# 2024 年第九届“数维杯”大学生数学

# 建模挑战赛论文

# 题目：基于移动平均与改进卡尔曼滤波器融合多重信号的飞行器定位研究

## 摘　要


针对全球卫星定位系统（GNSS）在室内、隧道或城市密集环境中的局限性，提出了一种创新的飞行器定位系统，该系统采用牛顿迭代算法融合多种机会信号以实现飞行器的自主定位。系统设计的核心在于利用环境中的无线电信号，通过解析信号中蕴含的位置和时间信息，从而在 GNSS 信号受限的情况下提供精确的导航定位。

在问题一中，建立了机会信号的数学模型，并确定了最少信号个数以唯一确定飞行器的位置。模型基于信号处理和定位理论，综合考虑了 TOA、TDOA、AOA、DFD 和 RSSI 信号，通过几何和代数方法分析，得出了在三维空间中定位所需的最少信号数量。其中，TOA、TDOA 和 DFD 信号至少需要 **4** 个发射源，而 AOA 信号至少需要 **2** 个发射源，RSSI 信号则至少需要 **4** 个发射源。在**特殊情况**下，如两发射源之间的距离为 **2D** 时 TOA 信号和 RSSI 信号最少可以用 **2** 个发射源确定飞行器的三维空间位置。

问题二中，设计了一种实时位置估计方法，通过数据预处理将 AOA **转换**为角度值，RSSI 转换为信号强度值，TOA 转换为距离估计，TDOA 转换为距离差估计，DFD 转换为频率差估计。随后，建立了基于三维坐标系的飞行器**位置参数模型**，采用**牛顿迭代法**求解**非线性方程**，以实现飞行器位置的实时估计。该方法利用发射源的初始位置坐标和飞行器的初始位置坐标，通过迭代逼近过程，快速找到满足精度要求的近似解，从而在无数据偏差的假设下，给出了飞行器在指定时间段内的**导航定位结果**。

问题三中，整合了 TOA、TDOA、DFD、AOA 和 RSSI 等多种信号信息，**定位误差分析**采用了**统计定位误差**和**克拉美罗界（CRLB）**两种方法，以评估定位系统的性能。利用**卡尔曼滤波器算法**处理数据并进行实时定位，通过预测和更新两个步骤，结合实时观测数据，提供连续且准确的定位结果。此外，设计了一种**机会信号筛选模型**，使用 **CRLB** 作为筛选标准，移除偏差较大的信号，以提高定位的准确性，并不断更新飞行器的位置估计。

问题四的分析中，建立了**评价判断方法**，以判断信号的**随机性偏差**和**常值飘移**，并设计了合理的机会信号筛选方法。通过**假设检验**和**置信区间**的概念来评估信号偏差的统计特性，设计了一种综合考虑偏差特性的信号筛选方法，以优化飞行器的定位结果。

将牛顿迭代算法应用于机会信号的实时筛选和定位问题，提出了一种新的信号融合模型，并通过卡尔曼滤波器算法实现了实时定位。此外，还对定位误差进行了深入分析，包括统计定位误差和克拉美罗界，为飞行器定位系统的优化提供了理论依据。通过仿真实验，验证了所提方法的有效性，为提高飞行器的自主导航能力提供了一种新的解决方案。



**关键词：**飞行器定位、牛顿迭代法、信号融合、卡尔曼滤波器、误差分析

# 目　　录

# 一、问题重述

## 1.1 问题背景

全球卫星定位系统（GNSS）的广泛应用为现代社会提供了极大的便利，但其在特定环境下的局限性也日益凸显。例如，在室内、隧道或城市密集环境中，GNSS 信号可能难以覆盖，而在紧急情况下，卫星信号可能受到干扰，导致导航失效。因此，开发一种独立于 GNSS 的自主导航技术显得尤为重要。机会信号导航技术利用环境中的无线电信号，为飞行器提供导航定位信息，成为一种潜在的解决方案。该技术通过解析信号中蕴含的位置和时间信息，实现飞行器的自主定位。然而，信号的偏差、噪声干扰和动态变化等因素，在实际应用中对导航定位的精度和稳定性构成挑战。因此，研究如何有效利用机会信号进行导航定位，对于提高飞行器的自主导航能力具有重要意义。

## 1.2 问题的提出

（1）问题 1 要求建立机会信号的数学表达式，并针对每一类信号确定唯一定位飞行器所需的最少信号数量。

（2）问题 2 要求根据附件 1 设计一种实时位置估计方法，并在无数据偏差的假设下，给出飞行器在指定时间段内（0-1 秒）的导航定位结果。

（3）问题 3 要求开发一种机会信号的实时筛选方法，以识别并排除偏差较大的信号，并据此给出飞行器指定时间段内的导航定位情况。

（4）问题 4 要求评价机会信号的偏差类型，并设计筛选方法，以给出飞行器在特定情况下的定位结果。

# 二、问题分析

## 2.1 问题一的分析

问题一要求我们建立机会信号的数学模型，并确定最少信号个数以唯一确定飞行器的位置。这一问题将基于信号处理和定位理论，考虑到每种信号类型提供的几何和物理信息。例如，TOA 和 TDOA 信号可以用来确定飞行器与发射源之间的距离或距离差，而 AOA 信号提供了角度信息，DFD 信号包含了频率变化信息，RSSI 信号则提供了信号强度信息。我们将利用这些信息构建数学模型，并通过几何和代数方法分析最少信号个数，以实现三维空间中的定位。

## 2.2 问题二的分析

问题二要求设计一种实时位置估计方法，并在无数据偏差的假设下，给出飞行器在指定时间段内的导航定位结果。通过预处理信号数据并转换为可用于定位的数值，进而利用牛顿迭代法对基于不同信号类型的非线性方程进行求解，以实现对飞行器在特定时间段内连续位置变化的精确估计。

## 2.3 问题三的分析

问题三要求开发一种实时筛选方法，以识别并排除偏差较大的信号，提高飞行器的导航定位精度。关键点在于构建一个信号质量评估机制，并通过该机制实时监测和筛选信号。解决思路涉及信号融合模型的建立，定位误差的统计分析，以及卡尔曼滤波器算法的应用，通过这些步骤实现对偏差信号的有效识别和排除，优化定位结果。

## 2.4 问题四的分析

问题四要求建立评价判断方法，以判断信号的随机性偏差和常值飘移，并设计合理的机会信号筛选方法。这一问题的分析将涉及到概率论和数理统计的知识。我们将使用假设检验和置信区间的概念来评估信号偏差的统计特性。对于随机性偏差，可能会采用基于最小二乘法的线性回归模型；对于常值飘移，则可能采用时间序列分析中的自适应滤波技术。完成偏差评估后，将设计一种综合考虑偏差特性的信号筛选方法，以优化飞行器的定位结果。

总体分析流程图如下:

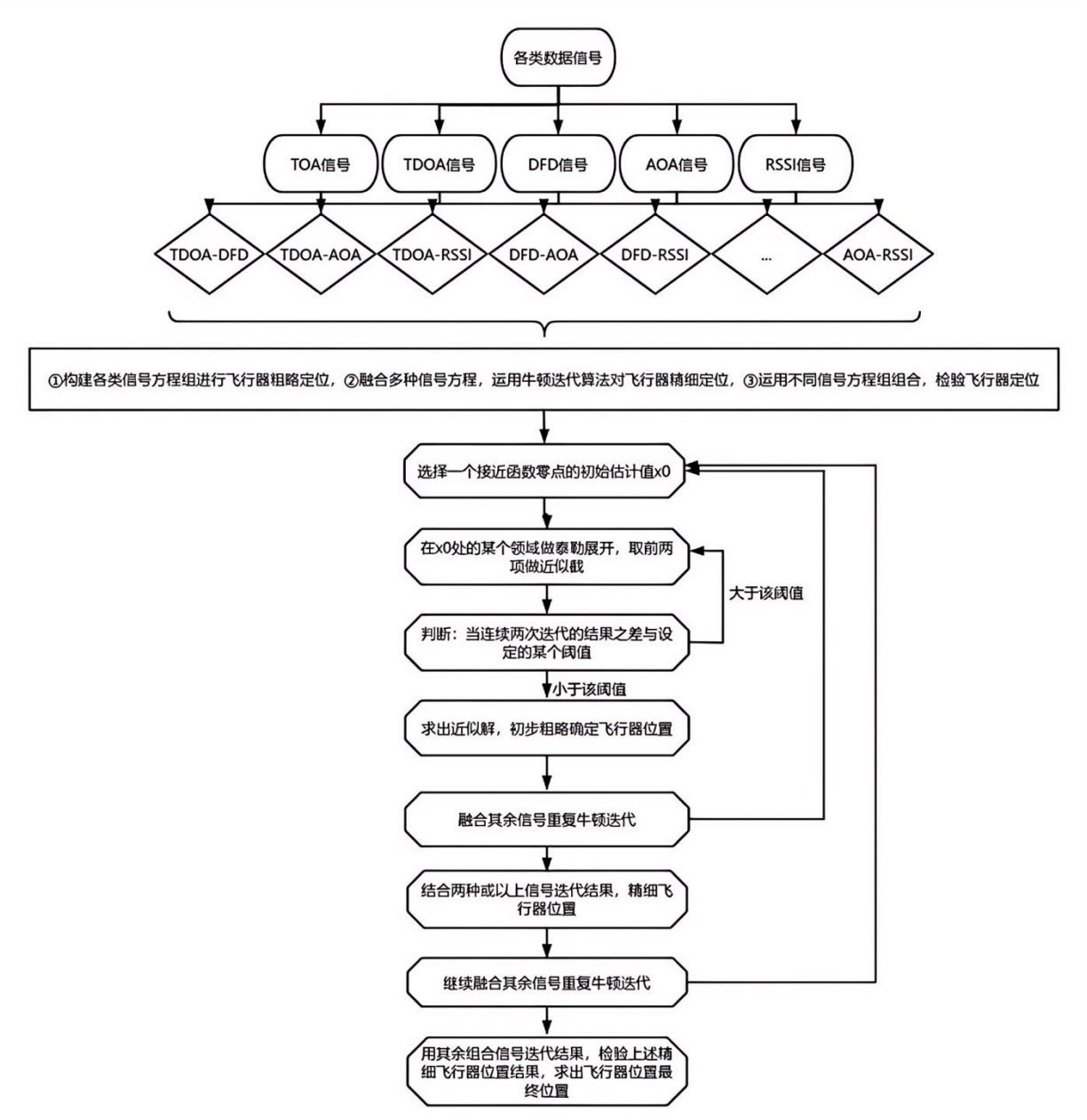


图 1 问题的总分析

## 三、模型假设

1．信号覆盖假设：假设飞行器在其飞行范围内始终能够接收到至少三个发射源的机会信号。

2．信号接收假设：飞行器的接收设备能够无误差地接收到所有机会信号，并且接收时间与发射时间的同步是精确的。

3．数据处理能力假设：飞行器的机载设备具有足够的计算能力，能够实时处理所有接收到的信号，并进行快速的定位解算。

4．算法稳定性假设：所设计的算法在面对不同发射源布局和信号特性时，均能保持稳定性和有效性。

5．模型普适性假设：建立的数学模型和算法适用于所有类型的飞行器，包括但不限于无人机、飞机、卫星等。

## 四、符号说明

| 符号 | 说明 |
|---|---|
| $\boldsymbol{P_0}$ | 信号强度 |
| $\mathbf{c}$ | 信号传播速度 |
| $\boldsymbol{f}$ | 发射源的发射信号频率 |
| $\mathbf{k}$ | 信号衰减系数 |
| $\mathbf{D}$ | 发射源与接收源的相对距离 |
| $\boldsymbol{d_0}$ | 标称距离 |
| $\boldsymbol{r_i}$ | 飞行器到节点 i 的距离 |
| $\boldsymbol{r_j}$ | 飞行器到节点 j 的距离 |
| $\boldsymbol{P_v}$ | 飞行器的位置 |
| $\boldsymbol{P_i}$ | 发射源的位置 |
| $\boldsymbol{S}$ | 接收源的位置 |
| $\boldsymbol{L}$ | 蒙特卡洛仿真次数 |
| $\hat{\mathbf{u}}_l$ | 第 i 次仿真对发射源位置的估计结果 |
| $\mathbf{Q}$ | 所有参数测量误差的协方差矩阵 |
| $\boldsymbol{X_k}$ | 第 k 个时间点的状态 |
| $\boldsymbol{F_k}$ | 状态转移矩阵 |
| $\boldsymbol{W_{k-1}}$ | 过程噪声 |
| $\boldsymbol{V_k}$ | 观测噪声 |
| $\boldsymbol{P_{k\|k-1}}$ | 预测误差协方差 |
| $\boldsymbol{Q_k}$ | 过程噪声协方差 |

