火箭残骸音爆定源问题的求解与优化

Github@ Lowmst

摘要

问题一 我们构建了单残骸音爆定源模型。假设出残骸发生音爆的位置和时间，共含四个未知数，则可以利用空间坐标距离公式以及残骸音爆位置和某一台设备之间的距离等于震动波速度×（音爆接收时间−音爆发生时间）列出方程，四台设备即可列出四个方程，从而求解出四个未知数，即 **最少只需四台设备** 即可确定残骸发射音爆的位置和时间。对于题目所给具体算例，将方程求解转化为优化问题并求解出结果，最终确定残骸发生音爆时的位置和时间分别为 **经度 110.728°，纬度 27.025°，高程 15215.429m，时间为-45.523s（相对于观测系统时钟 0 时）**

问题二 针对多残骸音爆定源，最核心的问题便是对设备接收到的时间数据进行分类，确定每台设备中的哪一条数据属于同一个残骸，然后可以由问题一模型进行求解。我们通过来源于到达时间差（Time Difference of Arrival，TDOA）算法的思想，确定了这些时间数据两两归属于同一残骸的必要条件。并且由于若某一组时间数据不属于同一个残骸，那么对于问题一模型的优化目标函数值则不能充分收敛至 0。通过这些条件约束，结合深度优先搜索（Depth First Search，DFS）搜索出这样的时间数据组合，得到一组求解出的目标函数值，选取其最小值即可确定某残骸所对应的时间数据，并套用问题一模型解出其坐标和时间。

问题三 为问题二的一个具体算例，将题中所给数据进行预处理后代入问题二的模型中进行求解,可以得到以下结果

表 **1**

残骸	经度(°)	纬度(°)	高程(m)	时间(s)
1	110.500	27.310	12513.956	12.000
2	110.300	27.650	11477.885	14.000
3	110.700	27.650	13468.163	15.000
4	110.500	27.950	11528.802	13.002

问题四 对问题三原题目中时间数据叠加一个均匀分布的随机误差以模拟设备存在的误差。将该数据代入问题三模型中进行求解计算，并将结果与问题三中求解的结果进行比较，计算残差（取两次计算中残骸位置的距离的平均值）如表所示

表 **2**　表格中为每个误差节点计算 **10** 次的平均值

误差(s)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	2.0	3.0
残差(km)	0.076	0.178	0.327	0.449	0.506	0.646	0.644	0.839	0.932	1.130	2.003	5.748

可以得出残差与误差大致成正相关关系，其中在 0.1~1.0 秒的误差下，残差可以在 1km 的距离之内，满足题意，而更大的误差下残差将大到无法接收的程度。

注意到在误差为 4.0 秒以上时将无法稳定求解，这是由于在部分特定的误差组合之下，时间数据将不满足 TDOA 时间差约束，导致求解算法无法正常工作。而在实际情形中，音爆接收时间误差通常小于 0.9 秒或远小于 0.9 秒，所以可以认为在 0.5 秒的时间误差下，模型可以较精确地定位。

关键字： 单、多残骸音爆定源　最优化算法　到达时间差　深度优先搜索　误差修正

一、 问题重述

1.1 问题背景

在火箭发射任务中，多级火箭的各级火箭或助推器完成任务后会分离并坠落地面，产生跨音速音爆。为了实现残骸的快速回收和准确定位，需要在残骸落区布置多台震动波监测设备，通过监测设备记录的音爆信号时间，确定残骸音爆发生时的空中位置和时间，并预测其落地点。多残骸同时坠落的复杂性以及设备记录误差带来的挑战，使得残骸的精准定位成为一个重要且复杂的问题。

1.2 问题提出

问题一 建立数学模型，分析如果要精准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，至少需要布置几台监测设备？假设某火箭一级残骸分离后，在落点附近布置了 7 台监测设备，合理选取各台设备三维坐标（经度、纬度、高程）、音爆抵达时间（相对于观测系统时钟 0 时）的数据，计算残骸发生音爆时的位置和时间。

问题二 火箭残骸除了一级残骸，还有两个或者四个助推器。在多个残骸发生音爆时，监测设备在监测范围内可能会采集到几组音爆数据。假设空中有 4 个残骸，每个设备按照时间先后顺序收到 4 组震动波。建立数学模型，分析如何确定监测设备接收到的震动波是来自哪一个残骸？如果要确定 4 个残骸在空中发生音爆时的位置和时间，至少需要布置多少台监测设备？

问题三 利用问题 2 所建立的数学模型，选取合适的数据，确定 4 个残骸在空中发生音爆时的位置和时间（4 个残骸产生音爆的时间可能不同，但互相差别不超过 5 s）。

问题四 假设设备记录时间存在 0.5s 的随机误差，请修正问题 2 所建立的模型以较精确地确定 4 个残骸在空中发生音爆时的位置和时间。通过对问题 3 表中数据叠加随机误差，给出修正模型的算例，并分析结果误差。如果时间误差无法降低，提供一种解决方案实现残骸空中的精准定位（误差 km），并自行根据问题 3 所计算得到的定位结果模拟所需的监测设备位置和音爆抵达时间数据，验证相关模型。

