目　录

1. 题目要求

已知两个基于同一个地球椭球模型建立的大地坐标系，分别在这两个坐标系中对若干个台站进行观测，每个台站的观测数据包括经度（λ）和纬度（φ）。根据这些数据，使用任意编程语言编写代码，将这两个大地坐标系转换为空间直角坐标系，然后求解两个空间直角坐标系之间转换所需要的7个转换参数值（即七参数，包括三个平移参数dx、dy、dz，三个旋转参数rx、ry、rz，和一个尺度变化参数m），并根据求解出的七参数计算其他台站在新的空间直角坐标系中的坐标值，最后给出精度指标进行精度评定。

2. 数据

数据来源：广东工业大学测绘22（1）班《大地测量学基础》课程编程实践作业。

数据内容：数据包括 50 个台站的两组经纬度坐标（两个基于同一个地球椭球模型的大地坐标系），其中 9 号台站、19 号台站和 29 号台站无数据，31—50 号台站只有其中一个坐标系的经纬度坐标数据。默认所有的点都在参考椭球面上，即H = 0。

数据使用：根据数据的特点，可以采用 1—30 号台站（除 9 号、19 号和 29 号外）共 27 个台站作为公共点，用以求解七参数和检验转换精度。将 31—50 号台站作为待转换的非公共点，转换后计算改正数加以改正，得到较为精确的坐标。

表 1　50 个台站的两组经纬度观测值

序号	台站名称	L1	B1	L2	B2
1	AIRP	85.3579	27.6971	85.28	27.693
2	BADG	102.235	51.7697	102.235	51.77
3	BAGH	92.1919	23.1617	92.192	23.162
4	BARK	92.3753	22.7256	92.375	22.726
5	BELT	83.8257	27.4574	83.826	27.457
6	BESI	84.3797	28.2286	84.38	28.229
7	BJNM	116.2241	40.2453	116.224	40.245
8	BRN2	87.2722	26.5197	87.272	26.52
9	—	—	—	—	—
10	CHNR	91.515	24.1897	91.515	24.19
11	DAMA	85.1077	27.6081	85.108	27.608
12	DIGH	92.067	23.2501	92.066	23.241
13	DMAU	84.2652	27.9734	84.265	27.973
14	GARM	70.3167	39.0065	70.317	39.006

15	GHER	84.4097	28.3746	84.41	28.375
16	IRKT	104.3162	52.219	104.316	52.219
17	JAFL	92.0191	25.1794	92.019	25.179
18	JAML	91.2371	25.0004	91.237	25
19	—	—	—	—	—
20	JURI	92.1353	24.4984	92.135	24.498
21	KAWA	84.1299	27.648	84.13	27.648
22	KIRT	85.2882	27.6819	85.288	27.682
23	KIT3	66.8854	39.1348	66.885	39.135
24	KMTR	78.1992	41.8678	78.199	41.868
25	KPTI	92.2171	22.4986	92.217	22.499
26	KRTV	78.6191	50.714	78.619	50.714
27	KSTU	92.7938	55.9933	92.794	55.993
28	KUMT	78.1904	41.8635	78.19	41.864
29	—	—	—	—	—
30	MANM	71.6804	37.5423	71.68	37.542
31	MKCH	91.8447	22.8512	—	—
32	MPUR	90.0301	24.6042	—	—
33	NEPJ	81.6154	28.0493	—	—
34	NR01	100.6622	30.1432	—	—
35	NR03	100.1014	29.9061	—	—
36	NR09	103.1693	31.4476	—	—
37	NVSK	83.2354	54.8406	—	—
38	OSHK	72.7775	40.5299	—	—
39	PODG	79.4849	43.3275	—	—
40	QIME	101.185	31.0229	—	—
41	RBIT	89.3923	26.8493	—	—
42	RMJT	86.55	27.3051	—	—
43	RMTE	86.5971	26.9909	—	—
44	RPUR	90.7632	23.0365	—	—
45	SHTZ	68.1228	37.5621	—	—
46	SITA	91.6625	22.6148	—	—
47	TASH	69.2956	41.328	—	—
48	ULAB	107.0523	47.8651	—	—
49	WARI	101.1254	30.8812	—	—
50	XIAN	109.2215	34.3687	—	—

3. 程序设计思路

3.1. 数据输入与预处理

首先读取包含大地坐标系经纬度坐标的 TXT 文件，然后进行数据处理和矩阵格式转换。输入数据为三个 TXT 文件，其中一个为公共点在原大地坐标系的坐标，一个为公共点在新大地坐标系的坐标，一个为非公共点在原大地坐标系的坐标。

1. 编写函数以读取数据文件；
2. 提取坐标数据并存储为矩阵，便于后续计算；
3. 针对空值、异常数据等情况，进行基础的错误检查和处理。

3.2. 大地坐标系转空间直角坐标系

利用地理坐标（经度、纬度、高程）将大地坐标转换为空间直角坐标。转换公式为

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (N+H)\cos B\cos L \\ (N+H)\cos B\sin L \\ [N(1-e^2)+H]\sin B \end{bmatrix} \tag{3-1}$$

L、B、H：分别为经度、纬度和高程，默认所有的点都在参考椭球面上，即$H = 0$。

X、Y、Z：空间直角坐标系中的坐标，表示点在空间中的位置。

a：地球椭球的长半轴。采用 WGS84 椭球，$a = 6378137(m)$。

b：地球的短半轴。

f：为地球的扁率，定义为

$$f = \frac{a-b}{a} \tag{3-2}$$

采用 WGS84 椭球，扁率f = 1/298.257223563

N：卯酉圈曲率半径，定义为

$$N = \frac{a}{\sqrt{1-(2f-f^2)\sin^2 B}} = \frac{a}{\sqrt{1-e^2\sin^2 B}} \tag{3-3}$$

e：地球的第一偏心率，用来描述地球椭球的扁平程度，定义为：

$$e = \sqrt{2f-f^2} = \sqrt{\frac{a^2-b^2}{a^2}} \tag{3-4}$$

3.3. 求解七参数

不同空间直角坐标系的坐标换算存在着三个平移参数dx、dy、dz，三个旋转参数rx、ry、rz，和一个尺度变化参数m。相应的坐标变换公式为：

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = (1+m)\begin{bmatrix} 1 & rz & -ry \\ -rz & 1 & rx \\ ry & -rx & 1 \end{bmatrix}\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \tag{3-5}$$

从这个模型可以看出，要求得这 7 个转换参数，至少需要 3 个公共点，当多于 3 个公共点时，可以按照最小二乘法求得 7 个参数的最或然值。

取

$$a_1 = m+1,\ a_2 = a_1 \times rx,\ a_3 = a_1 \times ry,\ a_4 = a_1 \times rz \tag{3-6}$$

则可将(3-5)式写为

$$\begin{bmatrix} \mathrm{X}_2 \\ \mathrm{Y}_2 \\ \mathrm{Z}_2 \end{bmatrix}_{转换值} = \begin{bmatrix} 1 & 0 & 0 & \mathrm{X}_1 & 0 & -Z_1 & \mathrm{Y}_1 \\ 0 & 1 & 0 & \mathrm{Y}_1 & Z_1 & 0 & -X_1 \\ 0 & 0 & 1 & Z_1 & -\mathrm{Y}_1 & X_1 & 0 \end{bmatrix}\begin{bmatrix} dx \\ dy \\ dz \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \tag{3-7}$$

取

$$\begin{bmatrix} V_{X_2} \\ V_{Y_2} \\ V_{Z_2} \end{bmatrix} = \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix}_{已知值} - \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix}_{转换值} \tag{3-8}$$

则写出如下误差方程：

$$\begin{bmatrix} V_{X_2} \\ V_{Y_2} \\ V_{Z_2} \end{bmatrix} = -\begin{bmatrix} 1 & 0 & 0 & \mathrm{X}_1 & 0 & -Z_1 & \mathrm{Y}_1 \\ 0 & 1 & 0 & \mathrm{Y}_1 & Z_1 & 0 & -X_1 \\ 0 & 0 & 1 & Z_1 & -\mathrm{Y}_1 & X_1 & 0 \end{bmatrix}\begin{bmatrix} dx \\ dy \\ dz \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix}_{已知值} \tag{3-9}$$

改写成矩阵形式为：

$$V = B \cdot \Delta X + L \tag{3-10}$$

其中，B 是系数矩阵，ΔX 是包含七参数的待估计向量，L 是观测向量。步骤如下：

1. 构建误差方程，构建系数矩阵 B；

2. 利用最小二乘法$V^T PV = min$的原则，可以解得：

$$\Delta X = -(B^T PB)^{-1} B^T PL \tag{3-11}$$

再由式(3-6)可进一步计算出最终的七参数。

3.4. 使用七参数转换坐标

使用计算得出的七参数代入转换公式(3-5)对大地坐标进行转换。

3.5. 使用配置法计算改正数并平差

由于公共点的坐标存在误差，求得的转换参数将受其影响，公共点坐标误差对转换参数的影响与点位的几何分布及点数的多少有关，因而为了求得较好的转换参数，应选择一定数量的精度较高且分布较均匀并有较大覆盖面的公共点。

(3-5)式为相似变换模型，当利用 3 个以上的公共点求解转换参数时存在多余观测，由于公共点误差的影响而使得转换的公共点的坐标值与已知值不完全相同，而实际工作中又往往要求所有的已知点的坐标值保持固定不变。为了解决这一矛盾，可采用配置法，将公共点的转换值改正为已知值，对非公共点的转换值进行相应的配置。具体方法是：

①计算公共点转换值的改正数$V =$ 已知值 $-$ 转换值，公共点的坐标采用已知值。

②采用配置法计算非公共点转换值的改正数

$$V' = \frac{\sum_{1}^{n} P_i V_i}{\sum_{1}^{n} P_i} \tag{3-12}$$

式中：n为公共点的个数，P为权，可根据非公共点与公共点的距离（S_i）来定权，常取

$$P = \frac{1}{S_i^2} \tag{3-13}$$

③使用改正数进行改正

3.6. 功能扩展

可添加许多新功能完善程序，例如：

1. 界面设计：设计简易界面输入坐标数据和显示转换结果；
2. 异常处理：识别并处理异常值；
3. 数据导出：转换结果的导出；

4. 程序实现过程

使用 Python 按照上述设计思路编写代码实现相关功能，由于代码中添加了 GUI 界面设计依据异常处理等繁杂内容，以下仅展示代码的核心算法部分，完整代码见附录。

4.1. 数据输入与预处理

编写 **read_file** 函数以读取数据文件并储存为矩阵，针对空值、异常数据等情况，进行基础的错误检查和处理。

```
def read_file(file_path):
    """
    从 TXT 文件读取经纬度和高程信息。
    文件格式：序号，台站名称，L1(经度)，B1(纬度)，H(高程，可选)
    :param file_path: 文件路径
    :return: 站点信息列表
    """
    stations = []  # 用于存储站点信息的列表
    try:
        # 以只读模式打开文件，指定编码为 utf-8
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                if not line.strip():  # 跳过空行
                    continue
                parts = line.strip().split(',')  # 将每行以逗号分隔为多个部分
                if len(parts) < 4:  # 如果数据列少于 4 列，跳过该行
                    continue
                num = parts[0]  # 序号
                name = parts[1]  # 台站名称
                lon = float(parts[2])  # 经度
                lat = float(parts[3])  # 纬度
                h = float(parts[4]) if len(parts) > 4 and parts[4].strip() else
```

```
0.0  # 读取高程，若无高程则默认值为 0.0
                # 将提取的信息作为字典存入 stations 列表
                stations.append({"num": num, "name": name, "lat": lat, "lon": lon,
"h": h})
    except Exception as e:
        # 异常处理，显示错误信息
        messagebox.showerror("文件读取错误", f"无法读取文件: {e}")
    return stations  # 返回站点信息列表
```

这段代码定义了一个 **read_file** 函数，从指定的 TXT 文件中读取站点的经纬度和高程信息。文件每行包含序号、台站名称、经度、纬度和可选的高程。函数逐行解析数据，若高程缺失则默认为 0，并将解析后的站点信息存入字典列表。若读取过程中出现异常，函数会弹出错误信息提示框。

4.2. 大地坐标系转空间直角坐标系

利用地理坐标（经度、纬度、高程）将大地坐标转换为空间直角坐标。

```
def lat_lon_to_ecef(lat, lon, h=0.0, a=6378137.0, f=1 / 298.257223563):
    """
    将纬度和经度转换为 ECEF 坐标，允许传入自定义的椭球参数。
    :param lat: 纬度
    :param lon: 经度
    :param h: 高程，默认为 0
    :param a: 长半径（地球椭球体的长半轴）
    :param f: 扁率（地球椭球体的扁率）
    :return: ECEF 坐标 (x, y, z)
    """
    # 将纬度和经度从度数转换为弧度，以便进行三角计算
    lat_rad = math.radians(lat)
    lon_rad = math.radians(lon)

    # 计算经纬度所在位置的卯酉圈曲率半径
    e2 = f * (2 - f)  # 第一偏心率的平方 e^2 = f * (2 - f)
    N = a / math.sqrt(1 - e2 * math.sin(lat_rad) ** 2)  # 卯酉圈曲率半径

    # 计算 ECEF 坐标
    x = (N + h) * math.cos(lat_rad) * math.cos(lon_rad)  # x 坐标
    y = (N + h) * math.cos(lat_rad) * math.sin(lon_rad)  # y 坐标
    z = (N * (1 - f) ** 2 + h) * math.sin(lat_rad)      # z 坐标

    return x, y, z  # 返回转换后的 ECEF 坐标
```

这段代码定义了 `lat_lon_to_ecef` 函数，将给定的纬度、经度和高程转换为地心地固坐标系（ECEF）下的三维坐标 (x, y, z)。函数首先将纬度和经度从度数转换为弧度，再根据椭球参数（长半径和扁率）计算卯酉圈曲率半径，然后利用三角计算得到对应的 ECEF 坐标值，并返回结果。

4.3. 求解七参数并评估拟合的精度

利用已知的公共点坐标，通过最小二乘法估算七参数

```
def ext_B(matrix):
    """
    根据输入的坐标矩阵生成系数矩阵 B，用于七参数计算。
    :param matrix: 输入的坐标矩阵 (n x 3)
    :return: 系数矩阵 B
    """
    n = matrix.shape[0]  # 获取坐标矩阵的行数
    result = []
    for i in range(n):
        X, Y, Z = matrix[i]  # 提取第 i 行的 X, Y, Z 坐标
        # 生成系数矩阵 B 的每一行对应的三行子矩阵
        row = [[1, 0, 0, X, 0, -Z, Y],
               [0, 1, 0, Y, Z, 0, -X],
               [0, 0, 1, Z, -Y, X, 0]]
        result.append(row)  # 将子矩阵添加到结果列表
    return np.vstack(result)  # 将所有子矩阵按行堆叠成最终的矩阵 B

def ext_L(matrix):
    """
    根据输入的坐标矩阵生成矩阵 L，用于七参数计算。
    :param matrix: 输入的坐标矩阵 (n x 3)
    :return: 矩阵 L
    """
    transposed_rows = []
    for row in matrix:
        transposed_rows.append(row.reshape(-1, 1))  # 将每行坐标转置为列向量
    stacked_matrix = np.vstack(transposed_rows)  # 将所有列向量堆叠成矩阵 L
    return stacked_matrix

# 七参数计算
def compute_seven_parameters(coords1, coords2):
    """
```

```
    根据输入的源坐标和目标坐标计算七参数（平移、旋转、尺度因子）。
    :param coords1: 源坐标集合
    :param coords2: 目标坐标集合
    :return: 七参数 dx, dy, dz (平移参数), rx_sec, ry_sec, rz_sec (旋转参数, 以秒为单位), m (尺度变化), xx (均方差)
    """
    Matrix1 = np.array(coords1)
    Matrix2 = np.array(coords2)
    B = -ext_B(Matrix1)  # 获取负的系数矩阵 B
    L = ext_L(Matrix2)  # 获取矩阵 L
    P = np.eye(B.shape[0])  # 权矩阵 P，初始化为单位矩阵
    deltaX = -np.linalg.inv(B.T @ P @ B) @ B.T @ P @ L  # 计算七参数的增量

    # 提取平移参数 dx, dy, dz
    dx = deltaX.reshape(-1)[0]
    dy = deltaX.reshape(-1)[1]
    dz = deltaX.reshape(-1)[2]

    # 提取尺度因子 s
    s = deltaX.reshape(-1)[3]

    # 计算旋转参数 rx, ry, rz 并将其除以尺度因子
    rx = deltaX.reshape(-1)[4] / deltaX.reshape(-1)[3]
    ry = deltaX.reshape(-1)[5] / deltaX.reshape(-1)[3]
    rz = deltaX.reshape(-1)[6] / deltaX.reshape(-1)[3]

    # 将尺度因子转换为百万分率 (ppm)
    m = (s - 1) * (10**6)

    # 将旋转参数从弧度转换为秒
    rx_sec = rx * 648000.0 / math.pi
    ry_sec = ry * 648000.0 / math.pi
    rz_sec = rz * 648000.0 / math.pi

    # 计算残差 V
    V = np.dot(B, deltaX) + L

    # 计算单位权中误差
    xx = math.sqrt(np.sum(V.T @ V) / (3 * Matrix1.shape[0] - 7))

    return dx, dy, dz, rx_sec, ry_sec, rz_sec, m, xx
```

这段代码实现了基于七参数模型的空间坐标转换参数计算 **compute_seven_para**

meters 函数通过输入的源坐标和目标坐标，首先生成计算所需的矩阵 B 和 L，接着使用最小二乘法求解平移参数、旋转参数（以秒为单位）和尺度变化（以百万分率 ppm 表示）。此外，该函数还计算了转换模型的单位权中误差，用于评估拟合的精度

