



《摄影测量学》 课间实习报告

学 院： 遥感信息工程学院

班 级： 20F10

学 号： 2020302131201

姓 名： 常耀文

指导教师： 刘亚文

2022 年 10 月 22 日

目录

1. 实习概述.....	3
1.1 单像空间后方交会概述.....	3
1.2 单像空间后方交会实习目的.....	3
1.3 实习所使用的编程语言与头文件使用.....	3
1.3.1 编程语言.....	3
1.3.1 使用的头文件.....	4
1.4 编程 IDE.....	4
2. 实习任务与实习数据.....	5
2.1 实习任务.....	5
2.2 实习数据.....	5
2.2.1 实习数据来源.....	5
3. 实习原理与平差模型.....	5
3.1 实习原理.....	5
3.2 平差模型.....	6
3.2.1 误差模型推导.....	6
3.2.2 误差方程系数推导.....	7
4. 实习思路与流程框图建立.....	8
4.1 实习思路.....	8
4.2 实习流程框图.....	9
5. 核心代码与中间结果展示.....	10
5.1 核心代码.....	10
5.2 中间结果展示.....	14
6. 实习结果与分析.....	17
6.1 实验数据.....	17
6.1 实验结果.....	17
6.3 实验分析.....	18
6.3.1 外方位元素计算结果分析.....	18
6.3.2 精度分析.....	18
6.3.3 迭代收敛速度分析.....	19
7. 实习心得.....	23

1. 实习概述

1.1 单像空间后方交会概述

在已知地面上若干点的地面坐标后，反求该相应摄影光束的外方位元素，即 X_s , Y_s , Z_s 和 ϕ , ω , κ 是一个单像空间后方交会问题。当用作摄影机的检定或其他较精确的测定时，还可借以同时求出摄影的内方位元素 x_0 , y_0 及 f ，这种原理的应用有以下几种情况：

- 一、在有些双像解析法运算或区域网解析加密运算中用作其中的一部分程序
- 二、在双像解析法运算或解析法加密运算以后，有时候根据所求得的地面点坐标值再反求各摄影光束的外方位元素，供上其他的生产工序的使用
- 三、通过恒星摄影或实验场摄影等措施确定摄影物镜的畸变差与内方位元素
- 四、利用恒星背景确定导弹的弹道或者人造卫星的空间方向

在之前的计算机视觉学习中，曾利用单像空间后方交会解算结果来进行双像解析算法的运行，因此对于单像空间后方交会有着一定的认识。

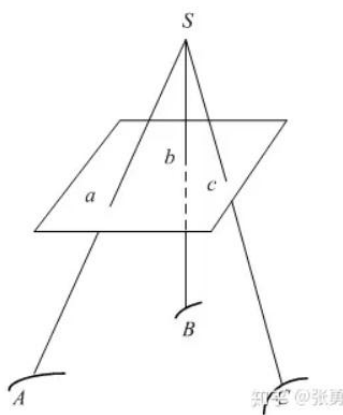


Fig1.单像空间后方交会概述

1.2 单像空间后方交会实习目的

编写单像空间后方交会程序，藉此检测学生对于所学知识的了解程度，同时在实践动手的过程中锻炼代码能力，提升编程技能

1.3 实习所使用的编程语言与头文件使用

1.3.1 编程语言

由于本次实习要求使用 C 或 C++ 编程语言进行实现相关的代码的内容，因此采用的主

要编程语言是 C 与 C++

1.3.1 使用的头文件

出于锻炼代码与深度学习矩阵与单像空间后方交会的目的考虑,本次实验引入的头文件并不包含矩阵运算的头文件,所有与矩阵相关的操作均由个人实现完成,因此实现的难度也大于直接引用<Eigen>库实现矩阵的相关操作,主要使用头文件都是在 C 或 C++编程中较为常见的头文件,头文件的使用见下图:

