基于多边测量算法的多个火箭残骸定位方法

课程名称 数学实验

上课班级：992304-001

姓名	学院	年级	专业	学号	点名册序号
王昱硕	计算机学院	**2022**	计算机科学与技术	**20220980**	**33**
张思帆	计算机学院	**2022**	计算机科学与技术	**20221799**	非本班
谈柯迪	大数据与软件学院	**2022**	软件工程	**20221817**	非本班

开 课 时 间 **2023** 至 **2024** 学年第 **2** 学期

基于多边测量算法的多个火箭残骸定位方法

摘要

随着现代火箭技术的快速发展，确保火箭发射阶段结束后残骸的安全落区管理变得尤为重要。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，在坠落至地面过程中，残骸会因速度超越音速而产生音爆。由此，可以通过在残骸理论落区内布置多台震动波监测设备，以接收不同火箭残骸从空中传来的跨音速音爆，然后根据音爆抵达的时间，定位空中残骸发生音爆时的位置，再采用弹道外推实现残骸落地点的快速精准定位。

对于问题一，由于已知各监测设备的坐标与音爆抵达时间，本文基于多边测量定位算法，分析得到至少需要布置四台监测设备以构建理论方程组，才能精准确定空中单个残骸发生音爆时的经度、纬度、高程和时间。由于音爆抵达时间存在误差，本文将上述模型转化为将该误差最小化的优化问题，使用粒子群算法，最终选取设备 A, B, E, G，得到残骸音爆的位置为经度 110.497850 度, 纬度 27.314287 度, 高程 1154.001033 米，音爆时间相对于零时刻为 19.093961 秒。

对于问题二，在问题一的模型基础上分析得到，计算得出各残骸音爆的时间误差平方和取最小值时使用的各设备接收到震动波即为各个监测设备接收到该残骸的震动波。在区分出不同残骸的数据后，基于问题一的基础模型，分析得到如果要确定 4 个残骸在空中发生音爆时的位置和时间，至少需要布置四台监测设备。

对于问题三，利用问题二中所建立的数学模型，本文区分了不同残骸，并确定了各个残骸发生音爆时的位置坐标、时间和选取的设备。

对于问题四，为了模拟设备记录时间存在 0.5 秒的随机误差，本文对监测时间数据添加了随机扰动（±0.5 秒），使用问题二的数学模型重新优化求解，分析得到即使存在此随机扰动，模型仍可在大部分情况下精确定位残骸，且最终结果在合理误差范围内，这证明本文数学模型具有稳定性。

最后，本文对所建立的模型进行了讨论和分析，综合评价了模型的优缺点。

关键字：**火箭残骸回收　多边测量定位　粒子群算法**

一、 问题重述

1.1 问题背景

运载火箭将卫星或飞船送入太空后，部分箭体进入更遥远的太空，部分坠入大海，部分坠落地面。火箭飞行过程中掉下来的部分称之为残骸。如今的绝大多数火箭都为多级火箭，在发射后几分钟内，在级间分离装置的控制下，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。由于上升的高度不高，很快就会坠落回地面。

虽然部分残骸在再入大气层过程中烧蚀销毁，但依有一部分残骸会坠落至地面，如不回收，可能会造成安全事故或者成为安全隐患。因此，残骸回收工作就显得十分必要。

为了能够快速定位残骸并进行回收，科研人员在残骸理论落区内布置数台可以接收震动波的监测设备，用于接收残骸在坠落至地面过程中产生的跨音速音爆。并根据不同残骸从空中传来的跨音速音爆震动波，以及音爆抵达的时间，基于此来定位空中残骸发生音爆时的位置。之后再采用弹道外推即可实现残骸落点的精确定位。如此定位后，残骸回收的难度大大降低。

1.2 问题要求

问题一 建立数学模型，分析至少需要几台设备才能够精确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。并从给定的多台设备的位置坐标和接收到的音爆抵达时间数据中选取合适的数据，计算单个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以外还会有两个或四个助推器，也就是会产生多个残骸。多个残骸发生音爆时，监测设备在监测范围内会收到多组音爆数据。题目假设空中有 4 个残骸，每个设备按照时间先后顺序接收到 4 组震动波。建立数学模型，分析震动波与残骸的对应关系，以及如果要确定 4 个残骸音爆时的位置坐标和时间，最少需要布置的设备数量。

问题三 通过给定的监测设备位置坐标和 4 个音爆抵达时间的数据表，结合问题 2 建立的数学模型，从表中选取合适数据，确定 4 个残骸在空中发生音爆时的位置坐标和时间（4 个残骸发生音爆的时间可能不同，但互相之间的误差不超过 5s）。

问题四 假设设备记录时间存在 0.5s 的随机误差，修正问题 2 建立的模型以精确确定 4 个残骸在空中发生音爆时的位置坐标和时间。并对问题 3 表中的数据叠加随机误差，给出修正算例并分析结果误差。如果无法降低误差，提供一种解决方案，自行模拟所需的监测设备数据并加以验证。

二、 问题分析

2.1 问题一分析

对于问题一，需要确定单个残骸发生音爆时的经度、纬度、高程和时间，数据表提供了每个监测设备的位置坐标和接收到的音爆抵达时间。基于每个设备的位置坐标，可以通过数学模型或者是测距算法确定一个方程组，根据方程组的未知数得出最少需要的监测设备数量，方程组的解就是残骸发生音爆时的位置坐标和时间。再根据方程组解的准确性来判断利用哪些设备可以得到最为精确的解，该组解即为单个残骸的位置坐标和音爆时间。

2.2 问题二分析

对于问题二，需要建立数学模型来确定多个残骸的震动波和设备接收到的音爆抵达时间之间的对应关系。问题中假定有 4 个残骸，每个设备会按时间先后顺序接收到 4 组震动波。该问题可以采用与问题一相似的解决方式，确定一个方程组来得到最少需要的监测设备数量。然后通过选取不同数量的监测设备，并通过将设备数据代入方程组，计算方程组最优解来确定某一组时间数据与残骸之间相关性，由此得到震动波带来的时间数据与残骸之间的对应关系。

2.3 问题三分析

对于问题三，需要将表中数据进行处理后代入问题二的数学模型中，并测试选取哪些设备可以得到最优解，再使用这些设备的数据进行计算，得到多个残骸在空中发生音爆时的位置和时间。最后需要互相比较确认差别是否超过 5s。

2.4 问题四分析

对于问题四，需要对表中的数据加入随机误差后再次利用问题二的模型和问题三的计算方式，测试问题二中数学模型的鲁棒性。如果叠加过随机误差的设备记录时间对模型以及计算结果没有产生较大误差，则模型无需修正，直接给出误差存在下的残骸音爆位置坐标以及时间即可。如果叠加过随机误差的设备记录时间对模型以及计算结果产生了较大误差，则需要修正问题二中的模型，并提供一种新的解决方案来实现空中的准确定位以及时间计算。再自行模拟需要的监测设备位置和音爆抵达时间数据，验证修正后的模型。