4.4. 使用七参数转换坐标

使用七参数将坐标从第一个坐标系转换到第二个坐标系。

```
def transform_to_second_coordinate_system(coords, params):
    """
    将坐标从第一个坐标系转换到第二个坐标系。
    :param coords: 第一个坐标系的坐标列表 (x, y, z)
    :param params: 七参数 (dx, dy, dz, rx, ry, rz, s)，s=1+m
    :return: 转换后的坐标列表
    """

    transformed = []  # 用于存储转换后的坐标
    dx, dy, dz, rx, ry, rz, s = params  # 将七参数分解为各自的变量

    # 遍历输入坐标并构建旋转矩阵 R
    for x, y, z in coords:
        # 构建旋转矩阵 R，使用 rx, ry, rz 作为旋转角度
        R = np.array([
            [np.cos(ry) * np.cos(rz), np.cos(ry) * np.sin(rz), -np.sin(ry)],
            [-np.cos(rx) * np.sin(rz) + np.sin(rx) * np.sin(ry) * np.cos(rz),
np.cos(rx) * np.cos(rz) + np.sin(rx) * np.sin(ry) * np.sin(rz),
             np.sin(rx) * np.cos(ry)],
            [np.sin(rx) * np.sin(rz) + np.cos(rx) * np.sin(ry) * np.cos(rz), -
np.sin(rx) * np.cos(rz) + np.cos(rx) * np.sin(ry) * np.sin(rz),
             np.cos(rx) * np.cos(ry)]
        ])

        # 应用七参数转换公式：s * R @ [x, y, z] + [dx, dy, dz]
        new_coords = s * R @ np.array([x, y, z]) + np.array([dx, dy, dz])
        transformed.append(new_coords)  # 将转换后的坐标添加到结果列表中

    return transformed  # 返回所有转换后的坐标
```

这段代码定义了一个名为 **transform_to_second_coordinate_system** 的函数，用于将给定的三维坐标从第一个坐标系转换到第二个坐标系。函数接收一个坐标列表和七个转换参数（包括平移、旋转和尺度因子），通过构建旋转矩阵并应用七参数转换

公式，计算转换后的新坐标，并将其存储在结果列表中，最后返回转换后的坐标集合。

4.5. 使用配置法计算改正数并平差

4.5.1. 计算非公共点的改正数

```
def compute_correction_with_seven_params(XYZ1, XYZ2, XYZ_non_public, params):
    """
    计算非公共点的改正数，使用七参数将 XYZ1 转换到坐标系 2 后再计算改正数
    :param XYZ1: 公共点在坐标系 1 中的坐标，大小为 n 行 3 列矩阵
    :param XYZ2: 公共点在坐标系 2 中的坐标，大小为 n 行 3 列矩阵
    :param XYZ_non_public: 非公共点在坐标系 1 中的坐标，大小为 t 行 3 列矩阵
    :param params: 七参数 (dx, dy, dz, rx, ry, rz, s)
    :return: 非公共点的改正数矩阵，大小为 t 行 3 列矩阵
    """
    # 使用七参数将公共点在坐标系 1 中的坐标 XYZ1 转换到坐标系 2 中的坐标
    transformed_XYZ1 = np.array(transform_to_second_coordinate_system(XYZ1,
params))

    # 步骤 1：计算改正数矩阵 V
    # 改正数矩阵 V 为转换到坐标系 2 中的公共点坐标与实际坐标的差值
    V = XYZ2 - transformed_XYZ1  # 大小为 n 行 3 列的矩阵

    # 获取公共点的数量 n 和非公共点的数量 t
    n = XYZ1.shape[0]  # 公共点数量
    t = XYZ_non_public.shape[0]  # 非公共点数量

    # 初始化距离矩阵 S 和权重矩阵 P
    S = np.zeros((t, n))  # t 行 n 列的距离矩阵
    P = np.zeros((t, n))  # t 行 n 列的权重矩阵

    # 步骤 2：计算每个非公共点与所有公共点之间的距离
    for i in range(t):
        for j in range(n):
            # 计算距离 S[i, j] = sqrt((X_i - X_j)^2 + (Y_i - Y_j)^2 + (Z_i - Z_j)^2)
            S[i, j] = np.linalg.norm(XYZ_non_public[i] - XYZ1[j])

    # 步骤 3：根据距离矩阵 S 计算权重矩阵 P
    # P = 1 / S^2，距离为零的权重设为无穷大，后续处理为 0
    with np.errstate(divide='ignore', invalid='ignore'):  # 忽略除以零的警告
        P = 1 / (S**2)
        P[np.isinf(P)] = 0  # 将无穷大的权重设为 0
```

```
    # 步骤 4：计算非公共点的改正数 Vf
    Vf = np.zeros((t, 3))  # 初始化非公共点的改正数矩阵，大小为 t 行 3 列

    for i in range(t):
        # 计算当前非公共点与所有公共点的权重和
        P_sum = np.sum(P[i])
        if P_sum != 0:
            # 根据加权平均计算非公共点的改正数
            # Vf[i] = (P[i] * V 的每一列) / P 的总和
            Vf[i] = np.dot(P[i], V) / P_sum

    return Vf
```

此函数使用给定的七参数将公共点坐标从坐标系 1 转换至坐标系 2，然后计算非公共点的改正数。通过计算非公共点和公共点之间的距离矩阵（S），得到权重矩阵（P），从而根据这些权重分配，计算每个非公共点的改正值。此方法对非公共点与公共点之间的距离敏感，并确保较近的公共点具有更高的权重。

4.5.2. 使用改正数进行平差

```
def compute_and_update_non_public_correction(self):
    """
    计算非公共点的改正数，并将结果更新到表格中
    """
    # 检查是否加载了公共点和非公共点坐标文件
    if not self.coords1 or not self.coords2 or not self.coords_to_transform:
        messagebox.showerror("错误", "请先加载公共点和非公共点坐标文件")
        return

    # 确保七参数已经计算
    if not hasattr(self, 'params') or self.params is None:
        messagebox.showerror("错误", "请先计算七参数")
        return

    # 将公共点和非公共点坐标转换为 numpy 数组
    XYZ1 = np.array(self.coords1)
    XYZ2 = np.array(self.coords2)
    XYZ_non_public = np.array([(x, y, z) for _, _, x, y, z in
self.coords_to_transform])

    # 步骤 1：使用七参数将非公共点转换到坐标系 2
```

```
    # 获取七参数 (dx, dy, dz, rx, ry, rz, s)
    params = compute_seven_parameters(self.coords1, self.coords2)

    # 转换非公共点坐标至坐标系 2
    transformed_non_public_points = transform_to_second_coordinate_system(
        [(x, y, z) for _, _, x, y, z in self.coords_to_transform], params
    )

    # 步骤 2：计算公共点的改正数
    corrections = compute_correction_with_seven_params(XYZ1, XYZ2,
XYZ_non_public, self.params)

    # 清空改正数表格的现有数据
    for row in self.correction_tree.get_children():
        self.correction_tree.delete(row)

    # 清空改正后结果表格的现有数据
    for row in self.corrected_tree.get_children():
        self.corrected_tree.delete(row)

    # 步骤 3：将计算结果插入到表格中
    for i, ((num, name, x, y, z), transformed_point, correction) in enumerate(
            zip(self.coords_to_transform, transformed_non_public_points,
corrections)):
        # 插入改正数到改正数表格
        self.correction_tree.insert("", "end", values=(
            num, name, f"{correction[0]:.8f}", f"{correction[1]:.8f}",
f"{correction[2]:.8f}"
        ))

        # 步骤 4：计算改正后的 X、Y、Z (七参数转换后的坐标 + 改正数)
        X_corrected = transformed_point[0] + correction[0]
        Y_corrected = transformed_point[1] + correction[1]
        Z_corrected = transformed_point[2] + correction[2]

        # 将改正后的 XYZ 转换回大地坐标系 (纬度、经度、高度)
        lat_corrected, lon_corrected, h_corrected =
ecef_to_lat_lon_ecef(X_corrected, Y_corrected, Z_corrected)

        # 插入改正后结果到改正后结果表格
        self.corrected_tree.insert("", "end", values=(
            num, name, f"{lat_corrected:.8f}", f"{lon_corrected:.8f}",
f"{h_corrected:.8f}", f"{X_corrected:.8f}",
```

```
            f"{Y_corrected:.8f}", f"{Z_corrected:.8f}"
        ))
```

此函数负责计算非公共点的改正数并更新到表格中。首先，确保加载了公共和非公共点数据，并检查七参数是否已计算。然后，它调用 **compute_correction_with_seven_params** 函数计算改正数，将结果插入到改正数表格中。接下来，将改正后的非公共点坐标转回大地坐标系，并将最终校正后的坐标插入至表格中，便于后续分析和展示。

4.5.3. 应用改正调整

```
def apply_correction_adjustment(self):
    """
    应用改正调整，将改正后的结果更新到表格中
    """
    # 检查是否加载了公共点和非公共点坐标文件
    if not self.coords1 or not self.coords2 or not self.coords_to_transform:
        messagebox.showerror("错误", "请先加载公共点和非公共点坐标文件")
        return

    # 确保七参数已经计算
    if self.params is None:
        # 计算七参数 (dx, dy, dz, rx, ry, rz, s)
        self.params = compute_seven_parameters(self.coords1, self.coords2)

    # 使用七参数将非公共点坐标转换到坐标系 2
    transformed_non_public_points = transform_to_second_coordinate_system(
        [(x, y, z) for _, _, x, y, z in self.coords_to_transform], self.params
    )

    # 计算非公共点的改正数矩阵
    corrections = compute_correction_with_seven_params(
        np.array(self.coords1), np.array(self.coords2),
        np.array([(x, y, z) for _, _, x, y, z in self.coords_to_transform]),
self.params
    )

    # 清空改正后结果表格的现有数据
    for row in self.corrected_tree.get_children():
        self.corrected_tree.delete(row)
```

```
    # 遍历每个非公共点，应用改正数并插入改正后结果到表格中
    for (num, name, _, _, _), transformed, correction in
zip(self.coords_to_transform,

transformed_non_public_points, corrections):
        # 计算改正后的 X、Y、Z 坐标
        X_corrected = transformed[0] + correction[0]
        Y_corrected = transformed[1] + correction[1]
        Z_corrected = transformed[2] + correction[2]

        # 将改正后的 XYZ 坐标转换为大地坐标系（纬度、经度、高度）
        lat_corrected, lon_corrected, h_corrected =
ecef_to_lat_lon_ecef(X_corrected, Y_corrected, Z_corrected)

        # 将结果插入改正后结果表格中
        self.corrected_tree.insert("", "end", values=(
            num, name, f"{lat_corrected:.8f}", f"{lon_corrected:.8f}",
f"{h_corrected:.8f}", f"{X_corrected:.8f}",
            f"{Y_corrected:.8f}", f"{Z_corrected:.8f}"
        ))
```

这个函数用于应用计算出的改正数来调整非公共点的坐标。它首先确保必要的坐标数据和七参数已经加载和计算。然后，它使用七参数模型转换非公共点的坐标，并获取这些点的改正数。最后，该函数将改正数应用到非公共点的坐标上，计算出调整后的坐标，并将这些信息更新到 GUI 的表格中，从而让用户能够看到改正后的坐标结果。这个过程有助于提高坐标数据的精度和可靠性。这个函数也是 GUI 的一部分，它在 **compute_and_update_non_public_correction** 函数的基础上进一步处理数据，确保用户可以看到最终的调整结果。它也负责处理用户界面的反馈，比如在缺少必要数据时显示错误消息。

4.6. 功能扩展

实际过程中添加了很多新功能，如界面设计、数据异常处理、导出处理等，详见参考附录的完整代码。

4.6.1. 界面设计

程序使用 Python 的 Tkinter 库来设计程序界面，界面设计了许多按钮、输入框、表格以输入和展示相应的数据。

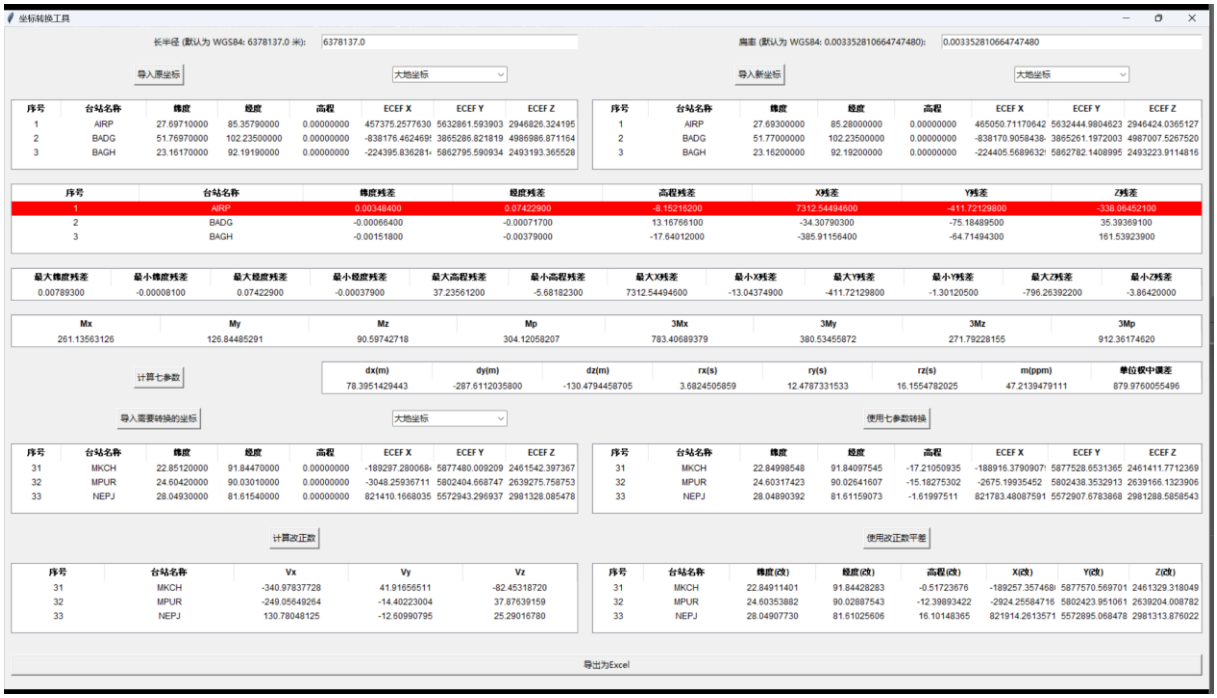


图 1　程序运行界面

代码中，设计了以下按钮、输入框、下拉框和表格：

1. 按钮 (*Button*):

file_button_1：用于导入原坐标文件。

file_button_2：用于导入新坐标文件。

compute_button：用于计算七参数。

file_button_3：用于导入需要转换的坐标文件。

compute_button_2：用于使用七参数转换坐标。

export_button：用于导出数据到 Excel 文件。

compute_button_x：用于计算改正数。

compute_button_y：用于使用改正数平差。

2. 输入框 (*Entry*):

a_entry：用于输入长半径，默认为 WGS84 椭球体的长半径。

f_entry：用于输入扁率，默认为 WGS84 椭球体的扁率。

3. 下拉框 (*Combobox*):

coord_type_combo_1：选择原坐标的类型（大地坐标或空间直角坐标）。

coord_type_combo_2：选择新坐标的类型（大地坐标或空间直角坐标）。

coord_type_combo_3：选择需要转换的坐标类型（大地坐标或空间直角坐标）。

4. 表格（*Treeview*）:

tree1：显示导入的原坐标数据。

tree2：显示导入的新坐标数据。

diff_tree：显示原坐标和新坐标之间的差值。

diff_sort_tree：显示差值的最大值和最小值。

residual_summary_tree：显示残差的汇总数据，如 Mx, My, Mz, Mp 等。

param_tree：显示计算出的七参数。

transform_input_tree：显示导入的需要转换的坐标数据。

transformed_output_tree：显示转换后的坐标数据。

correction_tree：显示计算出的改正数。

corrected_tree：显示应用改正数后的坐标数据。

这些 GUI 组件构成了程序的用户界面，允许用户进行文件导入、参数输入、数据查看和结果导出等操作。通过这些组件，用户可以与程序交互，完成坐标转换和数据处理的任务。

4.6.2. 异常处理

1. 文件读取错误处理

```
def read_file(file_path):
    stations = []
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                # ... (省略部分代码)
    except Exception as e:
        # 异常处理，显示错误信息
        messagebox.showerror("文件读取错误", f"无法读取文件: {e}")
```

2. 检查导入坐标系点数量是否一致

```
def compute_and_update_diff(self):
    if not self.coords1 or not self.coords2:
        messagebox.showerror("错误", "请先加载两个坐标系的文件")
        return

    if len(self.coords1) != len(self.coords2):
```

```
            messagebox.showerror("错误", "两个坐标系的点数量不一致")
            return
    ```
```

3. 检查是否已加载必要的文件

```
    def check_and_calculate_correction(self):
        if self.coords1 and self.coords2 and self.coords_to_transform:
            self.compute_and_update_non_public_correction()
    ```
```

4. 检查是否有计算出的七参数

```
    def compute_and_update_non_public_correction(self):
        if not hasattr(self, 'params') or self.params is None:
            if not self.coords1 or not self.coords2:
                messagebox.showerror("错误", "请先加载两个坐标系的文件")
                return
    ```
```

5. 检查差值是否超出预设的阈值

```
    def compute_and_update_diff(self):
        # ... (省略部分代码)
        warning_shown = False
        for row in self.diff_tree.get_children():
            values = self.diff_tree.item(row)["values"]
            x_diff, y_diff, z_diff = float(values[5]), float(values[6]),
float(values[7])

            if abs(x_diff) > mx3 or abs(y_diff) > my3 or abs(z_diff) > mz3:
                if not warning_shown:
                    messagebox.showwarning("数据异常", "残差表中存在异常值")
                    warning_shown = True
                self.diff_tree.tag_configure("highlight", background="red",
foreground="white")
                self.diff_tree.item(row, tags="highlight")
    ```
```

对于精度检测异常值，在残差表格中将对应台站数据标红便于剔除，除此之外还添加了许多窗口提示。

这些代码段展示了程序如何处理数据读取、文件加载、参数计算和结果展示过程中可能出现的异常情况。通过弹出错误或警告消息框，程序为用户提供了关于异常情况的即时反馈。这样的异常处理机制有助于用户了解数据问题并采取相应的纠正措施。

