正确甄别出每个数据的归属，而我们得到的目标函数值应当是所有可能归属中误差最小的，于是我们建立了以总误差最小，每个数据只能归属于一个残骸，每个残骸有 7 个数据，每台设备都有四个数据为约束条件的优化模型。在优化过程中，利用了信赖域反射算法（TRR）进行优化求解，在验证模型中使用了两个残骸，分别为问题一中的残骸以及以各监测设备的经纬度、高程的均值和时间 0 作为初值创建的残骸，并计算其音爆抵达时间，进行了数据分组求解，可以证明模型二与模型一均具有较优秀的执行能力。

问题三中，我们沿用问题二中建立的模型，实现对问题三提供的数据进行分组求解处理，最终我们得到了各个残骸之间的数据分布，分别为残骸 1(A1,B2,C4,D4,E3,F4,G2),残骸 2(A2,B3,C3,D2,E2,F2,G1),残骸 3(A3,B1,C1,D3,E1,F3,G4),残骸 4(A4,B4,C2,D1,E4,F1,G3)。并求出各个残骸的位置数据，详见下文。

问题四中，要求我们在加入随机误差下，也能保持高精度的坐标求解，首先我们对问题三中时间抵达数据加入了满足均值为 0，标准差为 0.5 的正态分布的随机误差。由于 TRR 算法可以解决非线性最小二乘问题，包括非线性数据拟合、优化等问题。它具有较高的收敛速度和稳定性，尤其适用于高维问题。所以本题我们仍然利用 TRR 算法进行优化，我们首先先以每个设备作为观测点，求解出各个设备的修正系数，随后加入模型二中的目标函数中，带入第三问数据进行求解，得到所有数据均满足题目需求，最后所得误差在理想范围以内，模型较为成功。

关键词：定位算法 梯度下降法 多目标优化 TRR 算法

一、 问题重述[i]

1.1 问题背景

21 世纪以来，随着科技的进步，火箭发射在世界范围的活动已逐渐趋于普遍，特别是在航天探索、通信卫星部署、科学研究以及军事应用等领域更为显著。绝大多数火箭为多级火箭，下面级火箭或助推器完成既定任务后，通过级间分离装置分离后坠落[1]。在坠落至地面过程中，残骸会产生跨音速音爆。准确定位残骸可以避免残骸对地面设施和人员造成伤害，确保公共安全。为了快速回收残骸，需要定位残骸精确位置，在残骸理论落区，通常会设置多台检测设备，通过监测设备接收到的音爆信号来定位残骸在空中发生音爆时的位置，进而实现残骸落地点的快速精准定位。

1.2 问题重述

问题一：题目已给出 7 台设备的经度，维度，高程和音爆抵达时间。要求分析要定位空中残骸爆炸点的位置所需的设备数量，并且需要确定一个模型，通过监测设备接收到的音爆信号来精确定位单个火箭残骸在空中发生音爆时的位置（经度、纬度和高度）和时间。

问题二：在考虑火箭残骸包括一级残骸和多个助推器的情况下，监测设备可能会同时接收到多个音爆信号。需要建立一个数学模型来分析如何区分这些音爆信号分别来自哪一个残骸，以及确定至少需要多少台监测设备来准确定位四个残骸在空中发生音爆的位置和时间。

问题三：给定了监测设备的坐标和四个不同时间点的音爆抵达时间，要求使用问题 2 中建立的数学模型来确定四个残骸在空中发生音爆的确切位置和时间。这些残骸产生音爆的时间可能不同，但时间差不超过 5 秒。

问题四：考虑到监测设备记录时间可能存在 0.5 秒的随机误差，需要修正问题 2 中的模型以更精确地确定残骸在空中发生音爆的位置和时间。需要通过模拟数据叠加随机误差来验证修正模型的效果，并在时间误差无法降低的情况下，提出一种解决方案来实现残骸空中的精准定位，并验证模型的有效性。

二、 问题分析

2.1 问题一的分析

问题一中要求我们选择合适的数据，计算残骸发生音爆的位置和时间。题目给出了七台设备的经度，纬度，高程和音爆抵达时间，我们假设残骸的四个参数为（x，y，z，t），分别为残骸爆炸的经度，维度，高程和音爆时间，我们利用了多边测量法来建立各个量之间的联系，即距离等于声速乘以某一设备音爆抵达时间减去音爆时间的差，可列出 7 个非线性方程组，根据算法思路，我们可以知道我们至少需要 4 台设备来检测才能得到其坐标。我们可以利用了梯度下降来求出改方程数值解，最终求出每个设备的误差，即为设备到残骸的距离除以声速减去音爆抵达时间加上音爆时间，确认精度。

2.2 问题二的分析

问题二要求我们确定 4 组震动波分别来自于哪个残骸，与第一题类似，该题我们仍然至少需要四个震动波检测器才能确认四个残骸的位置，其本质是找到声波数据可能的排列使得整体的误差最小。这道题与上道题不同的点在于我们并不知道每组数据是属于哪个残骸的，如果考虑全部数据，依次遍历是不太现实的，所以我们可以采取的方法为任取四组的数据进行排列组合求解使得整体误差最小的那组合，这样可以得到一组合理数据，随后利用得到的分组数据对剩余三组设备进行检验。

2.3 问题三的分析

利用问题二中建立的震动波区分模型对问题三中的数据进行处理，再利用问题一的模型对同一残骸的数据进行建立关系并处理求解，对得到的数据进行检验分析，再观察时间差是否都小于题目要求的 5s 即可。

2.4 问题四的分析

问题四在问题一，问题二的基础上加入了随机误差，我们需要找到一种改进算法，使我们尽可能的减小时间误差，并且尽可能把距离误差减小到 1km 以下。由于存在 0.5s 的随机误差，该误差无法判断是提早测量还是延后测量，所以我们随机多次计算，并取多次模拟的结果求平均为修正系数。利用 TRR 优化算法进行对加入修正系数的目标函数进行寻优求解，得出结果。

三、 模型假设

3.1 模型基本假设

针对本文问题，以及我们选用的模型，我们做出了如下假设：

(1) 震动波在传播过程中速度不变，为 340 m/s；

(2) 残骸爆炸点与各个设备的坐标足够接近，无需考虑地球曲率；

(3) 设备不会接收除了残骸爆炸引起的震动波以外的声波；

(4) 震动波从残骸爆炸点到设备接收的时间内，地球是不自转的，对数据没有影响。

四、 符号说明

符号	说明	单位
v	震动波传播速度	km/s
x_i	经度在笛卡尔坐标系中坐标	km
y_i	维度在笛卡尔坐标系中坐标	km
z_i	高程在笛卡尔坐标系中坐标	km
t_i	设备接收音爆时间	s
d_{ijk}	设备与检测器之间的欧式距离	km
err^2	求解后的距离误差	km
P_{ijk}	说明第 i 个设备的第 j 个时间数据是否属于第 k 个残骸的 0-1 变量	
ε_{ij}	设备时间误差	s
d_i	第 i 个设备的修正系数	km