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<math.h>
#include<fstream>
#include<string>
#include<vector>
#include<malloc.h>
using namespace std;
```

Fig2.程序使用的头文件

1.4 编程 IDE

为了实现编程任务,本次实习采用的编译器是 Microsoft Visual Studio 2022, Microsoft Visual Studio (简称 VS) 是美国微软公司的开发工具包系列产品。VS 是一个基本完整的开发工具集,它包括了整个软件生命周期所需要的大部分工具,如 UML 工具、代码管控工具、集成开发环境(IDE)等等。所写的目标代码适用于微软支持的所有平台,包括 Microsoft Windows、Windows Mobile、Windows CE、.NET Framework、.NET Compact Framework 和 Microsoft Silverlight 及 Windows Phone。



Fig3.Microsoft Visual Studio 2022

2. 实习任务与实习数据

2.1 实习任务

本次实习的任务是利用课本的数据与单张相片空间后方交会的知识,通过共线方程模型来计算本次数据下的外方位元素,并且对于实习结果进行精度评定

2.2 实习数据

2.2.1 实习数据来源

本次实习数据来源于课本 44 页的第 27 题,并且补充了 ($m=500000$) 的条件,初始已知的值由 x_0, y_0, f, m, X, Y, Z (其中 X, Y, Z 是已知的控制点坐标,共有 4 组控制点坐标), x_0, y_0 为 0

	影像坐标		地面坐标		
	$x(mm)$	$y(mm)$	$X(m)$	$Y(m)$	$Z(m)$
1	-86.15	-68.99	36589.41	25273.32	2195.17
2	-53.40	82.21	37631.08	31324.51	728.69
3	-14.78	-76.63	39100.97	24934.98	2386.50
4	10.46	64.43	40426.54	30319.81	757.31

Fig4.实习数据图表

3. 实习原理与平差模型

3.1 实习原理

本次实习以共线条件方程作为数学模型

设 S 为摄影中心,在世界坐标系下的坐标为 (X_s, Y_s, Z_s) ; M 为空间一点,在世界坐标系下的坐标为 (X, Y, Z) , m 是 M 在影像上的构象,其像平面和像空间辅助坐标分别为 $(x, y, -f)$, (X_m, Y_m, Z_m) , 此时可知 S, m, M 三点共线。可得 (式 1)

$$\frac{X_m}{X-X_S} = \frac{Y_m}{Y-Y_S} = \frac{Z_m}{Z-Z_S} = \lambda$$

(式 1)

再根据像平面坐标和像空间辅助坐标的关系有 (式 2)

$$\begin{bmatrix} x \\ y \\ -f \end{bmatrix} = R^T \begin{bmatrix} X_m \\ Y_m \\ Z_m \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} * \begin{bmatrix} X_m \\ Y_m \\ Z_m \end{bmatrix}$$

(式 2)

由上两式可解得共线方程式为

$$x - x_0 = -f \frac{a_1(X-X_S) + b_1(Y-Y_S) + c_1(Z-Z_S)}{a_3(X-X_S) + b_3(Y-Y_S) + c_3(Z-Z_S)}$$

$$y - y_0 = -f \frac{a_2(X-X_S) + b_2(Y-Y_S) + c_2(Z-Z_S)}{a_3(X-X_S) + b_3(Y-Y_S) + c_3(Z-Z_S)}$$

(式 3)

其中, x_0 、 y_0 、 f 是影像内方位元素; 表示像平面中心坐标和摄像机主距。

3.2 平差模型

在实验原理中, 我们已经建立了有关本次实习的平差模型, 即利用间接平差的方法建立最小二乘法的共线方程模型, 下面为了实习的进行有条不紊, 将对误差模型进行推导

3.2.1 误差模型推导

由共线条件方程, 以像点坐标为观测值, 可得:

$$\begin{cases} x = (x_0 - f \frac{\bar{X}}{\bar{Z}}) \\ y = (y_0 - f \frac{\bar{Y}}{\bar{Z}}) \end{cases}$$

$$\begin{cases} x + v_x = (x_0 - f \frac{\bar{X}}{\bar{Z}}) + d_x = (x) + d_x \\ y + v_y = (y_0 - f \frac{\bar{Y}}{\bar{Z}}) + d_y = (y) + d_y \end{cases}$$

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \end{bmatrix} - \begin{bmatrix} x - (x) \\ y - (y) \end{bmatrix}$$

根据间接平差方法列误差方程：

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial X_s} & \frac{\partial x}{\partial Y_s} & \frac{\partial x}{\partial Z_s} & \frac{\partial x}{\partial \varphi} & \frac{\partial x}{\partial \omega} & \frac{\partial x}{\partial \kappa} \\ \frac{\partial y}{\partial X_s} & \frac{\partial y}{\partial Y_s} & \frac{\partial y}{\partial Z_s} & \frac{\partial y}{\partial \varphi} & \frac{\partial y}{\partial \omega} & \frac{\partial y}{\partial \kappa} \end{bmatrix} \begin{bmatrix} \Delta X_s \\ \Delta Y_s \\ \Delta Z_s \\ \Delta \varphi \\ \Delta \omega \\ \Delta \kappa \end{bmatrix} - \begin{bmatrix} x - (\hat{x}) \\ y - (\hat{y}) \end{bmatrix}$$

$$V = Ax - L$$

最小化 $V^T P V$ 可以得到如下的参数估计公式：

$$A^T A x = A^T L$$

$$x = (A^T A)^{-1} A^T L$$

3.2.2 误差方程系数推导

根据式 3 以及旋转矩阵可得到：

$$\frac{\partial x}{\partial X_s} = \frac{1}{z} [a_1 f + a_3 (x - x_0)]$$

$$\frac{\partial x}{\partial Y_s} = \frac{1}{z} [b_1 f + b_3 (x - x_0)]$$

$$\frac{\partial x}{\partial Z_s} = \frac{1}{z} [c_1 f + c_3 (x - x_0)]$$

$$\frac{\partial y}{\partial X_s} = \frac{1}{z} [a_2 f + a_3 (y - y_0)]$$

$$\frac{\partial y}{\partial Y_s} = \frac{1}{z} [b_2 f + b_3 (y - y_0)]$$

$$\frac{\partial y}{\partial Z_s} = \frac{1}{z} [c_2 f + c_3 (y - y_0)]$$

$$\frac{\partial x}{\partial \varphi} = (y - y_0) \sin \omega - \left[\frac{(x - x_0)}{f} [(x - x_0) \cos \kappa - (y - y_0) \sin \kappa] + f \cos \kappa \right] \cos \omega$$

$$\frac{\partial x}{\partial \omega} = -f \sin \kappa - \frac{x - x_0}{f} [(x - x_0) \sin \kappa + (y - y_0) \cos \kappa]$$

$$\frac{\partial x}{\partial \kappa} = y - y_0$$

$$\frac{\partial y}{\partial \varphi} = -(x - x_0) \sin \omega - \left[\frac{y - y_0}{f} [(x - x_0) \cos \kappa - (y - y_0) \sin \kappa] - f \sin \kappa \right] \cos \omega$$

$$\frac{\partial y}{\partial \omega} = -f \cos \kappa - \frac{y - y_0}{f} [(x - x_0) \sin \kappa + (y - y_0) \cos \kappa]$$

$$\frac{\partial y}{\partial \kappa} = -(x - x_0)$$

4. 实习思路与流程框图建立

4.1 实习思路

本次实习的思路我采用了 UML 图的形式展示，UML 图直观易懂，为本次实习绘制的 UML 图如下：

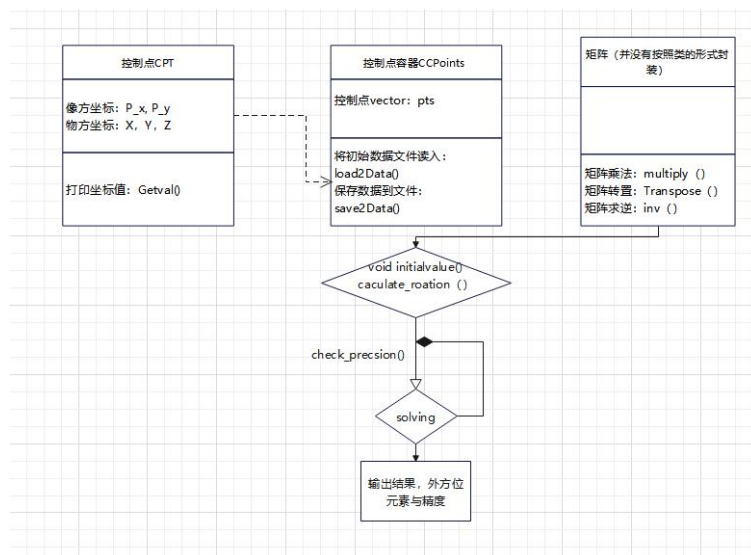


Fig5.实习 UML 图

本次的 UML 图体现了我对于实习操作的全部架构过程，在指导后续编程实践中具有重要的意义，其中主要设计了两个类控制点类 CPT，控制点容器类 CCPoints，其中矩阵因为时间原因，设计类过于复杂，没有完成相应的封装，但是本次实习需要用到的矩阵运算方式，如转置，求逆，乘法均实践完成，最后利用相应的设计函数初始化数据，开始迭代，在迭代的过程中使用了 check_precision() 函数控制迭代条件（线元素改正数 < 10e-3, 角元素改正数 < 10e-6），最后在迭代满足的情况下退出循环，输出相应的结果

4.2 实习流程框图

在建立实习的流程框图之前，我们需要明确空间后方交会的计算过程：

(1) 获取已知数据。计算比例尺 $1/m$ ，平均摄影距离 x_0 、航空摄影的航高 y_0 、内方位元素 f ；获取控制点的空间坐标 X_t, Y_t, Z_t 。

(2) 量测控制点的像点坐标并进行必要的影像坐标系误差改正，得到像点坐标。

(3) 确定未知数的初始值。在竖直航空摄影且地面控制点大体对称分布的情况下，按如下方法确定初始值：

$$\begin{aligned}Z_S^0 &= H = m * f \\X_S^0 &= \frac{1}{n} \sum_{i=1}^n X_{ti} \\Y_S^0 &= \frac{1}{n} \sum_{i=1}^n Y_{ti} \\ \varphi^0 &= \omega^0 = 0\end{aligned}$$

(4) 计算旋转矩阵 R ，利用角元素近似值按空间坐标系旋转变换公式计算方向余弦值，组成 R 阵

(5) 逐点计算像点坐标的近似值，利用未知数的近似值按照共线条件公式计算控制点像点坐标的近似值 $(x), (y)$

(6) 逐点计算误差方程式的系数与常数项，组成误差方程式

(7) 计算法方程的系数矩阵 ATA 与常数项 ATL ，组成法方程式

(8) 解求外方位元素，根据法方程，按公式求解外方位元素的改正数，并与相应的近似值求和，得到外方位元素新的近似值

(9) 检查计算是否收敛，将所求的外方位元素的改正数与规定的限差比较，通常对于 ϕ, ω, κ 的改正数给予限差，这个限差通常为 $0.1''$ ，当三个改正数与三个线元素改正数都小于限差时，迭代结束，否则用近似值重复 (4) ~ (8) 步骤的计算，直到满足要求为止

(10) 按照上述方法所求得影像外方位元素的精度可以通过法方程式中未知数的系数矩阵 $(ATA)^{-1}$ 来解求，此时视为不等精度观测，设单位权中误差为 m ，第 i 个元素的中误差为

$$m_i = \sqrt{Q_{ii}} m_0$$

但本例为等精度观测。

有 n 个点参与后方交会时，单位权中误差为：

$$m_0 = \pm \frac{\sqrt{[vv]}}{2n - 6}$$

当有了详细的计算流程后，建立流程框图编写程序就简单了，下方的图片是利用共线方程进行单像空间后方交会的计算机程序框图：

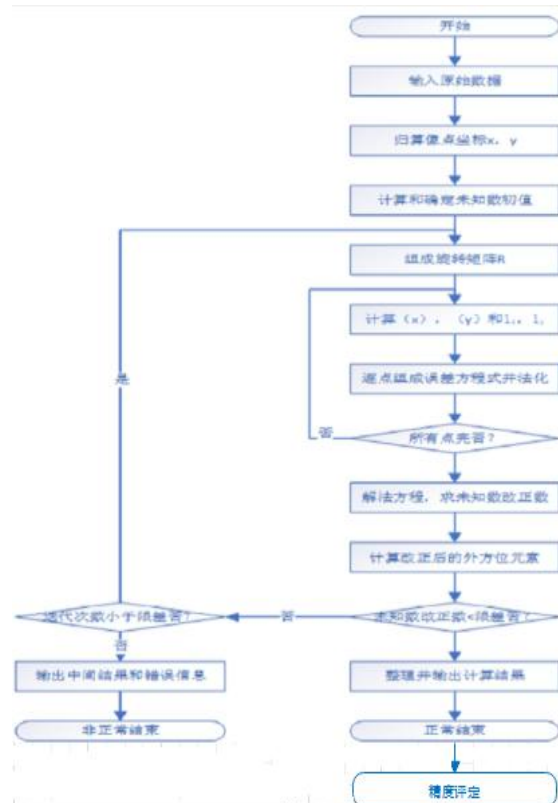


Fig6.实习流程框图

5. 核心代码与中间结果展示

5.1 核心代码

按照 UML 图，先构建了两个类——控制点类 CPt，控制点容器类 CCPoints，并为每类定义了相应的处理方法，代码展示如下：