# 五、问题一的模型建立与求解

## 5. 1 机会信号模型的建立

### 5.1.1 达到时间信息 TOA

TOA 信号提供了飞行器接收信号与信号发射之间的时间差。设$\boldsymbol{t_i}$为第 i 个发射源的信号发射时间，$\boldsymbol{t'_i}$为飞行器接收到信号的时间，则信号接收与发射之间的时间差 TOA 表示为：

$$TOA_i = t'_i - t_i \tag{1}$$

### 5.1.2 到达时间差信息 TDOA

TDOA 信号是指两个发射源同时发射同一信号到达接收器的时间差。设两个发射源分别为 i 和 j，那么，$\mathbf{TOA_i}$、$\mathbf{TOA_j}$表示为第 i 个和第 j 个发射源的 TOA，则两个发射源的到达时间差 TDOA 表示为：

$$TDOA_{ij} = |TOA_i - TOA_j| \tag{2}$$

### 5.1.3 多普勒频率差信息 DFD

DFD 信号是由于飞行器与信号源之间的相对速度导致的频率变化。根据题目设定坐标系为地面惯性系，**c** 是信号传播速度，设发射源为$(x_B, y_B, z_B)$，接受源为$(x_R, y_R, z_R)$，$\boldsymbol{f_i}$为第 **i** 个发射源的发射信号频率，$\boldsymbol{v_i}$为其发射源相对接收源的速度向量，$\boldsymbol{d_i}$为其发射源相对接收源的位移向量，可以表示为：

$$v_i = \frac{d_i}{TOA} \tag{3}$$

$$d_i = (x_B - x_R, y_B - y_R, z_B - z_R) \tag{4}$$

**D** 为发射源与接收源之间的距离,那么，多普勒频率差 DFD 可以表示为：

$$DFD_{ij} = \frac{f_i}{c} \times \left(\frac{v_i d_i}{D} - \frac{v_j d_j}{D}\right) \tag{5}$$

其中，发射源的发射信号频率$\boldsymbol{f}$为 $\mathbf{3\times10^8}$ Hz，信号传播速度 **c** 为光速 $\mathbf{3\times10^8}$m/s。

### 5.1.4 到达角度信息 AOA

AOA 信号提供了发射源信号的相对角度信息。根据题目设定坐标系为地面惯性系，设发射源为$(x_B, y_B, z_B)$，接受源为$(x_R, y_R, z_R)$，$\boldsymbol{\beta}$、$\boldsymbol{\alpha}$分别为发射源与接收源连线在水平和垂直方向的夹角，可以表示为：

$$\beta = \arctan\left(\frac{\sqrt{(x_B - x_R)^2 + (y_B - y_R)^2}}{|z_B - z_R|}\right) \tag{6}$$

$$\alpha = \arctan\left(\frac{y_B - y_R}{x_B - x_R}\right) \tag{7}$$

则发射源信号的相对角度信息 AOA 可以表示为：

$$\mathrm{AOA}(tan\beta) = \frac{\sqrt{(x_B - x_R)^2 + (y_B - y_R)^2}}{|z_B - z_R|} \tag{8}$$

$$\mathrm{AOA}(\tan\alpha) = \frac{y_B - y_R}{x_B - x_R} \tag{9}$$

#### 5.1.5 接收强度指标信息 RSSI

RSSI 信号为辐射源信号到达接受设备处的信号强度，反映了接收信号的强度，变化规律主要取决于不同传播环境中对应的信号衰减模型。设$\boldsymbol{P_0}$为标称信号强度，**e** 为信道衰减系数，**D** 为发射源与接收源的相对距离，$\boldsymbol{d_0}$为标称距离，则 RSSI 可以表示为：

$$RSSI = P_0\text{-}10\times \mathrm{k}\times \lg(\frac{D}{d_0}) \tag{10}$$

其中，标称信号强度$\boldsymbol{P_0}$为 **1000**，信道衰减系数 **k** 为 **20**，标称距离$\boldsymbol{d_0}$为 **1000** 米。

### 5. 2 机会信号最少信号个数的确定

#### 5.2.1 达到时间信息 TOA

每个发射源只能提供一个方程，题目中飞行器在三维空间的运动状态，即有四个未知数，分别为三维空间坐标和一个时间轴。因此，需要三个距离方程来解算三个空间坐标。所以，对于 TOA 信号，至少需要提供**四**个信号来确定飞行器在三维空间中的位置。

#### 5.2.2 到达时间差信息 TDOA

TDOA 信号基于到达时间差信息，表示为飞行器接收同一信号从两个发射源到达的时间差。该信号可以用来确定飞行器与两个发射源之间的距离差。在定位过程中，由于辐射源是非合作的，我们无法直接获得辐射源信号的发射时间。因此，在保证观测设备之间的时间同步的情况下，可以将 TOA（到达时间）转换为 TDOA（到达时间差）进行位置估计。TDOA 的参数模型可以表示为距离差的形式，即：

$$r_i - r_j = \sqrt{(x-s_{ix})^2+(y-s_{iy})^2+(z-s_{iz})^2} - \sqrt{(u-s_{jx})^2+(v-s_{jy})^2+(w-s_{jz})^2} \tag{11}$$

其中，$\boldsymbol{r_i}$和$\boldsymbol{r_j}$分别表示飞行器到节点 i 和节点 j 的**距离**，$\boldsymbol{(x,y,z)}$表示飞行器的位置，$\boldsymbol{(s_{ix},s_{iy},s_{iz})}$和$\boldsymbol{(s_{jx},s_{jy},s_{jz})}$表示节点 i 和节点 j 的**位置**。

由解析几何原理，如果一个点到两个固定点的距离之差是常数，则该点位于以这两个固定点为焦点的双曲线上。在 TDOA 定位中，每对节点 i 和节点 j 之间的距离差$\boldsymbol{r_i - r_j}$定义了一条双曲线。即一个 TDOA 参数可以确定一条双曲线，双曲线的两个焦点是两个发射源的位置。

在二维平面中定位飞行器时，根据双曲线性质，一个 TDOA 参数可以确定飞行器与两个发射源构成的双曲线。为了唯一确定飞行器的位置，需要至少两条双曲线相交于一点，这意味着至少需要三个发射源提供 TDOA 参数。所以，在三维空间中定位飞行器时，为了确保三维定位的准确性，即获得一个解，需要至少**四**个 TDOA 参数来确定飞行器的精确位置。

#### 5.2.3 多普勒频率差信息 DFD

根据题目提供的背景信息，对于运动目标，DFD 信号可用于确定目标的位置及运动特性。对于静止目标，发射源的运动可以用来确定目标的位置。在二维平面中，至少需要三个发射源才能确定目标位置，而在三维空间中，则至少需要四个发射源。

由于飞行器在三维空间中飞行，我们需要考虑三维空间中的定位问题。因此，基于 DFD 信号，为了确定飞行器的位置及运动特性，至少需要**四**个发射源的信号。这四个发射源通过提供它们与飞行器之间的 DFD 信号，允许飞行器计算出其在三维空间中的位置。

#### 5.2.4 到达角度信息 AOA

基于 AOA 的无源定位方法具备显著特性，即观测站本身并不释放任何电磁信号。该方法依赖于高精度的测向终端来捕获目标辐射源相对于观测站点的方位角数据，从而生成相应的定位示向线。在无噪声的理想环境中，这些示向线往往会相交于一个共同的点，此点即为目标的确切位置。

在 AOA 定位过程中，主要涵盖定位站点优选和目标位置解算两大关键环节。其中，首要任务是采取恰当的选站策略，以获取具备较高精度的定位观测站组合。随后，基于这一组合观测站所采集的方位角信息，结合定位算法，实现目标坐标的精确解算。

为实现在三维空间中对目标位置的唯一确定，至少需要**两**个角度信息来划定目标所在的位置。此外，通过引入更多额外的接收站，可以进一步缩小目标的潜在位置范围，显著提高定位的准确性。

#### 5.2.5 到达角度信息 RSSI

在定位 RSSI 信号时，为确保定位的准确性，通常需要获取至少**四**个发射源的信号强度信息，以便通过信号强度的差异来推断距离，进而解算位置。在实际应用中，提供信号强度信息仅为基础步骤，为实现精准定位，理论上还需结合其他信号源的信息进行综合分析。

综上所述，综合考虑以上 5 类信号，为了唯一确定飞行器的位置，我们得出以下结论：

TOA 信号：至少需要 **4** 个发射源。
TDOA 信号：至少需要 **4** 个发射源。
DFD 信号：至少需要 **4** 个发射源。
AOA 信号：至少需要 **2** 个发射源。
RSSI 信号：至少需要 **4** 个发射源。

在特殊情况下，例如两发射源之间的距离为 **2D**,通过定位算法算得两发射源到接收飞行器的距离都为 D，此时 TOA 信号和 RSSI 信号最少可以用 **2** 个发射源确定接受飞行器三维空间位置。

## 六、问题二的模型建立与求解

### 6. 1 数据预处理

在进行飞行器位置估计之前，首先需要对达到时间信息 TOA 和到达时间差信息 TDOA 数据进行数据转换：将 AOA 转换为角度值，RSSI 转换为信号强度值，TOA 转换为距离估计，TDOA 转换为距离差估计，DFD 转换为频率差估计。

### 6. 2 五种信号模型的建立与求解

#### 6.2.1 坐标系统的建立

首先，建立一个三维坐标系，其中

发射源 1 的初始位置坐标为(100,300,50)米
发射源 2 的初始位置坐标为(1000,100,150)米
发射源 3 的初始位置坐标为(-1000,-100,450)米
发射源 4 的初始位置坐标为(-200,-100,1050)米
接收飞行器的初始位置坐标为$(x_R, y_R, z_R)$米

基于收集的数据建立飞行器位置的参数模型,设定飞行器的位置为$P_v$=(x,y,z)，各个

发射源的位置分别为$P_i$=($x_i$,$y_i$,$z_i$),接受源为$S(x_R, y_R, z_R)$，其中 i=1,2,...,n。

### 6.2.2 达到时间信息 TOA 模型的建立与求解

先使用 TOA 测量，计算飞行器到每个接受源的距离为$d_i$=$\|P_v - S\|$。对于每个传感器,TOA 提供了飞行器到接受源的距离$d_i$，可以表示为

$$d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{12}$$

接着，根据 TOA 测量，建立距离方程：

$$d_i = c \times t_i \tag{13}$$

$$\begin{cases} d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} \\ d_2 = \sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} \\ \cdots \\ d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \end{cases} \tag{14}$$

