一、实习目的

使用面向对象语言 C++进行程序设计，编写出一个完整的单片空间后方交会程序对实验数据进行计算，输出改正后像点坐标和地面坐标，输出像片的外方位元素，并对其进行精度评定。实习目的为深入理解单片空间后方交会的原理，通过自我编程加强动手能力的培养，通过对实验结果的分析，增强学生综合运用所学知识解决实际问题的能力。

二、实习内容与步骤

原始数据：

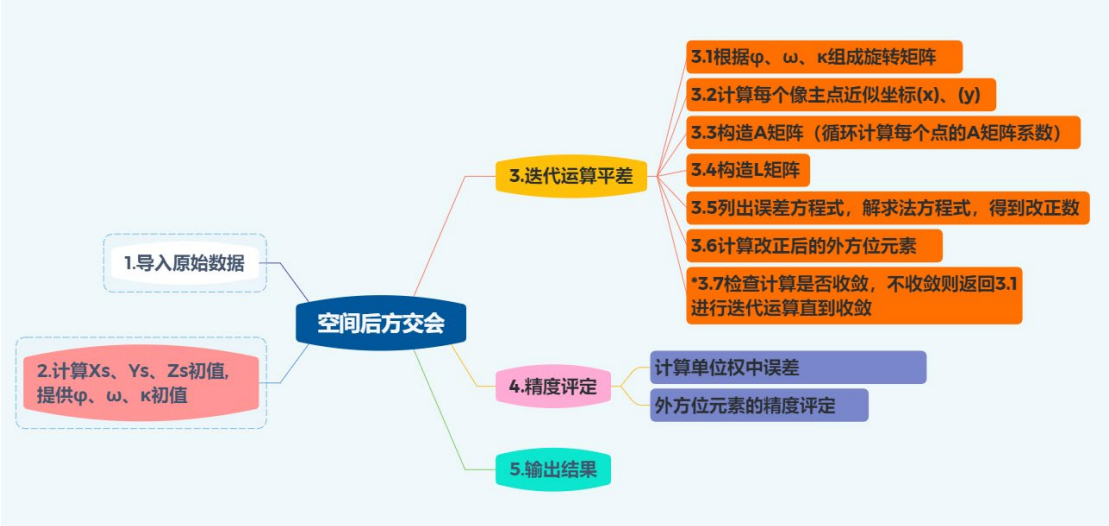
	影像坐标		地面坐标		
	x(mm)	y(mm)	X(m)	Y(m)	Z(m)
1	-86.15	-68.99	36589.41	25273.32	2195.17
2	-53.40	82.21	37631.08	31324.51	728.69
3	-14.78	-76.63	39100.97	24934.98	2386.50
4	10.46	64.43	40426.54	30319.81	757.31

$$x_0 = y_0 = 0;$$

$$f_k = 153.24mm;$$

$$m = 50000$$

流程图：



数据存储：**所有的数据变量和函数都存储在 SpaceResection 类里，创建 SpaceResection 类对象即可进行后方交会计算。**

```
struct portrayCoordinate//影像坐标
{
    double x,y;
};
struct floorCoordinate//地面坐标
{
    double X, Y,Z;
};
class SpaceResection
{
public://变量
    double m = 0,//像片比例尺
        f = 0,//焦距
        x0 = 0, y0 = 0;//内方位元素
    vector<portrayCoordinate>pic_XY;//像点坐标
    vector<floorCoordinate>floor_XY;//地面控制点坐标
    double pointnum=0;//地面控制点数量

    vector<double> M;// 保存六个值的中误差
    double   m0 = 0,     // 单位权中误差
           vv = 0;      // [vv]，即平方和

    double Xs = 0.0, Ys = 0.0, Zs = 0.0, t = 0.0, w = 0.0, k = 0.0;//外方位元素
```

函数实现：

```
public://函数
    SpaceResection();
    SpaceResection(double m, double f, double x0, double y0);

    void ReadCoordinate();//读取影像、地面坐标
    void GetStart();//初始化

    Matrix constructSR_R_Matrix(double a, double b, double c);//旋转矩阵R构造
    Matrix constructSR_A_Matrix(Matrix R, vector<double>&X, vector<double>&Y, vector<double>&Z);//矩阵A构造
    Matrix constructSR_L_Matrix(vector<double>X, vector<double>Y, vector<double>Z); // 矩阵L构造
    void correction(Matrix XX);//改正数
    bool CheckPrecison(Matrix &X);//收敛判断
    void AccuracyEvaluation(Matrix ATA, Matrix A, Matrix XX, Matrix L);//精度评定


    void calculate();//开始计算

    ~SpaceResection();
};
```