```
//定义控制点类
class CPt {
public:
    CPt() { p_x = p_y = 0; X = Y = Z = 0; };
    ~CPt() {}
    double p_x, p_y, X, Y, Z;
    string p_name;
    //打印得到的点的数据
    void Getval() {
        cout << p_name << " " << p_x << " " << p_y << " " << X << " " << Y << " " << Z << endl;
    }
};
```

Fig7.控制点类

```
//对控制点整体建类进行操作
class CPoints {
public:
    CPoints() { num = 0; };
    ~CPoints() {}
    int num;
    //存储控制点的容器
    vector<CPt> pts;
    //从文件中读取控制点坐标
    bool load2Data(const char* strFn) {
        ifstream ifile;
        ifile.open(strFn, ios::in);
        if (ifile.is_open()) { //判断文件是否成功打开
            int i = 0;
            CPt pt;
            ifile >> num;
            pts.clear();
            for (int i = 0; i < num; i++) {
                ifile >> pt.p_name >> pt.p_x >> pt.p_y >> pt.X >> pt.Y >> pt.Z;
                pt.p_x /= 1000; //将像点单位全部转化为m
                pt.p_y /= 1000;
                pts.push_back(pt);
            }
            ifile.close();
            return true;
        }
        else {
            return false;
        }
    }
    //将控制点坐标写入到文件
    bool save2Data(const char* strFn)
    {
        ofstream ofile;
        ofile.open(strFn, ios::out);
        if (!ofile.is_open()) { //判断文件是否成功打开
```

Fig7.控制点容器类

定义了控制点类后，对于控制点相关数据处理封装性较好，并且为了数据输入规范，建立了两个文件，一个是内方位元素 m，f 文件，一个是控制点数据文件，建立文件如下：

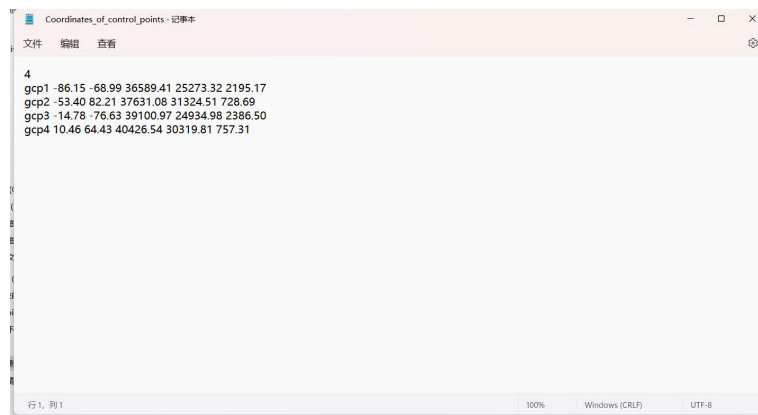


Fig8.控制点文件

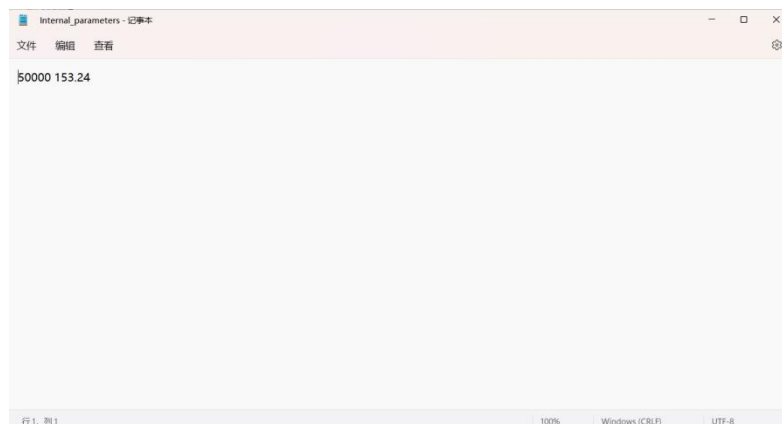


Fig9.内方位元素文件

在处理矩阵运算时，将矩阵用一维数组来表示，有效地简化了指针的使用与计算的复杂程度(在使用指向指针的指针来指向二维数组存在问题，因此查询网上的资料后改换了方法)，矩阵相关运算的代码较为复杂，而且不是我们本次实习的研究重点，因此在这里不放上详细的矩阵运算代码，只附上相关的函数，如下图（图中注明了相关的函数功能）：