完成方程构建后，对于形式复杂的非线性方程，可以运用牛顿迭代法进行模型的求解，当选取的初始值适当时，可以使结果快速向解靠拢，从而找到满足精度要求的近似解。

牛顿迭代法的公式如下：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{15}$$

其中，$x_n$为第 n 次迭代的近似解，$f(x_n)$为函数在$x_n$处的值，$f'(x_n)$为函数在$x_n$处的导数值。

对于$f(x)$，把其在$x$的某个领域做泰勒展开，如：

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x-x_0)^2}{2!} + \cdots + \frac{f^n(x_0)(x-x_0)^n}{n!} + R_n(x) \tag{16}$$

取前两项，令其等于 0,将其作为方程的近似解，

$$f(x_0) + f'(x_0)(x - x_0) = 0 \tag{17}$$

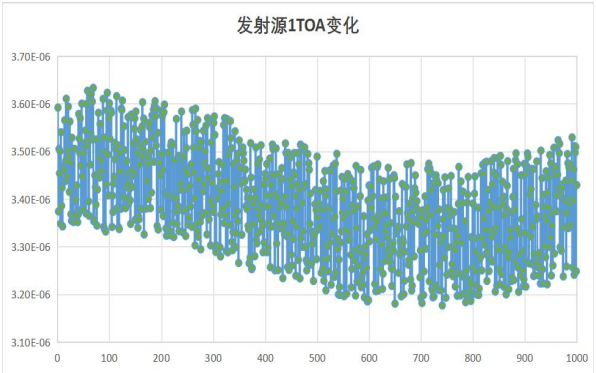


图 2 发射源 1TOA 变化图

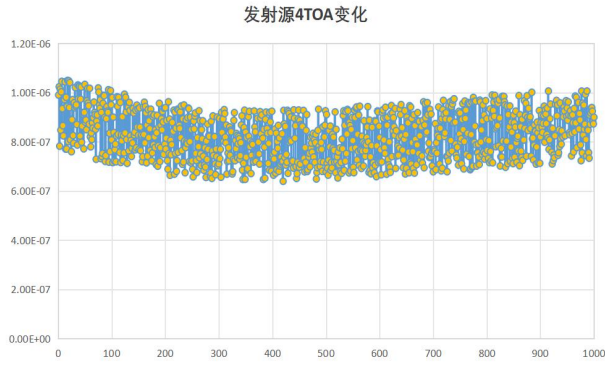


图 3 发射源 4TOA 变化图

依照上述过程重复，基于附件一的数据，通过不断使用切线来逼近函数的根，求出近似解。

### 6.2.3 到达时间差信息 TDOA 模型的建立与求解

TDOA 提供了两个接受源之间测量到的距离差的差值$\Delta d_{ij}$，可以表示为

$$\Delta d_{ij} = d_i - d_j = \sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} - \sqrt{(x-x_j)^2+(y-y_j)^2+(z-z_j)^2} \quad (18)$$

对于一对接受源$S_1$、$S_2$，计算飞行器到达这两个接受源的差值为

$$\Delta d_{12} = |d_1 - d_2| \quad (19)$$

接着，根据 TDOA[4]测量，建立距离差方程：

$$\Delta d_{ij} = c \times \Delta t_{ij} \quad (20)$$

其中 c 是信号传播速度，$\Delta t_{ij}$是 TDOA 测量值。

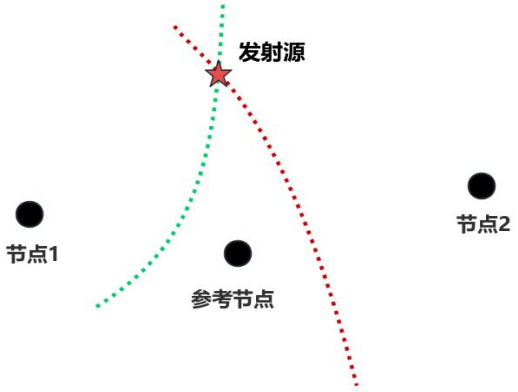


图 4 二维平面的 TDOA 定位示意图

由于 TDOA 给出的是距离差，每个 TDOA 测量可以构成一个双曲面[3]，则可以通过求解这些双曲面的交点来解决定位问题。设每 0.01s 都是一种静止状态，则可以认为每 0.01s 中接收飞行器的轨迹是以两发射源为焦点的双叶双曲线。

若(h,k,l)是双曲线的中心坐标，那么对于双叶双曲面$C'$可以表示为

$$C': \frac{(x-b)^2}{a^2} - \frac{(y-k)^2}{b^2} - \frac{(z-l)^2}{c^2} = 1 \quad (21)$$

根据附件一中接收情况一中的数据，有发射源一与发射源二和发射源三与发射源四的 TDOA 数据，同时有发射源一与发射源四的 TOA 数据，所以我们可以构建公式联立：

$$\begin{cases} \Delta d_{12} = d_1 - d_2 = \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} - \sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2} \\ \Delta d_{14} = d_1 - d_4 = \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} - \sqrt{(x-x_4)^2+(y-y_4)^2+(z-z_4)^2} \\ \Delta d_{34} = d_3 - d_4 = \sqrt{(x-x_3)^2+(y-y_3)^2+(z-z_3)^2} - \sqrt{(x-x_4)^2+(y-y_4)^2+(z-z_4)^2} \end{cases} \quad (22)$$

若(h,k,l)是双曲线的中心坐标，那么对于上述方程组可以建立多个双叶双曲面公式，其中双叶双曲面$C'$分别可以表示为

$$\begin{cases} C'_1: \frac{(x_1-b)^2}{a^2} - \frac{(y_1-k)^2}{b^2} - \frac{(z_1-l)^2}{c^2} = 1 \\ C'_2: \frac{(x_2-b)^2}{a^2} - \frac{(y_2-k)^2}{b^2} - \frac{(z_2-l)^2}{c^2} = 1 \\ C'_2: \frac{(x_3-b)^2}{a^2} - \frac{(y_3-k)^2}{b^2} - \frac{(z_3-l)^2}{c^2} = 1 \end{cases} \quad (23)$$

完成方程构建后，对于此类非线性方程，同理可以运用牛顿迭代法进行模型的求解，依照上述过程重复，通过不断使用切线来逼近函数的根，求出近似解。

### 6.2.4 多普勒频率差信息 DFD 模型的建立与求解

先计算出测量飞行器信号在两个接受源处的频率差[5]$\Delta f_{ij}$，根据 DFD 测量，建立频率差方程为：

$$\Delta f_{ij} = \frac{v}{c} \times \Delta d_{ij} \tag{24}$$

其中，v 是飞行器的速度，c 是信号传播速度。

设$v_0$为发射信号的频率，则为

$$f_{ij} = \frac{v_0}{c} \times \frac{\overrightarrow{v_i}\overrightarrow{d_i}}{|\overrightarrow{d_i}|} - \frac{v_0}{c} \times \frac{\overrightarrow{v_j}\overrightarrow{d_j}}{|\overrightarrow{d_j}|} \tag{25}$$

根据附件一中接收情况一中的数据，有发射源一与发射源二的 DFD 数据，所以可以构建公式联立：

$$\begin{aligned} f_{12} &= \frac{f_0}{c}\frac{\vec{v}_1 \bullet \vec{r}_1}{\| r_1 \|} - \frac{f_0}{c}\frac{\vec{v}_2 \bullet \vec{r}_2}{\| r_2 \|} \\ &= \frac{f_0}{c}\left[\frac{v_{x_1}(x-x_1)+v_{y_1}(y-y_1)+v_{z_1}(z-z_1)}{\sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2}} - \frac{v_{x_2}(x-x_2)+v_{y_2}(y-y_2)+v_{z_2}(z-z_2)}{\sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2}}\right] \end{aligned} \tag{26}$$

完成方程构建后，对于此类非线性方程，同理可以运用牛顿迭代法进行模型的求解，依照上述过程重复，通过不断使用切线来逼近函数的根，求出近似解。

### 6.2.5 到达角度信息 AOA 模型的建立与求解

设在每个传感器处测量飞行器信号的到达角度为$\theta_i$，再根据 AOA[4]测量建立方向方程：

$$\theta_i = \arctan\left(\frac{y-y_i}{x-x_i}\right) \tag{27}$$

对于 AOA 信号，设发射源 4 的 AOA($\tan\alpha$)信号设为 a，AOA($tan\beta$)信号设为 b，飞行器的坐标（x,y,z）建立如下方程式：

$$\begin{cases} y_4 - y = ax_1 - ax \\ b = sqrt\left(\dfrac{{d_4}^2 - \left(z_4 - z\right)^2}{z_4 - z}\right) \end{cases} \tag{28}$$

AOA 信号提供了发射源信号的相对角度信息。$\boldsymbol{\beta}$、$\boldsymbol{\alpha}$分别为发射源与接收源连线在水平和垂直方向的夹角，可以表示为：

$$\beta = \arctan\left(\frac{\sqrt{(x_B-x_R)^2+(y_B-y_R)^2}}{|z_B-z_R|}\right) \tag{29}$$

$$\alpha = \arctan\left(\frac{y_B-y_R}{x_B-x_R}\right) \tag{30}$$

即发射源信号的相对角度信息 AOA 可以表示为：

$$\mathrm{AOA}(tan\beta) = \frac{\sqrt{(x_B-x_R)^2+(y_B-y_R)^2}}{|z_B-z_R|} \tag{31}$$

$$\mathrm{AOA}(\tan\alpha) = \frac{y_B-y_R}{x_B-x_R} \tag{32}$$

在理想情况下，利用上式求解即可得到目标辐射源的真实位置。实际定位环境中，考虑到误差存在且双站所捕获的测量信息较少，可通过增加站点数量的方式获得更多测量信息并进行融合处理，从而得到高精度的目标定位解。

此时建立$\beta$关系式如下：

$$\begin{cases} \mathrm{AOA}(tan\beta_1) = \dfrac{\sqrt{(x_1 - x_R)^2 + (y_1 - y_R)^2}}{|z_B - z_R|} \\ \mathrm{AOA}(tan\beta_2) = \dfrac{\sqrt{(x_2 - x_R)^2 + (y_2 - y_R)^2}}{|z_B - z_R|} \\ \mathrm{AOA}(tan\beta_3) = \dfrac{\sqrt{(x_3 - x_R)^2 + (y_3 - y_R)^2}}{|z_3 - z_R|} \end{cases} \tag{33}$$

建立$\alpha$关系式如下：

$$\begin{cases} \mathrm{AOA}(\tan\alpha_1) = \dfrac{y_1 - y_R}{x_1 - x_R} \\ \mathrm{AOA}(\tan\alpha_2) = \dfrac{y_2 - y_R}{x_2 - x_R} \\ \mathrm{AOA}(\tan\alpha_3) = \dfrac{y_3 - y_R}{x_3 - x_R} \end{cases} \tag{34}$$

同理，完成方程构建后，对于此类非线性方程，可以运用牛顿迭代法进行模型的求解，依照上述过程重复，通过不断使用切线来逼近函数的根，求出近似解。

### 6.2.6 到达角度信息 RSSI 模型的建立与求解

对于 RSSI 信号，发射源 i 的 RSSI 信号 RSSIi[1]可以与接收源飞行器的（x,y,z）坐标建立下面的方程式：

$$\begin{gathered} \mathrm{RSSI}_i = \mathrm{R}_0 - 10 \times \mathrm{k} \times \lg(\frac{d_i}{d_0}) \\ 10\mathrm{k} \times \lg(\frac{d_i}{d_0}) = \mathrm{R}_0 - \mathrm{RSSI}_i \\ \frac{10\mathrm{k}}{10\mathrm{k}} \frac{d_i}{d_0} = e^{[\mathrm{R}_0 - \mathrm{RSSI}_i/10k]} \\ d_i = e^{[\mathrm{R}_0 - \mathrm{RSSI}_i/10k]} \times d_0 \end{gathered} \tag{35}$$