4.6.3. 数据导出

代码中与数据导出相关的代码主要集中在 *CoordinateConverterApp* 类的 *export_to_excel* 方法中。以下是这个方法的代码片段：

```
def export_to_excel(self):
    # 获取界面上所有数据
    params_data = [self.param_tree.item(item)["values"] for item in self.param_tree.get_children()]
    coords1_data = [self.tree1.item(item)["values"] for item in self.tree1.get_children()]
    coords2_data = [self.tree2.item(item)["values"] for item in self.tree2.get_children()]
    diff_data = [
        [item[0], item[1], round(float(item[2]), 6), round(float(item[3]), 6), round(float(item[4]), 6),
         round(float(item[5]), 6), round(float(item[6]), 6), round(float(item[7]), 6),]
        for item in [self.diff_tree.item(item)["values"] for item in self.diff_tree.get_children()]
    ]
    # ...（省略部分代码）

    # 数据字典，用于将不同表格的数据写入不同的 Excel Sheet
    data_dict = {
        "七参数": pd.DataFrame(params_data, columns=["dx(m)", "dy(m)", "dz(m)", "rx(s)", "ry(s)", "rz(s)", "m(ppm)" ,"单位权中误差"]),
        "原坐标": pd.DataFrame(coords1_data, columns=["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF Z"]),
        "新坐标": pd.DataFrame(coords2_data, columns=["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF Z"]),
        "残差": pd.DataFrame(diff_data, columns=["序号", "台站名称", "纬度残差", "经度残差", "高程残差", "X 残差", "Y 残差", "Z 残差",]),
        # ...（省略部分代码）
    }

    # 保存为 Excel 文件
    file_path = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel files", "*.xlsx")])
    if not file_path:
        return

    with pd.ExcelWriter(file_path, engine="openpyxl") as writer:
        # 创建汇总表
        summary_data = []
```

```
        # ...（省略部分代码）

        # 写入汇总表
        summary_df = pd.DataFrame(summary_data)
        summary_df.to_excel(writer,    sheet_name="  汇  总  ",    index=False,
header=False)

        # 写入单独的工作表
        for sheet_name, df in data_dict.items():
            df.to_excel(writer, sheet_name=sheet_name, index=False)

    # 自动调整列宽并设置单元格居中对齐
    workbook = load_workbook(file_path)
    for sheet_name in workbook.sheetnames:
        sheet = workbook[sheet_name]
        for column_cells in sheet.columns:
            max_length = 0
            column = column_cells[0].column_letter  # 获取列字母
            for cell in column_cells:
                try:
                    if cell.value:
                        max_length = max(max_length, len(str(cell.value)))
                        cell.alignment      =      Alignment(horizontal="center",
vertical="center")  # 设置居中对齐
                except:
                    pass
            adjusted_width = (max_length + 2)  # 加 2 留出额外的空白
            sheet.column_dimensions[column].width = adjusted_width

    workbook.save(file_path)
    messagebox.showinfo("导出成功", f"数据成功导出到 {file_path}")
```

首先收集 GUI 中各个表格的数据，并将它们组织成 *pandas* 的 *DataFrame* 结构。然后使用 *pandas* 的 *ExcelWriter* 将这些 *DataFrame* 写入不同的 Excel 工作表中。最后使用 *openpyxl* 库调整 Excel 列宽，并设置单元格的居中对齐样式。这个方法还负责创建一个汇总表，将所有重要的数据汇总到一个单独的工作表中，以便于用户查看和分析。此外，它还处理了文件保存对话框的显示和用户反馈消息的弹出。

5. 结果说明

程序运行结果输出到 EXCEL 表格中（见附件），下面展示部分结果并进行说明。

表 2　七参数（初始结果）

平移参数	$dx(m)$	78.3951429443
	$dy(m)$	-287.6112035800
	$dz(m)$	-130.4794458705
旋转参数	$rx(s)$	3.6824505859
	$ry(s)$	12.4787331533
	$rz(s)$	16.1554782025
尺度变化参数	$m(ppm)$	47.2139479111
单位权中误差		879.9760055496

表 3　转换后的 31—50 号台站坐标（初始结果）

序号	台站名称	纬度(°)	经度(°)	高程(m)	ECEF X(m)	ECEF Y(m)	ECEF Z(m)
31	*MKCH*	22.84998548	91.84097545	−17.21050935	−188916.37909079	5877528.65313650	2461411.77123694
32	*MPUR*	24.60317423	90.02641607	−15.18275302	−2675.19935452	5802438.35329131	2639166.13239065
33	*NEPJ*	28.04890392	81.61159073	−1.61997511	821783.48087591	5572907.67838686	2981288.58585435
34	*NR*01	30.14186833	100.65932063	−22.13982647	−1021094.55752004	5425100.08626035	3183972.52207814
35	*NR*03	29.90478682	100.09848259	−22.05659374	−970248.76238926	5447781.12342194	3161217.86112789
36	*NR*09	31.44619861	103.16660724	−21.95519790	−1240583.16417643	5303153.82417578	3308187.78631218
37	*NVSK*	54.84134020	83.23419753	34.42435673	433671.29013170	3655432.49305555	5191260.94752930
38	*OSHK*	40.53051923	72.77419255	24.78353017	1437648.47438464	4636897.18013823	4122952.42020940
39	*PODG*	43.32792157	79.48218295	15.69543598	848277.13997774	4568964.17602402	4354085.04898453
40	*QIME*	31.02158464	101.18220395	−21.75156452	−1060938.65740583	5366880.00477406	3267933.34569373
41	*RBIT*	26.84842105	89.38874114	−14.14128935	60750.27895588	5694154.73237599	2863234.39854712
42	*RMJT*	27.30439977	86.54635272	−10.08160141	341656.03577908	5661184.59866199	2908223.15695273
43	*RMTE*	26.99018182	86.59343319	−10.21575952	337948.04418853	5677321.56308884	2877241.58213622
44	*RPUR*	23.03535525	90.75944304	−15.97233041	−77837.25943159	5872047.06726505	2480316.74397454
45	*SHTZ*	37.56280798	68.11902433	32.93689181	1886573.22124075	4697504.70440388	3867112.24450196
46	*SITA*	22.61358384	91.65875260	−16.96374380	−170515.46144102	5888211.80992830	2437266.26357077
47	*TASH*	41.32881268	69.29216992	33.42091576	1696083.49106282	4486700.26074578	4189936.08951054
48	*ULAB*	47.86429786	107.05124707	3.18652086	−1257040.70101438	4098480.22513669	4706769.18973929
49	*WARI*	30.87988067	101.12259138	−21.82927119	−1056912.86358553	5375909.01037927	3254459.85626548
50	*XIAN*	34.36712188	109.21923164	−19.04694750	−1734916.95470793	4976623.70777409	3580123.73224043

表 4　非公共点改正数（初始结果）

序号	台站名称	Vx(m)	Vy(m)	Vz(m)
31	*MKCH*	−340.97837728	41.91656511	−82.45318720
32	*MPUR*	−249.05649264	−14.40223004	37.87639159
33	*NEPJ*	130.78048125	−12.60990795	25.29016780
34	*NR*01	−44.05895316	−13.02708278	14.41567168
35	*NR*03	−48.79463897	−12.74933456	14.54488500
36	*NR*09	−27.03280340	−15.36145253	14.95207184
37	*NVSK*	−66.18516273	33.57544710	−41.29836463
38	*OSHK*	−208.05080763	102.37884645	−69.73251384
39	*PODG*	−184.57370216	38.72477720	−19.21797225
40	*QIME*	−30.89342213	−13.82735618	14.37596443
41	*RBIT*	57.45393547	−27.65308161	35.44277459
42	*RMJT*	1026.40375284	−75.14244509	−2.20081172
43	*RMTE*	641.61829914	−59.34128140	25.53103803
44	*RPUR*	−302.17593273	21.12248019	−38.68139314
45	*SHTZ*	−234.88125526	123.76700864	−82.38659603
46	*SITA*	−340.41364611	7.64576827	−3.13448909
47	*TASH*	−222.31909650	116.81080464	−78.91294820
48	*ULAB*	−20.61950019	−44.55682691	20.54425755
49	*WARI*	−32.81504821	−13.70390870	14.37594956
50	*XIAN*	−22.15064253	−34.95492985	29.53436161

表 5　改正后 31—50 号台站坐标（初始结果）

序号	台站名称	纬度(改) (°)	经度(改) (°)	高程(改)	X(改) (m)	Y(改) (m)	Z(改) (m)
31	*MKCH*	22.84911401	91.84428283	−0.51723676	−189257.35746808	5877570.56970161	2461329.31804973
32	*MPUR*	24.60353882	90.02887543	−12.39893422	−2924.25584716	5802423.95106127	2639204.00878223
33	*NEPJ*	28.04907730	81.61025606	16.10148365	821914.26135716	5572895.06847891	2981313.87602215
34	*NR*01	30.14200186	100.65979504	−18.92455126	−1021138.61647320	5425087.05917756	3183986.93774983
35	*NR*03	29.90491854	100.09900315	−18.26894942	−970297.55702823	5447768.37408738	3161232.40601289
36	*NR*09	31.44635506	103.16692096	−21.66219395	−1240610.19697983	5303138.46272325	3308202.73838402
37	*NVSK*	54.84093896	83.23528209	15.37045481	433605.10496897	3655466.06850265	5191219.64916467
38	*OSHK*	40.52983021	72.77689568	6.96725814	1437440.42357702	4636999.55898468	4122882.68769556
39	*PODG*	43.32776865	79.48450757	5.69888700	848092.56627559	4569002.90080121	4354065.83101228
40	*QIME*	31.02173096	101.18254944	−20.83310509	−1060969.55082796	5366866.17741788	3267947.72165816
41	*RBIT*	26.84881665	89.38816012	−22.25773171	60807.73289135	5694127.07929438	2863269.84132171
42	*RMJT*	27.30443627	86.53595662	−22.71469803	342682.43953192	5661109.45621691	2908220.95614101
43	*RMTE*	26.99047346	86.58694526	−17.40788485	338589.66248767	5677262.22180745	2877267.11317425
44	*RPUR*	23.03494501	90.76238822	−7.97867771	−78139.43536432	5872068.18974524	2480278.06258140
45	*SHTZ*	37.56206949	68.12201334	4.36997497	1886338.33998549	4697628.47141252	3867029.85790593

46	*SITA*	22.61349693	91.66206008	−2.00867175	−170855.87508714	5888219.45569657	2437263.12908168
47	*TASH*	41.32809682	69.29514736	4.33113756	1695861.17196632	4486817.07155042	4189857.17656233
48	*ULAB*	47.86466561	107.05168516	−6.10086432	−1257061.32051456	4098435.66830977	4706789.73399683
49	*WARI*	30.88002490	101.12295575	−20.55828060	−1056945.67863374	5375895.30647057	3254474.23221504
50	*XIAN*	34.36747251	109.21958412	−23.60101951	−1734939.10535046	4976588.75284424	3580153.26660204

实际上，这个结果的误差非常大（下方精度评价中会给出相应的指标），由于代码中包含“检查差值是否超出预设的阈值”的部分，在程序界面已经给出提示并将误差过大的数据标红，所以需要剔除误差过大的数据以得到更加精确的结果（见下方精度评价和附录）。

6. 精度评价

6.1. 评价指标

6.1.1. 残差（*Residuals*）

残差是指转换后的坐标与实际坐标之间的差异。对于每个点，残差可以通过以下公式计算

$$\text{残差} = \text{实际坐标} - \text{转换后的坐标}$$

代码中分别计算了纬度残差、经度残差、高程残差以及 ECEF 坐标系中的 X、Y、Z 残差。

表 6　1—30 号台站坐标残差（初始结果）

序号	台站名称	纬度残差	经度残差	高程残差	X 残差	Y 残差	Z 残差
1	*AIRP*	0.003484	0.074229	−8.152162	7312.544946	−411.721298	−338.064521
2	*BADG*	−0.000664	−0.000717	13.167661	−34.307903	−75.184895	35.393691
3	*BAGH*	−0.001518	−0.00379	−17.64012	−385.911564	−64.714943	161.539239
4	*BARK*	−0.001651	−0.003411	−17.777435	−347.860444	−68.675159	175.455076
5	*BELT*	−0.000144	−0.004054	−5.681823	−398.62553	40.80397	16.784481
6	*BESI*	−0.000937	−0.003977	−6.432129	−392.699558	−4.999493	94.528888
7	*BJNM*	−0.001355	−0.001486	−7.862877	−73.143552	−137.707164	119.93903
8	*BRN2*	−0.001078	−0.00347	−11.293564	−347.569342	−26.720705	111.952398
10	*CHNR*	−0.001429	−0.00365	−16.96024	−369.433795	−59.164648	151.340693
11	*DAMA*	−0.000507	−0.003988	−7.77332	−393.882369	14.528662	53.349238
12	*DIGH*	0.007893	−0.002689	−17.513102	−288.052791	350.929165	−796.263922
13	*DMAU*	−0.000143	−0.003499	−6.308698	−342.736561	32.557717	16.960608

14	*GARM*	0.001168	−0.003855	28.776092	−294.450984	168.29117	−118.895563
15	*GHER*	−0.000932	−0.003965	−6.440709	−391.048861	−5.314808	93.894952
16	*IRKT*	−0.000455	−0.000379	14.101998	−13.043749	−53.55921	19.8848
17	*JAFL*	−0.000708	−0.003464	−17.504022	−348.309321	−29.78567	78.382323
18	*JAML*	−0.000673	−0.003508	−16.633254	−353.662446	−24.065776	74.606053
20	*JURI*	−0.000748	−0.003304	−17.657009	−333.998305	−30.75066	82.73399
21	*KAWA*	−0.000551	−0.003827	−6.15778	−378.050755	15.859994	56.98442
22	*KIRT*	−0.000713	−0.003475	−8.044987	−344.102675	−1.301205	73.692094
23	*KIT3*	0.000629	−0.003326	37.235612	−258.531457	126.884225	−77.670872
24	*KMTR*	0.000219	−0.002714	15.785052	−219.610189	50.468447	−28.653161
25	*KPTI*	−0.001653	−0.003632	−17.564643	−371.384356	−68.223587	175.841112
26	*KRTV*	0.000784	−0.001908	30.690769	−122.634419	73.748678	−78.998602
27	*KSTU*	0.000616	−0.000694	29.062196	−45.226877	38.426692	−62.440161
28	*KUMT*	−0.000081	−0.002514	15.79484	−207.979966	25.362058	−3.8642
30	*MANM*	0.000842	−0.003195	24.352238	−256.254161	124.473712	−88.903868

代码还计算了残差的最大值和最小值，以确定残差的范围。这些值可以帮助识别可能的异常值或系统性偏差。

表 7　残差最值（初始结果）

最大 X 残差	最小 X 残差	最大 Y 残差	最小 Y 残差	最大 Z 残差	最小 Z 残差
7312.544946	−13.043749	−411.721298	−1.301205	−796.263922	−3.8642

6.1.2. 点位中误差（*Mx*、*My*、*Mz*和*Mp*）

根据下面公式计算点位中误差

$$M_p = \pm\sqrt{(M_x^2 + M_y^2 + M_z^2)} \tag{6-1}$$

$$M_x = \pm\sqrt{\left(\frac{[VV]_x}{(n-1)}\right)} \tag{6-2}$$

$$M_y = \pm\sqrt{\left(\frac{[VV]_y}{(n-1)}\right)} \tag{6-3}$$

$$M_z = \pm\sqrt{\left(\frac{[VV]_z}{(n-1)}\right)} \tag{6-4}$$

表 8　点位中误差与限差（初始结果）

Mx	*My*	*Mz*	*Mp*	3*Mx*	3*My*	3*Mz*	3*Mp*
261.135631	126.844853	90.597427	304.120582	783.406894	380.534559	271.792282	912.361746

以点位中误差的三倍（$3Mp$）及其任一分量的三倍（$3Mx$、$3My$、$3Mz$）作为限差，当残差超过对应限差时，说明该台站的坐标值误差较大，可能存在一些问题，如数据错误或者发生地震等，此时对相应台站的信息进行标红，偏于后续剔除。

显然，表 7 中的残差最值明显超过了表 8 中的限差，说明只使用初始数据求解的转换参数误差很大，需要进行后续处理。

6.1.3. 单位权中误差（σ）

衡量在坐标转换过程中，单位权的误差大小的一个指标，用于**评估七个转换参数的精度**。

$$\sigma = \sqrt{\frac{\left[V^T PV\right]}{n-t}} \tag{6-5}$$

式中n为方程数量，由于一个点提供三个坐标（X、Y、Z），若记公共点个数为N，则总共有$n = 3N = 3 \times 27 = 81$，$t$为必要观测，由于需要得到七个参数，所以取$t = 7$，式中的$V$可以通过(3-8)或者(3-10)求得，假设数据中公共点的观测值都是同等精度的，则权阵P为单位阵。

使用初始数据计算的转换参数的点位中误差具体数值为879.9760055496（见表 2），远远超出一般工程所需要的精度要求，需要对初始数据进行处理。

6.2. 应用精度指标处理初始数据

由于代码中添加了异常处理，当使用初始数据进行解算时， 程序根据 6.1.2 的指标检测出超限数据并进行窗口提醒和高亮标记。出现异常的数据见下方表 9。

表 9　使用初始数据计算的残差表格中的异常值与限差

序号	台站名称	X 残差	Y 残差	Z 残差	残差
1	*AIRP*	−7312.544946	411.721298	338.064521	7331.924416
12	*DIGH*	288.052791	−350.929165	796.263922	916.6034708
限差		$3Mx$	$3My$	$3Mz$	$3Mp$
		783.406894	380.534559	271.792282	912.361746

根据这个结果，剔除初始数据中的 1 号台站（$AIDP$）和 12 号台站（$DIGH$），再次进行解算，结果仍然有残差超限。

表 10　剔除 1 号和 12 号台站计算得到的残差与限差

序号	台站名称	X 残差	Y 残差	Z 残差	残差
14	$GARM$	-16.204913	45.062811	-41.639426	63.45941962
28	$KUMT$	20.625781	-40.423112	42.173998	61.95237633
限差		$3Mx$	$3My$	$3Mz$	$3Mp$
		111.222226	34.794677	77.418104	139.909314

如果只考虑残差平方和开根号的结果与三倍点位中误差，则 14 和 28 号台站的精度已经符合要求。由于 14 号台站和 28 号台站残差的Y分量仍然超限，所以如果要求比较严格的情况下，需要每个点在每个方向分量上的坐标都不超限，那仍然需要进一步处理。

如果在此基础上剔除 28 号台站而不剔除 14 号台站，则 2 号台站、14 号台站和 23 号台站残差的X方向分量超限。

表 11　剔除 1 号、12 号和 28 号台站计算得到的残差与限差

序号	台站名称	X 残差	Y 残差	Z 残差	残差
2	$BADG$	-1.311981	-28.765367	24.138186	37.5741887
14	$GARM$	-13.722971	41.717415	-39.068473	58.77931805
23	$KIT3$	25.959981	-26.354726	21.811461	42.94452267
限差		$3Mx$	$3My$	$3Mz$	$3Mp$
		118.864565	25.401909	70.031919	140.280118

如果只剔除 14 号台站而不剔除 28 号台站，则所有点位精度全部合乎要求。

综上所述，初始数据中的 1 号台站和 12 号台站的坐标存在较大误差，可能是粗差，可能是由于地震或者错误记录等因素导致的，应当予以剔除，而初始数据中 14 号台站的坐标存在较小的误差，可能是观测精度不足等因素导致的，如果追求更高精度的结果，也可以剔除 14 号台站的坐标数据。