录入原始数据

将保存在地面坐标.txt 以及 影像坐标.txt 文件的影像坐标和地面控制点坐标读取，调用 ReadCoordinate 函数录入相应数据保存在两个数组中。

地面坐标.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

X	Y	Z
36589.41	25273.32	2195.17
37631.08	31324.51	728.69
39100.71	24934.98	2386.50
40426.54	30319.81	757.31

影像坐标.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(

x	y
-86.15	-68.99
-53.40	82.21
-14.78	-76.63
10.46	64.43

```
vector<portrayCoordinate>pic_XY;//像点坐标
vector<floorCoordinate>floor_XY;//地面控制点坐标
```

```
void SpaceResection::ReadCoordinate()
{
    FILE* fp = fopen("影像坐标.txt", "r");
    if (!fp)
    {
        cout << "读取失败";
        return;
    }
    fscanf_s(fp, "x\ty\t\n");

    while (!feof(fp))
    {
        portrayCoordinate p_XY;
        fscanf_s(fp, "%lf\t%lf\t\n", &p_XY.x, &p_XY.y);
        pic_XY.push_back(p_XY);
    }
    fclose(fp);
    this->pointnum = this->pic_XY.size();

    FILE* fs = fopen("地面坐标.txt", "r");
    if (!fs)
    {
        cout << "读取失败";
        return;
    }
    fscanf_s(fs, "X\tY\tZ\t\n");

    while (!feof(fs))
    {

        floorCoordinate fl_Cor;
        fscanf_s(fp, "%lf\t%lf\t%lf\t\n", &fl_Cor.X, &fl_Cor.Y,&fl_Cor.Z);
        floor_XY.push_back(fl_Cor);
    }
    fclose(fs);
```

2. 计算外方位元素初值

$$Z_S^0 = H = m \cdot f$$

$$X_S^0 = \frac{1}{n}\sum X_{ti}$$

$$Y_S^0 = \frac{1}{n}\sum Y_{ti}$$

$$\varphi^0 = \omega^0 = \kappa^0 = 0$$

m 为摄影比例尺字母，f 为航摄焦距，n 为控制点个数

调用 GetStart 函数进行初始化，注意影像坐标单位换算（f 已在 main 函数换算）

```
void SpaceResection::GetStart()//初始化，计算外方位元素的初值
{
    this->Zs = this->m * this->f;
    for (int i = 0; i <pointnum; i++)
    {
        this->pic_XY[i].x /= 1000;//单位换算 mm->m
        this->pic_XY[i].y /= 1000;
        this->Xs += this->floor_XY[i].X;
        this->Ys += this->floor_XY[i].Y;
    }
    this->Xs = this->Xs / pointnum;
    this->Ys = this->Ys / pointnum;
    t = w = k = 0;
}
```

3. 迭代运算后方交会

3.1. 根据ϕ、ω、κ组成旋转矩阵

原理：

$$R = R_\varphi R_\omega R_\kappa$$

$$= \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{bmatrix} \begin{bmatrix} \cos\kappa & -\sin\kappa & 0 \\ \sin\kappa & \cos\kappa & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

调用 constructSR_R_Matrix 函数进行旋转矩阵 R 的构建

```
Matrix SpaceResection::constructSR_R_Matrix(double a, double b, double c)
{
    Matrix R(3, 3);
    double sinA = sin(a), cosA = cos(a),
        sinB = sin(b), cosB = cos(b),
        sinC = sin(c), cosC = cos(c);
    R(0, 0) = cosA * cosC - sinA * sinB * sinC;
    R(0, 1) = -cosA * sinC - sinA * sinB * cosC;
    R(0, 2) = -sinA * cosB;
    R(1, 0) = cosB * sinC;
    R(1, 1) = cosB * cosC;
    R(1, 2) = -sinB;
    R(2, 0) = sinA * cosC + cosA * sinB * sinC;
    R(2, 1) = -sinA * sinC + cosA * sinB * cosC;
    R(2, 2) = cosA * cosB;
    return R;
}
```

3.2. 计算每个像主点近似坐标(x)、(y)

3.3. 构造 A 矩阵（循环计算每个点的 A 矩阵系数）

原理：

$$\begin{cases} x - x_0 = -f\dfrac{a_1(X_A-X_S)+b_1(Y_A-Y_S)+c_1(Z_A-Z_S)}{a_3(X_A-X_S)+b_3(Y_A-Y_S)+c_3(Z_A-Z_S)} = -f\dfrac{\bar{X}}{\bar{Z}} \\ y - y_0 = -f\dfrac{a_2(X_A-X_S)+b_2(Y_A-Y_S)+c_2(Z_A-Z_S)}{a_3(X_A-X_S)+b_3(Y_A-Y_S)+c_3(Z_A-Z_S)} = -f\dfrac{\bar{Y}}{\bar{Z}} \end{cases}$$