同理

$$\begin{cases} d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} \\ d_2 = \sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} \\ \cdots \\ d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \end{cases} \tag{36}$$

同理，完成方程构建后，对于此类非线性方程，可以运用牛顿迭代法进行模型的求解，依照上述过程重复，通过不断使用切线来逼近函数的根，求出近似解。

### 6. 3 结果

在不考虑数据偏差的情况下，计算出飞行器 0 秒至 10 秒的导航定位结果如下表所示。

表 1 飞行器 0 秒至 10 秒的导航定位结果

| 时间(s) | x | y | z |
|---|---|---|---|
| 4.77 | 283.07474543 | 46.37547111 | 1092.67156929 |
| 4.78 | 283.10355358 | 46.34813896 | 1092.63989429 |
| 4.79 | 283.13226415 | 46.32043823 | 1092.60816623 |
| 4.80 | 283.16107231 | 46.29299505 | 1092.57648243 |
| 4.81 | 283.18982629 | 46.26553322 | 1092.54478403 |
| 4.82 | 283.2185839 | 46.23795835 | 1092.51307736 |
| 4.83 | 283.24734013 | 46.21047892 | 1092.48137818 |
| 4.84 | 283.27611948 | 46.18299752 | 1092.4496849 |
| 4.85 | 283.30481255 | 46.15525091 | 1092.41795077 |
| 4.86 | 283.33366779 | 46.12798492 | 1092.38629252 |
| 4.87 | 283.36236628 | 46.10034126 | 1092.35456675 |
| 4.88 | 283.39110707 | 46.07273831 | 1092.32285456 |
| 4.89 | 283.41991113 | 46.04523871 | 1092.29116627 |
| 4.90 | 283.44862202 | 46.01756921 | 1092.25944242 |
| 4.91 | 283.47736096 | 45.99000424 | 1092.22773412 |
| 4.92 | 283.50621352 | 45.96263643 | 1092.19606603 |
| 4.93 | 283.53488745 | 45.93487432 | 1092.16432741 |
| 4.94 | 283.56369534 | 45.90738898 | 1092.13264066 |
| 4.95 | 283.59239104 | 45.87964849 | 1092.10090794 |
| 4.96 | 283.62117342 | 45.85223053 | 1092.06922209 |
| 4.97 | 283.64991369 | 45.82458905 | 1092.0375078 |
| 4.98 | 283.67867036 | 45.79695894 | 1092.00579815 |
| 4.99 | 283.7074362 | 45.76943178 | 1091.97409938 |
| 5.00 | 283.73611985 | 45.74162706 | 1091.94235916 |
| 5.01 | 283.76492764 | 45.71415803 | 1091.91067487 |
| 5.02 | 283.79363513 | 45.6864794 | 1091.87895069 |
| 5.03 | 283.8223462 | 45.65865563 | 1091.84721527 |
| 5.04 | 283.85116416 | 45.63131562 | 1091.81554499 |
| 5.05 | 283.87984536 | 45.60342677 | 1091.78379715 |
| 5.06 | 283.90862981 | 45.57590916 | 1091.75210512 |

飞行器的飞行轨迹如下图所示：

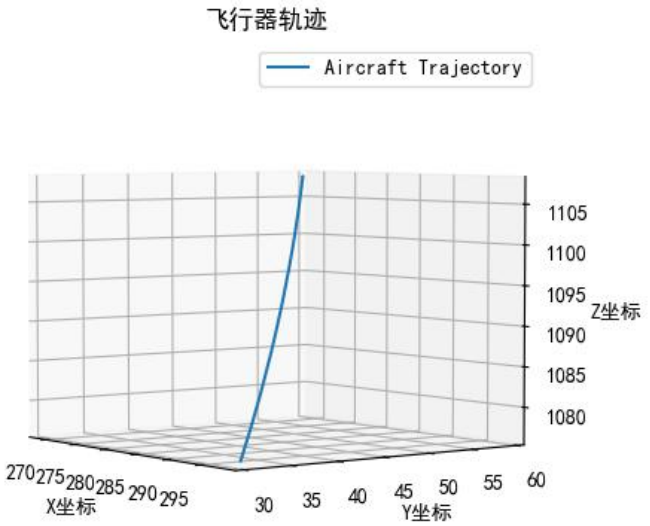


图 5 飞行器飞行轨迹

# 七、问题三的模型建立与求解

TOA、TDOA、DFD、AOA 和 RSSI 每种信号类型都提供了关于飞行器位置或运动状态的不同信息。为了提供更全面的解决方案，我们应该考虑结合这些不同类型的信号来优化定位结果[7]，以确保融合结果的准确性和可靠性。

## 7.1 定位误差分析

题目要求设计一种飞行器实时定位方法，并在存在偏差信号的情况下进行信号筛选和位置估计。这要求能够处理非线性的定位模型，同时具备良好的噪声抑制能力和实时性。因此，在进行定位误差分析时，我们采用两种主要方法：统计定位误差和克拉美罗界（CRLB）。

首先，通过**误差分析方法**来评估定位系统的性能；其次，我们使用**卡尔曼滤波器算法**来处理数据并进行实时定位；最后，通过筛选偏差信号来提高定位的准确性。

### 7.1.1 统计定位误差

统计定位误差通常通过均方误差（MSE）或均方根误差（RMSE）来衡量。这些指标可以通过多次实验并收集数据来计算得到。在应用中，通过收集到的定位数据，能计算估计位置与真实位置之间的差异。

$$\mathrm{MSE}\ (\hat{\mathbf{u}}) = \mathbb{E}\{(\hat{\mathbf{u}} - \mathbf{u})^2\} \tag{37}$$

其中，MSE 值越小，说明估计位置与真实位置之间的差异越小，则定位精度越高。

利用多次蒙特卡洛仿真对 MSE 进行统计，则定位的均方误差具体可以表示为：

$$\mathrm{MSE}\ (\hat{\mathbf{u}}) = \frac{1}{L}\sum_{l=1}^{L}(\hat{\mathbf{u}}_l - \mathbf{u})^2 \tag{38}$$

其中，L 表示蒙特卡洛仿真次数，$\hat{\mathbf{u}}_l$表示第 i 次仿真对发射源位置的估计结果。

则均方根误差是指均方误差的平均根：

$$\mathrm{RMSE}\ (\hat{\mathbf{u}}) = \sqrt{\frac{1}{L}\sum_{l=1}^{L}(\hat{\mathbf{u}}_l - \mathbf{u})^2} \tag{39}$$

### 7.1.2 克拉美罗界

克拉美罗界能提供一个理论上的定位误差下界，是一个基于测量噪声统计特性的误差界限。对于给定的定位系统，CRLB 可以帮助我们理解在最佳条件下可以达到的最小误差。

克拉美罗界 CRLB 是衡量参数估计性能的重要标准，是指无偏估计中方差最小的估计，可以通过 Fisher 矩阵间接求解 CRLB，即：

$$\mathrm{CRLB}\ (\mathbf{u}) = \mathrm{FIM}^{-1}\ (\mathbf{u}) \tag{40}$$

其中，Fisher 矩阵的公式为：

$$\mathrm{FIM}\ (\mathbf{u}) = \mathbb{E}\left[\left(\frac{\partial \ln p(\hat{\boldsymbol{\phi}}\mathbf{u})}{\partial \mathbf{u}}\right)^{T}\left(\frac{\partial \ln p(\hat{\boldsymbol{\phi}}\mathbf{u})}{\partial \mathbf{u}}\right)^{\tau}\right] \tag{41}$$

$p(\hat{\phi}u)$为包含发射源位置信息 u 的测量向量$\phi$的概率密度函数。假设参数测量误差符合高斯分布且相互独立，则有：

$$\text{FIM}(u) = \left[\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{u}^T}\right]^T \mathbf{Q}^{-1} \left[\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{u}^T}\right] \tag{42}$$

**Q** 为所有参数测量误差的协方差矩阵。根据算法，基于附件一的数据信息进行求解。

### 7.1.3 卡尔曼滤波器

针对第三问，我们使用卡尔曼滤波器算法来筛选出偏差较大的机会信号，并进行实时定位。卡尔曼滤波器是一种递归的估计算法，它利用一系列的观测数据，对于飞行器实时定位问题，可以有效地结合实时观测数据，提供连续且准确的定位结果。通过贝叶斯估计来递归地估计未知参数。

卡尔曼滤波器[8]的核心在于**预测**和**更新**两个步骤。在预测步骤中，滤波器预测了在下一个时间点的状态。在更新步骤中，滤波器利用实际观测数据来更新预测，从而得到一个更准确的估计。

其状态模型可以表示为

$$X_k = F_k X_{k-1} + W_{k-1} \tag{43}$$

其中，$X_k$是第 k 个时间点的状态，$F_k$是状态转移矩阵，$W_{k-1}$是过程噪声。

其观测模型可以表示为

$$Z_k = H_k X_k + V_k \tag{44}$$

其中，$Z_k$是第 k 个时间点的观测数据，$H_k$是状态转移矩阵，$V_k$是观测噪声。

### 7.2 机会信号筛选模型的建立与求解

首先，在求解之前，我们需要对卡尔曼滤波器进行初始化。选择飞行器的初始位置$x_0$，以及初始误差协方差矩阵$P_0$。

接着，基于附件一提供的 TDOA 对于时间步 k，运用卡尔曼滤波器进行求解，具体求解步骤如下：

**Step1）状态预测**

$$\widehat{X}_{k|k-1} = F_k \widehat{X}_{k-1} \tag{45}$$

其中，$\widehat{X}_{k|k-1}$是时间步 k 的预测状态，$F_k$是状态转移矩阵。

**Step2）误差协方差预测**

$$P_{k|k-1} = F_k P_{k-1} F_K^T + Q_k \tag{46}$$

其中，$P_{k|k-1}$是预测误差协方差，$Q_k$是过程噪声协方差。

**Step3）更新步骤**

当接收到时间步 k 的观测数据$Z_k$后，计算卡尔曼增益：

$$K_k = P_{k|k-1} H_K^T （H_k P_{k|k-1} H_K^T + R_k）^{-1} \tag{47}$$

其中，$K_k$是卡尔曼增益，$H_k$是观测矩阵，$R_k$是观测噪声协方差。

**Step4）更新状态估计**

根据观测数据修正观测状态的结果，表达式为：

$$\widehat{X}_k = \widehat{X}_{k|k-1} + K_k\ （Z_k - H_k\widehat{X}_{k|k-1}）\tag{48}$$

**Step5）更新误差协方差**

$$P_k = \ （I - K_kH_k）P_{k|k-1}\tag{49}$$

其中，I 是单位矩阵。

**Step6）筛选偏差信号**

使用 **CRLB** 作为筛选标准，比较每个信号测量的误差与 CRLB 的**比值**，如果比值超过某个阈值，则认为该信号存在较大偏差，将其从数据集中移除。

**Step7）实时定位**

重复上述预测和更新步骤，以实时接收的信号数据$\mathbf{Z_k}$为输入，不断更新飞行器的位置估计$\widehat{\mathbf{X}}_\mathbf{k}$。