三、 模型假设

为简化问题，本文做出以下假设：

- 假设 1：假设震动波的传播速度为 340 m/s 且为常数。
- 假设 2：假设计算两点间距离时可忽略地面曲率。
- 假设 3：假设纬度间每度距离值近似为 111.263 km，经度间每度距离值近似为 97.304 km。
- 假设 4：假设各监测设备的位置坐标和高程没有误差。
- 假设 5：假设各残骸音爆相互独立。

四、 符号说明

符号	说明	单位
LON	残骸音爆经度	$^\circ$
LAT	残骸音爆纬度	$^\circ$
(X, Y, Z)	残骸音爆坐标	m
(X', Y', Z')	加入误差后残骸音爆坐标	m
t_0	残骸音爆相对 0 时刻时间	s
t_0'	加入误差后残骸音爆相对 0 时刻时间	s
lon_i	监测设备 D_i 经度	$^\circ$
lat_i	监测设备 D_i 纬度	$^\circ$
(x_i, y_i, z_i)	监测设备 D_i 坐标	m
t_i	音爆抵达监测设备 D_i 测量时间	s
T_i	音爆抵达监测设备 D_i 实际时间	s
t_{ij}	监测设备 D_i 接收到第 j 个音爆的时间	s
d_i	残骸音爆位置与监测设备 D_i 间的距离	m
s	振动波传播速度	m/s
Δt_i	时间误差	s
e	时间误差平方和	s^2
e_i	监测设备 D_1 接收到第 i 个音爆的时间误差平方和	s^2
ΔD	加入误差后计算的残骸音爆位置与加入前计算的位置间的距离	m
ΔT	加入误差后计算的残骸音爆时间与加入前计算的时间的差值	s

五、 问题一的模型的建立和求解

5.1 建模准备

通过题目所给信息可知，需要通过一定数量监测设备三维坐标与音爆抵达时间，来确定残骸音爆位置坐标与时间。为了确定合适的数学模型，需要分析给定七台监测设备的特点，即它们是否互不共面。本文通过向量和行列式的方法计算验证四台监测设备是否四点共面，对于三阶行列式，它的行向量组成一个平行六面体，而其值就是这个平行六面体的体积，如果四点不共面，那么这个平行六面体的体积就不为 0。具体过程如下：首先根据假设，将设备经纬度转换为对应横纵坐标：

$$x_i = 1000 \times 97.304 \times lon_i \tag{1}$$

$$y_i = 1000 \times 111.263 \times lat_i \tag{2}$$

任选四台设备（此处以 A,B,C,D 为例），计算出三个行向量：

$$\begin{cases} \vec{a} = \left(x_2 - x_1,\ y_2 - y_1,\ z_2 - z_1\right) \\ \vec{b} = \left(x_3 - x_1,\ y_3 - y_1,\ z_3 - z_1\right) \\ \vec{c} = \left(x_4 - x_1,\ y_4 - y_1,\ z_4 - z_1\right) \end{cases} \tag{3}$$

根据上式写出行列式:

$$D = \begin{vmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{vmatrix} \tag{4}$$

最后判断行列式是否为 0，若不为 0 则代表四点不共面。通过遍历所有从七台设备选出四台的情况，使用 MATLAB 编写程序验证可得，七台监测设备互不共面。

5.2 模型建立

结合题目已知信息与数据中七台监测设备互不共面的特点，本文确定使用多边测量法来进行模型建立。由多边测量系统的工作原理[1] 可知，4 台设备瞄准同一靶标，在测量到 4 个距离后就可以确定靶标的位置；如果各个设备的基点位置已知，那么只需要 3 台设备。然而本题还需确定残骸音爆时间，等同于增加了一个未知量，因此至少需要布置 4 台监测设备。

结合本文假设分析得到，唯一影响残骸音爆位置坐标与音爆结果误差的因素只有监测设备监测到的音爆抵达时间。于是，该问题便转化为目标函数优化问题。通过从具体操作过程如下：

根据假设，将残骸音爆经纬度转换为对应横纵坐标：

$$X = 1000 \times 97.304 \times LON \tag{5}$$

$$Y = 1000 \times 111.263 \times LAT \tag{6}$$

得到 (X, Y, Z)，再结合上文所得 (x_i, y_i, z_i)，由空间中两点间距离公式：

$$d_i = \sqrt{(X - x_i)^2 + (Y - y_i)^2 + (Z - z_i)^2} \tag{7}$$

由速度计算公式：

$$s = \frac{d_i}{T_i} \tag{8}$$

解得 T_i，最后计算 Δt_i：

$$\Delta t_i = |t_0 + T_i - t_i| \tag{9}$$

为了使测得残骸音爆坐标与时间精确，需要总时间误差最小，为便于计算，将其转化为总时间误差平方和，即：

$$e = \sum_{i=1}^{n} (t_0 + T_i - t_i)^2 \tag{10}$$

5.3 模型求解

粒子群算法实现简单，可处理连续优化问题，且为多点搜索，在多样化与集中化间较为平衡[2]。基于本模型参数少，约束简单，无梯度的特点，又由前文分析得出至少需要 4 台监测设备的数据，本文采用粒子群算法遍历所有从 7 台设备选取数据的情况，以对目标函数优化求解，即计算出 e 取得最小值时 (X, Y, Z) 与 T 的解。

5.4 求解结果

根据上文分析，结合使用 MATLAB 编写程序计算所得结果，可以得出问题一求解结果：至少需要布置 4 台监测设备；选取设备 A, B, E, G，得到残骸音爆的位置为经度 110.497850 度, 纬度 27.314287 度, 高程 1154.001033 米，音爆时间相对于零时刻为 19.093961 秒。

六、 问题二的模型的建立和求解

6.1 模型建立

在问题一的模型基础上分析得到，使用监测设备接收到不同残骸音爆时间得出的 e 值一定大于使用监测设备接收到同一残骸音爆时间计算得出的 e 值。又由于无论以什么顺序遍历，使用同一组数据计算得出的 e 值一定相等。因此，为便于计算，每次遍历从第一台设备 D_1 接收到第 i 个音爆的时间开始，下文中第 i 个残骸指的是第 i 个抵达 D_1 的音爆对应的残骸。参考问题一模型10，问题二模型建立如下：

计算第 1 个残骸音爆的时间误差平方和：

$$e_1 = \sum_{i=1}^{n-1}\sum_{j=1}^{4}(t_0 + T_i - t_{ij})^2 \tag{11}$$

去除 e_1 取得最小值时使用的 t_{ij}，继续计算第 1 个残骸音爆的时间误差平方和：

$$e_2 = \sum_{i=1}^{n-1}\sum_{j=1}^{3}(t_0 + T_i - t_{ij})^2 \tag{12}$$

以此类推，算出全部 e_i。

6.2 模型求解结果

通过上述分析可得问题二求解结果：第 k 个残骸音爆的时间误差平方和 e_k 取最小值时使用的 t_{ij} 即为各个监测设备接收到该残骸的震动波，由此可确定监测设备接收到的震动波是来自哪一个残骸；而在区分出不同残骸后，便可利用实验一的模型，得出同样至少需要布置 4 台监测设备，能够确定 4 个残骸在空中发生音爆时的位置和时间。