表 四-1

五、 模型的建立、求解与评估

5.1 问题一模型的建立与求解

5.1.1 模型的建立

该题要求我们精准定位空中单个残骸发生音爆的位置和音爆时间，先对各个设备进行数据处理，将经纬度坐标系下坐标转化为笛卡尔坐标系中的坐标，假设各个设备的坐标处理后为(x_i, y_i, z_i, t_i)。设爆炸点坐标为(x, y, z, t)。

我们可以利用距离速度方程建立各个变量之间关系：

$$d_i = v * (t_i - t) \tag{5.1.1.1}$$

利用欧式距离公式，我们可以得到各个设备到爆炸点的距离 di 的公式：

$$d_i=\sqrt{(x_i-x)^2+(y_i-y)^2+(z_i-z)^2}\quad i=1,2,...,7 \tag{5.1.1.2}$$

我们可以将上式整理为：

$$v*t_i=d_i+v*t\quad i=1,2,...,7 \tag{5.1.1.3}$$

其为更容易拟合的形式，上述形式的左边为已知量，未知量在左侧，于是变成了一个容易计算的式子，我们只需要求解这 7 个非线性方程即能得到结果。

实际上，求出来的解为数值解，故每个设备都有一定误差，使得模型总误差最小，于是建立了对应的目标函数：

$$min\sum_{i=1}^{7}[\sqrt{(x_i-x)^2+(y_i-y)^2+(z_i-z)^2}-v*(t_i-t)]^2 \tag{5.1.1.4}$$

此处我们利用梯度下降法[2]来求解该非线性方程。其原理如下：

有目标函数 f(x),且 f(x)是 R4 上的连续可微函数，要求解的无约束最优化问题是，求得满足式(5.1.1.4)的(x,y,z,t)。而梯度下降法是一种迭代算法，我们构造一个序列 $x^{(0)},x^{(1)},x^{(2)},...$ 有 $f(x^{(k+1)})<f(x^{(k)}),k=0,1,2,...$只要上述过程不断执行下去，总能收敛到局部最小值，这就要求我们取多个初值，求得全局最优解。

以下为一个操作步骤：

首先，输入了目标函数 f(x)，梯度函数 g(x)= $\nabla f(x)$ 以及计算精度 ε 。

k=0,初始化 $x^{(0)}\in R^n$;

计算 f(x(k));

计算梯度函数， $\nabla f(x^{(k)})$ 当梯度下降的距离 $\|\nabla f(x^{(k)})\|<\varepsilon$，停止迭代,则可令 x*=x（k）；

如果没有能满足的条件，则要求λk，其满足：

$$f(x^{(k)}-\lambda_k\nabla f(x^{(k)}))=minf(x^{(k)}-\lambda_k\nabla f(x^{(k)}))(\lambda\geq 0) \tag{5.1.1.5}$$

令 $x^{(k+1)}=x^{(k)}-\lambda_k\nabla f(x^{(k)})$,并计算 $f(x^{(k+1)})$,当$\|f(x^{(k+1)})-f(x^{(k)})\|\leq\varepsilon$，或者$\|x^{(k+1)}-x^{(k)}\|\leq\varepsilon$时，停止迭代，令 $x^*=x^{(k+1)}$。

令 k=k+1，并返回步骤(3)。

从上式中可以发现迭代的三个停止条件：

（1） 梯度小于指定阈值；

（2） 目标函数变化小于指定阈值；

（3） 迭代变量 x 小于指定阈值；

值得注意的是，当目标函数是凸函数时，梯度下降法的解是全局最优解，这是没有问题的。但是在很多情况下，我们通过梯度下降到达的不一定是全局最低点，而是

局部最低点，所以我们要取多个初始点观察情况，找到适合本题的最优解。

5.1.2 模型的求解

在问题一中，我们取 x(0)=(mean(x_i),mean(y_i),mean(z_i),0)为初值，$\varepsilon=10^{-9}$，利用梯度下降法对式(4)进行求解。

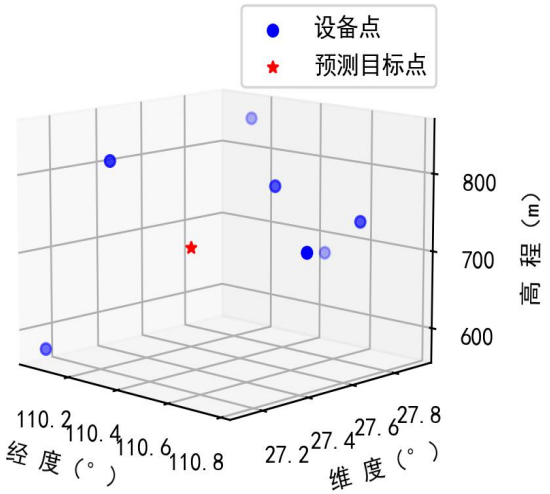

图 五-1 可视化经纬度

得到结果为：经度：110.615929 ° 维度：27.144525 ° 高程 1472.202286m 爆炸时间为 0。接下来我们对结果进行了误差分析，求出了每个设备的误差如下图所示：

设备	误差(km)
A	2.8222570403898573
B	0.808569052732951832
C	7.948605808986535
D	-4.427523082284381
E	13.85838246113045
F	-3.1326607268921265
G	-1.2756373735101079e-10

表 五-1

以下为随着迭代次数 Error 的变化：

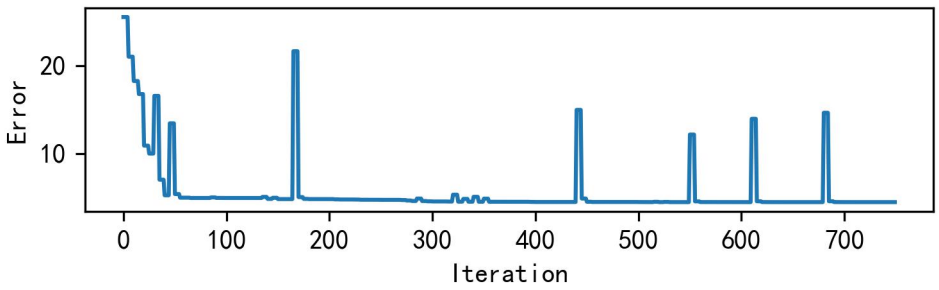

图 五-2 Iteration-error 图

观察误差数据可知，其大部分设备误差较小，仅有 C 与 E 设备误差较大，对于 7

个误差求出其均值为 4.485428310363489km，标准差为 4.313598912088789km,较为合理,我们认为是可以接收的误差。

5.2 问题二模型的建立与求解

5.2.1 模型的建立

同问题一，根据距离时间关系有：

$$d_{ijk}=v*(t_{ij}-t_k) \tag{5.2.1.1}$$

根据欧式距离公式，有：

$$d_{ijk}=\sqrt{(x_{ij}-x_k)^2+(y_{ij}-y_k)^2+(z_{ij}-z_k)^2} \quad i=1,2,\cdots,7 \ \ j,k=1,2,3 \tag{5.2.1.2}$$