## 7. 3 结果

对求解得到的数据结果进行可视化处理：

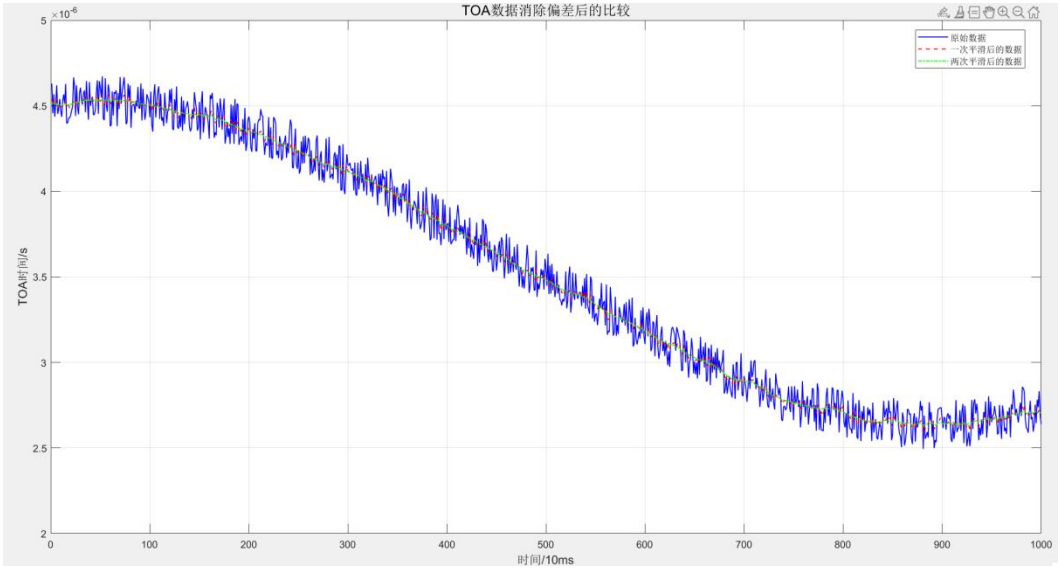


图 6TOA 数据消除偏差后的比较

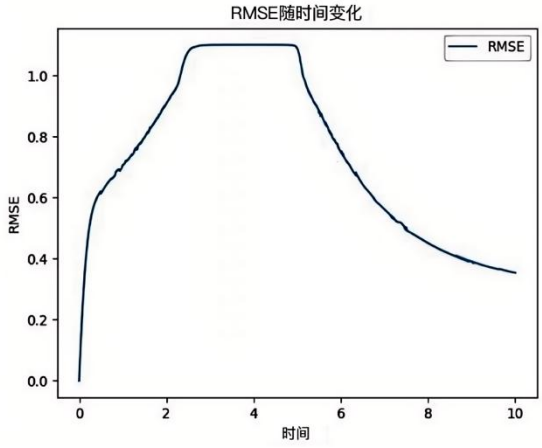


图 7RMSE 随时间变化

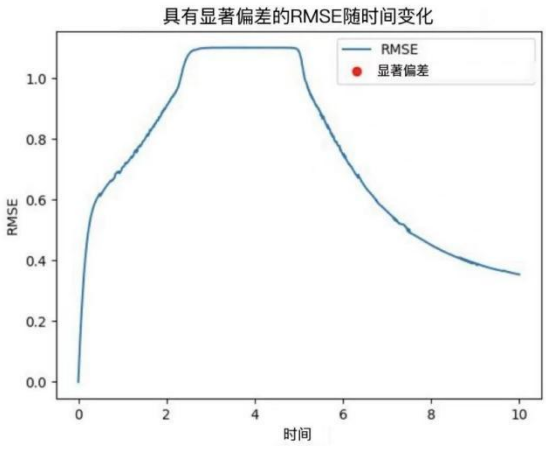


图 8 具有显著偏差的 RMSE 随时间变化

对求解得到的飞行器 0 秒至 10 秒的导航定位情况进行可视化：

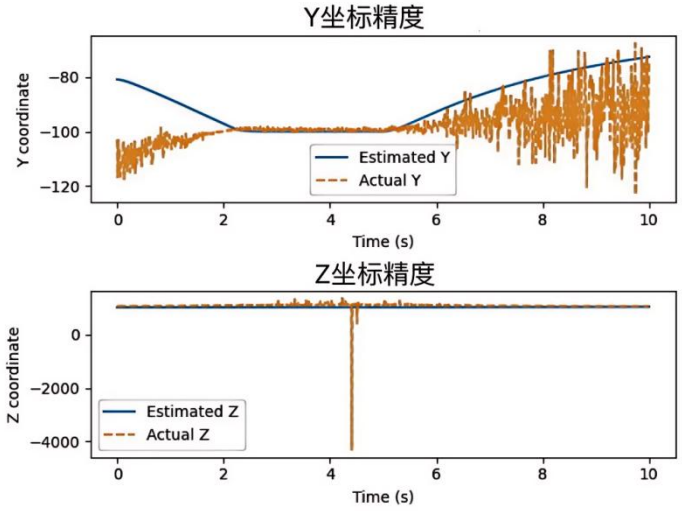


图 9 飞行器 0 秒至 10 秒的导航定位情况

此时飞行器 0 秒至 10 秒的导航定位情况如下表所示：

表 2 飞行器 0 秒至 10 秒的导航定位情况

| 时间(s) | x | y | z | RMSE |
|---|---|---|---|---|
| 2.00 | -160.2043005 | -97.16895234 | 1014.02776702 | 0.9098 |
| 2.01 | -160.06462965 | -97.25709119 | 1014.01089568 | 0.9118 |
| 2.02 | -159.92438522 | -97.34522643 | 1013.99415529 | 0.9143 |
| 2.03 | -159.78371099 | -97.43281208 | 1013.97786443 | 0.9145 |
| 2.04 | -159.64240213 | -97.52019941 | 1013.96165453 | 0.9168 |
| 2.05 | -159.49991856 | -97.60738165 | 1013.94582212 | 0.9215 |
| 2.06 | -159.35666918 | -97.69408875 | 1013.93036277 | 0.9235 |
| 2.07 | -159.21293477 | -97.7800377 | 1013.91529924 | 0.9235 |
| 2.08 | -159.06848302 | -97.86557412 | 1013.90030878 | 0.9257 |
| 2.09 | -158.92270193 | -97.95084015 | 1013.88563429 | 0.9310 |
| 2.10 | -158.77611084 | -98.03545712 | 1013.87136727 | 0.9329 |

## 八、问题四的模型建立与求解

首先，使用读取 Excel 文件中的数据，包括时间轴和各类信号值，完成物理常量的定义以及对发射源位置与速度的定义，为信号传播模型提供空间和运动信息。

### 8.1 信号模型的建立与求解

根据各类信号的物理特性，建立信号与飞行器位置之间的**关系模型**。例如，对于 TOA 信号，建立如下关系：

$$TOA_i = \frac{d_i}{c} \tag{50}$$

对于 RSSI 信号，建立如下关系：

$$\mathrm{RSSI}_i = s \times 10^{-\frac{k}{10}} \times log_{10} d_i \tag{51}$$

然后进行**残差函数**的定义，该函数计算观测值与根据飞行器位置计算的理论值之间的差异。初始化**卡尔曼滤波器**的状态，包括初始状态估计和协方差矩阵，以及过程和观测噪声的协方差矩阵。

模型的求解具体步骤如下：

**Step1）最小二乘法位置优化**

对每个时间步，使用最小二乘法优化算法，通过迭代求解**最小化**残差平方和的位置。

并根据回归分析的相关方法，计算均值和标准差来评估随机性偏差，其表达式分别为：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{52}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2} \tag{53}$$

**Step2）卡尔曼滤波器的应用**

将优化得到的位置作为观测数据，应用卡尔曼滤波器[8]进行轨迹平滑处理。

**Step3）RMSE 计算**

计算每个时间步的 RMSE，作为定位偏差的量化指标：

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (residuals_i)^2} \tag{54}$$

**Step4）显著偏差识别**

通过设定阈值，识别 RMSE 中的显著偏差点。

**Step5）常值飘移和随机性偏差识别**

通过滑动窗口分析 RMSE 序列，识别通过 RMSE 的移动平均值变化率的**常值飘移**和通过 RMSE 的移动标准差的**随机性偏差**。

8. 2 结果

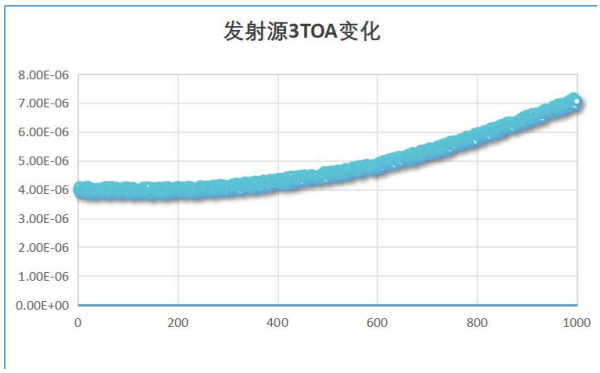


图 10 发射源 3TOA 数据可视化

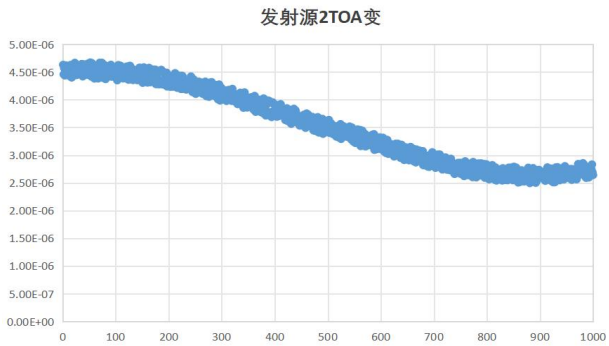


图 11 发射源 2TOA 数据可视化

完成评价判断方法的建立后进行求解，在接收情况 2 下的飞行器的飞行轨迹可视化如下图所示。

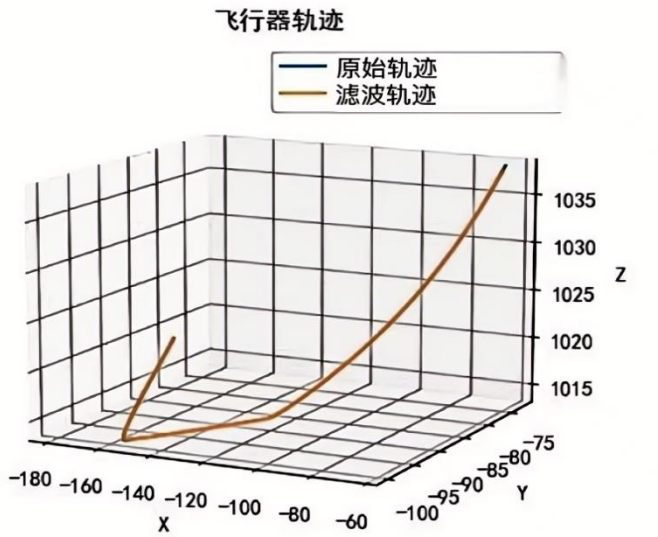


图 12 飞行器的飞行轨迹

在接收情况 2 下的飞行器 0 秒至 10 秒的部分定位结果如下表所示：

表 3 飞行器 0 秒至 10 秒的定位结果

| 时间(s) | x | y | z | RMSE |
|---|---|---|---|---|
| 5.90 | -89.28502837 | -93.59470011 | 1020.63942976 | 0.7768 |
| 5.91 | -89.17911991 | -93.51019945 | 1020.68080143 | 0.7780 |
| 5.92 | -89.07328386 | -93.42551337 | 1020.72199672 | 0.7781 |
| 5.93 | -88.9686692 | -93.34161044 | 1020.76349199 | 0.7710 |
| 5.94 | -88.86498729 | -93.25849399 | 1020.80457295 | 0.7639 |
| 5.95 | -88.76130072 | -93.1749883 | 1020.84606194 | 0.7658 |
| 5.96 | -88.65841542 | -93.0920854 | 1020.88725221 | 0.7601 |
| 5.97 | -88.55601482 | -93.00949036 | 1020.92837745 | 0.7570 |
| 5.98 | -88.45409772 | -92.92715788 | 1020.96947431 | 0.7541 |
| 5.99 | -88.35267311 | -92.84506277 | 1021.01055606 | 0.7513 |
| 6.00 | -88.25174485 | -92.76318835 | 1021.05163022 | 0.7486 |