七、 问题三的模型的建立和求解

7.1 模型建立

问题三使用的模型在问题二中已经建立，在此不再赘述。

7.2 模型求解

首先，根据给定的数据可以发现，没有两个残骸发生音爆时的位置坐标 (X, Y, Z) 与残骸音爆时间 T 较为接近，因此可以清楚地分辨不同残骸并计算其发生音爆时的位置坐标 (X, Y, Z) 与残骸音爆时间 T。在此基础上，可以进行求解。

通过问题二的分析可得，第 k 个残骸音爆的时间误差平方和 e_k 取最小值时使用的 t_{ij} 即为各个监测设备接收到该残骸的震动波，并由此可确定监测设备接收到的震动波是来自哪一个残骸。

因此需要选取表中全部时间数据进行排列组合找出第 k 个残骸以及其对应的误差平方和 e_k 最小值，这一步的目的是为了确定所有设备的时间数据中，哪些时间组合后可以与第 k 个残骸相对应，以此来找出可以确定四个残骸的四组时间数据组合，每找出一个残骸对应的时间数据组合，就将其从总时间数据中剔除，方便后续计算。

然后对每个残骸及其时间数据，再次利用采用粒子群算法遍历所有从 7 台设备选取数据的情况，以对目标函数优化求解，重新计算出在使用几台监测设备情况下可以得到最小的时间误差平方和 e_k，这一步的目的是为了确定使用 7 台监测设备中哪些设备的数据可以最为准确地求出单个残骸发生音爆时的位置坐标 (X, Y, Z) 与残骸音爆时间 T 的解。

7.3 求解结果

残骸序号不代表残骸发生音爆时的时间先后顺序，残骸序号只代表计算的先后顺序。可以通过最后求解出的残骸音爆时间 T 来确定残骸发生音爆的先后顺序。

使用 MATLAB 求解后，表格展示结果如下:

设备对应时间 / 残骸对应序号	A	B	C	D	E	F	G
1	100.767s	112.220s	188.020s	258.985s	118.443s	266.871s	163.024s
2	164.229s	169.362s	156.936s	141.409s	86.216s	166.270s	103.738s
3	214.850s	92.45s	75.560s	196.517s	78.600s	175.482s	210.306s
4	270.065s	196.583s	110.696s	94.653s	126.669s	67.274s	206.789s

表 1　残骸与时间数据组合对应关系

得到残骸 k 及其对应的时间数据组合后，我们可以利用粒子群算法遍历验证：对于单个残骸使用哪些设备的数据进行计算，可以得到最小的时间误差平方和 e_k。并在此基础上计算出该时间误差平方和 e_k 对应的残骸发生音爆时的位置坐标 (X, Y, Z) 与残骸音爆时间 T。

使用 MATLAB 求解后，表格展示结果如下:

残骸参数, 选取的设备 / 残骸对应序号	X	Y	Z	T	选取设备
1	110.499999°	27.310001°	12514.064024m	12.000737s	C, D, E, G
2	110.500000°	27.949998°	11528.687178m	13.001507s	A, B, D, E, F
3	110.299999°	27.650001°	11477.352615m	14.000559s	A, E, F, G
4	110.699999°	27.650001°	13468.539635m	14.999335s	A, B, C, E

表 2　残骸发生音爆时的位置坐标、时间和选取的设备

通过残骸发生音爆时的时间可以确定，残骸的序号恰好是残骸发生音爆的时间顺序。同时，进行简单运算可以发现，4 个残骸产生音爆的时间互相差别不超过 5s，符合题目要求。

八、 问题四的模型的建立和求解

8.1 模型建立

在建立问题四的模型之前，要先对问题二的模型进行测试。如果问题二的模型在存在 0.5s 的随机误差影响下依旧可以得到较为准确的结果，则无需对模型进行修正，也就无需建立新的模型。因此需要先对问题二的模型进行验证并分析误差，即判断各残骸和时间数据组合的对应关系是否与问题三的模型求解结果一致，以及各残骸前后位置坐标 (X, Y, Z) 与残骸音爆时间 T 相差是否过大。后者可通过加入误差后计算的残骸音爆位置与加入前计算的位置间的距离 ΔD，加入误差后计算的残骸音爆时间与加入误差前计算的时间差 ΔT 来作为衡量标准：

$$\Delta D = \sqrt{(X - X')^2 + (Y - Y')^2 + (Z - Z')^2} \tag{13}$$

$$\Delta T = |t_0 - t_0'| \tag{14}$$

8.2 模型求解

如要对问题二提出的模型进行测试，则测试过程与问题三的模型求解过程类似，只需在问题三中的 t_{ij} 加入 ±0.5s 的随机误差，再多次重复问题三的求解步骤。为不失一般性，本文进行了 50 次随机误差模拟，得到各残骸和时间数据组合的对应关系与问题三的模型求解结果关系表（相同则为 1，不同则为 0），如下所示：

次数个位 / 次数十位	0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1

表 3　各残骸和时间数据组合的对应关系与问题三的模型求解结果关系表

由表3可得各残骸和时间数据组合的对应关系与问题三的模型求解结果始终一致。基于此结果，再利用 MATLAB 计算每次模拟所得各残骸的 ΔD 与 ΔT 值，并绘制出折线图，以展示误差值随模拟次数增加的变化趋势，再计算误差平均值、平均值标准误差与标准差，从而评估模型的鲁棒性。