d_{ijk}表示第 i 个监测器的第 j 个音爆接收时间与第 k 个残骸的距离，x_k，y_k，z_k，t_k表示第 k 个残骸发生音爆的经度、维度、高程和音爆发生时间，对于每一个残骸，可列的 28 个方程中，只有 7 个，且每个监测器仅提供一个时间数据，可知误差平方为：

$$err^2=(d_{ijk}-v*t_{ij}+v*t_k)^2 \tag{5.2.1.3}$$

设变量 P_{ijk} 是 0-1 变量，当第 i 个设备第 j 个时间为第 k 个残骸对应的爆炸接受时间时，p_{ijk}为 1，否之为 0，所以我们可列目标函数[3]：

$$\min\sum_{i=1}^{7}\sum_{j=1}^{4}\sum_{k=1}^{4}err_{ijk}^2*P_{ijk} \tag{5.2.1.4}$$

有以下约束条件：

$$\sum_{j=1}^{4}P_{ijk}=1 \ \ i=1,2,\cdots,7 \quad k=1,2,3,4 \tag{5.2.1.5}$$

该约束表示，第 k 个残骸的震动波只会被一台设备接收一次，产生一个音爆接收时间。

$$\sum_{k=1}^{4}\sum_{j=1}^{4}P_{ijk}=4 \ \ ,k=1,2,3,4 \tag{5.2.1.6}$$

该约束表示，每个设备都会接收到四个不同的声音抵达时间。

综合上述，可列出区分残骸模型的目标函数和约束条件为：

$$min\sum_{i=1}^{7}\sum_{j=1}^{4}\sum_{k=1}^{4}\left(v*t_{ij}-\sqrt{(x_{ij}-x_k)^2+(y_{ij}-y_k)^2+(z_{ij}-z_k)^2}-v*t_k\right)^2*P_{ijk} \tag{5.2.1.7}$$

$$s.t.\{\sum_{j=1}^{4} P_{ijk}=1 \quad i=1,2,\cdots,7;\ k=1,2,3,4$$

$$\sum_{j=1}^{4}\sum_{k=1}^{4} P_{ijk}=4 \quad ,k=1,2,3,4$$

为了解决该非线性问题，我们利用了信赖域反射算法(trust-region-reflective,下文简称 TRR) TRR 方法是一种有效的优化算法，它特别适用于解决具有复杂约束和非线性特性的优化问题[4]。

TRR 方法的核心理念是：初始设定一个称为“信赖域半径”的参数，该参数定义了从当前迭代点出发的搜索范围的最大半径，并以此构建一个以当前点为中心，以该半径为界限的闭球形状的“信赖域”。在这个信赖域内，通过求解目标函数的二次近似问题，也就是信赖域子问题，来寻找最优的“候选移动”方向。如果这个候选移动能够显著降低目标函数的值，则接受这一移动作为实际的迭代步骤，并根据情况保持或扩展信赖域半径，以便进行下一轮迭代。反之，如果候选移动未能实现目标函数的足够下降，则表明二次近似模型与实际目标函数之间存在较大偏差，需要缩小信赖域半径，重新求解信赖域子问题以获得新的候选移动。这个过程将持续进行，直到达到某个停止准则为止。

针对最小化优化问题：

$$min \quad f(x) \quad x\in R^n \tag{5.2.1.8}$$

我们设 x_k 是第 k 次迭代点，记 $f_k=f(x_k),g_k=\nabla f(x_k),B_k$ 是 Hesse 矩阵$\nabla^2 f(x_k)$的第 k 次近似，则第 k 次迭代的信赖域子问题具有如下形式：

$$\begin{aligned} &min\ q_k(d)=g_k^T d+d^T B_k d/2, \\ &s.t.\|d\|\le \triangle_k \end{aligned} \tag{5.2.1.9}$$

其中△k 是信赖域半径，‖·‖是任意一种向量范数，在这我们取 2-范数，设子问题的最优解为 d_k，定义$\triangle f_k$ 为 f 在第 k 步的实际下降量：

$$\triangle f_k=f_k-f(x_k-d_k), \tag{5.2.1.10}$$

△q_k 为对应的预测下降量：

$$\triangle q_k=q_k(0)-q_k(d_k) \tag{5.2.1.11}$$

再定义它们的比值为：

$$r_k = \triangle f_k / \triangle q_k \tag{5.2.1.12}$$

一般的，我们有$\triangle q_k>0$。因此，若$r_k<0$，则$\triangle f_k<0, x_k+d_k$不能作为下一个迭代点，需要缩小信赖域半径重新求解子问题，若r_k比较接近 1，说明二次模型与目标函数在信赖域的范围内有很好的近似，此时$x_{k+1}=x_k+d_k$可以作为新的迭代点，同时下一次迭代时可以增加信赖域半径，对于其他情况，信赖域半径可以保持不变。

其算法步骤为，信赖域方法：

(1)选取初始参数 $0\le\eta_1<\eta_2<1, 0<\tau_1<1<\tau_2, 0\le\varepsilon\ll 1.\ x_0\in R^n.$

(2)取定$\tilde{\Delta}>0$为信赖域半径的上限，初始信赖域半径$\triangle 0\in(0,\Delta]$,令 k=0。

(3)计算$g_k=\nabla f(x_k)$，若$\|g_k\|\le\varepsilon$，停止迭代。

(4)求解子问题的解d_k。

(5)按式(17)计算r_k的值。

(6)校正信赖域半径

$$\triangle_{k+1}=\begin{cases} \tau_1\triangle_k, & 若 r_k\le\eta_1, \\ \triangle_k, & 若 \eta_1<r_k<\eta_2, \\ \min\{\tau_2\triangle_k, \tilde{\Delta}\}, & 若 r_k\ge\eta_2, \|d_k\|=\triangle_k \end{cases} \tag{5.2.1.13}$$

若$r_k>\eta_1$，则令$x_{k+1}=x_k+d_k$，更新B_k到B_{k+1}，令 k=k+1，转(2)；否则$x_{k+1}=x_k$，令 k=k+1，转(3)。

对于子问题，有求解子问题的光滑牛顿法：

选取δ，$\sigma\in(0,1), \mu_0>0, \lambda_0\ge 0. d_0\in R^n$，置$z_0=(\mu_0,\lambda_0,d_0)$，$\tilde{z}=(\mu_0,0,0)$

选取$\gamma\in(0,1)$使$\gamma\mu_0<1$及$\gamma\|H(z_0)\|<1$。令 j=0。

如果$\|H(z_j)\|=0$,算法终止；否则，计算$\beta_j=\beta(z_j)$。

求解下列方程组解的$\triangle z_j=$（$\triangle\mu_j, \triangle\lambda_j, \triangle d_j$）

$$H(z_j)+H'(z_j)\triangle z_j=\beta_j\tilde{z} \tag{5.2.1.14}$$

设 mj 为满足下式的最小非负整数：

$$\|H(z_j+\delta^{m_j}\triangle z_j)\|\leq[1-\sigma(1-\beta\mu_0)\delta^{m_j}]\|H(z_j)\| \tag{5.2.1.15}$$