# 九、模型评价、改进与推广

## 9.1 模型的优点

1．多信号融合：模型能够综合 TOA、TDOA、DFD、AOA 和 RSSI 等多种信号信息，提高了定位的准确性和鲁棒性。

2．实时性：采用卡尔曼滤波器算法，模型能够实时处理信号数据并更新飞行器的位置估计，满足了实时定位的需求。

3．误差分析：通过统计定位误差和克拉美罗界（CRLB）对定位系统的性能进行了全面的评估，提供了误差的理论下界。

4．信号筛选机制：设计了基于信号质量的实时筛选方法，能够识别并排除偏差较大的信号，优化了定位结果。

5．算法适用性：模型假设考虑了不同发射源布局和信号特性，设计的算法具有较好的普适性和稳定性。

### 9.2 模型的缺点

1．环境因素简化：在信号传播和定位计算中忽略了大气条件、建筑物遮挡等环境因素，可能会影响定位精度。

2．数据同步假设：假设了飞行器的接收设备能够无误差地接收所有机会信号，并且接收时间与发射时间的同步是精确的，这在实际应用中可能难以保证。

3．计算能力要求：模型假设飞行器的机载设备具有足够的计算能力进行实时处理，这可能限制了模型在计算能力较低的设备上的应用。

### 9.3 模型的改进与推广

1、改进之处主要包括以下几点：

(1) 算法优化：针对牛顿迭代法在求解非线性方程时可能出现的收敛速度慢或不收敛的问题，可以引入更高效的优化算法，如梯度下降法或拟牛顿法，以提高求解效率和稳定。

(2) 卡尔曼滤波器改进：对卡尔曼滤波器的参数进行自适应调整，以更好地适应不同飞行条件下的动态变化，提高滤波器的跟踪精度和鲁棒性。

(3) CRLB 在信号筛选中的应用：改进 CRLB 的计算方法，使其更精确地反映信号的可靠性，从而更有效地筛选出偏差较大的信号。

(4) 误差分析方法：结合蒙特卡洛仿真和引导滤波等技术，对定位误差进行更深入的分析，以识别和补偿系统性偏差。

(5) 机器学习集成：利用机器学习算法，如支持向量机（SVM）或神经网络，对信号特征进行学习，以自动识别和适应信号的动态变化。

(6) 多源数据融合：除了现有的信号类型，还可以考虑融合其他类型的传感器数据，如惯性导航系统（INS）数据，以提高定位的准确性和鲁棒性。

(7) 自适应阈值设定：在信号筛选过程中，采用自适应阈值代替固定阈值，根据实时信号环境动态调整筛选标准，以提高筛选的准确性。

2、模型推广主要可以从以下几点考虑：

(1) 多环境适应性：将模型应用于不同的地理和气候条件，如城市峡谷、山区、海洋等，以验证和调整模型在各种环境下的性能和鲁棒性。

(2) 跨技术整合：探索与现有的 GNSS 和其他导航系统的兼容性，实现无缝切换和数据融合，提高整体导航系统的可靠性。

(3) 商业应用拓展：将模型推广到商业领域，如自动驾驶车辆、无人机物流、智能交通管理系统等，以满足多样化的市场需求。

(4) 军事和安全领域：研究模型在军事导航、安全监控和紧急救援等高风险环境下的应用，增强这些场景下的定位精度和系统稳定性。

(5) 国际合作与标准化：与国际组织合作，推动模型的国际化应用，并参与相关标准的制定，以促进全球范围内的互操作性和数据共享。

# 十、参考文献

[1]郑辉,程水英,张茂乙.基于 GDOP 的多星单站无源雷达辐射源选取方法研究[J].火力与指挥控制,2023,48(11):102-108+115.

[2]耿傲婷.基于 AOA 的多站无源定位关键技术研究[D].南昌大学,2023.

[3]韩欢.基于深度学习的无源定位关键技术研究[D].电子科技大学,2023.

[4]张晨,王韦刚,许晨东,等.一种基于 TOA 的非视距误差优化定位算法[J].南京邮电大学学报(自然科学版),2022,42 (04):56-63.

[5]潘磊.基于多普勒频率差的机载无源定位技术研究[D].西南交通大学,2013.

[6]熊鑫. TDOA/AOA 组合定位技术及其在城市峡谷中的应用[D].江西理工大学,2023.

[7]朱春华,杨锦民.一种基于加权质心的 TOF 与 TDOA 联合定位算法[J].郑州大学学报(工学版),2023,44(03):50-55.

[8]裴勇军,徐晓新,毛梅娟.基于卡尔曼滤波算法的卫星导航高精度定位方法[J].北京测绘,2023,37(10):1412-1417.

[9]王晓燕,黄梓涵,汪涛,等.基于无线传感器网络与无迹卡尔曼滤波的室内定位系统设计[J].电子器件,2023,46(04):1104-1109.

[10]苏雅茜,崔超然,曲浩.基于自注意力移动平均线的时间序列预测[J].南京大学学报(自然科学),2022,58(04):649-657.

[11]艾青,杨俊杰,蒋伟,等.改进 PDR 与 RSSI 融合的室内定位方法[J].传感器与微系统,2023,42(12):75-78+82.

# 十一、附录

**附录 1**

介绍：问题一代码

```
import numpy as np
import pandas as pd
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Constants
C = 3e8    # Speed of light in m/s
f0 = 3e8    # Frequency in Hz
k = 20    # Channel attenuation coefficient
s = 100    # Nominal signal strength
nominal_distance = 1000    # Corresponding to nominal signal strength

# Initial positions and velocity vectors of the sources
source_positions = np.array([[100, 300, 50],
                             [1000, 100, 150],
                             [-1000, -100, 450],
                             [-200, -100, 1050]])
source_velocities = np.array([[0, 0, 0],
                              [0, 0, 0],
                              [0, 0, 8],
                              [20, 0, 0]])

# Load the data from Excel
file_path = 'D:/python/LDcode/database/飞行器接受情况 1.xlsx'
data = pd.read_excel(file_path)

# Extracting columns for different signals
times = data['时间 t（s）'].values
toa = data[['TOA(发射源 1)', 'TOA(发射源 4)']].values
tdoa = data[['TDOA(发射源 1,发射源 2)', 'TDOA(发射源 3,发射源 4)']].values
dfd = data[['DFD(发射源 1,发射源 2)']].values
aoa = {'alpha': data['AOA[tan(alpha)](发射源 4)'].values,
       'beta': data['AOA[tan(beta)](发射源 4)'].values}
rssi = data[['RSSI(发射源 1)', 'RSSI(发射源 2)', 'RSSI(发射源 3)']].values

# Initialize aircraft position (starting guess)
initial_guess = np.array([0, 0, 0])

# Function to calculate the residuals for Newton's method
def residuals(position, toa, tdoa, dfd, rssi, aoa, times):
    x, y, z = position
    res = []

    for t_idx, t in enumerate(times):
        toai = toa[t_idx]
        tdoaij = tdoa[t_idx]
        dfdij = dfd[t_idx]
        rssi_i = rssi[t_idx]
```

```
            aoa_alpha = aoa['alpha'][t_idx]
            aoa_beta = aoa['beta'][t_idx]

            for i in range(len(source_positions)):
                xi = source_positions[i, 0] + t * source_velocities[i, 0]
                yi = source_positions[i, 1] + t * source_velocities[i, 1]
                zi = source_positions[i, 2] + t * source_velocities[i, 2]
                di = np.sqrt((x - xi)**2 + (y - yi)**2 + (z - zi)**2)
                if i < len(toai) and not np.isnan(toai[i]):
                    res.append(toai[i] * C - di)
                if i < len(rssi_i) and not np.isnan(rssi_i[i]):
                    res.append(1000 * np.exp((s - rssi_i[i]) / (10 * k)) - di)
                if i == 3:
                    res.append(y - aoa_alpha * (x - xi))
                    res.append(np.sqrt(di**2 - (zi - z)**2) - aoa_beta * (zi - z))

            for j in range(i + 1, len(source_positions)):
                if j < len(tdoaij) and not np.isnan(tdoaij[j - 1]):
                    xj = source_positions[j, 0] + t * source_velocities[j, 0]
                    yj = source_positions[j, 1] + t * source_velocities[j, 1]
                    zj = source_positions[j, 2] + t * source_velocities[j, 2]
                    dj = np.sqrt((x - xj)**2 + (y - yj)**2 + (z - zj)**2)
                    res.append(tdoaij[j - 1] * C - (di - dj))
                if j < len(dfdij) and not np.isnan(dfdij[j - 1]):
                    vi_ri = np.dot(source_velocities[i], [x - xi, y - yi, z - zi]) / di
                    vj_rj = np.dot(source_velocities[j], [x - xj, y - yj, z - zj]) / dj
                    res.append(dfdij[j - 1] - (f0 / C) * (vi_ri - vj_rj))

        return res

# Generate time intervals from 0 to 10 seconds with a step of 0.01 seconds
time_intervals = np.arange(0, 10.01, 0.01)

# Store the results
positions = []

# Perform the optimization for each time step
for t in time_intervals:
    res_lsq = least_squares(residuals, initial_guess, args=(toa, tdoa, dfd, rssi, aoa, [t]))
    positions.append(res_lsq.x)
    initial_guess = res_lsq.x    # Update initial guess for next iteration

# Convert to numpy array
positions = np.array(positions)

# Print the positions at each time step
for idx, (t, pos) in enumerate(zip(time_intervals, positions)):
    print(f"Time {t:.2f} s: Position (x, y, z) = {pos}")

# Visualization
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(positions[:, 0], positions[:, 1], positions[:, 2], label='Aircraft Trajectory')
ax.set_xlabel('X')
ax.set_ylabel('Y')
```

```
ax.set_zlabel('Z')
ax.legend()
plt.show()
```