7. 总结

在本次《大地测量学基础》课程的编程实践作业中，我成功实现了从大地坐标到空间直角坐标的转换、七参数的解算以及坐标转换精度的评估与改正。项目主要包括对大地坐标系转换的全过程，从读取坐标数据、计算七参数，到应用平差法改正非公共点坐标，并最终对转换精度进行评价。通过对 27 个公共点的最小二乘法解算，完成了三平移、三旋转以及尺度变化七个转换参数的求解。之后在非公共点坐标的转换中，结合配

置法平差计算了改正数，有效提升了转换精度。在精度评价环节，依据残差、单位权中误差和点位中误差等指标，剔除了个别误差较大的异常台站，使得转换结果更加精准。在设计与实现过程中，特别加入了图形用户界面设计，以便直观呈现数据导入、七参数计算、坐标转换等主要功能，并通过异常处理机制提升了数据的有效性和程序的可操作性。整个过程严谨有序，完成了预期的编程任务。

此次实践的收获不仅仅在于掌握了坐标转换的理论与技术，更在于解决实际问题的过程中对大地测量的深入理解与体会。最开始在将地理坐标转换为空间直角坐标的过程中，每个公式的推导和每个变量的计算都让我感受到，坐标系统的构建并不只是数据上的简单变换，而是对地理坐标学理论的实践应用。在七参数解算过程中，通过逐步调整每个平移、旋转和尺度变化参数，切实感受到了参数变化对结果的显著影响，而这一过程让我体会到了测量精度的重要性。特别是当数据转换后通过精度评价指标发现异常台站，通过剔除这些异常数据显著提升了最终的转换精度时，成就感油然而生，整个解算和改正的过程让我深刻理解到数据清洗与处理对测量数据的重要性。

在程序设计过程中，界面设计和异常处理机制的加入为整个程序增色不少。设计图形用户界面时，面对一个个功能从思路到实现的过程，虽然调试花费了不少时间，但看到功能顺利运行、数据流畅展示时，顿时觉得所有的努力都是值得的。这不仅仅是一次编程实践，更是一次充满挑战和探索的旅程。经过这次实践，测绘数据处理在我心中不再只是抽象的计算公式和处理过程，而是充满成就感的专业技能。未来在大地测量与地理信息系统开发中，这次实践的经历无疑是一份宝贵的积累，坚定了我继续探索测绘技术、提升专业能力的决心。

附录 1 第一次剔除数据计算结果

下面展示剔除 1 号台站和 12 号台站数据的计算结果：

1. 七参数与单位权中误差

平移参数	$dx(m)$	-7.5401029000
	$dy(m)$	8.0802287041
	$dz(m)$	-2.8532848923
旋转参数	$rx(s)$	-0.2825168845
	$ry(s)$	-0.2598942274
	$rz(s)$	0.3197083797
尺度变化参数	$m(ppm)$	-0.7351238754
单位权中误差		25.4631134652

2. 残差

序号	台站名称	纬度残差	经度残差	高程残差	X 残差	Y 残差	Z 残差
2	*BADG*	−0.000291	−0.000075	−1.011495	0.205799	−25.331964	20.824632
3	*BAGH*	−0.000272	−0.000148	1.901959	−14.586234	−14.153305	26.919169
4	*BARK*	−0.000371	0.000253	1.967717	26.661361	−16.601834	37.148273
5	*BELT*	0.000418	−0.000346	0.429239	−31.708219	24.51548	−41.274057
6	*BESI*	−0.000382	−0.000347	0.421658	−35.866321	−16.987707	37.143043
7	*BJNM*	0.000321	0.000023	1.587933	−7.863668	20.475895	−28.264512
8	*BRN2*	−0.000279	0.000153	0.963474	14.537154	−15.361692	27.204906
10	*CHNR*	−0.000273	−0.000048	1.71714	−4.505772	−14.094062	26.908814
11	*DAMA*	0.000119	−0.000347	0.578709	−33.610878	8.476522	−11.917297
13	*DMAU*	0.000418	0.000154	0.433298	17.169781	19.711295	−41.08184
14	*GARM*	0.000501	−0.000351	−2.521759	−16.204913	45.062811	−41.639426
15	*GHER*	−0.000383	−0.000347	0.410375	−35.835922	−17.099056	37.106224
16	*IRKT*	0.000009	0.000124	−0.96125	7.869075	3.46071	0.125237
17	*JAFL*	0.000426	0.000051	1.671732	4.455536	18.736902	−43.426143
18	*JAML*	0.000426	0.000051	1.600033	4.797942	18.586293	−43.412602
20	*JURI*	0.000427	0.000251	1.75695	24.775286	18.943185	−43.753049
21	*KAWA*	0.000018	−0.000146	0.449236	−14.275539	1.98943	−1.958826
22	*KIRT*	−0.000081	0.000153	0.594007	14.679339	−5.936737	7.701034
23	*KIT3*	−0.000201	0.00035	−2.998909	23.169475	−22.698201	19.229053
24	*KMTR*	−0.000195	0.000143	−1.821309	8.923698	−15.257763	17.357015
25	*KPTI*	−0.000371	0.000053	1.972269	6.12277	−17.32011	37.203404
26	*KRTV*	0.000002	0.000034	−2.791203	2.711101	1.409988	2.03776
27	*KSTU*	0.000305	−0.000275	−2.295503	−18.571646	28.533401	−17.069747
28	*KUMT*	−0.000495	0.000343	−1.821871	20.625781	−40.423112	42.173998
30	*MANM*	0.000303	0.00035	−2.199908	36.325027	11.363894	−25.284647

3. 残差最值

最大纬度残差	最小纬度残差	最大经度残差	最小经度残差	最大高程残差	最小高程残差
0.000501	0.000002	−0.000351	0.000023	−2.998909	0.410375
最大 X 残差	最小 X 残差	最大 Y 残差	最小 Y 残差	最大 Z 残差	最小 Z 残差
36.325027	0.205799	45.062811	1.409988	−43.753049	0.125237

4. 点位中误差

Mx	My	Mz	Mp	$3Mx$	$3My$	$3Mz$	$3Mp$
37.074075	11.598226	25.806035	46.636438	111.222226	34.794677	77.418104	139.909314

5. 转换后坐标

序号	台站名称	纬度	经度	高程	ECEF X	ECEF Y	ECEF Z
31	$MKCH$	22.85122833	91.84465294	1.89301268	−189292.46943768	5877480.69063408	2461546.02330760
32	$MPUR$	24.60422516	90.03005279	1.49837842	−3043.47807444	5802404.87324055	2639278.91653355
33	$NEPJ$	28.04931550	81.61535519	0.07839739	821414.41733953	5572941.92371129	2981329.63869834
34	$NR01$	30.14322636	100.66214004	1.98504623	−1021351.38776640	5424996.04549379	3184114.82324274
35	$NR03$	29.90612631	100.10134064	1.96365249	−970511.17724095	5447680.33583160	3161358.54931598
36	$NR09$	31.44762621	103.16923722	2.03538491	−1240812.43669869	5303036.39343904	3308335.34209300
37	$NVSK$	54.84060224	83.23532841	−2.87734751	433604.52154811	3655486.41633331	5191183.14523131
38	$OSHK$	40.52990215	72.77744613	−2.34725197	1437392.24162412	4637001.64778828	4122882.70657091
39	$PODG$	43.32750490	79.48484085	−1.83711801	848068.65777608	4569022.21308734	4354039.34393697
40	$QIME$	31.02292570	101.18493903	1.92120259	−1061183.92238253	5366774.08332336	3268072.96252379
41	$RBIT$	26.84932255	89.39225120	1.18654751	60401.10951311	5694127.00776560	2863330.44057366
42	$RMJT$	27.30512000	86.54995277	0.79187654	341298.70590057	5661179.15565456	2908299.06058260
43	$RMTE$	26.99092031	86.59705304	0.83048538	337587.73996296	5677315.65702822	2877319.51047033
44	$RPUR$	23.03652729	90.76315365	1.74609694	−78217.08787521	5872007.53189129	2480443.12445494
45	$SHTZ$	37.56210001	68.12275095	−2.68902832	1886275.00907545	4697645.64656716	3867028.23951790
46	$SITA$	22.61482843	91.66245330	1.89545735	−170894.74543438	5888165.21182219	2437400.74238367
47	$TASH$	41.32799968	69.29554634	−2.87432762	1695830.53312196	4486830.48324953	4189844.31721715
48	$ULAB$	47.86511279	107.05222536	−0.09435756	−1257090.33033176	4098392.41824238	4706827.54533364
49	$WARI$	30.88122582	101.12533917	1.93385618	−1057159.84683775	5375803.22397193	3254600.04407391
50	$XIAN$	34.36872555	109.22143092	2.08634646	−1735080.66169622	4976478.75811210	3580282.50032038

6. 改正数

序号	台站名称	Vx	Vy	Vz
31	$MKCH$	4.35231523	−13.88266357	28.72164033
32	$MPUR$	3.66710609	3.71716994	−10.59013646
33	$NEPJ$	−15.47206082	3.13493698	−4.17427290
34	$NR01$	0.82847449	0.17970476	−1.40602319
35	$NR03$	0.97231852	0.20714109	−1.53946162
36	$NR09$	0.21949375	0.34859338	−1.32739637
37	$NVSK$	−2.18657963	5.13004110	−1.37051376

38	*OSHK*	6.78644325	9.99313836	−11.55156953
39	*PODG*	12.75471222	−22.70353211	24.24055767
40	*QIME*	0.48742656	0.20167268	−1.32922898
41	*RBIT*	1.73634420	−0.12221808	−1.79355982
42	*RMJT*	−3.76845284	−3.47557145	6.52724863
43	*RMTE*	0.33424706	−5.98645432	10.87336605
44	*RPUR*	3.76131711	−6.48403688	12.05582060
45	*SHTZ*	11.85038781	4.37143219	−7.26689425
46	*SITA*	6.75712195	−13.65421277	28.60543466
47	*TASH*	5.83815426	9.62176014	−10.76671925
48	*ULAB*	1.49382425	−3.93753087	3.81063281
49	*WARI*	0.53693807	0.19647728	−1.33582462
50	*XIAN*	−1.42323033	3.45551514	−5.21321406

7. 改正后坐标

序号	台站名称	纬度(改)	经度(改)	高程(改)	X(改)	Y(改)	Z(改)
31	*MKCH*	22.85151648	91.84461491	0.13127115	−189288.11712245	5877466.80797051	2461574.74494792
32	*MPUR*	24.60412427	90.03001656	0.46713058	−3039.81096835	5802408.59041049	2639268.32639709
33	*NEPJ*	28.04927867	81.61551553	−1.13841947	821398.94527870	5572945.05864827	2981325.46442544
34	*NR*01	30.14321528	100.66213124	1.29915976	−1021350.55929191	5424996.22519855	3184113.41721956
35	*NR*03	29.90611412	100.10133035	1.22505595	−970510.20492243	5447680.54297269	3161357.00985436
36	*NR*09	31.44761464	103.16923414	1.58976560	−1240812.21720494	5303036.74203242	3308334.01469664
37	*NVSK*	54.84055963	83.23537162	−1.21254599	433602.33496848	3655491.54637440	5191181.77471755
38	*OSHK*	40.52975546	72.77740455	−1.07182326	1437399.02806737	4637011.64092665	4122871.15500138
39	*PODG*	43.32778711	79.48463515	0.25113803	848081.41248830	4568999.50955522	4354063.58449463
40	*QIME*	31.02291495	101.18493361	1.32466131	−1061183.43495598	5366774.28499604	3268071.63329481
41	*RBIT*	26.84930853	89.39223372	0.28388975	60402.84585731	5694126.88554753	2863328.64701385
42	*RMJT*	27.30518765	86.54998865	0.50189631	341294.93744773	5661175.68008311	2908305.58783123
43	*RMTE*	26.99103215	86.59704609	0.45805142	337588.07421002	5677309.67057390	2877330.38383638
44	*RPUR*	23.03665056	90.76311780	0.45122668	−78213.32655810	5872001.04785442	2480455.18027554
45	*SHTZ*	37.56200157	68.12264492	−0.40312642	1886286.85946326	4697650.01799935	3867020.97262365
46	*SITA*	22.61511496	91.66239146	0.11530449	−170887.98831243	5888151.55760942	2437429.34781833
47	*TASH*	41.32786109	69.29552174	−1.67558190	1695836.37127622	4486840.10500967	4189833.55049791
48	*ULAB*	47.86516381	107.05222170	−0.08786787	−1257088.83650751	4098388.48071151	4706831.35596645
49	*WARI*	30.88121507	101.12533327	1.32476776	−1057159.30989967	5375803.42044921	3254598.70824929
50	*XIAN*	34.36866777	109.22143316	2.22341364	−1735082.08492655	4976482.21362724	3580277.28710632

附录 2 第二次剔除数据计算结果

下面展示剔除 1 号、12 号和 14 号台站数据的运行结果：

1. 七参数与单位权中误差

平移参数	$dx(m)$	-12.4290058960
	$dy(m)$	-0.1499539313
	$dz(m)$	-6.7432739490
旋转参数	$rx(s)$	-0.3122461964
	$ry(s)$	-0.1090425746
	$rz(s)$	0.6304963254
尺度变化参数	$m(ppm)$	0.6475312859
单位权中误差		24.7047345897

2. 残差

序号	台站名称	纬度残差	经度残差	高程残差	X 残差	Y 残差	Z 残差
2	*BADG*	−0.000264	−0.000013	0.396823	4.076829	−22.990252	17.875213
3	*BAGH*	−0.000269	−0.000165	1.798524	−16.397419	−14.00808	26.681013
4	*BARK*	−0.000369	0.000235	1.88319	24.81746	−16.517442	36.9833
5	*BELT*	0.000435	−0.000371	−0.275349	−34.00787	26.299348	−42.683057
6	*BESI*	−0.000364	−0.00037	−0.240638	−37.978609	−15.233771	35.677171
7	*BJNM*	0.000314	0.000066	3.890653	−3.588081	20.004103	−29.09662
8	*BRN2*	−0.000266	0.000132	0.478433	12.525454	−14.201163	26.16005
10	*CHNR*	−0.000268	−0.000065	1.551974	−6.272673	−13.767511	26.481201
11	*DAMA*	0.000135	−0.000369	−0.043947	−35.731213	10.065221	−13.254696
13	*DMAU*	0.000436	0.00013	−0.238939	15.003649	21.463702	−42.52412
15	*GHER*	−0.000364	−0.00037	−0.24836	−37.921636	−15.338451	35.624403
16	*IRKT*	0.000033	0.000191	0.618736	12.049684	5.70925	−2.761356
17	*JAFL*	0.000432	0.000035	1.544425	2.901355	19.068551	−43.948397
18	*JAML*	0.000432	0.000035	1.409947	3.105891	19.019086	−43.969056
20	*JURI*	0.000432	0.000235	1.641166	23.141784	19.202355	−44.175731
21	*KAWA*	0.000035	−0.00017	−0.234749	−16.508876	3.738508	−3.370169
22	*KIRT*	−0.000065	0.000131	−0.016334	12.592733	−4.371051	6.367163
23	*KIT3*	−0.000151	0.000306	−3.998342	21.432683	−17.260301	15.503982
24	*KMTR*	−0.00015	0.000129	−2.349699	8.548743	−11.38944	14.009308
25	*KPTI*	−0.000369	0.000035	1.876549	4.224429	−17.229413	37.057427
26	*KRTV*	0.000059	0.00004	−2.714662	4.111888	6.066395	−2.021682
27	*KSTU*	0.000351	−0.000214	−1.183905	−14.972854	32.32198	−20.84537
28	*KUMT*	−0.00045	0.000329	−2.350847	20.249241	−36.553889	38.826188
30	*MANM*	0.000347	0.000315	−3.168334	34.59743	15.902984	−28.59584

3. 残差最值

最大纬度残差	最小纬度残差	最大经度残差	最小经度残差	最大高程残差	最小高程残差
−0.00045	0.000033	−0.000371	−0.000013	−3.998342	−0.016334
最大 X 残差	最小 X 残差	最大 Y 残差	最小 Y 残差	最大 Z 残差	最小 Z 残差
−37.978609	2.901355	−36.553889	3.738508	−44.175731	−2.021682

4. 点位中误差

Mx	My	Mz	Mp	$3Mx$	$3My$	$3Mz$	$3Mp$
35.341546	16.245023	29.210875	48.64361	106.024638	48.735068	87.632624	145.930831

5. 转换后坐标

序号	台站名称	纬度	经度	高程	ECEF X	ECEF Y	ECEF Z
31	$MKCH$	22.85123100	91.84463444	1.76322102	−189290.56445043	5877480.51738955	2461546.24548746
32	$MPUR$	24.60423223	90.03003384	1.21415409	−3041.55867395	5802404.28994555	2639279.50984894
33	$NEPJ$	28.04933621	81.61532715	−0.74916683	821416.88078013	5572939.73160003	2981331.27484520
34	$NR01$	30.14322868	100.66214401	2.63091642	−1021351.84345815	5424996.39617637	3184115.37074470
35	$NR03$	29.90612903	100.10134349	2.55544130	−970511.51181602	5447680.64455533	3161359.10580524
36	$NR09$	31.44762700	103.16924663	2.92968871	−1240813.47043995	5303036.88826157	3308335.88326857
37	$NVSK$	54.84065922	83.23535987	−2.29199014	433601.94349110	3655481.83886708	5191187.27685598
38	$OSHK$	40.52995005	72.77741807	−3.13660107	1437393.31165254	4636997.06894044	4122886.23668895
39	$PODG$	43.32755043	79.48483246	−2.23105865	848068.64148968	4569018.39489153	4354042.75286640
40	$QIME$	31.02292850	101.18494517	2.62097890	−1061184.58227726	5366774.40142166	3268073.58858791
41	$RBIT$	26.84933350	89.39223420	0.85619405	60402.78962864	5694126.14687225	2863331.37446908
42	$RMJT$	27.30513463	86.54993193	0.26192132	341300.69186875	5661177.81948312	2908300.25733928
43	$RMTE$	26.99093443	86.59703186	0.30152942	337589.76777060	5677314.35322457	2877320.66400472
44	$RPUR$	23.03653136	90.76313383	1.52689597	−78215.05137607	5872007.18099329	2480443.45320891
45	$SHTZ$	37.56214724	68.12270861	−3.74250284	1886276.97824760	4697640.51209321	3867031.75290753
46	$SITA$	22.61483096	91.66243426	1.75262757	−170892.78125196	5888165.02911005	2437400.94617403
47	$TASH$	41.32805173	69.29551157	−3.70814615	1695831.68523931	4486825.29771482	4189848.10728408
48	$ULAB$	47.86512758	107.05228040	1.50491773	−1257094.22445673	4098391.07041972	4706829.83461114
49	$WARI$	30.88122851	101.12534499	2.62719486	−1057160.47771055	5375803.55043302	3254600.65575400
50	$XIAN$	34.36872170	109.22145296	3.61151834	−1735083.06976719	4976479.50695881	3580283.00895598