8.3 求解结果

各残骸 ΔD 与 ΔT 值随模拟次数增加的变化趋势图如下所示：

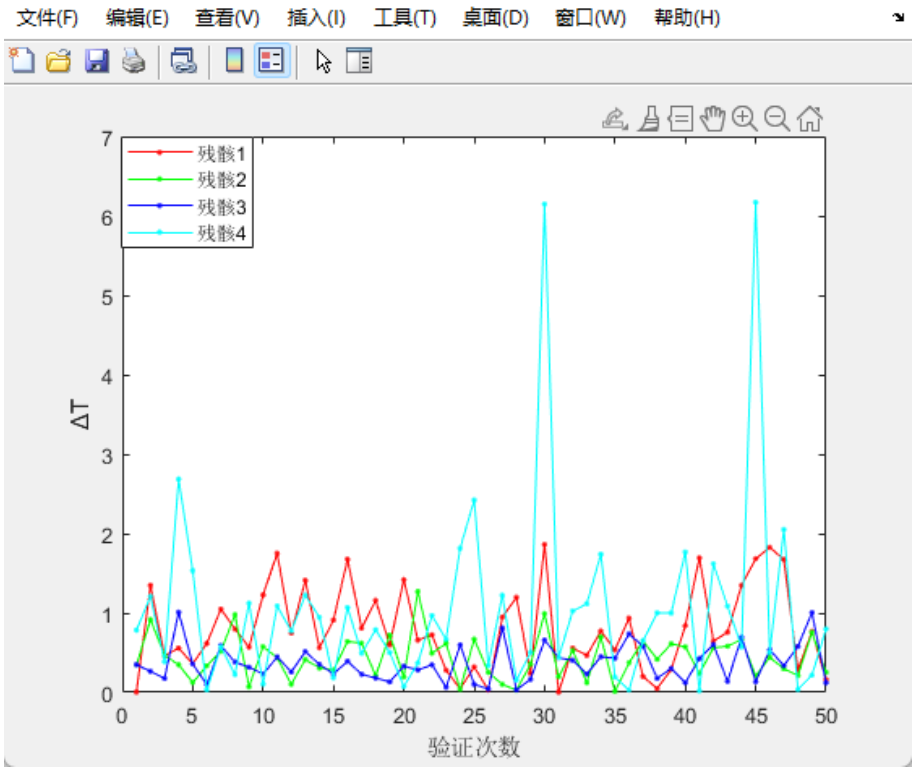

图 1　各残骸 ΔD 值随模拟次数变化趋势图

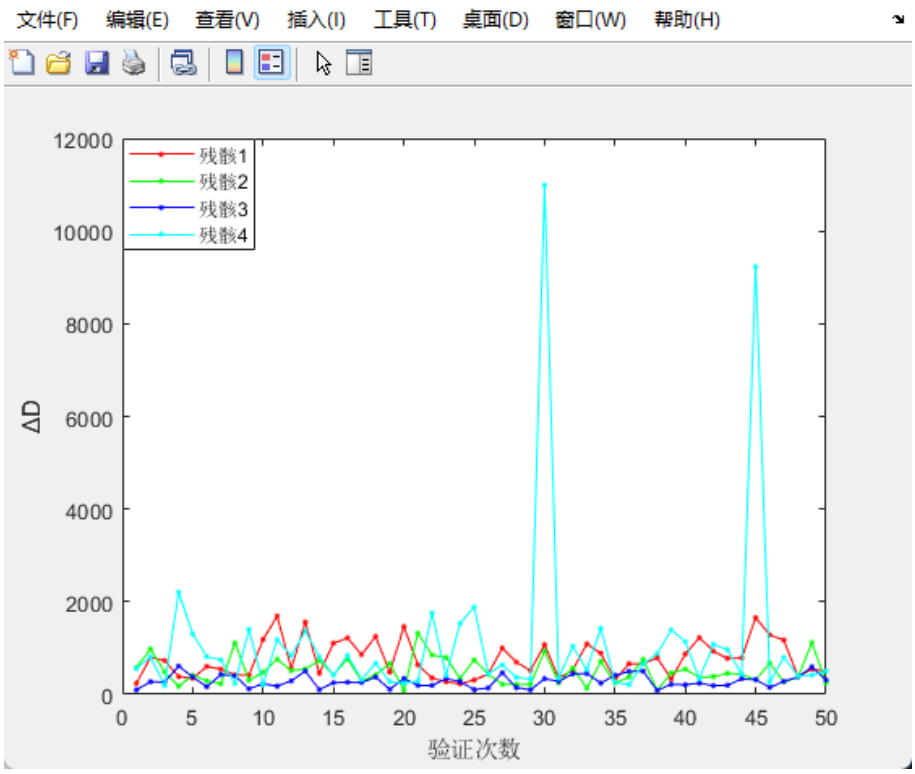

图 2　各残骸 ΔT 值随模拟次数变化趋势图

各残骸 ΔD 与 ΔT 误差平均值、平均值标准误差与标准差如下图所示：

描述统计

	N	平均值		标准差
	统计	统计	标准误差	统计
残骸1ΔD	50	751.7378	55.57389	392.96672
残骸2ΔD	50	502.3555	39.96181	282.57268
残骸3ΔD	50	285.5543	19.02130	134.50091
残骸4ΔD	50	1127.7458	271.91289	1922.71446
残骸1ΔT	50	.7966	.07580	.53596
残骸2ΔT	50	.4365	.03882	.27446
残骸3ΔT	50	.3678	.03296	.23309
残骸4ΔT	50	1.0487	.17343	1.22636
有效个案数（成列）	50			

图 3　各残骸 ΔD 与 ΔT 误差平均值、平均值标准误差与标准差

从图1与图2可以看出，95.5% 的 ΔD 在 1500m 内，90.5% 的 ΔT 在 1s 内。其中残骸 4 出现较大误差的异常值，推测可能是由于本模型最后才对残骸 4 的目标函数进行优化，这暗示本模型存在可能陷入局部最优而无法得出全局最优解的不足。根据图3分析

得到，尽管各残骸 ΔD 与 ΔT 标准差较大，随机误差带来的结果误差扰动明显，但从整体上看，ΔD 平均值低于 700m，ΔT 平均值低于 0.7s。这说明在大多数情况下，本模型依然能实现残骸空中的精确定位，所以无需对模型进行修正。

九、 模型的评价

9.1 模型的优点

• 优点 1：本数学模型基于多边测量定位算法，通过将问题转化为目标函数优化问题，利用粒子群算法进行求解，从而提高了结果的精确度。这种方法的一个重要特点是它能够在多变的实际环境中适应性强，即使在监测设备受限或环境噪声影响较大的情况下，也能保持较高的定位精度。此外，模型的高度抽象化处理使得复杂问题变得简单易懂，便于理解和应用。

• 优点 2：本数学模型建立过程完全基于数学公式的推导，确保了模型与实际情况之间的一致性。这种严格的数学推导过程不仅保证了模型的准确性和可靠性，而且提供了一种清晰的验证路径，使得模型的结果可以通过数学逻辑进行验证和复现。

• 优点 3：本数学模型模型证明只需至少四台监测设备即可实现对单个残骸的定位，这意味着系统具有较高的灵活性和可扩展性。在实际应用中，这使得模型可以根据实际情况调整监测设备的数量和布局，以适应不同的监测需求和环境条件，从而提供了一种经济高效的解决方案。

9.2 模型的缺点

• 缺点 1：本文的数学模型基于多边测量定位算法，这便限制了监测设备的布置位置。换言之，尽管通过问题一得到至少需要布置四台监测设备就能够得到残骸音爆位置与时间，但如果四台监测设备共面则无法构建方程组从而得到结果。

• 缺点 2：在震动波监测设备都布置在小范围的残骸理论落区内这一问题背景下，本文数学模型假设不考虑地球表面曲率，这同样限制了监测设备的布置位置。对于监测设备间距较远的实际情况，需要修正本模型，使用半正矢公式计算残骸音爆位置与各个监测设备间的距离。

• 缺点 3: 本数学模型是通过优化总时间误差平方和来求解残骸音爆位置与时间的，残骸音爆震动波传至各个设备的真实时间是影响最终结果的关键因素，该参数建立在震动波传播速度保持在 340m/s 不变的假设基础之上。然而，在复杂多变的现实场景中，很难排除空气密度不均匀和其他在残骸音爆位置与监测设备之间震动波对目标震动波造成的干扰。从实际应用的角度出发，需要在回收残骸前精确测定周边环境震动波的传播速度，并保持该环境的稳定。

- 缺点 4: 本文通过时间误差平方和取最小值时使用的同一组音爆抵达设备时间来区分不同残骸。然而，当两个或多个残骸在相近位置、相近时间音爆时，本模型便可能无法区分这些残骸。在这种情况下，可能要重新建立数学模型以区分残骸。

9.3 模型的改进

在优化求解时间误差平方和的最小值时，本模型使用的方法是直接全部遍历所有从 7 台设备选取数据的情况，这样的方法虽然简单直接，但并不高效。在设备数量较多或数据量较大时，全遍历方法的计算成本会急剧增加。因此，可能通过采用更高效的算法，以在更短的时间内找到近似的最优解。而在优化算法的选择上，本模型选用的是粒子群算法，有可能陷入局部最优，因此在实际应用中，可以结合多种算法的优点，如将粒子群算法与局部搜索算法结合，以提高找到全局最优解的概率。

参考文献

[1] 张国雄. 坐标测量技术发展方向[J]. 红外与激光工程, 2008, 37(S1):1-5.