**附录 2**

介绍：AOA 代码

```
import numpy as np
import pandas as pd
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Constants
C = 3e8    # Speed of light in m/s
f0 = 3e8    # Frequency in Hz
k = 20    # Channel attenuation coefficient
s = 100    # Nominal signal strength
nominal_distance = 1000    # Corresponding to nominal signal strength

# Initial positions and velocity vectors of the sources
source_positions = np.array([[100, 300, 50],
                             [1000, 100, 150],
                             [-1000, -100, 450],
                             [-200, -100, 1050]])
source_velocities = np.array([[0, 0, 0],
                              [0, 0, 0],
                              [0, 0, 8],
                              [20, 0, 0]])

# Load the data from Excel
file_path = 'D:/python/LDcode/database/飞行器接受情况 1.xlsx'
data = pd.read_excel(file_path)

# Extracting columns for different signals
times = data['时间 t（s）'].values
toa = data[['TOA(发射源 1)', 'TOA(发射源 4)']].values
tdoa = data[['TDOA(发射源 1,发射源 2)', 'TDOA(发射源 3,发射源 4)']].values
dfd = data[['DFD(发射源 1,发射源 2)']].values
aoa = {'alpha': data['AOA[tan(alpha)](发射源 4)'].values,
       'beta': data['AOA[tan(beta)](发射源 4)'].values}
rssi = data[['RSSI(发射源 1)', 'RSSI(发射源 2)', 'RSSI(发射源 3)']].values

# Initialize aircraft position (starting guess)
initial_guess = np.array([0, 0, 0])

# Function to calculate the residuals for Newton's method
def residuals(position, toa, tdoa, dfd, rssi, aoa, times):
    x, y, z = position
```

```
        res = []

        for t_idx, t in enumerate(times):
            toai = toa[t_idx]
            tdoaij = tdoa[t_idx]
            dfdij = dfd[t_idx]
            rssi_i = rssi[t_idx]
            aoa_alpha = aoa['alpha'][t_idx]
            aoa_beta = aoa['beta'][t_idx]

            for i in range(len(source_positions)):
                xi = source_positions[i, 0] + t * source_velocities[i, 0]
                yi = source_positions[i, 1] + t * source_velocities[i, 1]
                zi = source_positions[i, 2] + t * source_velocities[i, 2]
                di = np.sqrt((x - xi)**2 + (y - yi)**2 + (z - zi)**2)
                if i < len(toai) and not np.isnan(toai[i]):
                    res.append(toai[i] * C - di)
                if i < len(rssi_i) and not np.isnan(rssi_i[i]):
                    res.append(1000 * np.exp((s - rssi_i[i]) / (10 * k)) - di)
                if i == 3:
                    res.append(y - aoa_alpha * (x - xi))
                    res.append(np.sqrt(di**2 - (zi - z)**2) - aoa_beta * (zi - z))

            for j in range(i + 1, len(source_positions)):
                if j < len(tdoaij) and not np.isnan(tdoaij[j - 1]):
                    xj = source_positions[j, 0] + t * source_velocities[j, 0]
                    yj = source_positions[j, 1] + t * source_velocities[j, 1]
                    zj = source_positions[j, 2] + t * source_velocities[j, 2]
                    dj = np.sqrt((x - xj)**2 + (y - yj)**2 + (z - zj)**2)
                    res.append(tdoaij[j - 1] * C - (di - dj))
                if j < len(dfdij) and not np.isnan(dfdij[j - 1]):
                    vi_ri = np.dot(source_velocities[i], [x - xi, y - yi, z - zi]) / di
                    vj_rj = np.dot(source_velocities[j], [x - xj, y - yj, z - zj]) / dj
                    res.append(dfdij[j - 1] - (f0 / C) * (vi_ri - vj_rj))

        return res

    # Function to visualize AOA data accuracy
    def visualize_aoa_accuracy(positions, aoa, times):
        estimated_y = []
        estimated_z = []
        actual_y = []
        actual_z = []

        for idx, (pos, aoa_alpha, aoa_beta, t) in enumerate(zip(positions, aoa['alpha'],
aoa['beta'], times)):
            x, y, z = pos
            yi = source_positions[3, 1] + t * source_velocities[3, 1]
            zi = source_positions[3, 2] + t * source_velocities[3, 2]
            estimated_y.append(y)
```

```
                estimated_z.append(z)
                actual_y.append(yi - aoa_alpha * (x - source_positions[3, 0] - t *
source_velocities[3, 0]))
                actual_z.append(zi - np.sqrt((y - yi)**2 + (z - zi)**2) / aoa_beta)

            plt.figure()
            plt.subplot(2, 1, 1)
            plt.plot(times, estimated_y, label='Estimated Y')
            plt.plot(times, actual_y, label='Actual Y', linestyle='--')
            plt.xlabel('Time (s)')
            plt.ylabel('Y coordinate')
            plt.legend()
            plt.title('Y Coordinate Accuracy')

            plt.subplot(2, 1, 2)
            plt.plot(times, estimated_z, label='Estimated Z')
            plt.plot(times, actual_z, label='Actual Z', linestyle='--')
            plt.xlabel('Time (s)')
            plt.ylabel('Z coordinate')
            plt.legend()
            plt.title('Z Coordinate Accuracy')

            plt.tight_layout()
            plt.show()

    # Generate time intervals from 0 to 10 seconds with a step of 0.01 seconds
    time_intervals = np.arange(0, 10.01, 0.01)

    # Store the results
    positions = []

    # Perform the optimization for each time step
    for t in time_intervals:
        res_lsq = least_squares(residuals, initial_guess, args=(toa, tdoa, dfd, rssi, aoa,
[t]))
        positions.append(res_lsq.x)
        initial_guess = res_lsq.x    # Update initial guess for next iteration

    # Convert to numpy array
    positions = np.array(positions)

    # Print the positions at each time step
    for idx, (t, pos) in enumerate(zip(time_intervals, positions)):
        print(f"Time {t:.2f} s: Position (x, y, z) = {pos}")

    # Visualization of the trajectory
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(positions[:, 0], positions[:, 1], positions[:, 2], label='Aircraft Trajectory')
    ax.set_xlabel('X')
```

```
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
plt.title('Aircraft Trajectory')
plt.show()

# Visualize AOA accuracy
visualize_aoa_accuracy(positions, aoa, time_intervals)
```

**附录 3**

介绍：问题三代码

```
import numpy as np
import pandas as pd
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pykalman import KalmanFilter

# Constants
C = 3e8    # Speed of light in m/s
f0 = 3e8    # Frequency in Hz
k = 20    # Channel attenuation coefficient
s = 100    # Nominal signal strength
nominal_distance = 1000    # Corresponding to nominal signal strength

# Initial positions and velocity vectors of the sources
source_positions = np.array([[100, 300, 50],
                             [1000, 100, 150],
                             [-1000, -100, 450],
                             [-200, -100, 1050]])
source_velocities = np.array([[0, 0, 0],
                              [0, 0, 0],
                              [0, 0, 8],
                              [20, 0, 0]])

# Load the data from Excel
file_path = 'D:/python/LDcode/database/飞行器接受情况 2.xlsx'
data = pd.read_excel(file_path)

# Extracting columns for different signals
times = data['时间 t（s）'].values
toa = data[['TOA(发射源 2)', 'TOA(发射源 3)']].values
tdoa = data[['TDOA(发射源 3,发射源 4)']].values
dfd = data[['DFD(发射源 3,发射源 4)']].values
aoa = {'alpha': data['AOA[tan(alpha)](发射源 4)'].values,
       'beta': data['AOA[tan(beta)](发射源 4)'].values}
rssi = data[['RSSI(发射源 1)', 'RSSI(发射源 3)', 'RSSI(发射源 4)']].values
```

```
# Initialize aircraft position (starting guess)
initial_guess = np.array([0, 0, 0])

# Function to calculate the residuals for Newton's method
def residuals(position, toa, tdoa, dfd, rssi, aoa, times):
    x, y, z = position
    res = []

    for t_idx, t in enumerate(times):
        toai = toa[t_idx]
        tdoaij = tdoa[t_idx]
        dfdij = dfd[t_idx]
        rssi_i = rssi[t_idx]
        aoa_alpha = aoa['alpha'][t_idx]
        aoa_beta = aoa['beta'][t_idx]

        for i in range(len(source_positions)):
            xi = source_positions[i, 0] + t * source_velocities[i, 0]
            yi = source_positions[i, 1] + t * source_velocities[i, 1]
            zi = source_positions[i, 2] + t * source_velocities[i, 2]
            di = np.sqrt((x - xi)**2 + (y - yi)**2 + (z - zi)**2)
            if i < len(toai) and not np.isnan(toai[i]):
                res.append(toai[i] * C - di)
            if i < len(rssi_i) and not np.isnan(rssi_i[i]):
                res.append(1000 * np.exp((s - rssi_i[i]) / (10 * k)) - di)
            if i == 3:
                res.append(y - aoa_alpha * (x - xi))
                res.append(np.sqrt(di**2 - (zi - z)**2) - aoa_beta * (zi - z))

        for j in range(i + 1, len(source_positions)):
            if j < len(tdoaij) and not np.isnan(tdoaij[j - 1]):
                xj = source_positions[j, 0] + t * source_velocities[j, 0]
                yj = source_positions[j, 1] + t * source_velocities[j, 1]
                zj = source_positions[j, 2] + t * source_velocities[j, 2]
                dj = np.sqrt((x - xj)**2 + (y - yj)**2 + (z - zj)**2)
                res.append(tdoaij[j - 1] * C - (di - dj))
            if j < len(dfdij) and not np.isnan(dfdij[j - 1]):
                vi_ri = np.dot(source_velocities[i], [x - xi, y - yi, z - zi]) / di
                vj_rj = np.dot(source_velocities[j], [x - xj, y - yj, z - zj]) / dj
                res.append(dfdij[j - 1] - (f0 / C) * (vi_ri - vj_rj))

    return res

# Function to visualize AOA data accuracy
def visualize_aoa_accuracy(positions, aoa, times):
    estimated_y = []
    estimated_z = []
    actual_y = []
    actual_z = []
```

```
            for idx, (pos, aoa_alpha, aoa_beta, t) in enumerate(zip(positions, aoa['alpha'], aoa['beta'], times)):
                x, y, z = pos
                yi = source_positions[3, 1] + t * source_velocities[3, 1]
                zi = source_positions[3, 2] + t * source_velocities[3, 2]
                estimated_y.append(y)
                estimated_z.append(z)
                actual_y.append(yi - aoa_alpha * (x - source_positions[3, 0] - t * source_velocities[3, 0]))
                actual_z.append(zi - np.sqrt((y - yi)**2 + (z - zi)**2) / aoa_beta)

            plt.figure()
            plt.subplot(2, 1, 1)
            plt.plot(times, estimated_y, label='Estimated Y')
            plt.plot(times, actual_y, label='Actual Y', linestyle='--')
            plt.xlabel('Time (s)')
            plt.ylabel('Y coordinate')
            plt.legend()
            plt.title('Y Coordinate Accuracy')

            plt.subplot(2, 1, 2)
            plt.plot(times, estimated_z, label='Estimated Z')
            plt.plot(times, actual_z, label='Actual Z', linestyle='--')
            plt.xlabel('Time (s)')
            plt.ylabel('Z coordinate')
            plt.legend()
            plt.title('Z Coordinate Accuracy')

            plt.tight_layout()
            plt.show()

    # Generate time intervals from 0 to 10 seconds with a step of 0.01 seconds
    time_intervals = np.arange(0, 10.01, 0.01)

    # Store the results
    positions = []

    # Perform the optimization for each time step
    for t in time_intervals:
        res_lsq = least_squares(residuals, initial_guess, args=(toa, tdoa, dfd, rssi, aoa, [t]))
        positions.append(res_lsq.x)
        initial_guess = res_lsq.x  # Update initial guess for next iteration

    # Convert to numpy array
    positions = np.array(positions)

    # Kalman Filter setup
    kf = KalmanFilter(initial_state_mean=positions[0],
```

```
                                n_dim_obs=3,
                                observation_covariance=np.eye(3),
                                transition_covariance=np.eye(3) * 0.01)

    # Apply Kalman Filter
    filtered_state_means, filtered_state_covariances = kf.filter(positions)

    # Calculate RMSE as the indicator
    rmse = np.sqrt(np.mean((positions - filtered_state_means) ** 2, axis=1))

    # Print the positions at each time step
    for idx, (t, pos) in enumerate(zip(time_intervals, filtered_state_means)):
        print(f"Time {t:.2f} s: Filtered Position (x, y, z) = {pos}, RMSE =
{rmse[idx]:.4f}")

    # Visualization of the trajectory
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(positions[:, 0], positions[:, 1], positions[:, 2], label='Original Trajectory')
    ax.plot(filtered_state_means[:, 0], filtered_state_means[:, 1], filtered_state_means[:,
2], label='Filtered Trajectory')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.legend()
    plt.title('Aircraft Trajectory')
    plt.show()

    # Visualize AOA accuracy
    visualize_aoa_accuracy(filtered_state_means, aoa, time_intervals)

    # Plot RMSE over time
    plt.figure()
    plt.plot(time_intervals, rmse, label='RMSE')
    plt.xlabel('Time (s)')
    plt.ylabel('RMSE')
    plt.title('RMSE Over Time')
    plt.legend()
    plt.show()

    # Highlighting significant deviations
    threshold = np.mean(rmse) + 2 * np.std(rmse)
    significant_deviations = np.where(rmse > threshold)[0]

    # Plot the RMSE with significant deviations highlighted
    plt.figure()
    plt.plot(time_intervals, rmse, label='RMSE')
    plt.scatter(time_intervals[significant_deviations], rmse[significant_deviations],
color='red', label='Significant Deviations')
    plt.xlabel('Time (s)')
```

```
plt.ylabel('RMSE')
plt.title('RMSE Over Time with Significant Deviations')
plt.legend()
plt.show()

# Output indices of significant deviations
print(f"Significant deviations at indices: {significant_deviations}")
```