式中：x,y 为像点的像平面坐标；

x_0, y_0, f 为影像内方位元素；

X_S, Y_S, Z_S 为摄站点的物方空间坐标（影像外方位元素）；

X_A, Y_A, Z_A 为物方点的物方空间坐标（地面控制点坐标）；

将共线方程线性化，每个改正数前的参数可以通过共线方程求偏导数得到……

内方位元素已知，误差方程式可以简化为：

$$\begin{cases} v_x = a_{11}\Delta X_S + a_{12}\Delta Y_S + a_{13}\Delta Z_S + a_{14}\Delta\varphi + a_{15}\Delta\omega + a_{16}\Delta\kappa - l_x \\ v_y = a_{21}\Delta X_S + a_{22}\Delta Y_S + a_{23}\Delta Z_S + a_{24}\Delta\varphi + a_{25}\Delta\omega + a_{26}\Delta\kappa - l_y \end{cases}$$

总误差方程的矩阵形式为：$V = AX - L$

式中：

$$V = [v_x \quad v_y]^T$$

$$\mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \end{bmatrix}$$

$$\mathrm{X} = [\Delta X_S \quad \Delta Y_S \quad \Delta Z_S \quad \Delta\varphi \quad \Delta\omega \quad \Delta\kappa]$$

$$\mathrm{L} = [l_x \quad l_y]^T = [x-(x) \quad y-(y)]^T$$

求出每个点的像主点近似坐标(x)、(y)后，求出每个点的偏导数系数值，得到 A 矩阵。

```
Matrix SpaceResection::constructSR_A_Matrix(Matrix R, vector<double> &X, vector<double> &Y, vector<double> &Z)
{

    Matrix A(pointnum*2, 6);
    for (int i = 0; i < pointnum; i++)
    {

        Z[i]=(R(0, 2) * (floor_XY[i].X - Xs) + R(1, 2) * (floor_XY[i].Y - Ys) + R(2, 2) * (floor_XY[i].Z - Zs));
        //像主点近似坐标
        X[i]=(this->x0 - (this->f * (R(0, 0) * (floor_XY[i].X - Xs) + R(1, 0) * (floor_XY[i].Y - Ys) + R(2, 0) * (floor_XY[i].Z - Zs))) / Z[i]);
        Y[i]=(this->y0 - (this->f * (R(0, 1) * (floor_XY[i].X - Xs) + R(1, 1) * (floor_XY[i].Y - Ys) + R(2, 1) * (floor_XY[i].Z - Zs))) / Z[i]);

        //偏导数值
        A(i * 2, 0) = (R(0, 0) * f + R(0, 2) * (pic_XY[i].x - x0)) / Z[i];//a11=1/Z * (a1*f+a3(x-x0))
        A(i * 2, 1) = (R(1, 0) * f + R(1, 2) * (pic_XY[i].x - x0)) / Z[i];//a12=1/Z * (b1*f+b3(x-x0))
        A(i * 2, 2) = (R(2, 0) * f + R(2, 2) * (pic_XY[i].x - x0)) / Z[i];//a13=1/Z * (c1*f+c3(x-x0))

        A(i * 2 + 1, 0) = (R(0, 1) * f + R(0, 2) * (pic_XY[i].y - y0)) / Z[i];//a21=1/Z * (a2*f+a3(y-y0))
        A(i * 2 + 1, 1) = (R(1, 1) * f + R(1, 2) * (pic_XY[i].y - y0)) / Z[i];//a22=1/Z * (b2*f+b3(y-y0))
        A(i * 2 + 1, 2) = (R(2, 1) * f + R(2, 2) * (pic_XY[i].y - y0)) / Z[i];//a23=1/Z * (c2*f+c3(y-y0))

        A(i * 2, 3) = (pic_XY[i].y - y0) * sin(w) - ((pic_XY[i].x - x0) / f * ((pic_XY[i].x - x0) * cos(k) - (pic_XY[i].y - y0) * sin(k)) + f * cos(k)) * cos(w);//a14
        A(i * 2, 4) = -f * sin(k) -( (pic_XY[i].x - x0) / f * ((pic_XY[i].x - x0) * sin(k) + (pic_XY[i].y - y0) * cos(k)));//a15
        A(i * 2, 5) = pic_XY[i].y - y0;//a16

        A(i * 2 + 1, 3) = -(pic_XY[i].x - x0) * sin(w) - ((pic_XY[i].y - y0) / f * ((pic_XY[i].x - x0) * cos(k) - (pic_XY[i].y - y0) * sin(k)) - f * sin(k)) * cos(w);
        A(i * 2 + 1, 4) = -f * cos(k) - ((pic_XY[i].y - y0) / f * ((pic_XY[i].x - x0) * sin(k) + (pic_XY[i].y - y0) * cos(k)));
        A(i * 2 + 1, 5) = -(pic_XY[i].x - x0);


    }
    return A;
}
```