[2] 杨维李歧强. 粒子群优化算法综述[J]. 中国工程科学, 2004(05):87-94.

附录 A　文件列表

文件名	功能描述
Problem1.m	问题一程序代码
Problem3_1.m	问题三程序代码第一部分
Problem3_2.m	问题三程序代码第二部分
Problem3_3.m	问题三程序代码第三部分
Problem3_4.m	问题三程序代码第四部分
Problem4_1.m	问题四程序代码第一部分
Problem4_2.m	问题四程序代码第二部分
Problem4_3.m	问题四程序代码第三部分
Problem4_4.m	问题四程序代码第四部分
verify_diff.m	用于确定加入时间误差前后的位置差和时间差
determine_equipment_and_calculation.m	用于确定得到最优解所需的设备数量, 具体的设备，以及求解残骸发生音爆时的位置和时间
check_coplanarity.m	用于证明四点不共面
calculateFval.m	用于计算最优解并返回最后的结果
verify_optimal_solution.m	[函数] 用于验证最优解, 并得出对应的设备序号
verify_model.m	[函数] 用于验证未修正的模型是否可用于存在时间误差的数据
print_result.m	[函数] 用于按格式输出结果
error_analysis.sav	各残骸 ΔD 与 ΔT 数据整理
error_analysis.sps	各残骸 ΔD 与 ΔT 误差平均值、平均值标准误差与标准差

附录 B　代码

Problem1.m

```
% 先录入数据

% 全部经度
all_lon = [110.241, 110.780, 110.712, 110.251, 110.524,
   110.467, 110.047];

% 全部纬度
all_lat = [27.204, 27.456, 27.785, 27.825, 27.617, 27.921,
   27.121];

% 全部高程
all_height = [824, 727, 742, 850, 786, 678, 575];

% 全部时间
all_time = [100.767, 112.220, 188.020, 258.985, 118.443,
   266.871, 163.024];

% 从收集的数据中选取
x_i = [];
y_i = [];
z_i = [];
t_i = [];

a = [];
results = [];

for j = 4:7
    sort_num = nchoosek(1:7, j);

    for i=1:size(sort_num, 1)
        % 遍历所有情况查看目标函数最优值
        idx = sort_num(i,:);
```

```
30
31          x_i = all_lon(1, idx);
32          y_i = all_lat(1, idx);
33          z_i = all_height(1, idx);
34          t_i = all_time(1, idx);
35
36          objective_function = @(p) ...
37          sum( (sqrt(((p(1) - x_i) * 97.304 * 10 ^ 3).^2 + ...
38          ((p(2) - y_i) * 111.263 * 10 ^ 3).^2 + ...
39          ((p(3) - z_i).^2)) / 340 + p(4) - t_i ).^2 );
40
41          rng default;
42          options = optimoptions('particleswarm', 'Display', '
    off');
43          [result, fval] = particleswarm(objective_function, ...
44              4, [-500, -500, 500, -100], [500, 500, Inf, 500],
    options);
45
46          a = [fval, result, sort_num(i,:)];
47          if j == 4
48              results = [results; a, 0, 0, 0];
49          elseif j == 5
50              results = [results; a, 0, 0];
51          elseif j == 6
52              results = [results; a, 0];
53          elseif j == 7
54              results = [results; a];
55          end
56      end
57  end
58
59  % 对当前设备组合fval进行排序, fval越小, 说明越优, 结果越可靠
60  all_fval = [results(:, 1)];
61  [~, idx] = sort(all_fval);
62  results = results(idx,:);
```

```
63
64 fprintf('残骸发生音爆时，最有可能的位置为：经度%f°，纬度%f°，
       高程%fm \n', ...
65     results(1, 2), results(1, 3), results(1, 4));
66
67 fprintf('残骸发生音爆时，最有可能的时间为：%fs(相对于观测系统
       时钟0时)\n', ...
68     results(1, 5));
69
70 fprintf('得到此数据依据的设备为：%d，%d，%d，%d，%d，%d，%d(未
       使用设备用0代替)\n', ...
71     results(1, 6), results(1, 7), results(1, 8), results(1, 9)
       , results(1, 10), results(1, 11), results(1, 12));
72
73 fprintf('因为此组设备对应的目标函数值最优，为：%e \n', ...
74     results(1, 1));
```

Problem3_1.m

```
1  % 每个设备接收到四次音爆对应的时间
2  A_time = [100.767, 164.229, 214.850, 270.065];
3  B_time = [92.453,  112.220, 169.362, 196.583];
4  C_time = [75.560,  110.696, 156.936, 188.020];
5  D_time = [94.653,  141.409, 196.517, 258.985];
6  E_time = [78.600,  86.216,  118.443, 126.669];
7  F_time = [67.274,  166.270, 175.482, 266.871];
8  G_time = [103.738, 163.024, 206.789, 210.306];
9
10 all_time = [A_time;B_time;C_time;D_time;E_time;F_time;G_time];
11
12 % 计算最优解
13 results = calculateFval(all_time);
14
15 % 对当前设备组合fval进行排序，fval越小，说明越优，结果越可靠
16 % 排序完毕后进行输出
17 print_result(results);
```

Problem3_2.m

```m
% 去除假设的第1次音爆后的时间数据
A_time = [164.229, 214.850, 270.065];
B_time = [92.453,  169.362, 196.583];
C_time = [75.560,  110.696, 156.936];
D_time = [94.653,  141.409, 196.517];
E_time = [78.600,  86.216,  126.669];
F_time = [67.274,  166.270, 175.482];
G_time = [103.738, 206.789, 210.306];

all_time = [A_time;B_time;C_time;D_time;E_time;F_time;G_time];

% 计算最优解
results = calculateFval(all_time);

% 对当前设备组合fval进行排序，fval越小，说明越优，结果越可靠
% 排序完毕后进行输出
print_result(results);
```

Problem3_3.m

```m
% 去除假设的第1，2次音爆后的时间数据
A_time = [214.850, 270.065];
B_time = [92.453,  196.583];
C_time = [75.560,  110.696];
D_time = [94.653,  196.517];
E_time = [78.600,  126.669];
F_time = [67.274,  175.482];
G_time = [206.789, 210.306];

all_time = [A_time;B_time;C_time;D_time;E_time;F_time;G_time];

% 计算最优解
results = calculateFval(all_time);

% 对当前设备组合fval进行排序，fval越小，说明越优，结果越可靠
```

```
% 排序完毕后进行输出
print_result(results);
```