二、 问题分析

2.1 问题一

假设出残骸发生音爆的位置和时间，共含四个未知数，则可以利用空间坐标距离公式以及残骸音爆位置和某一台设备之间的距离等于震动波速度×（音爆接收时间–音爆发生时间）列出方程，四台设备即可列出四个方程，从而求解出四个未知数，即 **最少只需四台设备** 即可确定残骸发生音爆的位置和时间。对于题目所给具体算例，可以将方程求解转化为优化问题并求解出结果。

2.2 问题二

针对多残骸音爆定源，最核心的问题便是对设备接收到的时间数据进行分类，确定每台设备中的哪一条数据属于同一个残骸。然而若某一组时间数据不属于同一个残骸，那么对于问题一的优化目标函数值便不能充分收敛至0。如果能利用算法找到这样的数据组合，此问题即可转化为单残骸音爆定源问题。

2.3 问题三

其为问题二的一个具体算例，将题中所给数据代入问题二模型中进行求解即可得到结果。

2.4 问题四

该问题需要在二、三问的基础上考虑时间数据存在 0.5s 的误差，需对原时间矩阵叠加随机时间误差矩阵以进行更精确的求解。为研究时间误差对定位结果的影响，可在一定范围内改变误差大小阈值并进行相应求解，通过分析不同误差下的求解结果的残差的变化得出误差大小对定位结果的影响，并在可能的前提下，给出消除误差的方法。

三、 符号说明

表 **3**

符号	意义	单位
d	距离	km
x	横坐标	km
y	纵坐标	km
z	高坐标	km
t	时间（相对于观测系统时钟 0 时）	s
v	震动波速度	km/s
e	随机时间误差	s
T	时间矩阵或集合	-
E	随机时间误差矩阵或集合	-

四、 模型假设

1. 震动波的传播速度$v = 0.34$km/s。
2. 计算两点间距离时可忽略地面曲率。
3. 纬度间每度距离值近似为 111.263km，经度间每度距离值近似为 97.304km。
4. 以上述的假设可以建立以经度 110°，纬度 27°，高程 0m 为原点，x轴指向正东，y轴指向正北，z轴垂直于地面，单位长度为 1km 的空间直角坐标系以便于计算，且本文所述的所有计算均以此坐标系为基准。

   此处给出 **问题一** 及 **问题三** 中数据在此坐标系下转换后的坐标：

表 **4**　**问题一**

设备	横坐标x(km)	纵坐标y(km)	高坐标z(km)
A	23.450	22.698	0.824
B	75.897	50.736	0.727
C	69.280	87.342	0.742
D	24.423	91.792	0.850
E	50.987	68.649	0.786
F	45.441	102.473	0.678
G	4.573	13.463	0.575

表 **5**　**问题三**

设备	横坐标x(km)	纵坐标y(km)	高坐标z(km)
A	23.450	22.698	0.824
B	76.189	50.736	0.727
C	74.146	87.342	0.742
D	24.423	114.045	0.850
E	50.987	68.649	0.786
F	45.441	120.275	0.678
G	4.573	57.968	0.575

五、 模型建立与求解

5.1 问题一

5.1.1 单残骸音爆定源模型

考虑有n台监测设备接收单残骸的音爆震动波，若第i台监测设备的位置坐标为(x_i, y_i, z_i)，在t_i时间接收到音爆震动波，可以假设该残骸音爆发生位置坐标为(x_0, y_0, z_0)，发生时间为t_0。

设该残骸到第i台监测设备的距离为d_i，由空间中两点欧氏距离公式可知

$$d_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2} \tag{1}$$

由于震动波呈直线传播，由位移的速度时间公式可知

$$d_i = v(t_i - t_0) \tag{2}$$

即有方程

$$\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2} = v(t_i - t_0) \tag{3}$$

此时对此方程进行求解即可得出该残骸音爆发生的位置及时间。

令$f_i(x, y, z, t) = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} - v(t_i - t)$，则原方程化为

$$f_i(x_0, y_0, z_0, t_0) = 0 \tag{4}$$

对于这样含有 4 个未知量的方程，容易知道需要至少 4 个方程组成方程组才有可能获得唯一解。也就是 **至少需要布置 4 台监测设备** 才能确定残骸音爆的位置和时间。

为了求解此非线性方程组，往往采用数值计算的方法（如牛顿迭代法），但考虑到可能的时间数据误差以及对解的相关约束（如$z > 0$），不妨将此方程组求解问题转化为一个最优化问题。注意到对$\forall (x, y, z, t) \in \mathbb{R}^4$有

$$\lim_{(x,y,z,t)\to(x_0,y_0,z_0,t_0)} f_i(x, y, z, t) = 0 \tag{5}$$

故可以给出如下无约束优化模型

$$\min S(\boldsymbol{x}) = \sum_{i=1}^{n} [f_i(\boldsymbol{x})]^2, \quad \boldsymbol{x} = (x, y, z, t) \tag{6}$$