令 $\alpha_j=\delta^{m_j}$，$z_{j+1}=z_j+\alpha_j\triangle z_j$

令 j=j+1，转(1)。

利用上述方法可以用来求解式(12),得到合理分组的 P_{ijk}，当 P_{ijk} 为 1 时，表示第 i 个设备的第 j 个数据为第 k 个残骸。

5.2.2 模型的求解

我们想要求解问题二建立模型的有效性，以两组数据进行测试，一组是问题一残骸的音爆抵达时间，第二组为我们随意编织的一组数据，其经纬度，高程，与各个设备之间距离都较小，方便求解。最后将各个设备到残骸二的位置除以声速加上假设的残骸二音爆时间，即为残骸 2 音爆抵达各个设备的音爆抵达时间。数据齐全，利用建立的模型进行求解。

设残骸 2 坐标为(110.431713,27.561286,740.285714,0) ,由上可给出残骸 2 音爆传播到各个设备时，各个设备的音爆抵达时间为:

设备	残骸 2 音爆抵达时间(s)	残骸 1 音爆抵达时间(s)
A	129.03037222381494	100.767
B	105.4659678246523	112.220
C	108.60289456731553	188.020
D	100.60793450808694	258.985
E	32.09654495085216	118.443
F	118.14714192497237	266.871
G	181.33073749415746	163.024

图 五-3

利用模型二对这两组数据进行求解之后，发现能很好的分组，其中残骸 2 的音爆抵达时间分组为(A1,B1,C1,D1,F1,G1),残骸 1 的音爆抵达时间分组为(A2,B2,C2,D2,E2,F2,G2),这与我们初始分组一样，说明了本模型对于残骸区分的实用性，我们已知了残骸 1 的经度：110.615929，维度：27.144525°，高程 1472.202286m，爆炸时间为 0，通过求解得出残骸 2 的经度为 110.431695°，维度：27.561282°，高程：740.28031m，爆炸时间：0.001447s。

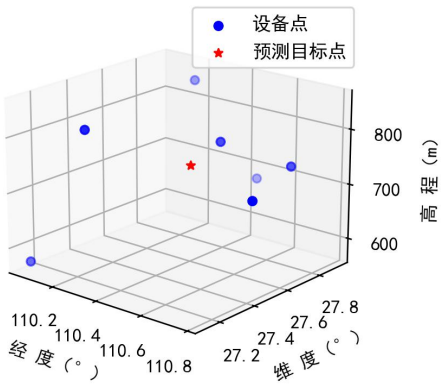

图 五-4 残骸 2 的经纬度可视化

其与各个设备之间的误差为(-3.418723792947276e-08,-0.0005218743149058014,0.002160524304521516,-0.0001077382725114262,4.0729908334924403e-10,-3.3623379636082973e-09,7.545053021829862e-05)，均值为：0.0004093758541474301km，0.0007352747635440498km，很好的说明了模型二能很好的区分不同残骸音爆，且证明了问题一对定位求解的精确性能。

5.3 问题三模型的建立与求解

5.3.1 模型的建立

问题三的模型以问题二的模型为基础，利用问题二的算法计算得到各个残骸的对应的监测设备的接收时间，然后将这些接收时间代入问题一的模型即可得到残骸的空间坐标、音爆时间以及误差，然后对误差进行分析，以期满足题目的要求。

根据题目要求以及第二建立的模型，我们可以建出残骸区分模型的目标函数：

$$min\sum_{i=1}^{7}\sum_{j=1}^{4}\sum_{k=1}^{4}(v*t_{ij}-\sqrt{(x_{ij}-x_k)^2+(y_{ij}-y_k)^2+(z_{ij}-z_k)^2}-v*t_k)$$

$$s.t.\{\sum_{j=1}^{4}P_{ijk}=1 \quad i=1,2,\cdots,7;\ k=1,2,3,4 \tag{5.3.1.1}$$

$$\sum_{j=1}^{4}\sum_{k=1}^{4}P_{ijk}=4$$

$$|t_{k1}-t_{k2}|\le 5$$

其中，t_{k1}，t_{k2} 表示各个残骸音爆时间，该约束表示每个设备有且仅有一个时间属于某一残骸，而残骸音爆发出的震动波在每个设备都有且仅有接收一次，共七次，且每个设备都有 4 个时间抵达时间，每个残骸音爆时间之间的差距小于等于 5s，满足题意。

5.3.2 模型的求解

利用模型三对问题三提供的数据进行求解，得到了各个残骸的对应音爆抵达时间如下图所示：

设备	残骸 1(s)	残骸 2(s)	残骸 3(s)	残骸 4(s)
A	100.767	164.229	214.850	270.065
B	112.220	169.362	92.453	196.583
C	188.020	156.936	75.560	110.696
D	258.985	141.409	196.517	94.653
E	118.443	86.216	78.600	126.669
F	266.871	166.270	175.482	67.274

G	163.024	103.738	210.306	206.789

表 五-2

得到了各个残骸音爆到达各个设备的音爆抵达时间，对其进行残骸定位得到各个残骸的坐标，如下表：

残骸	经度(°)	纬度(°)	高程(m)	音爆时间(s)
1	110.50000713	27.30998717	-10.86329791	12.11240132
2	110.29999956	27.65000005	11.47788504	14.00002058
3	110.69999873	27.65000009	13.46816555	14.99996543
4	110.4999985	27.94999754	11.52885651	13.00142566

表 五-3

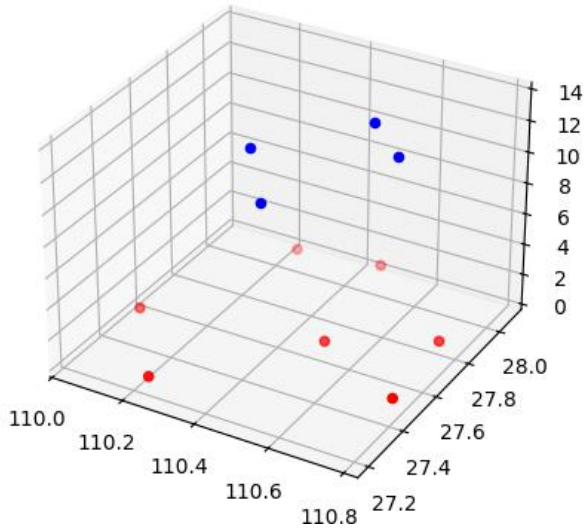

图 五-5 四个残骸的经纬度可视化

有误差图如下：

残骸	1	2	3	4
误差(km)	0.01022076859 4266053	5.75883523210 44796e-08	1.45286328556 2085e-08	1.56115942003 1689e-08