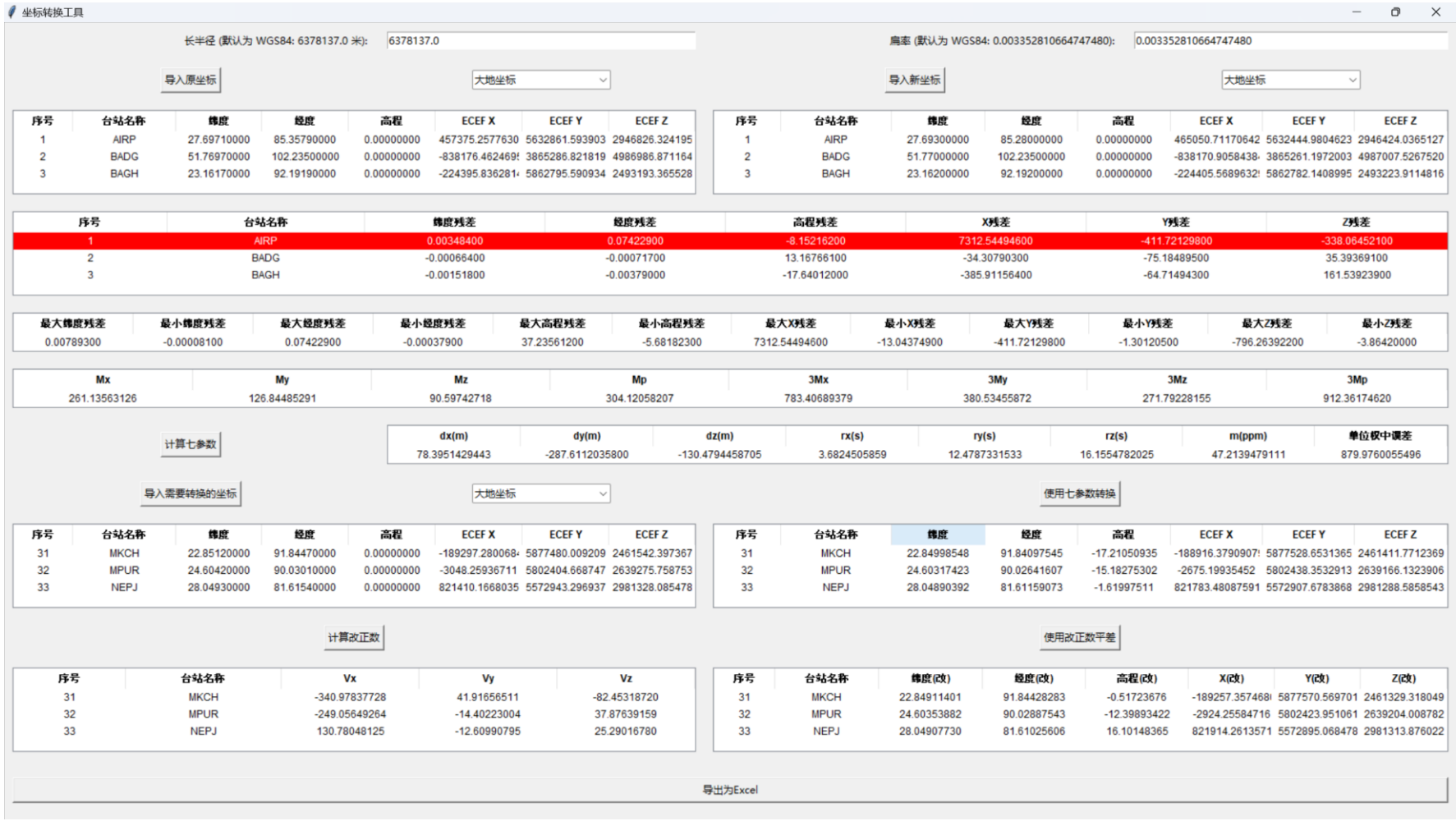
6. 改正数

序号	台站名称	Vx	Vy	Vz
31	$MKCH$	2.52058062	−13.74407993	28.50534267
32	$MPUR$	1.93867548	4.16203791	−11.12098901
33	$NEPJ$	−17.57025869	4.65753764	−5.42590505
34	$NR01$	−0.33226491	0.71143054	−1.97539754
35	$NR03$	−0.24976535	0.74007798	−2.10712951
36	$NR09$	−0.59283128	0.87455337	−1.92141533
37	$NVSK$	−0.34025523	7.64159337	−3.62584776

38	*OSHK*	17.99580525	−4.85117008	1.47564462
39	*PODG*	12.92393263	−20.31485592	22.32760061
40	*QIME*	−0.55420707	0.74035734	−1.91057724
41	*RBIT*	−0.08965760	0.72487401	−2.59857504
42	*RMJT*	−5.81778273	−2.06740274	5.32702490
43	*RMTE*	−1.69696132	−4.65472154	9.72661477
44	*RPUR*	1.98792609	−6.20866282	11.70275225
45	*SHTZ*	21.50863922	−7.20676414	3.14654416
46	*SITA*	4.92001791	−13.51094099	28.39095225
47	*TASH*	18.08145815	−7.82663593	5.00618020
48	*ULAB*	4.42335376	−2.30877363	1.72432831
49	*WARI*	−0.52288671	0.73396986	−1.91513816
50	*XIAN*	−0.88654243	3.91244068	−5.99212324

7. 改正后坐标

序号	台站名称	纬度(改)	经度(改)	高程(改)	X(改)	Y(改)	Z(改)
31	*MKCH*	22.85151666	91.84461421	0.09945865	−189288.04386981	5877466.77330962	2461574.75083013
32	*MPUR*	24.60412530	90.03001467	0.36718710	−3039.61999847	5802408.45198346	2639268.38885993
33	*NEPJ*	28.04928432	81.61551086	−1.49516323	821399.31052144	5572944.38913767	2981325.84894015
34	*NR*01	30.14320983	100.66214603	2.29671031	−1021352.17572306	5424997.10760691	3184113.39534715
35	*NR*03	29.90610908	100.10134469	2.17442865	−970511.76158137	5447681.38463331	3161356.99867573
36	*NR*09	31.44760758	103.16925060	2.76895084	−1240814.06327124	5303037.76281493	3308333.96185323
37	*NVSK*	54.84058503	83.23537914	−0.90959470	433601.60323587	3655489.48046045	5191183.65100822
38	*OSHK*	40.52995609	72.77719825	−1.64966995	1437411.30745779	4636992.21777037	4122887.71233357
39	*PODG*	43.32780541	79.48463008	0.27550872	848081.56542231	4568998.08003561	4354065.08046701
40	*QIME*	31.02290985	101.18494936	2.35083667	−1061185.13648433	5366775.14177900	3268071.67801067
41	*RBIT*	26.84930963	89.39223518	0.32840451	60402.69997104	5694126.87174626	2863328.77589405
42	*RMJT*	27.30518734	86.54998934	0.56077685	341294.87408602	5661175.75208038	2908305.58436417
43	*RMTE*	26.99103209	86.59704615	0.48579321	337588.07080927	5677309.69850303	2877330.39061949
44	*RPUR*	23.03665064	90.76311524	0.36899212	−78213.06344998	5872000.97233047	2480455.15596116
45	*SHTZ*	37.56216243	68.12245231	−0.77252653	1886298.48688682	4697633.30532908	3867034.89945169
46	*SITA*	22.61511502	91.66239024	0.07139746	−170887.86123405	5888151.51816906	2437429.33712628
47	*TASH*	41.32809110	69.29527648	−1.09943286	1695849.76669746	4486817.47107889	4189853.11346428
48	*ULAB*	47.86516136	107.05223293	0.43260270	−1257089.80110297	4098388.76164609	4706831.55893945
49	*WARI*	30.88120988	101.12534887	2.34890135	−1057161.00059726	5375804.28440287	3254598.74061585
50	*XIAN*	34.36865683	109.22144805	3.51916828	−1735083.95630962	4976483.41939949	3580277.01683274


坐标转换工具
长半径 (默认为 WGS84: 6378137.0 米):
6378137.0
扁率 (默认为 WGS84: 0.003352810664747480):
0.003352810664747480
导入原坐标
大地坐标
导入新坐标
大地坐标
序号 台站名称 纬度 经度 高程 ECEF X ECEF Y ECEF Z
1 AIRP 27.69710000 85.35790000 0.00000000 457375.2577630 5632861.593903 2946826.324195
2 BADG 51.76970000 102.23500000 0.00000000 -838176.462469 3865286.821819 4986986.871164
3 BAGH 23.16170000 92.19190000 0.00000000 -224395.836281 5862795.590934 2493193.365528
序号 台站名称 纬度 经度 高程 ECEF X ECEF Y ECEF Z
1 AIRP 27.69300000 85.28000000 0.00000000 465050.71170642 5632444.9804623 2946424.0365127
2 BADG 51.77000000 102.23500000 0.00000000 -838170.9058438 3865261.1972003 4987007.5267520
3 BAGH 23.16200000 92.19200000 0.00000000 -224405.5689632 5862782.1408995 2493223.9114816
序号 台站名称 纬度残差 经度残差 高程残差 X残差 Y残差 Z残差
1 AIRP 0.00348400 0.07422900 -8.15216200 7312.54494600 -411.72129800 -338.06452100
2 BADG -0.00066400 -0.00071700 13.16766100 -34.30790300 -75.18489500 35.39369100
3 BAGH -0.00151800 -0.00379000 -17.64012000 -385.91156400 -64.71494300 161.53923900
最大纬度残差 最小纬度残差 最大经度残差 最小经度残差 最大高程残差 最小高程残差 最大X残差 最小X残差 最大Y残差 最小Y残差 最大Z残差 最小Z残差
0.00789300 -0.00008100 0.07422900 -0.00037900 37.23561200 -5.68182300 7312.54494600 -13.04374900 -411.72129800 -1.30120500 -796.26392200 -3.86420000
Mx My Mz Mp 3Mx 3My 3Mz 3Mp
261.13563126 126.84485291 90.59742718 304.12058207 783.40689379 380.53455872 271.79228155 912.36174620
计算七参数
dx(m) dy(m) dz(m) rx(s) ry(s) rz(s) m(ppm) 单位权中误差
78.3951429443 -287.6112035800 -130.4794458705 3.6824505859 12.4787331533 16.1554782025 47.2139479111 879.9760055496
导入需要转换的坐标
大地坐标
使用七参数转换
序号 台站名称 纬度 经度 高程 ECEF X ECEF Y ECEF Z
31 MKCH 22.85120000 91.84470000 0.00000000 -189297.280068 5877480.009209 2461542.397367
32 MPUR 24.60420000 90.03010000 0.00000000 -3048.25936711 5802404.668747 2639275.758753
33 NEPJ 28.04930000 81.61540000 0.00000000 821410.1668035 5572943.296937 2981328.085478
序号 台站名称 纬度 经度 高程 ECEF X ECEF Y ECEF Z
31 MKCH 22.84998548 91.84097545 -17.21050935 -188916.3790907 5877528.6531365 2461411.7712369
32 MPUR 24.60317423 90.02641607 -15.18275302 -2675.19935452 5802438.3532913 2639166.1323906
33 NEPJ 28.04890392 81.61159073 -1.61997511 821783.48087591 5572907.6783868 2981288.5858543
计算改正数
使用改正数平差
序号 台站名称 Vx Vy Vz
31 MKCH -340.97837728 41.91656511 -82.45318720
32 MPUR -249.05649264 -14.40223004 37.87639159
33 NEPJ 130.78048125 -12.60990795 25.29016780
序号 台站名称 纬度(改) 经度(改) 高程(改) X(改) Y(改) Z(改)
31 MKCH 22.84911401 91.84428283 -0.51723676 -189257.357468 5877570.569701 2461329.318049
32 MPUR 24.60353882 90.02887543 -12.39893422 -2924.25584716 5802423.951061 2639204.008782
33 NEPJ 28.04907730 81.61025606 16.10148365 821914.2613571 5572895.068478 2981313.876022
导出为Excel

附录 4 导入文本文档格式

按照“序号”,“台站名称”,“经度”,“纬度”,“高程”的格式编辑导入文件的格式，其中高程留空默认为 0。

```
1,AIRP,85.3579,27.6971
2,BADG,102.235,51.7697
3,BAGH,92.1919,23.1617
4,BARK,92.3753,22.7256
5,BELT,83.8257,27.4574
6,BESI,84.3797,28.2286
7,BJNM,116.2241,40.2453
8,BRN2,87.2722,26.5197
10,CHNR,91.515,24.1897
11,DAMA,85.1077,27.6081
12,DIGH,92.067,23.2501
13,DMAU,84.2652,27.9734
14,GARM,70.3167,39.0065
15,GHER,84.4097,28.3746
16,IRKT,104.3162,52.219
17,JAFL,92.0191,25.1794
18,JAML,91.2371,25.0004
20,JURI,92.1353,24.4984
21,KAWA,84.1299,27.648
22,KIRT,85.2882,27.6819
23,KIT3,66.8854,39.1348
24,KMTR,78.1992,41.8678
25,KPTI,92.2171,22.4986
26,KRTV,78.6191,50.714
27,KSTU,92.7938,55.9933
28,KUMT,78.1904,41.8635
30,MANM,71.6804,37.5423
```

实际上，代码还扩展了直接输入空间直角坐标系坐标的功能，按照以下格式输入也可导入。

```
“序号”,“台站名称”,“X”,“Y”,“Z”
1,AIRP,457375.257763,5632861.593903,2946826.324196
2,BADG,-838176.462470,3865286.821819,4986986.871164
3,BAGH,-224395.836281,5862795.590935,2493193.365529
4,BARK,-243941.343987,5880857.736891,2448719.042967
5,BELT,609149.933610,5630843.245232,2923282.297744
6,BESI,550767.911225,5596746.985700,2998849.863737
7,BJNM,-2154249.264147,4373368.012022,4098813.069022
```

```
8,BRN2,271785.553321,5704376.397490,2830698.064565
10,CHNR,-153909.192488,5819334.587845,2597463.829523
11,DAMA,482358.837985,5635371.193369,2938090.398652
12,DIGH,-211475.772522,5859415.706982,2502191.326018
13,DMAU,563282.479906,5608889.848733,2973902.210609
14,GARM,1671599.084927,4672886.344104,3992877.786318
15,GHER,547090.089846,5589399.775239,3013096.079526
16,IRKT,-968254.146735,3794129.957898,5017769.053020
17,JAFL,-203489.242793,5772001.436685,2697072.245108
18,JAML,-124875.249211,5782647.216367,2679114.620625
20,JURI,-216375.512993,5803242.423888,2628616.216679
21,KAWA,578246.515799,5624279.518880,2942007.707015
22,KIRT,464291.556429,5633081.083755,2945334.841576
23,KIT3,1944758.279574,4556204.485449,4003935.969476
24,KMTR,972814.225161,4656274.261226,4234680.381346
25,KPTI,-228076.642125,5891166.815772,2425513.747106
26,KRTV,798538.799871,3967136.091729,4913460.394480
27,KSTU,-174274.001046,3571211.177802,5264025.047786
28,KUMT,973594.600700,4656436.792158,4234324.704086
30,MANM,1591559.514826,4806914.826610,3865287.653756
```