```
//矩阵转置运算
void Transpose(double *matrix, double TR_matrix[], int m, int n) {
    for(int i=0; i<m; i++)
```

Fig10.矩阵转置函数

```
//然后实现了我们的矩阵相乘。
void multiply(double *A, double *B, double *MultiplyR, int rows1, int cols1, int rows2, int cols2) {
    for (int i=0; i<rows1; i++) {
```

Fig11.矩阵相乘函数

```
//矩阵求逆运算，返回值是输入矩阵的逆矩阵
double* inv(double* a, int n)
{
    int* is, * js, i, j, k, l, u, v;
    double d, p;
    is = (int*)malloc(n * sizeof(int));
    js = (int*)malloc(n * sizeof(int));
    for (k = 0; k <= n - 1; k++)
```

Fig11.矩阵求逆函数

矩阵运算模块实现后，结合实习内容，定义了求解旋转矩阵的函数与对于外方位元素初始化的函数，具体实现过程如下图：

```
//定义初值计算函数,通过初始函数计算外方位元素
void initialvalue(CCPoints PT, const char*strFn, double*data) {
    double Xs0=0, Ys0=0, Zs0=0;
    double phi=0, omega=0, kappa=0;
    double f = 0, m = 0;
    int tnum = PT.num;
    double Xssum = 0;
    double Yssum = 0;
    double Zssum = 0;
    for (int i = 0; i < tnum; i++) {
        Xssum += PT.pts.at(i).X;
        Yssum += PT.pts.at(i).Y;
        Zssum += PT.pts.at(i).Z;
    }
    ifstream ifile;
    ifile.open(strFn, ios::in);
    if (ifile.is_open()) {
        ifile >> m >> f;
        ifile.close();
    }
    f = f/1000.0;
    double ztemp = 0;
    ztemp = m * f;
    Xs0 = Xssum / tnum;
    Ys0 = Yssum / tnum;
    Zs0 = Zssum / tnum + ztemp;
    data[0] = Xs0;
    data[1] = Ys0;
    data[2] = Zs0;
    data[3] = phi;
    data[4] = omega;
    data[5] = kappa;
    data[6] = m;
    data[7] = f;
}
```

Fig11.初始化外方位元素函数

这里使用了一个 data 数组，将全部的外方位元素与内方位元素全部存储到了 data 数组中，方便后续的调用与输出，也是之后编程的基础，在求解旋转矩阵的过程中就使用到了这

个思路，如下图：

```
//计算旋转矩阵
void caculate_roation(double* og_data, double* roation) { //传入初值向量
    //利用旋转矩阵公式计算旋转矩阵
    roation[0] = cos(og_data[3]) * cos(og_data[5]) - sin(og_data[3]) * sin(og_data[4]) * sin(og_data[5]);
    roation[1] = (-1.0) * cos(og_data[3]) * sin(og_data[5]) - sin(og_data[3]) * sin(og_data[4]) * cos(og_data[5]);
    roation[2] = (-1.0) * sin(og_data[3]) * cos(og_data[4]);
    roation[3] = cos(og_data[4]) * sin(og_data[5]);
    roation[4] = cos(og_data[4]) * cos(og_data[5]);
    roation[5] = -sin(og_data[4]);
    roation[6] = sin(og_data[3]) * cos(og_data[5]) + cos(og_data[3]) * sin(og_data[4]) * sin(og_data[5]);
    roation[7] = (-1.0) * sin(og_data[3]) * sin(og_data[5]) + cos(og_data[3]) * sin(og_data[4]) * cos(og_data[5]);
    roation[8] = cos(og_data[3]) * cos(og_data[4]);
    return;
}
```

Fig12.求解旋转矩阵

在所有的预备工作做完后，就需要执行实习流程中的求解误差方程，严格按照实习思路中的后方空间交会的计算流程执行，代码展示如下：