表 五-4

综合上述求解可知，本模型的精度极高，且满足题目要求，是个有效精确的结果。

5.4 问题四模型的建立与求解

5.4.1 模型的建立

问题四中要求我们求解带有随机误差的定位问题，已知设备记录时间存在 0.5 s 的随机误差，故有以下关系存在：

$$t'_{ij}=t_{ij}+\varepsilon_{ij} \quad i=1,2,\cdots,7;j=1,2,3,4 \tag{5.4.1.1}$$

t'_{ij}为第 i 个设备记录的第 j 个数据的时间，它与真实时间 t_{ij}中存在一个加性误差

ε_{ij},根据题目信息，我们可设误差 $\varepsilon_{ij}\sim N(0,0.5)$,可列距离时间关系式：

$$d'_{ijk}=v*(t_{ij}-t_k)=v*(t'_{ij}-\varepsilon_{ij}-t_k) \tag{5.4.1.2}$$

将式(7)与式(23)联合在一起为：

$$\sqrt{(x_{ij}-x_k)^2+(y_{ij}-y_k)^2+(z_{ij}-z_k)^2}=v*(t'_{ij}-\varepsilon_{ij}-t_k) \tag{5.4.1.3}$$

其中各变量与问题二中相同，不再赘述。

接下来求解修正系数，其算式为：

$$\xi_i=\left(\sum_{j=1,j!=i}^{7}(d_{ijk}+v*\varepsilon_{ij}-\sqrt{(x_j-x'_i)^2+(y_j-y'_i)^2+(z_j-z'_i)^2})\right)/6$$
$$d_i=mean(\xi_i) \tag{5.4.1.4}$$

可算出每个设备的修正系数，d_i 表示第 i 个监测站的修正系数，其中 x'_i, y'_i, z'_i, t'_i是在算法求解附加误差后的修正参数，ξ_i附加后第 i 个监测站理论距离与真实距离服从的分布，d_i 是在算法求解附加误差后的修正系数，为附加后第 i 个监测站理论距离与真实距离服从的分布，

综合上述模型以及本题分析，我们能得出最终的目标函数以及约束：

$$min\sum_{i=1}^{7}\sum_{j=1}^{4}\sum_{k=1}^{4}(v*t_{ij}-d'_{ijk}-d_i-v*t_k)^2*P_{ijk}$$
$$s.t.\begin{cases}\sum_{j=1}^{4}P_{ijk}=1\\ \sum_{j=1}^{4}\sum_{k=1}^{4}p_{ijk}=4\\ |t_{k1}-t_{k2}|\le 5\end{cases} \tag{5.4.1.5}$$

5.4.2 模型的求解

我们能得到求解的修正系数如下：

设备	A	B	C	D	E	F	G
修正系数	-27.484422	-0.0869819	-0.11760478	-1.76995084	-3.11069287	-0.09401921	-0.69041337

表 五-5

以及残骸归属结果如下图所示：

监测站	第一组数据	第二组数据	第三组数据	第四组数据
A	0	1	2	3
B	2	0	1	3
C	2	3	1	0
D	3	1	2	0
E	2	1	0	3
F	3	1	2	0
G	0	0	3	2

表 五-6

利用定位模型，可得以下结果：

残骸	经度	维度	高程	时间	误差
1	110.500	27.309	-10.769	12.115	0.003398
2	110.300	27.650	11.186	14.191	0.007042
3	110.699	27.650	13.277	15.377	0.007193
4	110.500	27.951	12.254	12.076	0.001813

表 五-7

我们发现误差基本控制在 1km 以内，说明本题解决较为优秀。

六、 模型的推广

6.1 模型的优点

- **收敛速度快、误差较小**

由于采用了梯度下降法、TRR 算法等局部优化算法，并且提前应用 PSO 算法求出全局最优解可能存在的位置作为初值，因此促成了该模型的收敛速度较快、分析误差较小的优点。

- **时间、空间复杂度低**

由于目标函数的简单以及模型结构的高效，且运用了 TRR 等优化算法，节省了穷举花费的时间，造就了整个模型时间和空间复杂度较低的特点。

6.2 模型的缺点

- 模型的普适性应用不强

由于模型的求解过程中，忽略了包括大地曲率、音速随温度的变化、多路径误差等会影响最终结果的因素，因此如果要将模型应用于现实生活中，需要进行进一步的抗干扰优化。

6.3 模型的改进

6.3.1 声源定位模型的改进

- PSO（Particle Swarm Optimization）算法

粒子群优化（PSO）是一种基于群体智能的优化算法，其灵感来源于鸟群或鱼群的社会行为。[7]声源定位问题可以归结为一个优化问题，即找到使得目标函数最小化的位置坐标。在声源定位中，这个目标函数可以设置为声源与监测设备的计算出的欧

式距离（由 x,y,z 确定）和声波距离（由 t 确定）的差值。

数学上，PSO 算法通过模拟一群粒子在解空间中寻找最优解的过程进行优化。每个粒子代表声源定位中的一个潜在解，即声源可能的位置。粒子们通过个体和群体的经验来更新自己的时空位置和速度。

其数学原理可以归纳如下：

1.初始化：

随机生成粒子群中每个粒子的位置和速度。位置表示潜在声源的坐标，速度决定了粒子探索解空间的方向和步长。

2.目标函数：

目标函数表现该粒子表示的时空坐标（x,y,z,t）带入非线性方程组中计算得到的欧式距离和声波距离的差值。可以表示如下：

$$V_{i+1}=wV_i+c_1r_1(P_{best}-X_i)+c_2r_2(G_{best}-X_i)$$

其中，V_i是当前速度，X_i是当前位置，P_{best}和G_{best}分别是个体最佳和全局最佳位置，w为惯性权重，c_1和c_2为学习因子，r_1和r_2为随机数。

位置更新公式：

$$X_{i+1}=X_i+V_{i+1}$$

新位置由旧位置和新速度决定。

4.确定个人和全局最优：

每个粒子在更新自己的时候会评估其位置对应的目标函数值(适应度)，如果当前位置的函数值优于 pbest，则更新 pbest。

同时，如果当前位置的函数值优于 gbest，则更新全体粒子的 gbest。

5.终止条件：

算法运行到达预设的最大迭代次数，或者 gbest 满足一定的精度需求时停止迭代。

通过上述过程，粒子群算法能够有效地在解空间中进行并行搜索，最终收敛于声源的最优估计位置。这个方法对初始值的选择不敏感，而且能很好地适应复杂的优化问题，其在声源定位中能很好地收敛于全局最优解。

- TDOA（Time Difference Of Arrival）算法

该方法是基于各监测设备与待定位残骸之间的距离之差通过求解非线性双曲方程组来推断待残骸相对于各监测设备的相对位置的定位方法。本题目中由于声音传播速度是已知且恒定的，传播距离与传播时间之间是成正比例关系的，而音爆来自同一残骸时候（或者间隔是已知的），则传播时间又进一步与监测设备的接收时间成比例关系。因此只需要测量残骸产生的音爆波到达各监测设备的到达时间之差即可得到对应的距离之差。[5]