对此进行最优化求解，即可得到最满足对原方程组残差最小的解，即为残骸音爆发生的位置和时间。

5.1.2 求解

针对问题一所给出的数据进行求解，经反复地权衡并考虑可能存在的脏数据，而且验证了相关数据的合理性之后，最终选择了 A、B、C、D、F、G 这 6 台监测设备进行求解。

对于最优化算法，此处介绍模拟退火算法和牛顿迭代法：

模拟退火算法

模拟退火算法是基于蒙特卡洛迭代求解策略的一种随机优化算法，其出发点是基于物理中固体物质的退火过程和一般组合优化问题之间的相似性，从某一较高初温出发，伴随温度参数的不断下降，结合概率突跳特性在求解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。

STEP 1 考虑优化函数f，给定初始温度T_0，终止温度$T_m > 0$，温度下降率α，此时T_0应尽量大，T_m应尽量小，α应尽量接近1^-，使得温度下降速度尽量慢，能有足够多的轮数进行迭代。

STEP 2 使当前温度$T = T_0$，采取随机方式在f的定义域中选取初始解$\boldsymbol{x}_0$，并使最优解$\boldsymbol{x} := \boldsymbol{x}_0$，接下来采取随机方式在$\boldsymbol{x}$的邻域中选取新解$\boldsymbol{x}'$，若$f(\boldsymbol{x}') < f(\boldsymbol{x})$时，则$\boldsymbol{x} := \boldsymbol{x}'$，否则有$e^{-\frac{|f(\boldsymbol{x}')-f(\boldsymbol{x})|}{T}}$的概率使$\boldsymbol{x} := \boldsymbol{x}'$。

STEP 3 当前温度进行下降，即$T = \alpha T$，若$T < T_m$，则终止计算，否则继续选取新解，进入 **STEP 2** 计算。

牛顿迭代法

牛顿法又称为牛顿—拉弗森方法，是一种在实数域和复数域上近似求解方程的方法，其利用函数曲线的切线来逐步逼近逼近方程的根。牛顿法是一种迭代求解方法，在实际使用时，要选择合适的初始猜测值以确保算法的收敛性和处理可能的数值问题。

STEP 1 通过分析题意知该无约束优化问题为

$$\min_{\boldsymbol{x}\in\mathbb{R}^4} S(\boldsymbol{x}) \quad \boldsymbol{x} = (x, y, z, t) \tag{7}$$

其中$\boldsymbol{x}^* = \left((x_0)^*, (y_0)^*, (z_0)^*, (t_0)^*\right)$为该目标函数的极小点。

STEP 2 设第k次迭代值为$\boldsymbol{x}^k$，则可在$\boldsymbol{x}^k$附近进行二阶泰勒展开

$$S(\boldsymbol{x}) = S\left(\boldsymbol{x}^k\right) + \left(g_k\left(\boldsymbol{x} - \boldsymbol{x}^k\right)\right)^T + \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^k\right)^T H\left(\boldsymbol{x}^k\right)\left(\boldsymbol{x} - \boldsymbol{x}^k\right) \tag{8}$$

其中，$g_k = g\left(\boldsymbol{x}^k\right) = \nabla S\left(\boldsymbol{x}^k\right)$是$S(\boldsymbol{x})$的梯度向量在点$\boldsymbol{x}^k$的值，$H\left(\boldsymbol{x}^k\right)$是$S(\boldsymbol{x})$的海塞矩阵

$$H(\boldsymbol{x}) = \left[\frac{\partial^2 S}{\left(\partial x_i\right)\left(\partial x_j\right)}\right]_{n\times n} \tag{9}$$

在点$\boldsymbol{x}^k$处的值。函数$S(\boldsymbol{x})$有极值的必要条件是在极值点处一阶导数为 0，即梯度向量为 0。特别的当$H(\boldsymbol{x})$是正定矩阵时，函数$S(\boldsymbol{x})$的极值为极小值。

5.1.3 求解结果

在本题中，将利用一种更加优化的模拟退火算法（双模拟退火算法），经编程求解后可以得到最优解$(70.850, 2.814, 15.215, -45.523)$，即残骸音爆发生位置坐标为$(70.850, 2.814, 15.215)$，转换回经纬度坐标为