```
//最小二乘法求解误差方程的求解函数过程，并且对计算结果进行精度评定，用一维矩阵简化二维矩阵，在之前使用二维矩阵作形参时出现了错误，抽象一维数组为二维数组
void solving(double* og_data, CCPoints PT) {
    double X0[4] = {0.0};
    double Y0[4] = {0.0};
    double Z0[4] = {0.0};
    double x0 = 0;
    double y0 = 0;
    double v_i[2];
    double v[8*1]; //改正数矩阵
    double A[8 * 6] = {0.0};
    double L[8*1] = {0.0};
    double cor_roation[6 * 1] = {0.0}; //初始改正值矩阵，赋予初值使得初始迭代条件成立，赋1是为了表达式成立
    double AT[6*2*4];
    double ATA[6*6];
    double ATL[6];
    double *ATA_inv = NULL;
    int ct = 0; //定义迭代次数，为迭代设置最大迭代次数
    while (ct < MAXtimes && !control_prescion(cor_roation, ct)) { //这个条件后边把迭代的条件写完后写，最大迭代次数为4
        double R[9];
        caculate_roation(og_data, R);
        for (int i = 0; i < N; i++) { // N为控制点数
            Z0[i] = R[2] * (PT.pts.at(i).X - og_data[0]) + R[5] * (PT.pts.at(i).Y - og_data[1]) + R[8] * (PT.pts.at(i).Z - og_data[2]);
            X0[i] = x0 - og_data[7] * (R[0] * (PT.pts.at(i).X - og_data[0]) + R[3] * (PT.pts.at(i).Y - og_data[1]) + R[6] * (PT.pts.at(i).Z - og_data[2]));
            Y0[i] = y0 - og_data[7] * (R[1] * (PT.pts.at(i).X - og_data[0]) + R[4] * (PT.pts.at(i).Y - og_data[1]) + R[7] * (PT.pts.at(i).Z - og_data[2]));
        }
        //计算误差方程式系数
    }
}
```

Fig13.误差方程求解

需要注意的是，在求解误差方程的过程中，需要控制迭代的次数，在这里为迭代设计迭代的精度检验函数，如下图：

```
//分析迭代条件
bool control_prescion(double *cor_roation, int k) {
    if (k == 0) {
        return false;
    }
    else {
        bool flag; //建立flag，flag用于控制限差
        flag = fabs(cor_roation[0]) < 1.0e-3 && fabs(cor_roation[1]) < 1.0e-3 && fabs(cor_roation[2]) < 1.0e-3 && fabs(cor_roation[3]) < 1.0e-6 && fabs(cor_roation[4]) < 1.0e-6 && fabs(cor_roation[5]) < 1.0e-6 && fabs(cor_roation[6]) < 1.0e-6 && fabs(cor_roation[7]) < 1.0e-6 && fabs(cor_roation[8]) < 1.0e-6;
        return flag;
    }
}
```

Fig14.分析迭代条件

在迭代完成后，需要对于迭代最终的结果进行精度分析，按照间接平差的相关理论知识，利用已有的数据计算协因数阵 Q，并算出单位权中误差，由于题目视为等权的情况，所以协因数阵的对角阵与单位权中误差相乘即可得出相应的元素中误差，编程实现如下：

```
//精度评定
double Q[6] = { 0.0 };
double m0 = 0; //单位权中误差
double vv = 0;
for (int i = 0; i < 6; i++) {
    Q[i] = ATA[i * 6 + i];
}
double v[8] = {0.0};
multiply(A, cor_roation, v, 8, 6, 6, 1);
for (int i = 0; i < 8; i++) {
    v[i] -= L[i];
}

for (int i = 0; i < 8; i++) {
    vv = vv + v[i] * v[i];
}
m0 = sqrt(vv / (2 * N - 6));
double M[6] = { 0.0 }; // 保存六个值的中误差

for (int i = 0; i < 6; i++) {
    double Qi = Q[i];
    M[i] = m0 * sqrt(Qi);
    if (i > 2) {
        M[i] = M[i] * 180 * 3600 / PI; // 转换为角度制
    }
}
```

Fig15.外方位元素精度评定

5.2 中间结果展示

本次实习中重要的中间结果展示如下：

- 控制点容器所含的控制点 vector ppt.pts，如下图所示：

ppt.pts	{ size=4 }	std::vector<CPt,std::...
[capacity]	4	unsigned __int64
[allocator]	allocator	std::_Compressed_p...
[0]	{p_x=-0.086150000000000004 p_y=-0.068989999999999996 X=3...	CPt
p_x	-0.086150000000000004	double
p_y	-0.068989999999999996	double
X	36589.4100000000003	double
Y	25273.3200000000000	double
Z	2195.17000000000001	double
p_name	"qcp1"	std::string

Fig16.部分控制点数据

ppt 是实体化的 CCPoints 类型的对象，ppt.pts 存储着全部的控制点数据，方便了之后的误差方程计算

- 存储外方位元素与内方位部分元素的数组 data0，结果如下：

data0	0x00000003831df770 {38437.000000000000, 27963.15499999999...	double[9]
[0]	38437.000000000000	double
[1]	27963.154999999999	double
[2]	9178.9175000000014	double
[3]	0.0000000000000000	double
[4]	0.0000000000000000	double
[5]	0.0000000000000000	double
[6]	50000.000000000000	double
[7]	0.15324000000000002	double
[8]	0.0000000000000000	double

Fig17 初始的 data0

外方位元素的数组 data0 是后续误差处理与求解的基础，无论后续是求解旋转矩阵还是改正外方位元素，都需要依赖 data0 数组来实现，data0 数组的组织如下图：


```
data[0] = Xs0;
data[1] = Ys0;
data[2] = Zs0;
data[3] = phi;
data[4] = omega;
data[5] = kappa;
data[6] = m;
data[7] = f;
```

Fig18.data0 数组的组织格式

- 误差方程式中的 A 阵，因为每次计算结果 A 阵都会随着迭代次数的改变而改变，因此只选取了第一次的 A 阵作为展示，L 阵同理，如下图：

A	0x0000008a30efeba0 (-2.1942374062063382e-05, -0.0000000000...	double[48]
[0]	-2.1942374062063382e-05	double
[1]	-0.0000000000000000	double
[2]	1.2335783903985645e-05	double
[3]	-0.20167267097363614	double
[4]	-0.038785490080918819	double
[5]	-0.06898999999999996	double
[6]	-0.0000000000000000	double
[7]	-2.1942374062063382e-05	double
[8]	9.8786503950779987e-06	double
[9]	-0.038785490080918819	double
[10]	-0.18429990668232837	double
[11]	0.086150000000000004	double
[12]	-1.8134423008137947e-05	double
[13]	-0.0000000000000000	double
[14]	6.3193564906980300e-06	double
[15]	-0.17184845732184809	double
[16]	0.028647963978073603	double
[17]	0.08220999999999991	double
[18]	0.0000000000000000	double
[19]	-1.8134423008137947e-05	double

Fig19.A 阵

L	0x0000008a30efed40 (-0.045609489106672396, -0.00996863426...	double[8]
[0]	-0.045609489106672396	double
[1]	-0.0099686342647697587	double
[2]	-0.038785105809281489	double
[3]	0.021253766549480468	double
[4]	-0.029759462437342839	double
[5]	-0.0083130072650864084	double
[6]	-0.025741771407655854	double
[7]	0.021548185310820964	double

Fig20.L 阵

- 每次迭代后的外方位元素也都在迭代的过程中保留了，也保留了外方位元素的改正数，下图是外方位元素在迭代过程中的中间值：