Problem3_4.m

```
% 去除假设的第1, 2, 3次音爆后的时间数据
time = [270.065, 196.583, 110.696, 94.653, 126.669, 67.274,
   206.789];

all_time = time';

% 计算最优解
results = calculateFval(all_time);

% 对当前设备组合fval进行排序, fval越小, 说明越优, 结果越可靠
% 排序完毕后进行输出
print_result(results);
```

Problem4_1.m

```
% 加入了验证程序
% 用于验证4_1在多次重复实验情况下结果的稳定程度

% 四个残骸->7个设备的对应时间。
first_wreckage_time  = [100.767, 112.22, 188.02, 258.985,
   118.443, 266.871, 163.024];
second_wreckage_time  = [164.229, 169.362, 156.936, 141.409,
   86.216, 166.27, 103.738];
third_wreckage_time = [214.85, 92.453, 75.56, 196.517, 78.6,
   175.482, 210.306];
fourth_wreckage_time = [270.065, 196.583, 110.696, 94.653,
   126.669, 67.274, 206.789];

% 用于存放每次验证后的结果
all_situation = [];

for i = 1:50
    fprintf('目前正在进行第%d次验证\n', i);
```

```
    % 用于使用不同的种子，在每次验证时生成不同的随机数
    rng(i);
    % 使用rand函数生成在[-0.5, 0.5]区间内的随机误差
    random_error = -0.5 + (0.5 + 0.5) * rand(4, 7);

    % 叠加了随机误差后的时间
    error_first_wreckage_time  = first_wreckage_time  + random_error(1, 1:7);
    error_second_wreckage_time = second_wreckage_time + random_error(2, 1:7);
    error_third_wreckage_time  = third_wreckage_time  + random_error(3, 1:7);
    error_fourth_wreckage_time = fourth_wreckage_time + random_error(4, 1:7);

    % 重复实验3的步骤检验模型是否需要修正
    all_error_time = [error_first_wreckage_time; error_second_wreckage_time; ...
    error_third_wreckage_time; error_fourth_wreckage_time]';

    A_time = all_error_time(1, 1:4);
    B_time = all_error_time(2, 1:4);
    C_time = all_error_time(3, 1:4);
    D_time = all_error_time(4, 1:4);
    E_time = all_error_time(5, 1:4);
    F_time = all_error_time(6, 1:4);
    G_time = all_error_time(7, 1:4);

    all_time = [A_time;B_time;C_time;D_time;E_time;F_time;G_time];

    % 计算最优解
    results = calculateFval(all_time);

    isCorrect = verify_model(results(1, 6:12), 1);
```

```
    all_situation = [all_situation, isCorrect];
end

disp(all_situation);
```

Problem4_2.m

```
% 加入了验证程序
% 用于验证4_2在多次重复实验情况下结果的稳定程度

% 去除假设的第1个残骸后剩余残骸与设备的对应时间
second_wreckage_time  = [164.229, 169.362, 156.936, 141.409,
   86.216, 166.27, 103.738];
third_wreckage_time = [214.85, 92.453, 75.56, 196.517, 78.6,
   175.482, 210.306];
fourth_wreckage_time = [270.065, 196.583, 110.696, 94.653,
   126.669, 67.274, 206.789];

% 用于存放每次验证后的结果
all_situation = [];

for i = 1:50
    fprintf('目前正在进行第%d次验证\n', i);
    % 用于使用不同的种子，在每次验证时生成不同的随机数
    rng(i);
    % 使用rand函数生成在[-0.5, 0.5]区间内的随机误差
    random_error = -0.5 + (0.5 + 0.5) * rand(4, 7);

    % 叠加了随机误差后的时间
    error_second_wreckage_time = second_wreckage_time +
   random_error(2, 1:7);
    error_third_wreckage_time  = third_wreckage_time  +
   random_error(3, 1:7);
    error_fourth_wreckage_time = fourth_wreckage_time +
   random_error(4, 1:7);

```

```
    % 重复实验3的步骤检验模型是否需要修正
    all_error_time = [error_second_wreckage_time;
error_third_wreckage_time; error_fourth_wreckage_time]';

    A_time = all_error_time(1, 1:3);
    B_time = all_error_time(2, 1:3);
    C_time = all_error_time(3, 1:3);
    D_time = all_error_time(4, 1:3);
    E_time = all_error_time(5, 1:3);
    F_time = all_error_time(6, 1:3);
    G_time = all_error_time(7, 1:3);

    all_time = [A_time;B_time;C_time;D_time;E_time;F_time;
G_time];

    % 计算最优解
    results = calculateFval(all_time);

    % 验证结果
    isCorrect = verify_model(results(1, 6:12), 2);
    all_situation = [all_situation, isCorrect];
end

disp(all_situation);
```

Problem4_3.m

```
% 加入了验证程序
% 用于验证4_3在多次重复实验情况下结果的稳定程度

% 去除假设的第1, 2个残骸后剩余残骸与设备的对应时间
third_wreckage_time = [214.85, 92.453, 75.56, 196.517, 78.6,
   175.482, 210.306];
fourth_wreckage_time = [270.065, 196.583, 110.696, 94.653,
   126.669, 67.274, 206.789];

```

```
% 用于存放每次验证后的结果
all_situation = [];

for i = 1:50
    fprintf('目前正在进行第%d次验证\n', i);
    % 用于使用不同的种子，在每次验证时生成不同的随机数
    rng(i);
    % 使用rand函数生成在[-0.5, 0.5]区间内的随机误差
    random_error = -0.5 + (0.5 + 0.5) * rand(4, 7);

    % 叠加了随机误差后的时间
    error_third_wreckage_time  = third_wreckage_time  + random_error(3, 1:7);
    error_fourth_wreckage_time = fourth_wreckage_time + random_error(4, 1:7);

    % 重复实验3的步骤检验模型是否需要修正
    all_error_time = [error_third_wreckage_time; error_fourth_wreckage_time]';

    A_time = all_error_time(1, 1:2);
    B_time = all_error_time(2, 1:2);
    C_time = all_error_time(3, 1:2);
    D_time = all_error_time(4, 1:2);
    E_time = all_error_time(5, 1:2);
    F_time = all_error_time(6, 1:2);
    G_time = all_error_time(7, 1:2);

    all_time = [A_time;B_time;C_time;D_time;E_time;F_time; G_time];

    % 计算最优解
    results = calculateFval(all_time);

    isCorrect = verify_model(results(1, 6:12), 3);
```

```
    all_situation = [all_situation, isCorrect];
end

disp(all_situation);
```

Problem4_4.m

```
% 加入了验证程序
% 用于验证4_4在多次重复实验情况下结果的稳定程度

% 最后一个残骸与设备的对应时间
fourth_wreckage_time = [270.065, 196.583, 110.696, 94.653,
   126.669, 67.274, 206.789];

% 用于存放每次验证后的结果
all_situation = [];

for i = 1:50
    fprintf('目前正在进行第%d次验证\n', i);
    % 用于使用不同的种子，在每次验证时生成不同的随机数
    rng(i);
    % 使用rand函数生成在[-0.5, 0.5]区间内的随机误差
    random_error = -0.5 + (0.5 + 0.5) * rand(4, 7);

    % 叠加了随机误差后的时间
    error_fourth_wreckage_time = fourth_wreckage_time +
   random_error(4, 1:7);

    % 重复实验3的步骤检验模型是否需要修正
    all_error_time = error_fourth_wreckage_time';

    all_time = all_error_time;

    % 计算最优解
    results = calculateFval(all_time);