附录 5 完整代码

```
import math
import numpy as np
import pandas as pd
# import tkinter.font as tkfont
import tkinter as tk
from tkinter import filedialog, messagebox
from tkinter import ttk  # 用于表格显示
# from openpyxl.utils import get_column_letter
from openpyxl import load_workbook
from openpyxl.styles import Alignment
# from openpyxl.styles import PatternFill

# WGS84 椭球体常量
a = 6378137.0  # 赤道半径，单位：米
f = 1 / 298.257223563  # 扁率
b = a * (1 - f)  # 极半径
e2 = f * (2 - f)  # 第一偏心率的平方

def lat_lon_to_ecef(lat, lon, h=0.0, a=6378137.0, f=1 / 298.257223563):
    """
    将纬度和经度转换为 ECEF 坐标，允许传入自定义的椭球参数。
    :param lat: 纬度
    :param lon: 经度
    :param h: 高程，默认为 0
    :param a: 长半径
    :param f: 扁率
    :return: ECEF 坐标 (x, y, z)
    """
    # 将纬度和经度转换为弧度
    lat_rad = math.radians(lat)
    lon_rad = math.radians(lon)

    # 计算经纬度所在的卯酉圈曲率半径
    e2 = f * (2 - f)  # 第一偏心率的平方
    N = a / math.sqrt(1 - e2 * math.sin(lat_rad) ** 2)

    # 计算 ECEF 坐标
    x = (N + h) * math.cos(lat_rad) * math.cos(lon_rad)
    y = (N + h) * math.cos(lat_rad) * math.sin(lon_rad)
    z = (N * (1 - f) ** 2 + h) * math.sin(lat_rad)
```

```
    return x, y, z

def ecef_to_lat_lon_ecef(x, y, z, a=6378137.0, f=1 / 298.257223563):
    """
    将 ECEF 坐标转换为经纬度和高程，允许传入自定义的椭球参数。
    :param x: ECEF x 坐标
    :param y: ECEF y 坐标
    :param z: ECEF z 坐标
    :param a: 长半径
    :param f: 扁率
    :return: (纬度, 经度, 高程)
    """
    e2 = f * (2 - f)

    # 计算经度
    lon = math.atan2(y, x)  # 经度（弧度）

    # 计算 P
    p = math.sqrt(x ** 2 + y ** 2)

    # 计算初始纬度
    lat = math.atan2(z, (1 - f) * p)  # 初始纬度（弧度）

    # 迭代计算精确的纬度
    for _ in range(100):
        N = a / math.sqrt(1 - e2 * math.sin(lat) ** 2)
        h = p / math.cos(lat) - N
        lat = math.atan2(z + e2 * N * math.sin(lat), p)

    # 转换为度
    lat = math.degrees(lat)
    lon = math.degrees(lon)

    return lat, lon, h

def read_file(file_path):
    """
    从 TXT 文件读取经纬度和高程信息。
    文件格式: 序号, 台站名称, L1(经度), B1(纬度), H(高程, 可选)
    :param file_path: 文件路径
    :return: 站点信息列表
    """
    stations = []
```

```
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                if not line.strip():  # 跳过空行
                    continue
                parts = line.strip().split(',')
                if len(parts) < 4:
                    continue
                num = parts[0]
                name = parts[1]
                lon = float(parts[2])
                lat = float(parts[3])
                h = float(parts[4]) if len(parts) > 4 and parts[4].strip() else 0.0  # 如果没有高程，默认为0
                stations.append({"num": num, "name": name, "lat": lat, "lon": lon, "h": h})
    except Exception as e:
        # 异常处理，显示错误信息
        messagebox.showerror("文件读取错误", f"无法读取文件: {e}")
    return stations

def ext_B(matrix):
    n = matrix.shape[0]
    result = []
    for i in range(n):
        X, Y, Z = matrix[i]
        row = [[1, 0, 0, X, 0, -Z, Y],
               [0, 1, 0, Y, Z, 0, -X],
               [0, 0, 1, Z, -Y, X, 0]]
        result.append(row)
    return np.vstack(result)

def ext_L(matrix):
    transposed_rows = []
    for row in matrix:
        transposed_rows.append(row.reshape(-1, 1))
    stacked_matrix = np.vstack(transposed_rows)
    return stacked_matrix

#七参数计算
def compute_seven_parameters(coords1,coords2):
    Matrix1 = np.array(coords1)
    Matrix2 = np.array(coords2)
    B = -ext_B(Matrix1)
    L = ext_L(Matrix2)
```

```
    P = np.eye(B.shape[0])
    deltaX = -np.linalg.inv(B.T @ P @ B) @ B.T @ P @ L
    dx = deltaX.reshape(-1)[0]
    dy = deltaX.reshape(-1)[1]
    dz = deltaX.reshape(-1)[2]
    s = deltaX.reshape(-1)[3]
    rx = deltaX.reshape(-1)[4] / deltaX.reshape(-1)[3]
    ry = deltaX.reshape(-1)[5] / deltaX.reshape(-1)[3]
    rz = deltaX.reshape(-1)[6] / deltaX.reshape(-1)[3]
    return dx, dy, dz, rx, ry, rz, s

def compute_seven_parameters_sec(coords1,coords2):
    Matrix1 = np.array(coords1)
    Matrix2 = np.array(coords2)
    B = -ext_B(Matrix1)
    L = ext_L(Matrix2)
    P = np.eye(B.shape[0])
    deltaX = -np.linalg.inv(B.T @ P @ B) @ B.T @ P @ L
    dx = deltaX.reshape(-1)[0]
    dy = deltaX.reshape(-1)[1]
    dz = deltaX.reshape(-1)[2]
    s = deltaX.reshape(-1)[3]
    rx = deltaX.reshape(-1)[4] / deltaX.reshape(-1)[3]
    ry = deltaX.reshape(-1)[5] / deltaX.reshape(-1)[3]
    rz = deltaX.reshape(-1)[6] / deltaX.reshape(-1)[3]
    m = (s - 1) * (10**6)
    rx_sec = rx * 648000.0 / math.pi
    ry_sec = ry * 648000.0 / math.pi
    rz_sec = rz * 648000.0 / math.pi
    V = np.dot(B, deltaX) + L
    xx = math.sqrt(np.sum(V.T @ V) / (3 * Matrix1.shape[0] - 7))
    return dx, dy, dz, rx_sec, ry_sec, rz_sec, m, xx

def transform_to_second_coordinate_system(coords, params):
    """
    将坐标从第一个坐标系转换到第二个坐标系。
    :param coords: 第一个坐标系的坐标列表 (x, y, z)
    :param params: 七参数 (dx, dy, dz, rx, ry, rz, s)
    :return: 转换后的坐标列表
    """

    transformed = []
    dx, dy, dz, rx, ry, rz, s = params
```

```
    # 构建旋转矩阵
    for x, y, z in coords:
        R = np.array([
            [np.cos(ry) * np.cos(rz), np.cos(ry) * np.sin(rz), -np.sin(ry)],
            [-np.cos(rx)  *  np.sin(rz)+np.sin(rx)*np.sin(ry)*np.cos(rz),  np.cos(rx)  *  np.cos(rz)  +
np.sin(rx) * np.sin(ry) * np.sin(rz),
             np.sin(rx)*np.cos(ry)],
            [np.sin(rx)  *  np.sin(rz)+np.cos(rx)*np.sin(ry)*np.cos(rz),  -np.sin(rx)  *  np.cos(rz)  +
np.cos(rx) * np.sin(ry) * np.sin(rz),
             np.cos(rx)*np.cos(ry)]
        ])

        # 应用七参数转换
        new_coords = s * R @ np.array([x, y, z]) + np.array([dx, dy, dz])
        transformed.append(new_coords)

    return transformed

def compute_correction_with_seven_params(XYZ1, XYZ2, XYZ_non_public, params):
    """
    计算非公共点改正数，使用七参数将 XYZ1 转换为坐标系 2 后再计算改正数
    :param XYZ1: 公共点坐标系 1 的坐标
    :param XYZ2: 公共点坐标系 2 的坐标
    :param XYZ_non_public: 非公共点坐标
    :param params: 七参数 (dx, dy, dz, rx, ry, rz, s)
    :return: 非公共点的改正数矩阵
    """
    # 使用七参数将 XYZ1 转换为坐标系 2 下的坐标
    transformed_XYZ1 = np.array(transform_to_second_coordinate_system(XYZ1, params))

    # 步骤 1：计算改正数矩阵 V = XYZ2 - (转换后的 XYZ1)
    V = XYZ2 - transformed_XYZ1  # n 行 3 列矩阵

    # 获取公共点数量 n 和非公共点数量 t
    n = XYZ1.shape[0]
    t = XYZ_non_public.shape[0]

    # 初始化用于存储距离矩阵 S 和权重矩阵 P
    S = np.zeros((t, n))
    P = np.zeros((t, n))

    # 步骤 2：计算非公共点与所有公共点之间的距离
```

```
    for i in range(t):
        for j in range(n):
            # 计算距离 S[i, j] = sqrt((X_i - X_j)^2 + (Y_i - Y_j)^2 + (Z_i - Z_j)^2)
            S[i, j] = np.linalg.norm(XYZ_non_public[i] - XYZ1[j])

    # 步骤 3：根据距离矩阵 S 计算权重矩阵 P
    # P = 1 / S^2，如果距离为零，则权重设为无穷大（表示此非公共点与公共点重合）
    with np.errstate(divide='ignore', invalid='ignore'):  # 忽略除以零的警告
        P = 1 / (S**2)
        P[np.isinf(P)] = 0  # 避免无穷大的权重，设为 0

    # 步骤 4：计算非公共点的改正数 Vf
    Vf = np.zeros((t, 3))  # 初始化非公共点改正数矩阵，大小为 t 行 3 列

    for i in range(t):
        # 权重的总和
        P_sum = np.sum(P[i])
        if P_sum != 0:
            # 计算非公共点的改正数 Vf[i] = (P[i] * V 的每一列) / P 的总和
            Vf[i] = np.dot(P[i], V) / P_sum

    return Vf

# Tkinter GUI 部分
class CoordinateConverterApp:

    def export_to_excel(self):
        # 获取界面上所有数据
        params_data = [self.param_tree.item(item)["values"] for item in self.param_tree.get_children()]
        coords1_data = [self.tree1.item(item)["values"] for item in self.tree1.get_children()]
        coords2_data = [self.tree2.item(item)["values"] for item in self.tree2.get_children()]
        diff_data = [
            [item[0], item[1], round(float(item[2]), 6), round(float(item[3]), 6), round(float(item[4]),
6),
             round(float(item[5]), 6), round(float(item[6]), 6), round(float(item[7]), 6),
             ]
            for item in [self.diff_tree.item(item)["values"] for item in self.diff_tree.get_children()]
        ]
        diff_sort_data=[[round(float(item[0]), 6), round(float(item[1]), 6), round(float(item[2]), 6),
round(float(item[3]), 6), round(float(item[4]), 6),
             round(float(item[5]), 6), round(float(item[6]), 6), round(float(item[7]), 6),
             round(float(item[8]), 6), round(float(item[9]), 6), round(float(item[10]), 6),
round(float(item[11]), 6),]
```

```
            for item in
            [self.diff_sort_tree.item(item)["values"] for item in self.diff_sort_tree.get_children()]
        ]

        # 获取 Mp, Mx, My, Mz 表格数据
        residual_summary_data = [
            [round(float(item[0]),    6),    round(float(item[1]),    6),    round(float(item[2]),    6),
round(float(item[3]), 6), round(float(item[4]), 6), round(float(item[5]), 6), round(float(item[6]), 6),
round(float(item[7]), 6)]
            for item in
            [self.residual_summary_tree.item(item)["values"]            for            item            in
self.residual_summary_tree.get_children()]
        ]

        coords_to_transform_data = [self.transform_input_tree.item(item)["values"] for item in
                                    self.transform_input_tree.get_children()]
        transformed_data = [self.transformed_output_tree.item(item)["values"] for item in
                            self.transformed_output_tree.get_children()]

        # 新增：获取改正数表格数据
        correction_data = [
            self.correction_tree.item(item)["values"]
            for item in self.correction_tree.get_children()
        ]

        # 新增：获取改正后数据表格数据
        corrected_data = [
            self.corrected_tree.item(item)["values"]
            for item in self.corrected_tree.get_children()
        ]

        if not params_data or not coords1_data or not coords2_data or not transformed_data or not
coords_to_transform_data:
            messagebox.showerror("错误", "没有可导出的数据")
            return

        # 数据字典，用于将不同表格的数据写入不同的 Excel Sheet
        data_dict = {
            "七参数": pd.DataFrame(params_data, columns=["dx(m)", "dy(m)", "dz(m)", "rx(s)", "ry(s)",
"rz(s)", "m(ppm)" ,"单位权中误差"]),
            "原坐标": pd.DataFrame(coords1_data,
                                columns=["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF
Z"]),
```

```
            "新坐标": pd.DataFrame(coords2_data,
                                columns=["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF
Z"]),
            "残差": pd.DataFrame(diff_data,
                               columns=["序号", "台站名称", "纬度残差", "经度残差", "高程残差", "X 残差", "Y 残差", "Z
残差",]),
            "残差最值": pd.DataFrame(diff_sort_data,
                                  columns=["最大纬度残差", "最小纬度残差", "最大经度残差", "最小经度残差", "最大高程
残差", "最小高程残差",
                                           "最大 X 残差", "最小 X 残差", "最大 Y 残差", "最小 Y 残差", "最大 Z 残差", "最小
Z 残差"]),
            "点位中误差": pd.DataFrame(residual_summary_data, columns=["Mx", "My", "Mz", "Mp", "3Mx", "3My",
"3Mz", "3Mp"]),  # 新增的表格数据
            "转换前坐标": pd.DataFrame(coords_to_transform_data,
                                    columns=["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y",
                                             "ECEF Z"]),
            "转换后坐标": pd.DataFrame(transformed_data,
                                    columns=["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y",
                                             "ECEF Z"]),
            # 新增：改正数和改正后数据
            "改正数": pd.DataFrame(correction_data, columns=["序号", "台站名称", "Vx", "Vy", "Vz"]),
            "改正后坐标": pd.DataFrame(corrected_data,
                                    columns=["序号", "台站名称", "纬度(改)", "经度(改)", "高程(改)", "X(改)",
                                             "Y(改)", "Z(改)"])
        }

        # 保存为 Excel 文件
        file_path = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel files",
"*.xlsx")])
        if not file_path:
            return

        with pd.ExcelWriter(file_path, engine="openpyxl") as writer:
            # 创建汇总表
            summary_data = []

            # 添加七参数到汇总表
            summary_data.append(["七参数"])
            summary_data.append(["dx(m)", "dy(m)", "dz(m)", "rx(s)", "ry(s)", "rz(s)", "m(ppm)" ,"单位权中
误差"])
            summary_data.extend(params_data)
            summary_data.append([])  # 空行分隔
```

```
            # 添加原坐标系数据到汇总表
            summary_data.append(["原坐标"])
            summary_data.append(["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF Z"])
            summary_data.extend(coords1_data)
            summary_data.append([])  # 空行分隔

            # 添加新坐标系数据到汇总表
            summary_data.append(["新坐标"])
            summary_data.append(["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF Z"])
            summary_data.extend(coords2_data)
            summary_data.append([])  # 空行分隔

            # 添加差值数据到汇总表
            summary_data.append(["残差"])
            summary_data.append(["序号", "台站名称", "纬度残差", "经度残差", "高程残差", "X 残差", "Y 残差", "Z 残
差"])
            summary_data.extend(diff_data)
            summary_data.append([])  # 空行分隔

            summary_data.append(["残差最值"])
            summary_data.append(["最大纬度残差", "最小纬度残差", "最大经度残差", "最小经度残差", "最大高程残差", "
最小高程残差", "最大 X 残差", "最小 X 残差", "最大 Y 残差", "最小 Y 残差", "最大 Z 残差", "最小 Z 残差"])
            summary_data.extend(diff_sort_data)
            summary_data.append([])  # 空行分隔

            # 添加 Mp, Mx, My, Mz 到汇总表
            summary_data.append(["点位中误差"])
            summary_data.append(["Mx", "My", "Mz", "Mp","3Mx", "3My", "3Mz", "3Mp"])
            summary_data.extend(residual_summary_data)
            summary_data.append([])  # 空行分隔

            # 添加“读取坐标系 1 的文件”数据到汇总表
            summary_data.append(["转换前坐标"])
            summary_data.append(["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF Z"])
            summary_data.extend(coords_to_transform_data)
            summary_data.append([])  # 空行分隔

            # 添加转换后坐标数据到汇总表
            summary_data.append(["转换后坐标"])
            summary_data.append(["序号", "台站名称", "纬度", "经度", "高程", "ECEF X", "ECEF Y", "ECEF Z"])
            summary_data.extend(transformed_data)
            summary_data.append([])  # 空行分隔
```

```
            summary_data.append(["改正数"])
            summary_data.append(["序号", "台站名称", "Vx", "Vy", "Vz"])
            summary_data.extend(correction_data)
            summary_data.append([])  # 空行分隔

            summary_data.append(["改正后坐标"])
            summary_data.append(["序号", "台站名称", "纬度(改)", "经度(改)", "高程(改)", "X(改)", "Y(改)",
"Z(改)"])
            summary_data.extend(corrected_data)



            # 写入汇总表
            summary_df = pd.DataFrame(summary_data)
            summary_df.to_excel(writer, sheet_name="汇总", index=False, header=False)

            # 写入单独的工作表
            for sheet_name, df in data_dict.items():
                df.to_excel(writer, sheet_name=sheet_name, index=False)

        # 自动调整列宽并设置单元格居中对齐
        workbook = load_workbook(file_path)
        for sheet_name in workbook.sheetnames:
            sheet = workbook[sheet_name]
            for column_cells in sheet.columns:
                max_length = 0
                column = column_cells[0].column_letter  # 获取列字母
                for cell in column_cells:
                    try:
                        if cell.value:
                            max_length = max(max_length, len(str(cell.value)))
                            cell.alignment = Alignment(horizontal="center", vertical="center")  # 设置居中对
齐
                    except:
                        pass
                adjusted_width = (max_length + 2)  # 加 2 留出额外的空白
                sheet.column_dimensions[column].width = adjusted_width

            # 获取 3Mx、3My 和 3Mz 的值
            mx3, my3, mz3 = 0, 0, 0
            if self.residual_summary_tree.get_children():
                mx3, my3, mz3 = (

float(self.residual_summary_tree.item(self.residual_summary_tree.get_children()[0])["values"][4]),
```

```
float(self.residual_summary_tree.item(self.residual_summary_tree.get_children()[0])["values"][5]),

float(self.residual_summary_tree.item(self.residual_summary_tree.get_children()[0])["values"][6])
            )



        workbook.save(file_path)
        messagebox.showinfo("导出成功", f"数据成功导出到 {file_path}")

    def __init__(self, root):
        self.root = root
        self.root.title("坐标转换工具")

        # 设置窗口初始大小
        self.root.geometry("1600x900")
        # 允许窗口横向缩放
        self.root.columnconfigure(0, weight=1)
        self.root.columnconfigure(1, weight=1)
        self.root.columnconfigure(2, weight=1)
        self.root.columnconfigure(3, weight=1)

        # 允许窗口纵向扩展
        self.root.rowconfigure(0, weight=0)
        self.root.rowconfigure(1, weight=0)
        self.root.rowconfigure(2, weight=1)
        self.root.rowconfigure(3, weight=1)
        self.root.rowconfigure(4, weight=0)
        self.root.rowconfigure(5, weight=0)
        self.root.rowconfigure(6, weight=0)
        self.root.rowconfigure(7, weight=0)
        self.root.rowconfigure(8, weight=1)
        self.root.rowconfigure(10, weight=1)
        self.root.rowconfigure(9, weight=0)

        # 创建 Treeview 样式
        style = ttk.Style()
        style.configure("Treeview.Heading", font=("Arial", 9, "bold"))  # 调整表头字体为 9 号
        style.configure("Treeview", rowheight=20, font=("Arial", 9))  # 调整行高和内容字体为 9 号

        # 第 1 行：长半径 和 扁率输入框
        tk.Label(root, text="长半径 (默认为 WGS84: 6378137.0 米):").grid(row=0, column=0, sticky='e',
padx=10, pady=10)
```

```
        self.a_entry = tk.Entry(root)
        self.a_entry.grid(row=0, column=1, padx=10, pady=10, sticky='ew')
        self.a_entry.insert(0, "6378137.0")

        tk.Label(root, text="扁率 (默认为 WGS84: 0.0033528106647474 80):").grid(row=0, column=2, sticky='e',
padx=10,
                                                                    pady=10)
        self.f_entry = tk.Entry(root)
        self.f_entry.grid(row=0, column=3, padx=10, pady=10, sticky='ew')
        self.f_entry.insert(0, "0.0033528106647474 80")

        # 第 2 行：导入原坐标 和 导入新坐标按钮
        self.file_button_1 = tk.Button(root, text="导入原坐标", command=self.load_file_1)
        self.file_button_1.grid(row=1, column=0, padx=10, pady=10)

        self.coord_type_var_1 = tk.StringVar()
        self.coord_type_combo_1 = ttk.Combobox(root, textvariable=self.coord_type_var_1, state="readonly")
        self.coord_type_combo_1['values'] = ("大地坐标", "空间直角坐标")
        self.coord_type_combo_1.grid(row=1, column=1, padx=10, pady=10)
        self.coord_type_combo_1.current(0)

        self.file_button_2 = tk.Button(root, text="导入新坐标", command=self.load_file_2)