```
外方位元素的初值是(单位m):  
Xs=38437.00000, Ys=27963.15500, Zs=9178.91750,  $\phi$ =0.00000,  $\omega$ =0.00000,  $\kappa$ =0.00000  
  
第1轮迭代: 外方位元素为:  
Xs=39815.97569, Ys=27473.73850, Zs=7560.47078,  $\phi$ =-0.00494,  $\omega$ =0.00175,  $\kappa$ =-0.05359  
改正数值为:  
 $\Delta X_s = 1378.98$   $\Delta Y_s = -489.417$   $\Delta Z_s = -1618.45$   
 $\Delta \phi = -0.00493929$   $\Delta \omega = 0.0017456$   $\Delta \kappa = -0.05359$   
  
第2轮迭代: 外方位元素为:  
Xs=39795.46719, Ys=27476.43328, Zs=7572.10688,  $\phi$ =-0.00399,  $\omega$ =0.00212,  $\kappa$ =-0.06761  
改正数值为:  
 $\Delta X_s = -20.5085$   $\Delta Y_s = 2.69479$   $\Delta Z_s = 11.6361$   
 $\Delta \phi = 0.000950327$   $\Delta \omega = 0.000369611$   $\Delta \kappa = -0.0140153$   
  
第3轮迭代: 外方位元素为:  
Xs=39795.45228, Ys=27476.46234, Zs=7572.68594,  $\phi$ =-0.00399,  $\omega$ =0.00211,  $\kappa$ =-0.06758  
改正数值为:  
 $\Delta X_s = -0.0149043$   $\Delta Y_s = 0.0290565$   $\Delta Z_s = 0.579052$   
 $\Delta \phi = 2.03637e-06$   $\Delta \omega = -1.31993e-06$   $\Delta \kappa = 2.73067e-05$   
  
第4轮迭代: 外方位元素为:  
Xs=39795.45229, Ys=27476.46224, Zs=7572.68591,  $\phi$ =-0.00399,  $\omega$ =0.00211,  $\kappa$ =-0.06758  
改正数值为:  
 $\Delta X_s = 6.46432e-06$   $\Delta Y_s = -9.44744e-05$   $\Delta Z_s = -2.14633e-05$   
 $\Delta \phi = -2.82643e-09$   $\Delta \omega = 1.23151e-08$   $\Delta \kappa = -5.93806e-09$ 
```

Fig21.外方位元素的迭代中间值

- 外方位元素的精度与单位权中误差，根据平差理论，获得的外方位元素的精度与单位权中误差展示结果如下图（由于角元素习惯使用角度表达，因此将中误差的单位由弧度转变为了角度）

```
单位权中误差: 7.25942e-06  
  
六个外方位元素的精度:  
Xs: 1.10739 m  
Ys: 1.24952 m  
Zs: 0.488128 m  
 $\phi$ : 36.8442'  
 $\omega$ : 33.3038'  
 $\kappa$ : 14.859'
```

Fig22 外方位元素的单位权中误差与精度

6. 实习结果与分析

6.1 实验数据

	影像坐标		地面坐标		
	X (mm)	Y (mm)	X (m)	Y (m)	Z (m)
1	-86.15	-68.99	36589.41	25273.32	2195.17
2	-53.40	82.21	37631.08	31324.51	728.69
3	-14.78	-76.63	39100.97	24934.98	2386.50
4	10.46	64.43	40426.54	30319.81	757.31

$$f=153.24\text{mm}, \quad x_0=y_0=0$$

6.1 实验结果

$$X_s=39795.45\text{m}$$

$$Y_s=27476.46\text{m}$$

$$Z_s=7572.69\text{m}$$

$$R=\begin{pmatrix} 0.99771 & 0.06753 & 0.00399 \\ -0.06753 & 0.99772 & -0.00211 \\ -0.00412 & 0.00184 & 0.99999 \end{pmatrix}$$

$$\varphi=-0.00399\text{rad}$$

$$\omega=0.00211\text{rad}$$

$$\kappa=-0.06758\text{rad}$$

在实验结果中，可能由于 C++在输出时保留位数的问题，可能存在实验的结果与答案存在差异的情况，当控制了输出位数，结果输出与答案相同，说明了输出结果是正确的，另外，由于在程序中输出外方位角元素的精度时将弧度转化为了角度，符合我们的一般使用习惯，代码执行的实验结果如下图所示：

```
最后输出的外方位元素为：
Xs = 39795.45   Ys = 27476.46   Zs = 7572.69   φ = -0.0039869317   ω = 0.0021139057   κ = -0.0675779767
单位权中误差：7.25942e-06

六个外方位元素的精度：
Xs: 1.10739 m
Ys: 1.24952 m
Zs: 0.488128 m
φ: 36.8442'
ω: 33.3038'
κ: 14.859'

输出最终的旋转矩阵为：
0.997709 0.0675344 0.00398691
-0.0675264 0.997715 -0.0021139
-0.00412056 0.00183984 0.99999
```

Fig23 实验代码执行结果

6.3 实验分析

6.3.1 外方位元素计算结果分析

在求解外方位元素初始值时，做了两点假设。第一，假设控制点在影像覆盖范围内均匀分布；第二，假设影像竖直对地摄影。当倾斜摄影或者近景交向摄影时，上述代码中求解外方位元素的方法将会失效，导致迭代过程不收敛，无法求解正确的外方位元素。而本次实验是基于影像竖直对地摄影，因此迭代过程是收敛的，问题本身是可解的。

结合原始的课本数据资料，发现课本数据给出的外方位元素的结果与程序运行结果相同（当程序与课本保留相同的数据位数后，两者之间的数值相同），说明程序运行结果正确。在计算外方位元素时，我们不断迭代，并且改正初值，直到，满足退出迭代的条件（线元素改正数 $<1.0\text{e-}3$ ，角元素改正数 $<1.0\text{e-}6$ ），对每次的迭代改正数进行分析，不难发现，到第四次满足退出迭代条件，因此可知迭代总次数为 4，输出第四次计算结果。

```
Δ Xs = 1378.98   Δ Ys = -489.417   Δ Zs = -1618.45  
Δ φ = -0.00493929   Δ ω = 0.0017456   Δ κ = -0.05359
```

Fig24 第一次迭代改正数

```
改正数值为：  
Δ Xs = -20.5085   Δ Ys = 2.69479   Δ Zs = 11.6361  
Δ φ = 0.000950327   Δ ω = 0.000369611   Δ κ = -0.0140153
```

Fig25 第二次迭代改正数