```

```
    % 验证结果
    isCorrect = verify_model(results(1, 6:12), 4);
    all_situation = [all_situation, isCorrect];
end

disp(all_situation);
```

verify_diff.m

```
% 此代码用于确定加入时间误差前后的位置差和时间差

% 四个残骸->7个设备的对应时间。
first_wreckage_time  = [100.767, 112.22, 188.02, 258.985,
   118.443, 266.871, 163.024];
second_wreckage_time  = [164.229, 169.362, 156.936, 141.409,
   86.216, 166.27, 103.738];
third_wreckage_time = [214.85, 92.453, 75.56, 196.517, 78.6,
   175.482, 210.306];
fourth_wreckage_time = [270.065, 196.583, 110.696, 94.653,
   126.669, 67.274, 206.789];

single_wreckage_position_diff = [];
single_wreckage_time_diff = [];

all_position_diff = [];
all_time_diff = [];
for j = 1:4
    for i = 1:50
        fprintf('目前正在进行第%d个残骸的', j);
        fprintf('第%d次验证\n', i);
        % 用于使用不同的种子，在每次验证时生成不同的随机数
        rng(i);
        % 使用rand函数生成在[-0.5, 0.5]区间内的随机误差
        random_error = -0.5 + (0.5 + 0.5) * rand(4, 7);

        % 叠加了随机误差后的时间
```

```
24        error_first_wreckage_time  = first_wreckage_time  +
  random_error(1, 1:7);
25        error_second_wreckage_time = second_wreckage_time +
  random_error(2, 1:7);
26        error_third_wreckage_time  = third_wreckage_time  +
  random_error(3, 1:7);
27        error_fourth_wreckage_time = fourth_wreckage_time +
  random_error(4, 1:7);
28        if j == 1
29            result1 = verify_optimal_solution(
  first_wreckage_time);
30            result2 = verify_optimal_solution(
  error_first_wreckage_time);
31        elseif j == 2
32            result1 = verify_optimal_solution(
  second_wreckage_time);
33            result2 = verify_optimal_solution(
  error_second_wreckage_time);
34        elseif j == 3
35            result1 = verify_optimal_solution(
  third_wreckage_time);
36            result2 = verify_optimal_solution(
  error_third_wreckage_time);
37        elseif j == 4
38            result1 = verify_optimal_solution(
  fourth_wreckage_time);
39            result2 = verify_optimal_solution(
  error_fourth_wreckage_time);
40        end
41
42        x_diff = ( result1(1, 2) - result2(1, 2) ) * 97.304 *
  10 ^ 3;
43        y_diff = ( result1(1, 3) - result2(1, 3) ) * 111.263 *
   10 ^ 3;
44        z_diff = result1(1, 4) - result2(1, 4);
```

```
45
46          position_diff = sqrt(x_diff^2 + y_diff^2 + z_diff^2);
47          single_wreckage_position_diff = [
    single_wreckage_position_diff, position_diff];
48
49          t_diff = result1(1, 5) - result2(1, 5);
50          single_wreckage_time_diff = [single_wreckage_time_diff
    , t_diff];
51      end
52      all_position_diff = [all_position_diff;
    single_wreckage_position_diff];
53      all_time_diff = [all_time_diff; single_wreckage_time_diff
    ];
54      single_wreckage_position_diff = [];
55      single_wreckage_time_diff = [];
56  end
57
58  figure(1);
59  plot(all_position_diff(1,:), '-r.');
60  hold on;
61  plot(all_position_diff(2,:), '-g.');
62  hold on;
63  plot(all_position_diff(3,:), '-b.');
64  hold on;
65  plot(all_position_diff(4,:), '-c.');
66
67  xlabel('验证次数');
68  ylabel('ΔD');
69
70  legend('残骸1', '残骸2', '残骸3', '残骸4');
71
72  all_time_diff = abs(all_time_diff);
73
74  figure(2);
75  plot(all_time_diff(1,:), '-r.');
```

```
hold on;
plot(all_time_diff(2,:), '-g.');
hold on;
plot(all_time_diff(3,:), '-b.');
hold on;
plot(all_time_diff(4,:), '-c.');

xlabel('验证次数');
ylabel('ΔT');

legend('残骸1', '残骸2', '残骸3', '残骸4');
```

determine_equipment_and_calculation.m

```
% 此代码用于确定得到最优解所需的设备数量，具体的是哪几个设备
% 以及求解残骸发生音爆时的位置和时间

% 四个残骸->7个设备的对应时间。
first_wreckage_time  = [100.767, 112.22, 188.02, 258.985,
   118.443, 266.871, 163.024];
second_wreckage_time = [270.065, 196.583, 110.696, 94.653,
   126.669, 67.274, 206.789];
third_wreckage_time  = [164.229, 169.362, 156.936, 141.409,
   86.216, 166.27, 103.738];
fourth_wreckage_time = [214.85, 92.453, 75.56, 196.517, 78.6,
   175.482, 210.306];

% 使用rand函数生成在[-0.5, 0.5]区间内的随机误差
% 暂时使用default种子，控制数据方便后续进行
rng('default');
random_error = -0.5 + (0.5 + 0.5) * rand(4, 7);

% 叠加了随机误差后的时间
error_first_wreckage_time  = first_wreckage_time  +
   random_error(1, 1:7);
error_second_wreckage_time = second_wreckage_time +
```

```
    random_error(2, 1:7);
error_third_wreckage_time  = third_wreckage_time  +
    random_error(3, 1:7);
error_fourth_wreckage_time = fourth_wreckage_time +
    random_error(4, 1:7);

results = verify_optimal_solution(error_third_wreckage_time);
```

check_coplanarity.m

```
% 此代码的作用是用来证明四点不共面
% 原理是如果四点共面则围成的体体积为0
% 体积不为0则四点不共面

% 示例使用
p1 = [110.241*97.304*10^3, 27.204*111.263*10^3, 824];
p2 = [110.780*97.304*10^3, 27.456*111.263*10^3, 727];
p3 = [110.712*97.304*10^3, 27.785*111.263*10^3, 742];
p4 = [110.251*97.304*10^3, 27.825*111.263*10^3, 850];

% p1, p2, p3, p4 是 1x3 向量，代表四个点的坐标
% isNotCoplanar 是一个布尔值，true 表示四点不共面，false 表示四点共面

% 构造矩阵 A
A = [p1 1; p2 1; p3 1; p4 1];

% 计算行列式
detA = det(A);

% 判断行列式是否为零
if abs(detA) > 1e-10
    isNotCoplanar = true;
    disp('四点不共面');
else
    isNotCoplanar = false;
```

```
    disp('四点共面');
end
```