经度 110.728°，纬度 27.025°，高程 15215.429m，时间-45.523s（相对于观测系统时钟 0 时）

有示意图如下

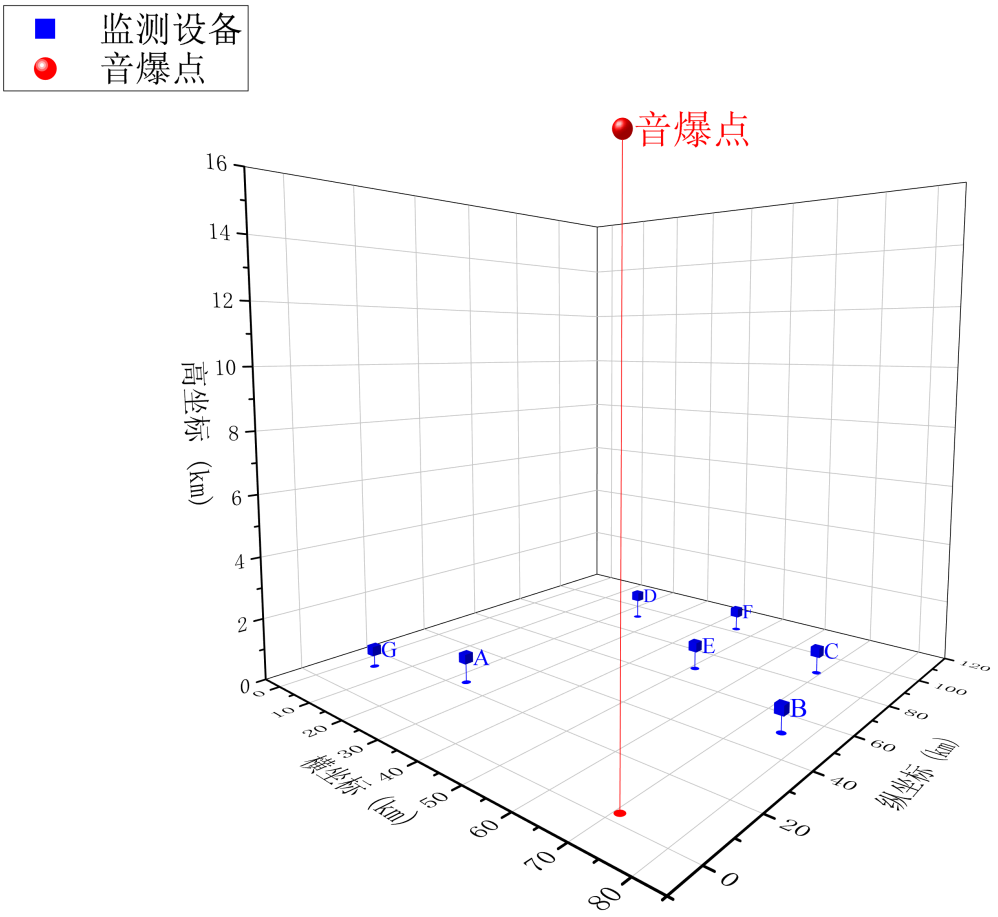


图 **1**

5.2 问题二

5.2.1 多残骸音爆定源模型

考虑有n台监测设备接收m个残骸的音爆震动波，若第i台监测设备的位置坐标为(x_i, y_i, z_i)，依次在$t_{i1}, t_{i2}, \cdots, t_{ij}, \cdots, t_{im}$时间接收到各残骸的音爆震动波。有时间矩阵

$$T = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1j} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2j} & \cdots & t_{2m} \\ \vdots & \vdots & & \vdots & & \vdots \\ t_{i1} & t_{i2} & \cdots & t_{ij} & \cdots & t_{im} \\ \vdots & \vdots & & \vdots & & \vdots \\ t_{n1} & t_{n2} & \cdots & t_{nj} & \cdots & t_{nm} \end{pmatrix} \tag{10}$$

假设第j个残骸的音爆震动波在集合

$$T_j = \left\{ t_{1k_1}, t_{2k_2}, \cdots, t_{ik_i}, \cdots, t_{nk_n} \right\} \quad (k_i \in \{1, 2, \cdots, m\}) \tag{11}$$

中时间被各监测设备所接收，那么此问题就转化为了m个单残骸音爆定源问题。

由上述可知，此模型应求解出每个T_j所包含的时间数据，且$\bigcap_{j=1}^{m} T_j = \varnothing, \bigcup_{j=1}^{m} T_j = \left\{ t_{ij} | 1 < i < n, 1 < j < m \right\}$。

对于T中的任意组合$\left\{ t_{1k_1}, t_{2k_2}, \cdots, t_{ik_i}, \cdots, t_{nk_n} \right\}$ $(k_i \in \{1, 2, \cdots, m\})$，若组合中时间数据并不都归属于同一残骸，那么在进行单残骸音爆定源求解时其中的优化目标函数值便不能充分收敛至 0。那么通过对所有这样的组合进行单残骸音爆定源求解操作，选取其中目标函数值最小的组合即为归属于同一残骸的时间数据组合。这是时间复杂度为$O(m^n)$的算法。

5.2.2 优化

5.2.2.1 到达时间差

不妨回到单残骸音爆定源模型上来。

仍然考虑有n台监测设备接收单残骸的音爆震动波，若第i台监测设备的位置坐标为(x_i, y_i, z_i)，在t_i时间接收到音爆震动波，第j台监测设备的位置坐标为(x_j, y_j, z_j)，在t_j时间接收到音爆震动波，可以假设该残骸音爆发生位置坐标为(x_0, y_0, z_0)，发生时间为t_0。