```
Δ Xs = -0.0149043   Δ Ys = 0.0290565   Δ Zs = 0.579052  
Δ φ = 2.03637e-06   Δ ω = -1.31993e-06   Δ κ = 2.73067e-05
```

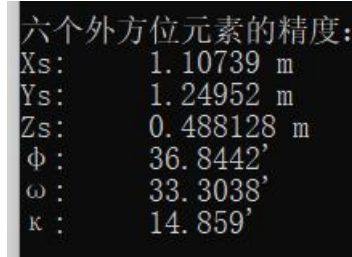
Fig26 第三次迭代改正数

```
改正数值为：  
Δ Xs = 6.46432e-06   Δ Ys = -9.44744e-05   Δ Zs = -2.14633e-05  
Δ φ = -2.82643e-09   Δ ω = 1.23151e-08   Δ κ = -5.93806e-09
```

Fig27 第四次迭代改正数

6.3.2 精度分析

由于本次实习采用的是基于最小二乘法的间接平差法，因此精度评定的结果按照间接平差的方法计算，通过公式计算出相应的结果后，发现线元素的误差分别为 1.10739m，1.24952m，0.488128m，线元素误差在 1m 级，符合精度要求，角元素的中误差为 36.8442'，33.3038'，14.859'，角元素误差在 30'级，同样符合精度要求。

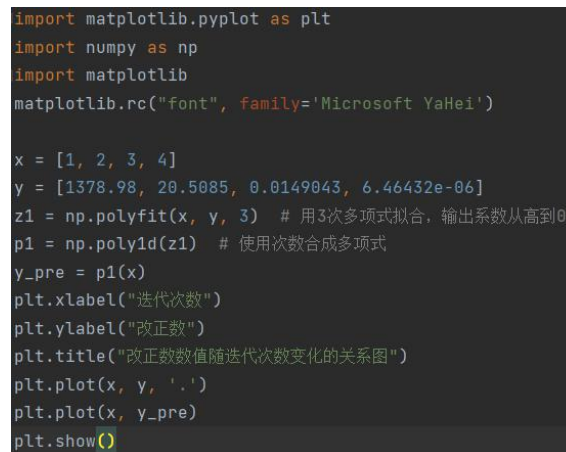


六个外方位元素的精度:
Xs: 1.10739 m
Ys: 1.24952 m
Zs: 0.488128 m
Φ: 36.8442'
ω: 33.3038'
κ: 14.859'

Fig28 外方位元素精度

6.3.3 迭代收敛速度分析

由于每次迭代，改正数数值变化较大，为了分析迭代收敛速度，利用 python 对迭代的改正数与迭代次数之间建立一种拟合关系，这里使用的是三次项拟合数据，建立了关于不同外方位元素改正数与迭代次数之间的关系，具体代码如下：



```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
matplotlib.rc("font", family='Microsoft YaHei')