因此，题目一中，对声源定位进行计算时，可以根据到达时间建立双曲线方程组。假设我们以监测设备 A 作为参考设备（参考设备相关参数下标为 1），从剩余的监测设备中任取两个监测设备的数据（相关参数下标依次为 2，3）。

残骸坐标记录为：(x, y, z)

各监测设备坐标记录为：(x_i, y_i, z_i)

各监测设备到残骸的欧式距离记录为：$R_i=\sqrt{(x_i-x)^2+(y_y-y)^2+(z_i+z)^2}$

各监测设备与参考设备到残骸的声波距离之差记为：$R_{i,1}=v*|t_i-t_1|$

$$R_{2,1}=R_2-R_1$$

基于以上定义，可得如下非线性双曲线方程组：

$$R_{3,1}=R_3-R_1$$

TDOA 定位问题就归结为求解以上两个双曲面方程组的交点的问题。对于该非线性方程可以进行线性化处理，对方程两侧进行泰勒级数展开并保留前两项。

经由以上处理之后，再对最后的线性方程进行梯度下降优化或者牛顿法可以更快地收敛到最优解，并且线性处理之后可以极大地减少局部最优解的出现[6]。

6.3.2 多信号区分模型的改进

- 神经网络（Neural Network）算法

针对问题背景，由于火箭发射不属于频发事件，故在一次监测中，可以认为多个残骸都是从同一个火箭上分离的。那么，对于监测设备接收到的多组音爆时间所确定的多个残骸的时空位置，实际上是一个火箭的发射轨迹的时空函数 f(x, y, z, t)上的多个点。

由于火箭种类的不同，发射任务的差异，发射地点的环境影响等等因素，都会引起上述时空函数的变化，而这也是导致监测设备中对于不同残骸音爆的接收时间差异看似随机的原因。但是，基于神经网络的特性，我们还是有可能从大量的数据中找到不同残骸接收时间之间的差异。

数据集的制作需要人为收集火箭发射时残骸音爆的时空位置以及地面基站接收到音爆的时间，并做好标记，再交由神经网络训练，是存在提高多信号区分模型精度的可能性。

6.3.3 误差优化模型的改进

- 蒙特卡洛方法

现阶段影响误差优化模型的最大因素是修正系数的不确定性，代入的修正系数不同会极大地影响最后的优化结果。因此求出一个合适的修正系数是十分关键的。

已知修正系数的得出过程如下：

预先设置 0.5s 以内的时间误差（满足某种分布，例如正态分布），然后将添加了时间误差的数据代入目标函数，可以求解出来该时间误差对应的修正系数。

以上过程意味着，只要知道时间误差的分布函数，便可以利用蒙特卡洛方法算出对应的修正系数的分布函数。利用修正系数的分布函数就能得到准确的修正系数的期望值。

随着蒙特卡洛方法次数的增加，得到的修正系数的分布函数就更加精确，依据分布函数得到的修正系数的期望就更加准确，误差优化模型就能因此更为准确。

因此依据蒙特卡洛方法和高算力的支持，有进一步优化模型误差的可能性。

6.4 模型的推广

本问题的模型实质上是基于多基站接收信号的时间差对信号源进行定位的模型，在生产工作中的许多领域都能加以运用：

- 室内定位与导航：

在无法接收 GPS 信号的封闭室内环境中，通过部署多个无线信号接收天线，可以利用各个天线对手机信号接收的时间差进行定位，帮助人们在商场、机场等大型建筑内更精确的定位与导航。

- 地震学：

通过分析地震波到达不同地震监测站点的时间，可以确定地震的震源位置，可以更好地确定灾害严重地区以及帮助地震救援的快速响应。同时对于震源时空位置的记录分析，有助于科学上对于地球板块运动的深入了解和研究。

- 医学影像（PET 技术）：

利用药物或者手术对病灶部分进行放射性标记，发射性示踪剂发出的高能粒子同样可以视作一种信号，通过患者身体外部的多个高精度的探测仪对这种信号进行捕捉，分析其抵达不同探测仪器的时间差，便可以计算出患者体内病灶的空间位置以及大小，以此来判断病情的严重程度或者为进一步的治疗研究提供数据支持。

- 水下声纳系统：

当声波发射器向水下环境发射声波时，声波会在遇到物体后反射回来，被声纳探测器捕捉。在装配有声纳传感器的多个浮标或声纳阵列，能够接收来自目标物体反射回来的声波。通过计算声波从发射到接收的时间差，考虑海水的温度、盐度和深度对声速的影响，可以使用多传感器时间差定位模型来估算目标物体的距离和方位。这种方法可以用来定位海底沉船、航空器黑匣子、潜艇以及进行海洋生物群体和地形的调查。

七、 参考文献

[1] 卢玉.空间碎片主动清除的法律问题和对策研究[D].华东政法大学,2023.DOI:10.27150/d.cnki.ghdzc.2023.000017.

[2] 吴秋峰,刘振忠. Stiefel 流形上的梯度下降法[J]. 应用数学学报,2012,35(4):719-727. DOI:10.3969/j.issn.0254-3079.2012.04.013.

[3] 陈豫禹. 基于 AOA 的静态目标无源定位算法研究[D]. 四川:电子科技大学,2023.

[4] THU MINH LE, FATAHI, BEHZAD. Trust-region reflective optimisation to obtain soil visco-plastic properties[J]. Engineering computations: International journal for computer-aided engineering and software,2016,33(2):410-442. DOI:10.1108/EC-11-2014-0236.

[5] 胡森康,徐铮,刘伟,等. 一种精确的超声波定位系统[J]. 物理与工程,2021,31(1):120-123. DOI:10.3969/j.issn.1009-7104.2021.01.026.

[6] 洪伟,蔺诚毅,陈婷. 城市环境下无人机TDOA定位中到达时间差误差统计模型[J]. 火控雷达技术,2019,48(1):38-41. DOI:10.19472/j.cnki.1008-8652.2019.01.008.

[7] 申莉,汪晓东. 一种基于粒子群优化算法的非线性方程求解方法[C]. //第六届中国 Rough 集与软计算学术研讨会(CRSSC'2006). 2006:202-205.