设该残骸到第i台监测设备的距离为d_i，到第j台监测设备的距离为d_j，有

$$\begin{aligned} d_i - d_j = &\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2} \\ &- \sqrt{(x_j - x_0)^2 + (y_j - y_0)^2 + (z_j - z_0)^2} \end{aligned} \tag{12}$$

同时有

$$\begin{aligned} d_i - d_j &= v(t_i - t_0) - v\big(t_j - t_0\big) \\ &= v\big(t_i - t_j\big) \end{aligned} \tag{13}$$

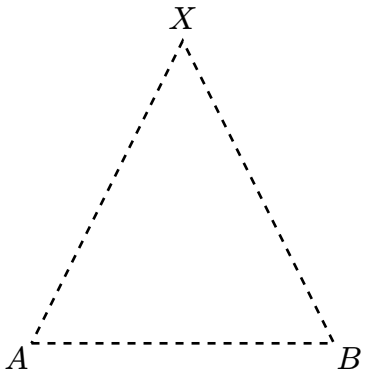
即有方程

$$\begin{aligned} &\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2} - \sqrt{\big(x_j - x_0\big)^2 + \big(y_j - y_0\big)^2 + \big(z_j - z_0\big)^2} \\ &= v\big(t_i - t_j\big) \end{aligned} \tag{14}$$

此方程通过一对监测设备接收到音爆震动波时间之差，使其与残骸音爆发生时间无关。[1]

5.2.2.2 某两个接收时间归属于同一残骸的必要条件

同上述情景，在第i台、第j台监测设备和该残骸音爆位置组成的三角形ABX上，如图所示



有$d_i = |AX|, d_j = |BX|$，两监测设备之间的距离$d_{ij} = |AB|$，在三角形中有$|AB| > ||AX| - |BX||$即

$$d_{ij} > \big|d_i - d_j\big| \tag{15}$$

由(13) 式可得

$$d_{ij} > \big|v\big(t_i - t_j\big)\big| \tag{16}$$

即两台监测设备分别接收到同一残骸音爆的时间数据t_i, t_j，则可推出$d_{ij} > \big|v\big(t_i - t_j\big)\big|$。

从多残骸音爆定源模型的角度来看，则是两台监测设备接收到的某个时间数据能归属到同一残骸音爆的必要条件。

[1]这样含有 3 个未知量的方程，容易知道需要至少 3 个方程组成方程组才有可能获得唯一解。但这并不意味着至少仅 3 台监测设备即可确定残骸音爆的位置和时间，要求解这样的方程组至少需要有 3 对两两无关的监测设备提供的数据，即仍然是至少 4 台监测设备。

5.2.2.3 深度优先搜索

深度优先搜索（DFS）是一种遍历或搜索图或树数据结构的算法，其核心思想是尽可能深入地探索每个分支，直至到达一个未被访问的节点或叶节点为止，然后回溯并探索其他未访问的分支。

在 DFS 核心函数中，首先维护一个存储了当前已搜索出的时间数据组合的栈，然后可以对某一设备的时间数据进行遍历：对每个栈内每个已搜索的数据都对新数据进行一次必要性判断并依此放入栈中，随后递归式地进行核心函数，搜索下一台设备的时间数据；递归结束后，移除栈中最后的数据（进行回溯），进入遍历中的下一个数据。

通过 DFS 递归函数的算法，从设备 B 开始（设备 A 的时间数据按序对应各残骸），最终能够搜索出相对于逐项枚举而言足够少的组合，大幅优化了算法时间复杂度。[2]

5.3 问题三

此问题为问题二的一个具体算例，经编程求解后得到

表 **6**　各残骸所对应在各监测设备的音爆接收时间

残骸	各设备接收时间(s)						
	A	B	C	D	E	F	G
1	100.767	112.220	188.020	258.985	118.443	266.871	163.024
2	164.229	169.362	156.936	141.409	86.216	166.270	103.738
3	214.850	92.453	75.560	196.517	78.600	175.482	210.306
4	270.065	196.583	110.696	94.653	126.669	67.274	206.789

表 **7**　各残骸音爆发生的位置坐标与时间

残骸	横坐标x(km)	纵坐标y(km)	高坐标z(km)	时间(s)
1	48.652	34.491	12.514	12.000
2	29.191	72.321	11.477	14.000
3	68.112	72.321	13.468	15.000
4	48.652	105.700	11.529	13.002

表 8　各残骸音爆发生的经纬度坐标、高程及时间

残骸	**经度(°)**	**纬度(°)**	**高程(m)**	**时间(s)**
1	**110.500**	**27.310**	**12513.956**	**12.000**
2	**110.300**	**27.650**	**11477.885**	**14.000**
3	**110.700**	**27.650**	**13468.163**	**15.000**
4	**110.500**	**27.950**	**11528.802**	**13.002**