calculateFval.m

```
function results = calculateFval(all_time)
% 此函数的作用是用来计算最优解Fval并返回最后的结果results

% 录入数据
% 全部经度
all_lon = [110.241, 110.783, 110.762, 110.251, 110.524,
   110.467, 110.047];

% 全部纬度
all_lat = [27.204, 27.456, 27.785, 28.025, 27.617, 28.081,
   27.521];

% 全部高程
all_height = [824, 727, 742, 850, 786, 678, 575];

num = size(all_time, 2);

A_time = all_time(1, 1:num);
B_time = all_time(2, 1:num);
C_time = all_time(3, 1:num);
D_time = all_time(4, 1:num);
E_time = all_time(5, 1:num);
F_time = all_time(6, 1:num);
G_time = all_time(7, 1:num);

x_i = all_lon;
y_i = all_lat;
z_i = all_height;
t_i = [];

a = [];
```

```
results = [];

% 逐次执行4选1，7个时间作为一组，使用问题1的公式进行计算，一共执行4^6次，选出最优的一组时间;
% 这一组时间对应的就是同一个残骸;
idx1 = 1;
t_i(1, 1) = A_time(1, idx1);
for idx2 = 1:num
    t_i(1, 2) = B_time(1, idx2);
    for idx3 = 1:num
        t_i(1, 3) = C_time(1, idx3);
        for idx4 = 1:num
            t_i(1, 4) = D_time(1, idx4);
            for idx5 = 1:num
                t_i(1, 5) = E_time(1, idx5);
                for idx6 = 1:num
                    t_i(1, 6) = F_time(1, idx6);
                    for idx7 = 1:num
                        t_i(1, 7) = G_time(1, idx7);
                        objective_function = @(p) ...
                        sum( (sqrt(((p(1) - x_i) * 97.304 * 10 ^ 3).^2 + ...
                        ((p(2) - y_i) * 111.263 * 10 ^ 3).^2 + ...
                        ((p(3) - z_i).^2)) / 340 + p(4) - t_i).^2 );

                        rng default;
                        options = optimoptions('particleswarm', 'Display', 'off');
                        [result, fval] = particleswarm(objective_function, 4, [100, 20, 500, -100], [200, 40, Inf, 500], options);

                        a = [fval, result, t_i];
```

```
                        results = [results; a];
                    end
                end
            end
        end
    end
end
```

verify_optimal_solution.m

```
function results = verify_optimal_solution(all_time)
% 此函数用于验证最优解，即使用多少个设备可以求出最优解

% 先录入数据
% 全部经度
all_lon = [110.241, 110.783, 110.762, 110.251, 110.524,
   110.467, 110.047];

% 全部纬度
all_lat = [27.204, 27.456, 27.785, 28.025, 27.617, 28.081,
   27.521];

% 全部高程
all_height = [824, 727, 742, 850, 786, 678, 575];

% 从收集的数据中选取
x_i = [];
y_i = [];
z_i = [];
t_i = [];

a = [];
results = [];

for j = 4:7
    sort_num = nchoosek(1:7, j);
```

```
25
26      for i=1:size(sort_num, 1)
27          % 遍历所有情况查看目标函数最优值
28          idx = sort_num(i,:);
29
30          x_i = all_lon(1, idx);
31          y_i = all_lat(1, idx);
32          z_i = all_height(1, idx);
33          t_i = all_time(1, idx);
34
35          objective_function = @(p) ...
36          sum( (sqrt(((p(1) - x_i) * 97.304 * 10 ^ 3).^2 + ...
37          ((p(2) - y_i) * 111.263 * 10 ^ 3).^2 + ...
38          ((p(3) - z_i).^2)) / 340 + p(4) - t_i ).^2 );
39
40          rng default;
41          options = optimoptions('particleswarm', 'Display', '
    off');
42          [result, fval] = particleswarm(objective_function, ...
43              4, [100, 20, 500, -100], [200, 40, Inf, 500],
    options);
44
45          a = [fval, result, sort_num(i,:)];
46          if j == 4
47              results = [results; a, 0, 0, 0];
48          elseif j == 5
49              results = [results; a, 0, 0];
50          elseif j == 6
51              results = [results; a, 0];
52          elseif j == 7
53              results = [results; a];
54          end
55      end
56  end
57
```

```
% 对当前设备组合fval进行排序，fval越小，说明越优，结果越可靠
all_fval = [results(:, 1)];
[~, idx] = sort(all_fval);
results = results(idx,:);

fprintf('残骸发生音爆时，最有可能的位置为：经度%f°，纬度%f°，高程%fm \n', ...
    results(1, 2), results(1, 3), results(1, 4));

fprintf('残骸发生音爆时，最有可能的时间为：%fs(相对于观测系统时钟0时)\n', ...
    results(1, 5));

fprintf('得到此数据依据的设备为：%d，%d，%d，%d，%d，%d，%d(未使用设备用0代替)\n', ...
    results(1, 6), results(1, 7), results(1, 8), results(1, 9), results(1, 10), results(1, 11), results(1, 12));

fprintf('因为此组设备对应的目标函数值最优，为：%e \n', ...
    results(1, 1));
```

verify_model.m

```
function isCorrect = verify_model(new_time, order)
% 此函数用于验证未修正的模型是否可用于存在时间误差的数据
% 验证的原理是比较求出的原来的时间数据组合和存在误差的时间数据组合
% 是否为同一组合，如果为同一组合，return 1 否则 return 0

% new_time为加入时间误差后的数据，order为对应的残骸顺序

% 导入旧的时间数据
first_wreckage_time  = [100.767, 112.22, 188.02, 258.985, 118.443, 266.871, 163.024];
second_wreckage_time  = [164.229, 169.362, 156.936, 141.409, 86.216, 166.27, 103.738];
```

```
third_wreckage_time = [214.85, 92.453, 75.56, 196.517, 78.6, 175.482, 210.306];
fourth_wreckage_time = [270.065, 196.583, 110.696, 94.653, 126.669, 67.274, 206.789];

if order == 1
    diff_abs = abs(new_time - first_wreckage_time);
elseif order == 2
    diff_abs = abs(new_time - second_wreckage_time);
elseif order == 3
    diff_abs = abs(new_time - third_wreckage_time);
elseif order == 4
    diff_abs = abs(new_time - fourth_wreckage_time);
end

if all(diff_abs(:) <= 0.5)
    isCorrect = 1;
else
    isCorrect = 0;
end
```

print_result.m

```
function print_result(results)
% 此函数的作用是按一定的格式输出结果

    all_fval = [results(:, 1)];
    [~, idx] = sort(all_fval);
    results = results(idx,:);

    fprintf(['该残骸发生音爆时，对应七个设备的接收时间为: A: %.3fs, B: %.3fs, C: %.3fs,' ...
        ' D: %.3fs, E: %.3fs, F: %.3fs, G: %.3fs (相对于观测系统时钟0时)\n'], ...
        results(1, 6), results(1, 7), results(1, 8), results(1, 9), results(1, 10), ...
```

```
        results(1, 11), results(1, 12));

    fprintf('因为此组时间对应的目标函数值最优，为: %e \n', ...
        results(1, 1));
end
```

error_analysis.sps

```
* Encoding: UTF-8.
DESCRIPTIVES VARIABLES=残骸1ΔD 残骸2ΔD 残骸3ΔD 残骸4ΔD 残骸1ΔT
    残骸2ΔT 残骸3ΔT 残骸4ΔT
  /STATISTICS=MEAN STDDEV SEMEAN.
```