**附录 4**

介绍：问题四代码

```
import numpy as np
import pandas as pd
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pykalman import KalmanFilter

# Constants
C = 3e8    # Speed of light in m/s
f0 = 3e8    # Frequency in Hz
k = 20    # Channel attenuation coefficient
s = 100    # Nominal signal strength
nominal_distance = 1000    # Corresponding to nominal signal strength

# Initial positions and velocity vectors of the sources
source_positions = np.array([[100, 300, 50],
                             [1000, 100, 150],
                             [-1000, -100, 450],
                             [-200, -100, 1050]])
source_velocities = np.array([[0, 0, 0],
                              [0, 0, 0],
                              [0, 0, 8],
                              [20, 0, 0]])

# Load the data from Excel
file_path = 'D:/python/LDcode/database/飞行器接受情况 2.xlsx'
data = pd.read_excel(file_path)

# Extracting columns for different signals
times = data['时间 t（s）'].values
toa = data[['TOA(发射源 2)', 'TOA(发射源 3)']].values
tdoa = data[['TDOA(发射源 3,发射源 4)']].values
dfd = data[['DFD(发射源 3,发射源 4)']].values
aoa = {'alpha': data['AOA[tan(alpha)](发射源 4)'].values,
       'beta': data['AOA[tan(beta)](发射源 4)'].values}
rssi = data[['RSSI(发射源 1)', 'RSSI(发射源 3)', 'RSSI(发射源 4)']].values
```

```
# Initialize aircraft position (starting guess)
initial_guess = np.array([0, 0, 0])

# Function to calculate the residuals for Newton's method
def residuals(position, toa, tdoa, dfd, rssi, aoa, times):
    x, y, z = position
    res = []

    for t_idx, t in enumerate(times):
        toai = toa[t_idx]
        tdoaij = tdoa[t_idx]
        dfdij = dfd[t_idx]
        rssi_i = rssi[t_idx]
        aoa_alpha = aoa['alpha'][t_idx]
        aoa_beta = aoa['beta'][t_idx]

        for i in range(len(source_positions)):
            xi = source_positions[i, 0] + t * source_velocities[i, 0]
            yi = source_positions[i, 1] + t * source_velocities[i, 1]
            zi = source_positions[i, 2] + t * source_velocities[i, 2]
            di = np.sqrt((x - xi)**2 + (y - yi)**2 + (z - zi)**2)
            if i < len(toai) and not np.isnan(toai[i]):
                res.append(toai[i] * C - di)
            if i < len(rssi_i) and not np.isnan(rssi_i[i]):
                res.append(1000 * np.exp((s - rssi_i[i]) / (10 * k)) - di)
            if i == 3:
                res.append(y - aoa_alpha * (x - xi))
                res.append(np.sqrt(di**2 - (zi - z)**2) - aoa_beta * (zi - z))

        for j in range(i + 1, len(source_positions)):
            if j < len(tdoaij) and not np.isnan(tdoaij[j - 1]):
                xj = source_positions[j, 0] + t * source_velocities[j, 0]
                yj = source_positions[j, 1] + t * source_velocities[j, 1]
                zj = source_positions[j, 2] + t * source_velocities[j, 2]
                dj = np.sqrt((x - xj)**2 + (y - yj)**2 + (z - zj)**2)
                res.append(tdoaij[j - 1] * C - (di - dj))
            if j < len(dfdij) and not np.isnan(dfdij[j - 1]):
                vi_ri = np.dot(source_velocities[i], [x - xi, y - yi, z - zi]) / di
                vj_rj = np.dot(source_velocities[j], [x - xj, y - yj, z - zj]) / dj
                res.append(dfdij[j - 1] - (f0 / C) * (vi_ri - vj_rj))

    return res

# Function to visualize AOA data accuracy
def visualize_aoa_accuracy(positions, aoa, times):
    estimated_y = []
    estimated_z = []
    actual_y = []
    actual_z = []
```

```
            for idx, (pos, aoa_alpha, aoa_beta, t) in enumerate(zip(positions, aoa['alpha'], aoa['beta'], times)):
                x, y, z = pos
                yi = source_positions[3, 1] + t * source_velocities[3, 1]
                zi = source_positions[3, 2] + t * source_velocities[3, 2]
                estimated_y.append(y)
                estimated_z.append(z)
                actual_y.append(yi - aoa_alpha * (x - source_positions[3, 0] - t * source_velocities[3, 0]))
                actual_z.append(zi - np.sqrt((y - yi)**2 + (z - zi)**2) / aoa_beta)

            plt.figure()
            plt.subplot(2, 1, 1)
            plt.plot(times, estimated_y, label='Estimated Y')
            plt.plot(times, actual_y, label='Actual Y', linestyle='--')
            plt.xlabel('Time (s)')
            plt.ylabel('Y coordinate')
            plt.legend()
            plt.title('Y Coordinate Accuracy')

            plt.subplot(2, 1, 2)
            plt.plot(times, estimated_z, label='Estimated Z')
            plt.plot(times, actual_z, label='Actual Z', linestyle='--')
            plt.xlabel('Time (s)')
            plt.ylabel('Z coordinate')
            plt.legend()
            plt.title('Z Coordinate Accuracy')

            plt.tight_layout()
            plt.show()

    # Generate time intervals from 0 to 10 seconds with a step of 0.01 seconds
    time_intervals = np.arange(0, 10.01, 0.01)

    # Store the results
    positions = []

    # Perform the optimization for each time step
    for t in time_intervals:
        res_lsq = least_squares(residuals, initial_guess, args=(toa, tdoa, dfd, rssi, aoa, [t]))
        positions.append(res_lsq.x)
        initial_guess = res_lsq.x  # Update initial guess for next iteration

    # Convert to numpy array
    positions = np.array(positions)

    # Kalman Filter setup
    kf = KalmanFilter(initial_state_mean=positions[0],
```

```
                              n_dim_obs=3,
                              observation_covariance=np.eye(3),
                              transition_covariance=np.eye(3) * 0.01)

    # Apply Kalman Filter
    filtered_state_means, filtered_state_covariances = kf.filter(positions)

    # Calculate RMSE as the indicator
    rmse = np.sqrt(np.mean((positions - filtered_state_means) ** 2, axis=1))

    # Print the positions at each time step
    for idx, (t, pos) in enumerate(zip(time_intervals, filtered_state_means)):
        print(f"Time {t:.2f} s: Filtered Position (x, y, z) = {pos}, RMSE =
{rmse[idx]:.4f}")

    # Visualization of the trajectory
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(positions[:, 0], positions[:, 1], positions[:, 2], label='Original Trajectory')
    ax.plot(filtered_state_means[:, 0], filtered_state_means[:, 1], filtered_state_means[:,
2], label='Filtered Trajectory')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.legend()
    plt.title('Aircraft Trajectory')
    plt.show()

    # Visualize AOA accuracy
    visualize_aoa_accuracy(filtered_state_means, aoa, time_intervals)

    # Plot RMSE over time
    plt.figure()
    plt.plot(time_intervals, rmse, label='RMSE')
    plt.xlabel('Time (s)')
    plt.ylabel('RMSE')
    plt.title('RMSE Over Time')
    plt.legend()
    plt.show()

    # Highlighting significant deviations
    threshold = np.mean(rmse) + 2 * np.std(rmse)
    significant_deviations = np.where(rmse > threshold)[0]

    # Plot the RMSE with significant deviations highlighted
    plt.figure()
    plt.plot(time_intervals, rmse, label='RMSE')
    plt.scatter(time_intervals[significant_deviations], rmse[significant_deviations],
color='red', label='Significant Deviations')
    plt.xlabel('Time (s)')
```

```
    plt.ylabel('RMSE')
    plt.title('RMSE Over Time with Significant Deviations')
    plt.legend()
    plt.show()

    # Output indices of significant deviations
    print(f"Significant deviations at indices: {significant_deviations}")

    # Identify Constant Drift and Random Bias
    window_size = 50    # Window size for moving analysis
    constant_drift_indices = []
    random_bias_indices = []

    for i in range(len(rmse) - window_size + 1):
        window = rmse[i:i + window_size]
        mean_rmse = np.mean(window)
        std_rmse = np.std(window)

        # Check for constant drift
        if abs(np.mean(np.diff(window))) > 0.01:    # Change threshold as needed
            constant_drift_indices.append(i + window_size // 2)
        # Check for random bias
        elif std_rmse > 0.5:    # Change threshold as needed
            random_bias_indices.append(i + window_size // 2)

    # Plot RMSE with bias annotations
    plt.figure()
    plt.plot(time_intervals, rmse, label='RMSE')
    plt.scatter(time_intervals[constant_drift_indices], rmse[constant_drift_indices],
color='blue', label='Constant Drift')
    plt.scatter(time_intervals[random_bias_indices], rmse[random_bias_indices],
color='green', label='Random Bias')
    plt.xlabel('Time (s)')
    plt.ylabel('RMSE')
    plt.title('RMSE Over Time with Bias Detection')
    plt.legend()
    plt.show()

    # Output bias indices
    print(f"Constant drift indices: {constant_drift_indices}")
    print(f"Random bias indices: {random_bias_indices}")

    # Visualize deviation for each signal type
    def visualize_signal_deviation(signal_name, signal_values, rmse):
        plt.figure()
        plt.plot(time_intervals, signal_values, label=f'{signal_name} Values')
        plt.xlabel('Time (s)')
        plt.ylabel(f'{signal_name} Values')
        plt.title(f'{signal_name} Deviation Over Time')
        plt.legend()
```

```
        plt.show()

    # Visualize deviation for TOA
    for idx in range(toa.shape[1]):
        visualize_signal_deviation(f'TOA(发射源{idx + 2})', toa[:, idx], rmse)

    # Visualize deviation for TDOA
    for idx in range(tdoa.shape[1]):
        visualize_signal_deviation(f'TDOA(发射源{idx + 3},发射源{idx + 4})',
tdoa[:, idx], rmse)

    # Visualize deviation for DFD
    for idx in range(dfd.shape[1]):
        visualize_signal_deviation(f'DFD(发射源{idx + 3},发射源{idx + 4})', dfd[:,
idx], rmse)

    # Visualize deviation for RSSI
    for idx in range(rssi.shape[1]):
        visualize_signal_deviation(f'RSSI(发射源{idx + 1})', rssi[:, idx], rmse)

    # Visualize deviation for AOA alpha
    visualize_signal_deviation('AOA[tan(alpha)](发射源 4)', aoa['alpha'], rmse)

    # Visualize deviation for AOA beta
    visualize_signal_deviation('AOA[tan(beta)](发射源 4)', aoa['beta'], rmse)
```