有示意图如下

[2]以问题三为例，算法计算量可以减少约 3/4

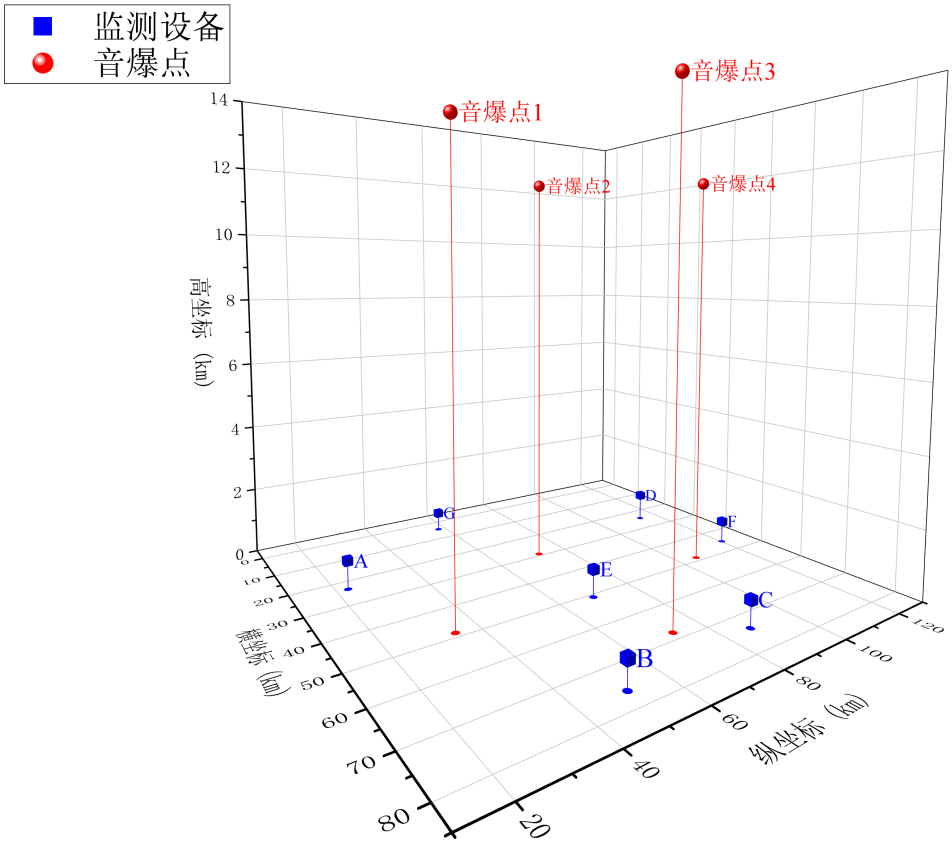


图 **2**

5.4 问题四

由原题假设设备记录时间存在随机误差，可以构建一个随机误差矩阵E，那么此时设备记录的时间矩阵

$$T' = T + E \tag{17}$$

若随机误差阈值为λ，则其中E中元素e_{ij}为区间$[-\lambda, \lambda]$中的随机值，并服从均匀分布。

鉴于本文中算法均由最优化算法支撑，实际上就是在解空间中搜索最优解，在内部算法中较难进行优化消除随机误差的影响[3]。此处将给出在不同的误差阈值下求解出的结果的残差[4]（取与残骸音爆准确位置之间距离的平均值）

表 **9**　表格中为每个误差节点计算 **10** 次残差的平均值

误差(s)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	2.0	3.0
残差(km)	0.076	0.178	0.327	0.449	0.506	0.646	0.644	0.839	0.932	1.130	2.003	5.748