        self.file_button_2.grid(row=1, column=2, padx=10, pady=10)

        self.coord_type_var_2 = tk.StringVar()
        self.coord_type_combo_2 = ttk.Combobox(root, textvariable=self.coord_type_var_2, state="readonly")
        self.coord_type_combo_2['values'] = ("大地坐标", "空间直角坐标")
        self.coord_type_combo_2.grid(row=1, column=3, padx=10, pady=10)
        self.coord_type_combo_2.current(0)

        # 第 3 行：原坐标和新坐标表格
        self.tree1 = ttk.Treeview(root, columns=("num", "name", "lat", "lon", "h", "x", "y", "z"),
show="headings",
                                  height=10)
        self.tree1.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")
        self.tree1.heading("num", text="序号")
        self.tree1.heading("name", text="台站名称")
        self.tree1.heading("lat", text="纬度")
        self.tree1.heading("lon", text="经度")
        self.tree1.heading("h", text="高程")
        self.tree1.heading("x", text="ECEF X")
        self.tree1.heading("y", text="ECEF Y")
        self.tree1.heading("z", text="ECEF Z")
```

```
        self.tree1.column("num", width=50, anchor='center')
        self.tree1.column("name", width=100, anchor='center')
        self.tree1.column("lat", width=80, anchor='center')
        self.tree1.column("lon", width=80, anchor='center')
        self.tree1.column("h", width=80, anchor='center')
        self.tree1.column("x", width=80, anchor='center')
        self.tree1.column("y", width=80, anchor='center')
        self.tree1.column("z", width=80, anchor='center')

        self.tree2 = ttk.Treeview(root, columns=("num", "name", "lat", "lon", "h", "x", "y", "z"),
show="headings",
                                  height=10)
        self.tree2.grid(row=2, column=2,columnspan=2, padx=10, pady=10, sticky="nsew")
        self.tree2.heading("num", text="序号")
        self.tree2.heading("name", text="台站名称")
        self.tree2.heading("lat", text="纬度")
        self.tree2.heading("lon", text="经度")
        self.tree2.heading("h", text="高程")
        self.tree2.heading("x", text="ECEF X")
        self.tree2.heading("y", text="ECEF Y")
        self.tree2.heading("z", text="ECEF Z")
        self.tree2.column("num", width=50, anchor='center')
        self.tree2.column("name", width=100, anchor='center')
        self.tree2.column("lat", width=80, anchor='center')
        self.tree2.column("lon", width=80, anchor='center')
        self.tree2.column("h", width=80, anchor='center')
        self.tree2.column("x", width=80, anchor='center')
        self.tree2.column("y", width=80, anchor='center')
        self.tree2.column("z", width=80, anchor='center')

        # 第 4 行：残差表格，横跨两列
        # 修改残差表格，增加最大值和最小值的列
        self.diff_tree = ttk.Treeview(root, columns=(
            "num", "name", "lat_diff", "lon_diff", "h_diff", "x_diff", "y_diff", "z_diff"),
                                      show="headings", height=10)

        self.diff_tree.grid(row=3, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

        # 现有列
        self.diff_tree.heading("num", text="序号")
        self.diff_tree.heading("name", text="台站名称")
        self.diff_tree.heading("lat_diff", text="纬度残差")
        self.diff_tree.heading("lon_diff", text="经度残差")
```

```
        self.diff_tree.heading("h_diff", text="高程残差")
        self.diff_tree.heading("x_diff", text="X 残差")
        self.diff_tree.heading("y_diff", text="Y 残差")
        self.diff_tree.heading("z_diff", text="Z 残差")



        self.diff_tree.column("num", width=50, anchor='center')
        self.diff_tree.column("name", width=100, anchor='center')
        self.diff_tree.column("lat_diff", width=80, anchor='center')
        self.diff_tree.column("lon_diff", width=80, anchor='center')
        self.diff_tree.column("h_diff", width=80, anchor='center')
        self.diff_tree.column("x_diff", width=80, anchor='center')
        self.diff_tree.column("y_diff", width=80, anchor='center')
        self.diff_tree.column("z_diff", width=80, anchor='center')




        self.diff_sort_tree = ttk.Treeview(root, columns=("lat_max", "lat_min", "lon_max", "lon_min",
"h_max", "h_min", "x_max", "x_min", "y_max", "y_min", "z_max",
            "z_min"), show="headings", height=1)
        self.diff_sort_tree.grid(row=4, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

        # 新增的最大值最小值列
        self.diff_sort_tree.heading("lat_max", text="最大纬度残差")
        self.diff_sort_tree.heading("lat_min", text="最小纬度残差")
        self.diff_sort_tree.heading("lon_max", text="最大经度残差")
        self.diff_sort_tree.heading("lon_min", text="最小经度残差")
        self.diff_sort_tree.heading("h_max", text="最大高程残差")
        self.diff_sort_tree.heading("h_min", text="最小高程残差")
        self.diff_sort_tree.heading("x_max", text="最大 X 残差")
        self.diff_sort_tree.heading("x_min", text="最小 X 残差")
        self.diff_sort_tree.heading("y_max", text="最大 Y 残差")
        self.diff_sort_tree.heading("y_min", text="最小 Y 残差")
        self.diff_sort_tree.heading("z_max", text="最大 Z 残差")
        self.diff_sort_tree.heading("z_min", text="最小 Z 残差")

        self.diff_sort_tree.column("lat_max", width=80, anchor='center')
        self.diff_sort_tree.column("lat_min", width=80, anchor='center')
        self.diff_sort_tree.column("lon_max", width=80, anchor='center')
        self.diff_sort_tree.column("lon_min", width=80, anchor='center')
        self.diff_sort_tree.column("h_max", width=80, anchor='center')
        self.diff_sort_tree.column("h_min", width=80, anchor='center')
        self.diff_sort_tree.column("x_max", width=80, anchor='center')
```

```
        self.diff_sort_tree.column("x_min", width=80, anchor='center')
        self.diff_sort_tree.column("y_max", width=80, anchor='center')
        self.diff_sort_tree.column("y_min", width=80, anchor='center')
        self.diff_sort_tree.column("z_max", width=80, anchor='center')
        self.diff_sort_tree.column("z_min", width=80, anchor='center')



        # 在第 4 行初始化残差表格下方添加新的表格
        self.residual_summary_tree = ttk.Treeview(root, columns=("Mx", "My", "Mz", "Mp","3Mx", "3My",
"3Mz", "3Mp"), show="headings", height=1)
        self.residual_summary_tree.grid(row=5, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")
        self.residual_summary_tree.heading("Mx", text="Mx")
        self.residual_summary_tree.heading("My", text="My")
        self.residual_summary_tree.heading("Mz", text="Mz")
        self.residual_summary_tree.heading("Mp", text="Mp")
        self.residual_summary_tree.heading("3Mx", text="3Mx")
        self.residual_summary_tree.heading("3My", text="3My")
        self.residual_summary_tree.heading("3Mz", text="3Mz")
        self.residual_summary_tree.heading("3Mp", text="3Mp")
        self.residual_summary_tree.column("Mx", width=80, anchor='center')
        self.residual_summary_tree.column("My", width=80, anchor='center')
        self.residual_summary_tree.column("Mz", width=80, anchor='center')
        self.residual_summary_tree.column("Mp", width=80, anchor='center')
        self.residual_summary_tree.column("3Mx", width=80, anchor='center')
        self.residual_summary_tree.column("3My", width=80, anchor='center')
        self.residual_summary_tree.column("3Mz", width=80, anchor='center')
        self.residual_summary_tree.column("3Mp", width=80, anchor='center')

        # 第 5 行：计算七参数 和 七参数表格
        self.compute_button = tk.Button(root, text="计算七参数", command=self.compute_seven_parameters)
        self.compute_button.grid(row=6, column=0, padx=10, pady=10)

        self.param_tree = ttk.Treeview(root, columns=("dx", "dy", "dz", "rx", "ry", "rz", "m", "xx"),
show="headings",
                                       height=1)
        self.param_tree.grid(row=6, column=1, columnspan=3, padx=10, pady=10, sticky="nsew")
        self.param_tree.heading("dx", text="dx(m)")
        self.param_tree.heading("dy", text="dy(m)")
        self.param_tree.heading("dz", text="dz(m)")
        self.param_tree.heading("rx", text="rx(s)")
        self.param_tree.heading("ry", text="ry(s)")
        self.param_tree.heading("rz", text="rz(s)")
```

```
        self.param_tree.heading("m", text="m(ppm)")
        self.param_tree.heading("xx", text="单位权中误差")
        self.param_tree.column("dx", width=100, anchor='center')
        self.param_tree.column("dy", width=100, anchor='center')
        self.param_tree.column("dz", width=100, anchor='center')
        self.param_tree.column("rx", width=100, anchor='center')
        self.param_tree.column("ry", width=100, anchor='center')
        self.param_tree.column("rz", width=100, anchor='center')
        self.param_tree.column("m", width=100, anchor='center')
        self.param_tree.column("xx", width=100, anchor='center')

        # 第 6 行：导入需要转换的坐标 和 使用七参数转换
        self.file_button_3 = tk.Button(root, text="导入需要转换的坐标", command=self.load_file_to_transform)
        self.file_button_3.grid(row=7, column=0, padx=10, pady=10)

        self.coord_type_var_3 = tk.StringVar()
        self.coord_type_combo_3 = ttk.Combobox(root, textvariable=self.coord_type_var_3, state="readonly")
        self.coord_type_combo_3['values'] = ("大地坐标", "空间直角坐标")
        self.coord_type_combo_3.grid(row=7, column=1, padx=10, pady=10)
        self.coord_type_combo_3.current(0)

        self.compute_button_2 = tk.Button(root, text="使用七参数转换", command=self.transform_coords)
        self.compute_button_2.grid(row=7, column=2, columnspan=2, padx=10, pady=10)

        # 第 7 行：转换前后表格
        self.transform_input_tree = ttk.Treeview(root, columns=("num", "name", "lat", "lon", "h", "x",
"y", "z"),
                                                 show="headings", height=10)
        self.transform_input_tree.grid(row=8, column=0, columnspan=2,padx=10, pady=10, sticky="nsew")
        self.transform_input_tree.heading("num", text="序号")
        self.transform_input_tree.heading("name", text="台站名称")
        self.transform_input_tree.heading("lat", text="纬度")
        self.transform_input_tree.heading("lon", text="经度")
        self.transform_input_tree.heading("h", text="高程")
        self.transform_input_tree.heading("x", text="ECEF X")
        self.transform_input_tree.heading("y", text="ECEF Y")
        self.transform_input_tree.heading("z", text="ECEF Z")
        self.transform_input_tree.column("num", width=50, anchor='center')
        self.transform_input_tree.column("name", width=100, anchor='center')
        self.transform_input_tree.column("lat", width=80, anchor='center')
        self.transform_input_tree.column("lon", width=80, anchor='center')
        self.transform_input_tree.column("h", width=80, anchor='center')
        self.transform_input_tree.column("x", width=80, anchor='center')
```

```
        self.transform_input_tree.column("y", width=80, anchor='center')
        self.transform_input_tree.column("z", width=80, anchor='center')

        self.transformed_output_tree = ttk.Treeview(root, columns=("num", "name", "lat", "lon", "h", "x",
"y", "z"),
                                                    show="headings", height=10)
        self.transformed_output_tree.grid(row=8, column=2, columnspan=2, padx=10, pady=10, sticky="nsew")
        self.transformed_output_tree.heading("num", text="序号")
        self.transformed_output_tree.heading("name", text="台站名称")
        self.transformed_output_tree.heading("lat", text="纬度")
        self.transformed_output_tree.heading("lon", text="经度")
        self.transformed_output_tree.heading("h", text="高程")
        self.transformed_output_tree.heading("x", text="ECEF X")
        self.transformed_output_tree.heading("y", text="ECEF Y")
        self.transformed_output_tree.heading("z", text="ECEF Z")
        self.transformed_output_tree.column("num", width=50, anchor='center')
        self.transformed_output_tree.column("name", width=100, anchor='center')
        self.transformed_output_tree.column("lat", width=80, anchor='center')
        self.transformed_output_tree.column("lon", width=80, anchor='center')
        self.transformed_output_tree.column("h", width=80, anchor='center')
        self.transformed_output_tree.column("x", width=80, anchor='center')
        self.transformed_output_tree.column("y", width=80, anchor='center')
        self.transformed_output_tree.column("z", width=80, anchor='center')

        # 新增第 9 行：改正数表格
        self.correction_tree = ttk.Treeview(root, columns=("num", "name", "Vx", "Vy", "Vz"),
show="headings", height=10)
        self.correction_tree.grid(row=10, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")
        self.correction_tree.heading("num", text="序号")
        self.correction_tree.heading("name", text="台站名称")
        self.correction_tree.heading("Vx", text="Vx")
        self.correction_tree.heading("Vy", text="Vy")
        self.correction_tree.heading("Vz", text="Vz")

        self.correction_tree.column("num", width=50, anchor='center')
        self.correction_tree.column("name", width=100, anchor='center')
        self.correction_tree.column("Vx", width=80, anchor='center')
        self.correction_tree.column("Vy", width=80, anchor='center')
        self.correction_tree.column("Vz", width=80, anchor='center')

        # 新增第 10 行：改正后结果表格
        self.corrected_tree = ttk.Treeview(root, columns=(
        "num", "name", "lat_corrected", "lon_corrected", "h_corrected", "X_corrected", "Y_corrected",
```

```
"Z_corrected"),
                                    show="headings", height=10)
        self.corrected_tree.grid(row=10, column=2, columnspan=2, padx=10, pady=10, sticky="nsew")
        self.corrected_tree.heading("num", text="序号")
        self.corrected_tree.heading("name", text="台站名称")
        self.corrected_tree.heading("lat_corrected", text="纬度(改)")
        self.corrected_tree.heading("lon_corrected", text="经度(改)")
        self.corrected_tree.heading("h_corrected", text="高程(改)")
        self.corrected_tree.heading("X_corrected", text="X(改)")
        self.corrected_tree.heading("Y_corrected", text="Y(改)")
        self.corrected_tree.heading("Z_corrected", text="Z(改)")

        self.corrected_tree.column("num", width=50, anchor='center')
        self.corrected_tree.column("name", width=100, anchor='center')
        self.corrected_tree.column("lat_corrected", width=100, anchor='center')
        self.corrected_tree.column("lon_corrected", width=100, anchor='center')
        self.corrected_tree.column("h_corrected", width=100, anchor='center')
        self.corrected_tree.column("X_corrected", width=80, anchor='center')
        self.corrected_tree.column("Y_corrected", width=80, anchor='center')
        self.corrected_tree.column("Z_corrected", width=80, anchor='center')

        # 修改计算改正数按钮的 command
        self.compute_button_x = tk.Button(root, text="计算改正数", command=self.compute_correction)
        self.compute_button_x.grid(row=9, column=0, columnspan=2, padx=10, pady=10)

        # 修改使用改正数平差按钮的 command
        self.compute_button_y    =    tk.Button(root,    text="   使   用   改   正   数   平   差   ",
command=self.apply_correction_adjustment)
        self.compute_button_y.grid(row=9, column=2, columnspan=2, padx=10, pady=10)

        # 第 8 行：导出为 Excel 按钮，居中
        self.export_button = tk.Button(root, text="导出为 Excel", command=self.export_to_excel)
        self.export_button.grid(row=11, column=0, columnspan=4, padx=10, pady=20, sticky="ew")

        # 坐标系数据初始化
        self.coords1 = []
        self.coords2 = []
        self.coords_to_transform = []

    def check_and_calculate_correction(self):
        """
        检查是否已导入所有必要文件并计算非公共点改正数
        """
```

```
        # 确保公共点 1、公共点 2 和非公共点数据都已导入
        if self.coords1 and self.coords2 and self.coords_to_transform:
            self.compute_and_update_non_public_correction()

    def load_file_1(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
        if file_path:
            stations = read_file(file_path)
            if not stations:
                messagebox.showerror("错误", "文件内容无效或空")
                return

            # 清空之前的数据
            for row in self.tree1.get_children():
                self.tree1.delete(row)
            self.coords1 = []  # 重置原坐标列表
            self.params = None  # 重置七参数

            # 判断选择的数据类型
            coord_type = self.coord_type_var_1.get()

            # 获取用户输入的椭球参数
            a = float(self.a_entry.get())
            f = float(self.f_entry.get())

            for station in stations:
                if coord_type == "大地坐标":
                    lat = station['lat']
                    lon = station['lon']
                    h = station['h']
                    x, y, z = lat_lon_to_ecef(lat, lon, h, a, f)
                    self.coords1.append((x, y, z))
                    self.tree1.insert("", "end", values=(
                    station['num'], station['name'], f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x:.8f}",
f"{y:.8f}",
                    f"{z:.8f}"))
                else:
                    x, y, z = station['lon'], station['lat'], station['h']
                    lat, lon, h = ecef_to_lat_lon_ecef(x, y, z)
                    self.coords1.append((x, y, z))
                    self.tree1.insert("", "end", values=(
                    station['num'], station['name'], f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x:.8f}",
f"{y:.8f}",
```

```
                f"{z:.8f}"))

        if self.coords2:
            self.compute_and_update_diff()

    def load_file_2(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
        if file_path:
            stations = read_file(file_path)
            if not stations:
                messagebox.showerror("错误", "文件内容无效或空")
                return

            # 清空之前的数据
            for row in self.tree2.get_children():
                self.tree2.delete(row)
            self.coords2 = []  # 重置新坐标列表
            self.params = None  # 重置七参数

            coord_type = self.coord_type_var_2.get()
            a = float(self.a_entry.get())
            f = float(self.f_entry.get())

            for station in stations:
                if coord_type == "大地坐标":
                    lat = station['lat']
                    lon = station['lon']
                    h = station['h']
                    x, y, z = lat_lon_to_ecef(lat, lon, h, a, f)
                    self.coords2.append((x, y, z))