x = [1, 2, 3, 4]
y = [1378.98, 20.5085, 0.0149043, 6.46432e-06]
z1 = np.polyfit(x, y, 3) # 用3次多项式拟合，输出系数从高到0
p1 = np.poly1d(z1) # 使用系数合成多项式
y_pre = p1(x)
plt.xlabel("迭代次数")
plt.ylabel("改正数")
plt.title("改正数数值随迭代次数变化的关系图")
plt.plot(x, y, '.')
plt.plot(x, y_pre)
plt.show()
```

Fig29 建立迭代改正数与迭代次数的关系代码

代码建立完成后，利用在 C++ 程序中获得的中间结果-迭代次数与外方位元素的改正数作为 python 程序的输入数据，从而完成相应的分析，下面将具体展示每一个外方位元素的收敛速度：

✓ X_s 迭代改正数与迭代次数的关系图像

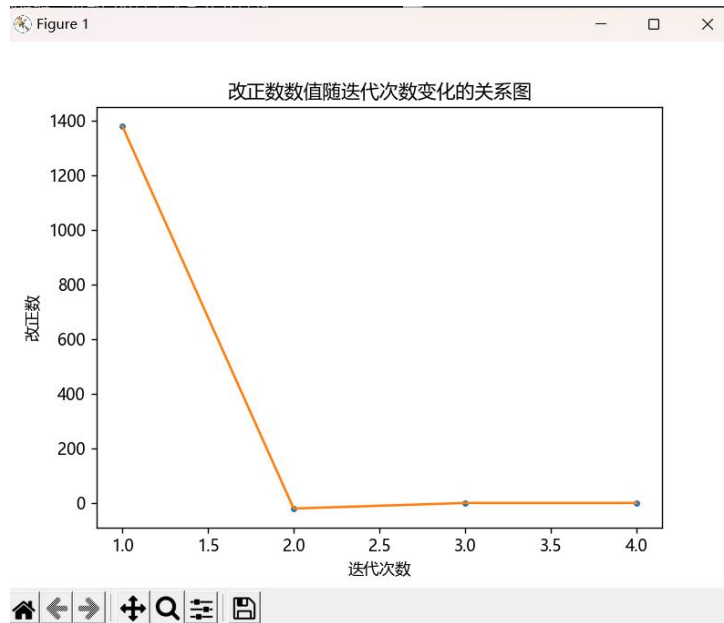


Fig30 X_s 迭代改正数与迭代次数的关系图像

✓ Y_s 迭代改正数与迭代次数的关系图像

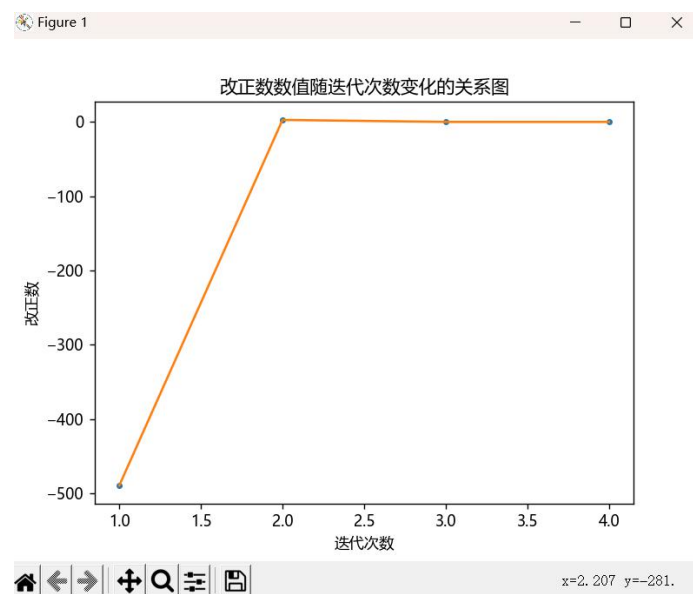


Fig31 Y_s 迭代改正数与迭代次数的关系图像

✓ Z_s 迭代改正数与迭代次数的关系图像

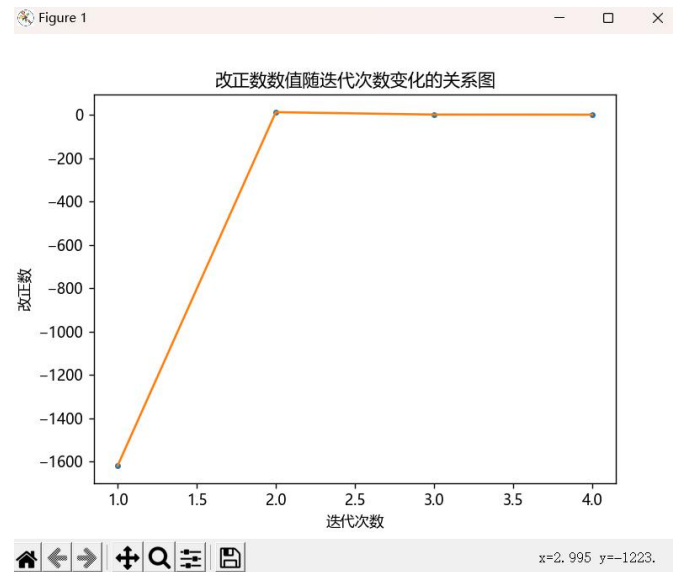


Fig32 Z_s 迭代改正数与迭代次数的关系图像

✓ φ 迭代改正数与迭代次数的关系图像

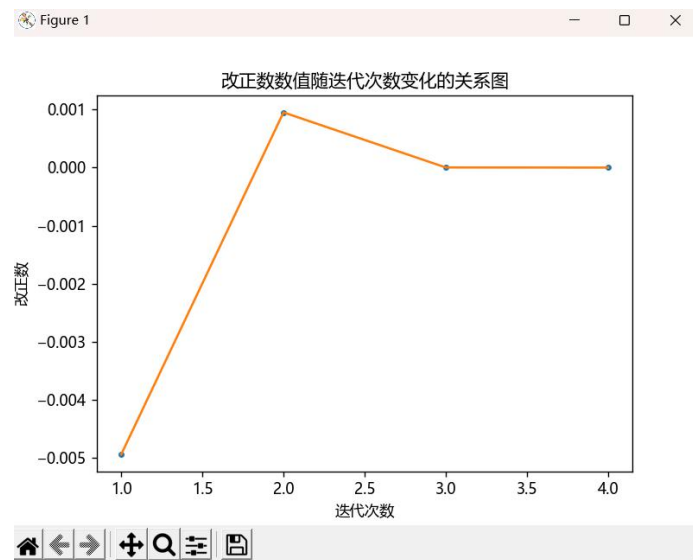


Fig33 φ 迭代改正数与迭代次数的关系图像

✓ ω 迭代改正数与迭代次数的关系图像

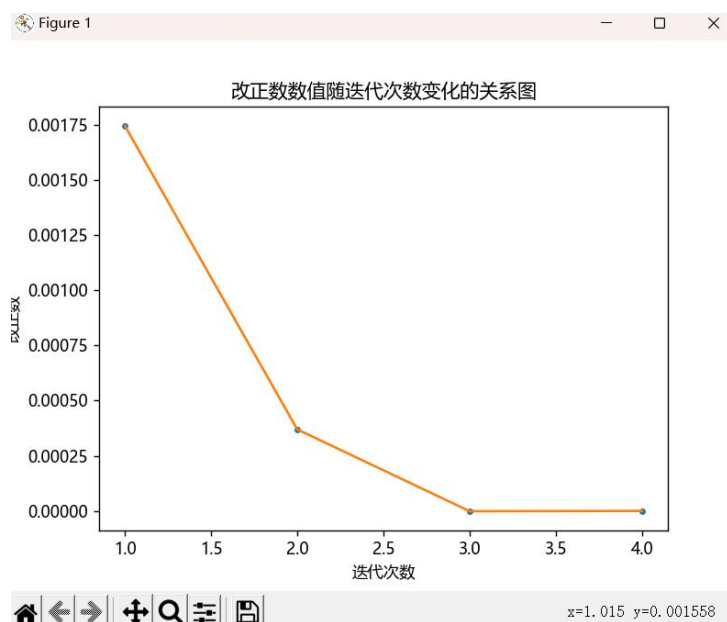


Fig34 ω 迭代改正数与迭代次数的关系图像

✓ κ 迭代改正数与迭代次数的关系图像

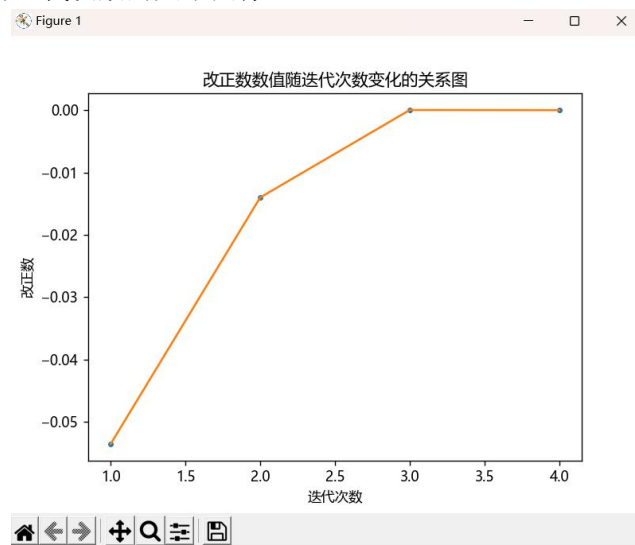


Fig35 κ 迭代改正数与迭代次数的关系图像

通过对于以上的图像分析，不难发现，外方位元素改正数与迭代次数的关系存在着快收敛的性质，当迭代次数为在 1 与 2 之间时，收敛速度是最快的（所有的外方位元素均使用），而且全部的外方位元素到迭代第三次时开始收敛向 0，反映出了最小二乘法计算外方位元素的方法是可行的，也即符合两个假设条件，实验成功进行

7. 实习心得

本次实习使用了 C, C++, python 编程语言, 极大地锻炼了代码能力, 尽管在编程的过程中遇到了许多的问题, 比如使用 new 动态分配二维数组的内存时不一定地址连续, 指向指针的指针不一定可以代表矩阵....., 但是通过资料的查询和请教同学, 很好地解决了在实习中遇到的问题, 收获了许多知识。总的来说, 本次实习要求的任务都已充分完成, 并在实习过程中有不断试错、不断探究的过程, 是一次完整而有收获的实习, 并且在实习过程中, 及时使用了老师教授的知识, 丰满了摄影测量的知识体系, 对于接下来的课程学习具有重要的意义。

综上, 本次实习兼顾了理论与实践, 做到了实践体现真知, 对于我有极大的裨益。