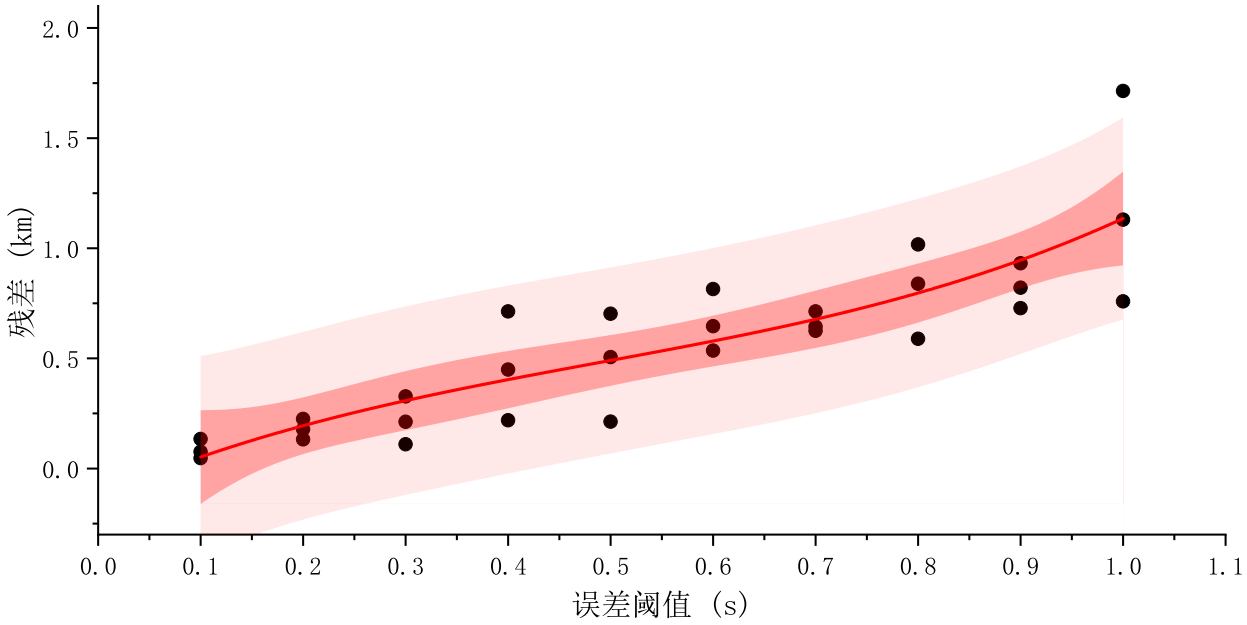


图 **3**　另一组随机误差下计算的结果（三次）

可以得出残差与误差大致成正相关关系，其中在 0.1~1.0 秒的误差下，残差可以在 1km 的距离之内，满足题意，而更大的误差下残差将大到无法接受的程度。

而在计算中注意到在误差为 4.0 秒以上时将无法稳定求解，这是由于在部分特定的误差组合之下，时间数据将不满足到达时间差约束，导致求解算法无法正常工作。而在实际情形中，音爆接收时间误差通常小于 0.9 秒或远小于 0.9 秒，所以可以认为在 0.5 秒的时间误差下，模型可以较精确地定位。

[3]实际上可以由外部进行新的最优化求解出该随机误差，并消除其影响。在$\mathbb{R}^{n\times m}$空间中进行搜索，结合误差阈值约束进行最优化求解，但由于核心求解算法自身求解时间无法忽略，求解时间将巨幅延长。

[4]因数据随机性，此结果没有唯一性，仅从数据拟合角度进行分析。

六、 模型的评价与改进

1. 牛顿迭代法收敛速度快，可快速得到最优解，但易陷入局部最优解，采用模拟退火算法进行全局搜索可有效提高结果的稳定性性和准确性，但代价是求解时间延长。除此之外，也有诸如盆地跳跃法、双模拟退火算法等更优化高效的算法可以选用。
2. 问题二、三中，在 DFS 算法中结合到达时间差必要条件进行搜索，相比于直接枚举，计算次数从 4890 降到 1345，时间从数分钟缩短至 20 秒左右，计算高效；但仍然没有找到一种时间复杂度更低的一种算法，受限于此，误差也难以计算消去。
3. 在对问题四误差进行分析时，实际上有一种理论上可以求解并消去误差的方法，见上文注释[3]
4. 对于更复杂的情况（如更大规模的多残骸系统），可能计算复杂度过高，可引入更高效的聚类或数据挖掘技术来提高模型的稳定性和效率。
5. 结合其他传感器数据（如 GPS、雷达），通过数据融合技术，提升模型对复杂坠落环境的适应能力和定位准确性。

七、 参考文献

[1] 王强, 李伟, 龚建泽, 丁思炜, 和 雷鹏, 《基于火箭残骸实时定位信息的落点计算模型》, 计算机测量与控制, 2021.

[2] 廖彦杰, 薛松柏, 龚小维, 和 叶琪玮, 《火箭整流罩残骸定位跟踪系统》. 2021 年.

[3] 王洪, 刘昌忠, 汪学刚, 和 吴宏刚, 《一种多点定位的目标位置精确解算方法》, 航空学报, 卷 32, 期 7, 页 6–7, 2011.

八、 附录

8.1 Python 实现单残骸音爆定源求解

```python
from utils import *

import numpy as np
from scipy import optimize as opt

data = 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]
    ]
)

locs = convert(data[:, 0:3], 'xyz')
times =   data[:, 3]
v = 0.34

@np.vectorize(excluded=[0])
def f(x, i):
    return distance(locs[i, :], x[0:3]) - (times[i] - x[3]) * v

def s(x):
    return np.sum(f(x, [0,1,2,3,5,6])**2)

result = opt.dual_annealing(s, bounds=[[0,200],[0,200],[0,20],[-50,50]])
print(convert(np.array(result.x[0:3]), 'blh'))
print(result.x[3])
```