                    self.tree2.insert("", "end", values=(
                    station['num'], station['name'], f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x:.8f}",
f"{y:.8f}",
                    f"{z:.8f}"))
                else:
                    x, y, z = station['lon'], station['lat'], station['h']
                    lat, lon, h = ecef_to_lat_lon_ecef(x, y, z)
                    self.coords2.append((x, y, z))
                    self.tree2.insert("", "end", values=(
                    station['num'], station['name'], f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x:.8f}",
f"{y:.8f}",
                    f"{z:.8f}"))
```

```
        if self.coords1:
            self.compute_and_update_diff()

def compute_and_update_diff(self):
    """
    计算差值并更新到差值表格，包括纬度、经度、高程和 ECEF 的差值
    """
    # 清空差值表格
    for row in self.diff_tree.get_children():
        self.diff_tree.delete(row)

    # 确保七参数已经计算
    if not hasattr(self, 'params') or self.params is None:
        if not self.coords1 or not self.coords2:
            messagebox.showerror("错误", "请先加载两个坐标系的文件")
            return

        if len(self.coords1) != len(self.coords2):
            messagebox.showerror("错误", "两个坐标系的点数量不一致")
            return

        # 计算七参数并存储在 self.params 中
        self.params = compute_seven_parameters(self.coords1, self.coords2)

    # 进行坐标转换，使用计算出的 self.params
    for row1, row2 in zip(self.tree1.get_children(), self.tree2.get_children()):
        # 获取 tree1 和 tree2 的每一行值
        values1 = self.tree1.item(row1)["values"]
        values2 = self.tree2.item(row2)["values"]

        # 提取 ECEF 坐标并进行七参数转换
        x1, y1, z1 = float(values1[5]), float(values1[6]), float(values1[7])
        transformed_coords = transform_to_second_coordinate_system([(x1, y1, z1)], self.params)[0]

        # 提取坐标系 2 的 ECEF 坐标
        x2, y2, z2 = float(values2[5]), float(values2[6]), float(values2[7])

        # 计算转换后的差值（X, Y, Z）
        x_diff = round(x2 - transformed_coords[0], 6)
        y_diff = round(y2 - transformed_coords[1], 6)
        z_diff = round(z2 - transformed_coords[2], 6)

        # 转换为纬度、经度和高程
```

```
            lat1,   lon1,   h1   =   ecef_to_lat_lon_ecef(transformed_coords[0],   transformed_coords[1],
transformed_coords[2])
            lat2, lon2, h2 = float(values2[2]), float(values2[3]), float(values2[4])

            # 计算纬度、经度、高程的差值
            lat_diff = round(lat1 - lat2, 6)
            lon_diff = round(lon1 - lon2, 6)
            h_diff = round(h1 - h2, 6)

            # 插入差值到新的表格（包含纬度、经度、高程、X, Y, Z 差值）
            self.diff_tree.insert("", "end", values=(
                values1[0],    values1[1],    f"{lat_diff:.8f}",    f"{lon_diff:.8f}",    f"{h_diff:.8f}",
f"{x_diff:.8f}", f"{y_diff:.8f}", f"{z_diff:.8f}",
            ))



            # 清空残差表格
            for row in self.diff_sort_tree.get_children():
                self.diff_sort_tree.delete(row)

            lat_diffs, lon_diffs, h_diffs = [], [], []
            x_diffs, y_diffs, z_diffs = [], [], []

            for row1, row2 in zip(self.tree1.get_children(), self.tree2.get_children()):
                # coord_type1 = self.coord_type_var_1.get()
                # coord_type2 = self.coord_type_var_2.get()
                #
                # if coord_type1 == "大地坐标":
                #     a = float(self.a_entry.get())
                #     f = float(self.f_entry.get())
                #     values1 = self.tree1.item(row1)["values"]
                #     x, y, z = lat_lon_to_ecef(float(values1[2]), float(values1[3]), float(values1[4]), a,
f)
                #     values2 = (values1[0], values1[1], values1[2], values1[3], values1[4], x, y, z)
                values1 = self.tree1.item(row1)["values"]
                values2 = self.tree2.item(row2)["values"]

                # 提取 ECEF 坐标并进行转换
                x1, y1, z1 = float(values1[5]), float(values1[6]), float(values1[7])
                x2, y2, z2 = float(values2[5]), float(values2[6]), float(values2[7])
                # 计算差值
                transformed_coords = transform_to_second_coordinate_system([(x1, y1, z1)], self.params)[0]
                # 计算转换后的差值（X, Y, Z）
```

```
            x_diff = round(x2 - transformed_coords[0], 6)
            y_diff = round(y2 - transformed_coords[1], 6)
            z_diff = round(z2 - transformed_coords[2], 6)

            # 转换为纬度、经度和高程
            # 转换为纬度、经度和高程
            lat1, lon1, h1 = ecef_to_lat_lon_ecef(transformed_coords[0], transformed_coords[1],
                                                  transformed_coords[2])
            lat2, lon2, h2 = float(values2[2]), float(values2[3]), float(values2[4])

            lat_diff = round(lat1 - lat2, 6)
            lon_diff = round(lon1 - lon2, 6)
            h_diff = round(h1 - h2, 6)

            # 记录每次的差值
            lat_diffs.append(lat_diff)
            lon_diffs.append(lon_diff)
            h_diffs.append(h_diff)
            x_diffs.append(x_diff)
            y_diffs.append(y_diff)
            z_diffs.append(z_diff)

        # 计算最大值和最小值
        # lat_max, lat_min = max(lat_diffs, key=abs), min(lat_diffs, key=abs)
        # lon_max, lon_min = max(lon_diffs, key=abs), min(lon_diffs, key=abs)
        # h_max, h_min = max(h_diffs, key=abs), min(h_diffs, key=abs)
        # x_max, x_min = max(x_diffs, key=abs), min(x_diffs, key=abs)
        # y_max, y_min = max(y_diffs, key=abs), min(y_diffs, key=abs)
        # z_max, z_min = max(z_diffs, key=abs), min(z_diffs, key=abs)
        lat_max, lat_min = f"{max(lat_diffs, key=abs):.8f}", f"{min(lat_diffs, key=abs):.8f}"
        lon_max, lon_min = f"{max(lon_diffs, key=abs):.8f}", f"{min(lon_diffs, key=abs):.8f}"
        h_max, h_min = f"{max(h_diffs, key=abs):.8f}", f"{min(h_diffs, key=abs):.8f}"
        x_max, x_min = f"{max(x_diffs, key=abs):.8f}", f"{min(x_diffs, key=abs):.8f}"
        y_max, y_min = f"{max(y_diffs, key=abs):.8f}", f"{min(y_diffs, key=abs):.8f}"
        z_max, z_min = f"{max(z_diffs, key=abs):.8f}", f"{min(z_diffs, key=abs):.8f}"


        # 插入最大值最小值到表格
        self.diff_sort_tree.insert("", "end", values=(
            lat_max, lat_min, lon_max, lon_min, h_max, h_min, x_max, x_min, y_max, y_min, z_max, z_min
        ))

        # 初始化残差平方和
```

```
            vv_x = vv_y = vv_z = 0
            n = len(self.tree1.get_children())

            for row1, row2 in zip(self.tree1.get_children(), self.tree2.get_children()):
                ...
                # 计算平方和
                vv_x += x_diff ** 2
                vv_y += y_diff ** 2
                vv_z += z_diff ** 2

            # 计算 Mx, My, Mz, Mp
            Mx = math.sqrt(vv_x / (n - 1))
            My = math.sqrt(vv_y / (n - 1))
            Mz = math.sqrt(vv_z / (n - 1))
            Mp = math.sqrt(Mx ** 2 + My ** 2 + Mz ** 2)
            Mx3 = 3 * Mx
            My3 = 3 * My
            Mz3 = 3 * Mz
            Mp3 = 3 * Mp
            # # 计算 Mx, My, Mz, Mp
            # Mx = math.sqrt(vv_x / (3*n - 7))
            # My = math.sqrt(vv_y / (3*n - 7))
            # Mz = math.sqrt(vv_z / (3*n - 7))
            # Mp = math.sqrt(Mx ** 2 + My ** 2 + Mz ** 2)
            # Mx3 = 3 * Mx
            # My3 = 3 * My
            # Mz3 = 3 * Mz
            # Mp3 = 3 * Mp

            # 插入结果到表格
            for row in self.residual_summary_tree.get_children():
                self.residual_summary_tree.delete(row)

            self.residual_summary_tree.insert("", "end", values=(f"{Mx:.8f}", f"{My:.8f}", f"{Mz:.8f}",
f"{Mp:.8f}",f"{Mx3:.8f}", f"{My3:.8f}", f"{Mz3:.8f}", f"{Mp3:.8f}"))

        mx3, my3, mz3 = float(
            self.residual_summary_tree.item(self.residual_summary_tree.get_children()[0])["values"][4]),
\

float(self.residual_summary_tree.item(self.residual_summary_tree.get_children()[0])["values"][5]), \

float(self.residual_summary_tree.item(self.residual_summary_tree.get_children()[0])["values"][6])
```

```
        # 检查差值是否超出 3Mx、3My、3Mz，并标记界面上的超出值
        # 初始化标志变量，确保警告窗口只弹出一次
        warning_shown = False

        for row in self.diff_tree.get_children():
            values = self.diff_tree.item(row)["values"]
            x_diff, y_diff, z_diff = float(values[5]), float(values[6]), float(values[7])

            # 检查是否有异常值
            if abs(x_diff) > mx3 or abs(y_diff) > my3 or abs(z_diff) > mz3:
                # 如果没有弹出过警告窗口，则弹出
                if not warning_shown:
                    messagebox.showwarning("数据异常", "残差表中存在异常值")
                    warning_shown = True  # 设置标志为 True，防止再次弹出窗口

                # 标记并高亮异常值
                for i, diff in enumerate([x_diff, y_diff, z_diff]):
                    if abs(diff) > [mx3, my3, mz3][i]:
                        self.diff_tree.tag_configure("highlight", background="red", foreground="white")
                        self.diff_tree.item(row, tags="highlight")



    def compute_seven_parameters(self):

        if not self.coords1 or not self.coords2:
            messagebox.showerror("错误", "请先加载两个坐标系的文件")
            return

        if len(self.coords1) != len(self.coords2):
            messagebox.showerror("错误", "两个坐标系的点数量不一致")
            return

        seven_params = compute_seven_parameters_sec(self.coords1, self.coords2)

        for row in self.param_tree.get_children():
            self.param_tree.delete(row)

        # 插入七参数结果
        self.param_tree.insert("", "end",
                               values=(f"{seven_params[0]:.10f}",                f"{seven_params[1]:.10f}",
f"{seven_params[2]:.10f}",
```

```
                                f"{seven_params[3]:.10f}",                 f"{seven_params[4]:.10f}",
f"{seven_params[5]:.10f}",
                                f"{seven_params[6]:.10f}", f"{seven_params[7]:.10f}"))




    def load_file_to_transform(self):
        """
        加载需要转换的 txt 文件，读取经纬度和高程或 XYZ，并显示在表格中。
        """
        file_path = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
        if file_path:
            stations = read_file(file_path)
            if not stations:
                messagebox.showerror("错误", "文件内容无效或空")
                return

            for row in self.transform_input_tree.get_children():
                self.transform_input_tree.delete(row)

            # 判断选择的数据类型
            coord_type = self.coord_type_var_3.get()

            # 获取用户输入的椭球参数
            a = float(self.a_entry.get())
            f = float(self.f_entry.get())

            self.coords_to_transform = []
            for station in stations:
                if coord_type == "大地坐标":
                    # 大地坐标转换为 ECEF
                    lat = station['lat']
                    lon = station['lon']
                    h = station['h']
                    x, y, z = lat_lon_to_ecef(lat, lon, h, a, f)
                    self.coords_to_transform.append((station['num'], station['name'], x, y, z))
                    # 插入经纬度和计算的 ECEF 坐标
                    self.transform_input_tree.insert("", "end", values=(
                        station['num'], station['name'], f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x:.8f}",
f"{y:.8f}",
                        f"{z:.8f}"
                    ))
```

```
            else:
                # 空间直角坐标直接使用
                x, y, z = station['lon'], station['lat'], station['h']
                lat, lon, h = ecef_to_lat_lon_ecef(x, y, z)  # ECEF 转为经纬度
                self.coords_to_transform.append((station['num'], station['name'], x, y, z))
                # 插入 XYZ 和计算的经纬度
                self.transform_input_tree.insert("", "end", values=(
                    station['num'], station['name'], f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x:.8f}",
f"{y:.8f}",
                    f"{z:.8f}"
                ))
        # self.check_and_calculate_correction()

    def transform_coords(self):
        """
        将加载的坐标通过七参数转换到新的坐标系，并将 ECEF 坐标转换为经纬度和高程。
        """

        if not self.coords_to_transform:
            messagebox.showerror("错误", "请先加载需要转换的文件")
            return

        if not self.param_tree.get_children():
            messagebox.showerror("错误", "请先计算七参数")
            return

        # 获取七参数
        params = compute_seven_parameters(self.coords1, self.coords2)

        # 转换坐标
        transformed_coords = transform_to_second_coordinate_system(
            [(x, y, z) for _, _, x, y, z in self.coords_to_transform], params
        )

        # 清除转换后的表格
        for row in self.transformed_output_tree.get_children():
            self.transformed_output_tree.delete(row)

        # 将转换后的坐标以及台站名称插入到表格中
        for (num, name, x_old, y_old, z_old), (x_new, y_new, z_new) in zip(self.coords_to_transform,
                                                                           transformed_coords):
            # ECEF 转为 经纬度和高程
            lat, lon, h = ecef_to_lat_lon_ecef(x_new, y_new, z_new)
```

```
            # 插入转换后的数据，包括台站名称
            self.transformed_output_tree.insert("", "end", values=(
                num, name, f"{lat:.8f}", f"{lon:.8f}", f"{h:.8f}", f"{x_new:.8f}", f"{y_new:.8f}",
f"{z_new:.8f}"
            ))

    def compute_and_update_non_public_correction(self):
        """
        计算非公共点的改正数，并更新结果到表格中
        """
        if not self.coords1 or not self.coords2 or not self.coords_to_transform:
            messagebox.showerror("错误", "请先加载公共点和非公共点坐标文件")
            return

        # 确保七参数已经计算
        if not hasattr(self, 'params') or self.params is None:
            messagebox.showerror("错误", "请先计算七参数")
            return

        # 将坐标转换为 numpy 数组
        XYZ1 = np.array(self.coords1)
        XYZ2 = np.array(self.coords2)
        XYZ_non_public = np.array([(x, y, z) for _, _, x, y, z in self.coords_to_transform])

        # Step 1: 使用七参数转换非公共点坐标 (coords_to_transform)
        #  transformed_non_public_points  =  self.transform_to_second_coordinate_system2(XYZ_non_public,
self.params)
        # 获取七参数
        params = compute_seven_parameters(self.coords1, self.coords2)
        # 转换坐标
        transformed_non_public_points = transform_to_second_coordinate_system(
            [(x, y, z) for _, _, x, y, z in self.coords_to_transform], params
        )
        # Step 2: 计算公共点间的改正数
        corrections = compute_correction_with_seven_params(XYZ1, XYZ2, XYZ_non_public, self.params)

        # 清空现有的改正数表格数据
        for row in self.correction_tree.get_children():
            self.correction_tree.delete(row)

        # 清空改正后结果表格数据
        for row in self.corrected_tree.get_children():
```

```
            self.corrected_tree.delete(row)

        # Step 3: 处理计算结果，插入到表格中
        for i, ((num, name, x, y, z), transformed_point, correction) in enumerate(
                zip(self.coords_to_transform, transformed_non_public_points, corrections)):
            # 插入改正数表格
            self.correction_tree.insert("", "end", values=(
                num, name, f"{correction[0]:.8f}", f"{correction[1]:.8f}", f"{correction[2]:.8f}"
            ))

            # Step 4: 计算改正后的 X、Y、Z (七参数转换后的值 + 改正数)
            X_corrected = transformed_point[0] + correction[0]
            Y_corrected = transformed_point[1] + correction[1]
            Z_corrected = transformed_point[2] + correction[2]

            # 将 X_corrected, Y_corrected, Z_corrected 转换回大地坐标系
            lat_corrected, lon_corrected, h_corrected = ecef_to_lat_lon_ecef(X_corrected, Y_corrected,
Z_corrected)

            # 插入改正后结果表格
            self.corrected_tree.insert("", "end", values=(
                num, name, f"{lat_corrected:.8f}", f"{lon_corrected:.8f}", f"{h_corrected:.8f}",
f"{X_corrected:.8f}",
                f"{Y_corrected:.8f}", f"{Z_corrected:.8f}"
            ))

    def compute_correction(self):
        if not self.coords1 or not self.coords2 or not self.coords_to_transform:
            messagebox.showerror("错误", "请先加载公共点和非公共点坐标文件")
            return

        self.params = compute_seven_parameters(self.coords1, self.coords2)

        XYZ1 = np.array(self.coords1)
        XYZ2 = np.array(self.coords2)
        XYZ_non_public = np.array([(x, y, z) for _, _, x, y, z in self.coords_to_transform])

        corrections = compute_correction_with_seven_params(XYZ1, XYZ2, XYZ_non_public, self.params)

        # 清空改正数表格
        for row in self.correction_tree.get_children():
            self.correction_tree.delete(row)
```

```
        for i, (num, name, correction) in enumerate(
                zip([s[0] for s in self.coords_to_transform], [s[1] for s in self.coords_to_transform],
corrections)):
            self.correction_tree.insert("", "end", values=(
            num, name, f"{correction[0]:.8f}", f"{correction[1]:.8f}", f"{correction[2]:.8f}"))

    def apply_correction_adjustment(self):
        if not self.coords1 or not self.coords2 or not self.coords_to_transform:
            messagebox.showerror("错误", "请先加载公共点和非公共点坐标文件")
            return

        if self.params is None:
            self.params = compute_seven_parameters(self.coords1, self.coords2)

        transformed_non_public_points = transform_to_second_coordinate_system(
            [(x, y, z) for _, _, x, y, z in self.coords_to_transform], self.params
        )
        corrections = compute_correction_with_seven_params(
            np.array(self.coords1), np.array(self.coords2),
            np.array([(x, y, z) for _, _, x, y, z in self.coords_to_transform]), self.params
        )

        # 清空改正后结果表格
        for row in self.corrected_tree.get_children():
            self.corrected_tree.delete(row)

        for (num, name, _, _, _), transformed, correction in zip(self.coords_to_transform,
                                                                 transformed_non_public_points, corrections):
            X_corrected = transformed[0] + correction[0]
            Y_corrected = transformed[1] + correction[1]
            Z_corrected = transformed[2] + correction[2]
            lat_corrected, lon_corrected, h_corrected = ecef_to_lat_lon_ecef(X_corrected, Y_corrected,
Z_corrected)
            self.corrected_tree.insert("", "end", values=(
            num,    name,    f"{lat_corrected:.8f}",    f"{lon_corrected:.8f}",    f"{h_corrected:.8f}",
f"{X_corrected:.8f}",
            f"{Y_corrected:.8f}", f"{Z_corrected:.8f}"))

if __name__ == "__main__":
    root = tk.Tk()
    app = CoordinateConverterApp(root)
    root.mainloop()
```