附录

1.问题一代码：python 实现

误差计算函数 obj

```
def objective_function(variables, points, times_received):
    signal_origin = variables[:3]
    signal_time = variables[3]
    euclid_distances = np.linalg.norm(points - signal_origin, axis=1)
    sound_distances = (times_received - signal_time) * SPEED_OF_SOUND
    return np.mean(np.abs(euclid_distances - sound_distances
```

主函数

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from matplotlib_inline import backend_inline
backend_inline.set_matplotlib_formats('svg')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


# 设置全局变量
SPEED_OF_SOUND = 0.34  # 声速 km/s
X_LENGTH = 97.304      # 单位经度长度 km
Y_LENGTH = 111.263     # 单位纬度长度 km

# 导入初始数据
DATA_LIST = np.array([
    [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]
])
DATA_DF = pd.DataFrame(DATA_LIST, columns=['longitude', 'latitude', 'altitude', 'time_received'])

# 数据预处理
DATA_DF['x'] = DATA_DF['longitude'] * X_LENGTH
```

```
    DATA_DF['y'] = DATA_DF['latitude'] * Y_LENGTH
    DATA_DF['z'] = DATA_DF['altitude'] / 1000

    # 计算位置和优化目标函数
    def calculate_position(df, tol=1e-9):
        points = df[['x', 'y', 'z']].to_numpy()
        times_received = df['time_received'].to_numpy()

        def objective_function(variables, points, times_received):
                signal_origin = variables[:3]
                signal_time = variables[3]
                euclid_distances = np.linalg.norm(points - signal_origin, axis=1)
                sound_distances = (times_received - signal_time) *
SPEED_OF_SOUND
                return np.mean(np.abs(euclid_distances - sound_distances))

        # 限制函数与边界条件
        def constraint_function(variables, times_received):
            signal_time = variables[3]
            return times_received - signal_time
        constraints = {'type': 'ineq', 'fun':constraint_function}
        bounds = [(-180*97.304, 180*97.304), (-90*111.263, 90*111.263), (0, 10000),
(0, np.min(times_received))]

        # 初始猜测
        x0 = np.zeros(4)
        x0[:3] = np.mean(points, axis=0)  # 假设初始位置为所有监测设备位置的平
均值
        x0[3] = 0  # 假设初始信号发出时间为 0

        # 实施优化
        result = minimize(objective_function, x0,
                          args=(points, times_received),
                          method='L-BFGS-B', options={'gtol': tol},
                          constraints=constraints, bounds=bounds
                          )

        if result.success:
            RESULT_POINT = pd.DataFrame({'longitude': result.x[0]/97.304, 'latitude':
result.x[1]/111.263, 'altitude': result.x[2]*1000,
                                         'time_explosion': result.x[3],
'x':result.x[0], 'y':result.x[1], 'z':result.x[2]}, index=['Position'])
            return RESULT_POINT  # 返回信号源位置
        else:
```

```
            raise ValueError("Optimization failed: " + result.message)


    # 误差函数
    def calculate_errors(df, target_position):
        errors = []

        target_xyz = np.array(calculated_position.loc['Position'].to_list()[-3:])
        target_t = calculated_position.loc['Position','time_explosion']
        for index, row in df.iterrows():
            error = np.sqrt(np.sum((row[['x', 'y', 'z']] - target_xyz) ** 2)) - (row['time_received'] - target_t) * SPEED_OF_SOUND
            errors.append(error)
        errors_abs = np.abs(errors)

        return errors, np.mean(errors_abs), np.std(errors_abs)

    # 绘图函数
    def plot_devices_and_target(df, target_position):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')

        # 获取设备坐标点
        xs = df['longitude']
        ys = df['latitude']
        zs = df['altitude']

        # 绘制设备点
        ax.scatter(xs, ys, zs, c='blue', label='设备点', depthshade=True)

        # 绘制信号源点
        target_position = target_position.loc['Position'].to_dict()
        ax.scatter(target_position['longitude'], target_position['latitude'], target_position['altitude'], c='red', marker='*', label='预测目标点')

        # 设置图例
        ax.legend()
        ax.set_xlabel('经 度（°）')
        ax.set_ylabel('维 度（°）')
        ax.set_zlabel('高 程（m）')

        plt.show()
```

```
    if __name__ == '__main__':

        # 调用函数进行计算
        calculated_position = calculate_position(DATA_DF)
        print(f'监测设备信息:\n{DATA_DF}\n预测目标点信息:\n{calculated_position}')
        print(calculate_errors(DATA_DF, calculated_position))

        # 调用函数进行绘图
        plot_devices_and_target(DATA_DF, calculated_position)
```

2. 问题二代码：python 实现

误差计算函数 obj

```
def obj(a, y0, y1, y2, y3, y4, y5, y6):
y=np.array([y0,y1,y2,y3,y4,y5,y6])
d=np.sqrt(((phi-a[0])*97.304)**2+((theta-a[1])*111.263)**2+(h-a[2])**2)
error=y-d-0.340*a[3]
return error
```

优化函数 opt

```
def opt(x):
res=99999
target = []
a=[]
for i in range(7):
    target.append(t[i,x[i]])
target=np.array(target)*0.340
for i in range(7):
    res0=0
    a0=np.random.rand(4)
    fit_res=least_squares(obj,a0,args=(target))
    a0=fit_res.x
    for j in range(2):
        fit_res=least_squares(obj,a0,args=(target))
        a0=fit_res.x
    res0=(obj(a0,*target)*obj(a0,*target)).sum()
    if res0<res:
        res=res0
        a=a0
return res,a
```

反馈递归函数 backtrack

```
def backtrack(path,sel,m):
global res_tol,n,solution,path1
if len(path)==6:
    n+=1
    if n%20==0:
        print('当前最优值:',res_tol,'当前的迭代次数为:',n)
    for i in range(m):
        for j in range(1,7):
            if path[j-1]==a_best[i,j]:
                return
    path=[sel]+path
```

```
        res0,a=opt(path)
        if res0<res_tol:
            res_tol=res0
            solution=a
            path1=path
        return
    for i in range(2):
        backtrack(path+[i],sel,m)
```

结果显示函数 display_results

```
def display_results(a_best, ress, location):
print("最优参数集合 a_best:")
for i, a in enumerate(a_best):
    print(f"起点 {i} 的最优参数: {a}")
print("\n 每个起点的最优误差 ress:")
for i in range(2):   # 只打印前两个元素
    print(f"起点{i}的最优误差为{ress[i]}")

print("\n 位置 location:")
for i, loc in enumerate(location):
    print(f"起点 {i} 的位置: {loc}")
```

主函数

```
import numpy as np
from scipy.optimize import least_squares
phi = np.array([110.241, 110.780, 110.712, 110.251, 110.524, 110.467, 110.047])
theta = np.array([27.204, 27.456, 27.785, 27.825, 27.617, 27.921, 27.121])
h = np.array([824, 727, 742, 850, 786, 678, 575]) / 1000
t = np.zeros((7,7))
x1_data = (np.mean(phi), np.mean(theta), np.mean(h), 0)
print(x1_data)
for i in range(7):
    t[i][0] = np.array((np.sqrt(((phi[i] - x1_data[0])*111.263) ** 2 + ((theta[i] -
x1_data[1])*97.304) ** 2 + (h[i] - x1_data[2]) ** 2)) / 0.340)
print(f"设点坐标为{x1_data}")
print(f"设点时间为{t[:,0]}")
t[:,1]=np.array([100.767,112.220,188.02,258.985,118.443,266.871,163.024])
res_tol = 100000
solution = 0
print('开始')
n = 0
a_best = np.zeros((2, 7))
```