8.2 Python 实现多残骸音爆定源求解

```python
from utils import *
import numpy as np
import scipy.optimize as opt
import time as t

# 核心求解函数
def solve(locs, times):

    # 取音速为 0.34km/s
    v = 0.34

    # 计算各监测设备之间的距离
    dists = np.zeros((7, 7))
    for dev_idx_1 in range(6):
        for dev_idx_2 in range(dev_idx_1 + 1, 7):
            dists[dev_idx_1, dev_idx_2] = distance(locs[dev_idx_1], locs[dev_idx_2]
            dists[dev_idx_2, dev_idx_1] = dists[dev_idx_1, dev_idx_2]

    # 维护 7*4 个布尔矩阵，表示 times 中某时间数据相对于其余数据是否应被选取为同一组
     selected = [[np.full_like(times, True, dtype=bool) for i in range(4)] for j
in range(7)]
    for dev_idx_1 in range(7):
        for time_idx_1 in range(4):
            for dev_idx_2 in range(7):
                for time_idx_2 in range(4):
                                            selected[dev_idx_1][time_idx_1][dev_idx_2,
time_idx_2] = (abs(times[dev_idx_1, time_idx_1] - times[dev_idx_2, time_idx_2]) <=
(dists[dev_idx_1, dev_idx_2])/v)

    # 最终选取的 4 组时间数据，初始化为 List
    anss = []

    # 以设备 A 的 4 个时间数据为参照，设置 4 个循环得到 4 组最终数据
    for main_idx in range(4):
        tmp_anss = []
        tmp_ans = []

        # DFS 核心函数
        def dfs(dev_idx):
            if dev_idx == 7:
                tmp_anss.append(list(tmp_ans))
                return
            for time_idx in range(4):
                if (not check_selected(dev_idx, time_idx)) or (not check_ans(dev_idx,
time_idx)):
```

```
                    continue
                tmp_ans.append(time_idx)
                dfs(dev_idx + 1)
                tmp_ans.pop() # 去除栈中最后一位

        def check_selected(dev_idx, time_idx):
            for i in range(len(tmp_ans)):
                if not selected[dev_idx][time_idx][i, tmp_ans[i]]:
                    return False
            return True

        def check_ans(dev_idx, time_idx):
            for ans in anss:
                try:
                    if time_idx == ans[dev_idx]:
                        return False
                except:
                    return True
            return True

        tmp_ans.append(main_idx)
        dfs(1)

        tmp_anss_x = []
        tmp_anss_fun = []

        for tmp_ans in tmp_anss:
            time = [times[dev_idx, tmp_ans[dev_idx]] for dev_idx in range(len(tmp_a

            @np.vectorize(excluded=[0])
            def f(x, i):
                return distance(locs[i, :], x[0:3]) - (time[i] - x[3]) * v

            def s(x):
                return np.sum(f(x, [0,1,2,3,4,5,6])**2)

            constraint = [{'type': 'ineq', 'fun': lambda x: x[2]}]
            result = opt.minimize(s, [0, 0, 12, 0], constraints=constraint)

            tmp_anss_x.append(result.x)
            tmp_anss_fun.append(result.fun)

        ans_idx = tmp_anss_fun.index(min(tmp_anss_fun))

        return_val = tmp_anss[ans_idx]
        return_val.append(tmp_anss_x[ans_idx])
        return_val.append(tmp_anss_fun[ans_idx])
```

```
            anss.append(return_val)
        return anss

    # 原始数据输入
    data = np.array(
        [
            [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]
        ]
    )
    # 取转换为空间直角坐标系的监测设备坐标
    locs = convert(data[:, 0:3], 'xyz')
    # 取各监测设备接收的音爆抵达时间
    times_ori = data[:, 3:]
    print(solve(locs, times_ori))
```

8.3 Python 工具函数 utlis.py

```python
import numpy as np

def distance(p_1: np.ndarray, p_2: np.ndarray):
    if p_1.shape == p_2.shape:
        return np.sqrt(np.sum(np.square(p_1 - p_2)))


@np.vectorize(signature='(n),()->(n)')
def convert(coordinates: np.ndarray, type: str):
    base_lon = 110
    base_lat = 27
    converted = np.zeros(3)
    match type:
        case 'xyz':
            converted[0] = (coordinates[0] - base_lon) * 97.304
            converted[1] = (coordinates[1] - base_lat) * 111.263
            converted[2] = coordinates[2] / 1000
        case 'blh':
            converted[0] = coordinates[0] / 97.304 + base_lon
            converted[1] = coordinates[1] / 111.263 + base_lat
            converted[2] = coordinates[2] * 1000
    return converted
```

8.4 Matlab 实现模拟退火算法

```matlab
function best_solution = SA(fun, init_solution)

    best_solution = init_solution; % 最优解

    init_temperature = 1; % 初始温度
    temperature = init_temperature;
    last_temperature = 10^(-30); % 终止温度
    rate = 0.99999; % 降温速度

    while temperature > last_temperature
        next_solution = zeros(1, length(best_solution));
        for i = 1 : 1 : length(best_solution)
            next_solution(i) = best_solution(i) + (rand-0.5)/100;
        end
        if fun(next_solution) < fun(best_solution)
            best_solution = next_solution;
        else
            if rand < exp(-(fun(next_solution) - fun(best_solution)) / temperature)
                best_solution = next_solution;
            end
        end
        temperature = temperature * rate;
    end
end
```