```
    path2 = []
    location = []
    ress = []
    for i in range(2):
        path1 = []
        res_tol = 999999
        solution = 0
        backtrack([], i, i)
        location.append(solution)
        ress.append(res_tol)
        a_best[i, :] = path1
    ress=np.array(ress)
    print(f"设点坐标为{x1_data}")
    print(f"设点时间为{t[:,0]}")
    print(ress)
    display_results(a_best, ress, location)
```

3. 问题三代码：python 实现

误差计算函数 obj

```
def obj(a,y0,y1,y2,y3,y4,y5,y6):
y=np.array([y0,y1,y2,y3,y4,y5,y6])
d=np.sqrt(((phi-a[0])*97.304)**2+((theta-a[1])*111.263)**2+(h-a[2])**2)
error=y-d-0.340*a[3]
return error
```

优化函数 opt

```
def opt(x):
res=999999
target = []
a=[]
for i in range(7):
    target.append(t[i,x[i]])
target=np.array(target)*0.340
for i in range(3):
    res0=0
    a0=np.random.rand(4)
    fit_res=least_squares(obj,a0,args=(target))
    a0=fit_res.x
    for j in range(2):
        fit_res=least_squares(obj,a0,args=(target))
        a0=fit_res.x
    res0=(obj(a0,*target)*obj(a0,*target)).sum()
    if res0<res:
        res=res0
        a=a0
return res,a
```

结果显示函数 display_results

```
def display_results(a_best, ress, location):
print("最优参数集合 a_best:")
for i, a in enumerate(a_best):
    print(f"起点 {i} 的最优参数: {a}")
print("\n 每个起点的最优误差 ress:")
for i in range(4):  # 只打印前两个元素
    print(f"起点{i}的最优误差为{ress[i]}")
print("\n 位置 location:")
for i, loc in enumerate(location):
    print(f"起点 {i} 的位置: {loc}")
```

反馈递归函数 backtrack

```
def opt(x):
res=999999
target = []
a=[]
for i in range(7):
    target.append(t[i,x[i]])
target=np.array(target)*0.340
for i in range(3):
    res0=0
    a0=np.random.rand(4)
    fit_res=least_squares(obj,a0,args=(target))
    a0=fit_res.x
    for j in range(2):
        fit_res=least_squares(obj,a0,args=(target))
        a0=fit_res.x
    res0=(obj(a0,*target)*obj(a0,*target)).sum()
    if res0<res:
        res=res0
        a=a0
return res,a
```

主函数

```
import numpy as np
from scipy.optimize import least_squares
from numpy.random import randint,rand
phi=np.array([110.241,110.783,110.762,110.251,110.524,110.467,110.047])
theta=np.array([27.204,27.456,27.785,28.025,27.617,28.081,27.521])
h=np.array([824,727,742,850,786,678,575])/1000
t=np.array([[100.767,164.229,214.850,270.065],
[92.453,112.220,169.362,196.583],
[75.560,110.696,156.936,188.020],
[94.653,141.409,196.517,258.985],
[78.600,86.216,118.443,126.669],
[67.274,166.270,175.482,266.871],
[103.738,163.024,206.789,210.306]])
res_tol=99999
solution=0
print('开始')
n=0
a_best=np.zeros((4,7))
path2=[]
location=[]
ress=[]
```

```
    for i in range(4):
        path1=[]
        res_tol=99999
        solution=0
        backtrack([],i,i)
        location.append(solution)
        ress.append(res_tol)
        a_best[i,:]=path1
    display_results(a_best, ress,location)
```

4. 问题四代码：python 实现

误差计算函数 obj

```
def obj(a,y0,y1,y2,y3,y4,y5,y6):
y=np.array([y0,y1,y2,y3,y4,y5,y6])
d=np.sqrt(((phi-a[0])*97.304)**2+((theta-a[1])*111.263)**2+(h-a[2])**2)
error=y-d-0.340*a[3]-d_xiu
return error
```

优化函数 opt

```
def opt(x):
resago=99999
a=[]
target=[]
for i in range(7):
    target.append(t[i,x[i]])
target=np.array(target)*0.340

for i in range(2):
    res0=0
    a0=np.random.rand(4)
    fit_res=least_squares(obj,a0,args=(target))
    a0=fit_res.x
    for j in range(2):
        fit_res=least_squares(obj,a0,args=(target))
        a0=fit_res.x
    res0=(obj(a0,*target)*obj(a0,*target)).sum()
    if res0<resago:
        resago=res0
        a=a0
return resago,a
```

反馈递归函数 backtrack

```
def backtrack(path,sel,m):
global res_tol,n,solution,path1
if len(path)==6:
    n+=1
    if n%20==0:
        print('当前最优值:',res_tol,'当前的迭代次数为:',n)
    for i in range(m):
        for j in range(1,7):
            if path[j-1]==a_best[i,j]:
                return
    path=[sel]+path
    res0,a=opt(path)
```

```
            if res0<res_tol:
                res_tol=res0
                solution=a
                path1=path
            return
        for i in range(4):
            backtrack(path+[i],sel,m)
```

结果显示函数 display_results

```
def display_results(a_best, ress,location):
    print("最优参数集合  a_best:")
    for i, a in enumerate(a_best):
        print(f"起点  {i}  的最优参数: {a}")
    print("\n 每个起点的最优误差  ress:")
    for i in range(4):   #  只打印前两个元素
        print(f"起点{i}的最优误差为{ress[i]}")
    print("\n 位置  location:")
    for i, loc in enumerate(location):
        print(f"起点  {i}  的位置: {loc}")
```

主函数

```
    import numpy as np
    from scipy.optimize import least_squares
    from numpy.random import randint,rand
d_xiu=np.array([-38.65840166,-7.11606118,-0.09229454, -0.69497375 , -2.77534127,
   -0.11463343 , -0.73265696])
phi=np.array([110.241,110.783,110.762,110.251,110.524,110.467,110.047])
theta=np.array([27.204,27.456,27.785,28.025,27.617,28.081,27.521])
h=np.array([824,727,742,850,786,678,575])/1000
t=np.array([[100.767,164.229,214.850,270.065],
[92.453,112.220,169.362,196.583],
[75.560,110.696,156.936,188.020],
[94.653,141.409,196.517,258.985],
[78.600,86.216,118.443,126.669],
[67.274,166.270,175.482,266.871],
[103.738,163.024,206.789,210.306]])
t+=np.random.normal(0,0.5*0.340,size=t.shape)
res_tol=99999
solution=0
print('开始')
n=0
a_best=np.zeros((4,7))
location=[]
```

```
ress=[]
path2=[]
for i in range(4):
        path1=[]
        res_tol=99999
        solution=0
        backtrack([],i,i)
        location.append(solution)
        ress.append(res_tol)
        a_best[i,:]=path1
display_results(a_best, ress,location)
```