3.4. 构造 L 矩阵

原理：

$$\mathrm{L} = [l_x \quad l_y]^T = [x-(x) \quad y-(y)]^T$$

对每个点的像点坐标减去近似像点坐标即 L 矩阵系数

```
Matrix SpaceResection::constructSR_L_Matrix(vector<double>X, vector<double>Y, vector<double>Z)
{
    Matrix L(pointnum * 2, 1);
    for (int i = 0; i < pointnum; i++)
    {
        L(i * 2, 0) = pic_XY[i].x - X[i];
        L(i * 2+1, 0) = pic_XY[i].y - Y[i];
    }
    return L;
}
```

3.5. 列出误差方程式，解求法方程式，得到改正数

原理：根据最小二乘间接平差原理，列出法方程式：

$$\boldsymbol{A^T PAX = A^T PL}$$

式中 P 为观测值权矩阵，反映观测值的量测精度。对所有像点坐标的观测值，一般认为是等精度量测，则 P 为单位矩阵，由此得到法方程解的表达式：

$$\mathrm{X} = (A^T A)^{-1} A^T L$$

代码：

```
ATA = A.transpose() * A;
ATL = A.transpose() * L;
XX= ATA.inverse()* ATL;
```

3.6. 计算改正后的外方位元素

原理：

$$\begin{cases} X_S = X_S^0 + \Delta X_S^1 + \Delta X_S^2 + \cdots \\ Y_S = Y_S^0 + \Delta Y_S^1 + \Delta Y_S^2 + \cdots \\ Z_S = Z_S^0 + \Delta Z_S^1 + \Delta Z_S^2 + \cdots \\ \varphi = \varphi^0 + \Delta\varphi^1 + \Delta\varphi^2 + \cdots \\ \omega = \omega^0 + \Delta\omega^1 + \Delta\omega^2 + \cdots \\ \kappa = \kappa^0 + \Delta\kappa^1 + \Delta\kappa^2 + \cdots \end{cases}$$

每次计算结束加上新的改正数，反复趋近。

代码：

```
void SpaceResection::correction(Matrix XX)
{
    Xs = Xs + XX(0, 0);
    Ys = Ys + XX(1, 0);
    Zs = Zs + XX(2, 0);
    t = t + XX(3, 0);
    w = w + XX(4, 0);
    k = k + XX(5, 0);
}
```

3.7. 检查计算是否收敛，不收敛则返回 3.1 进行迭代运算直到收敛

原理：当改正数$\boldsymbol{\Delta X_S, \Delta Y_S, \Delta Z_S}$皆小于限差（$\boldsymbol{10^{-3}}$）

$\boldsymbol{\Delta\varphi, \Delta\omega, \Delta\kappa}$皆小于限差（$\boldsymbol{10^{-6}}$）

代码：当改正数皆小于其规定限差时，返回 Ture 结束迭代。

```
#define PRECISION1 1.0e-3
#define PRECISION2 1.0e-6
```

```
bool SpaceResection::CheckPrecison(Matrix& X)
{
    bool Boolean;
    Boolean = { fabs(X(0,0)) < PRECISION1 &&fabs(X(1,0))<PRECISION1&& fabs(X(2,0)) < PRECISION1
        && fabs(X(3,0)) < PRECISION2 && fabs(X(4,0)) < PRECISION2 && fabs(X(5,0)) < PRECISION2 };

    return Boolean;
}
```

3.8. 总计算代码

```
int Count=0;//迭代次数
cout << "******开始迭代******" << endl;
do{
    Count++;
    if (Count ==30) {
        cout << "迭代次数超限，可能不收敛" << endl;
        break;
    }
    Matrix R=constructSR_R_Matrix(this->t, this->w, this->k);//构造R矩阵

    vector<double>X(pointnum), Y(pointnum), Z(pointnum);       //像主点近似坐标
    A=constructSR_A_Matrix(R,X,Y,Z);//构造A矩阵
    L=constructSR_L_Matrix(X,Y,Z);//构造L矩阵

    //X=inv(A^T *A) * A^T * L
    ATA = A.transpose() * A;
    ATL = A.transpose() * L;
    XX= ATA.inverse()* ATL;
    correction(XX);

    cout << "迭代次数第" << Count << "次" << endl;
    cout << "外方位元素" << endl;
    cout << "\tXs =" << Xs << "\tYs =" << Ys << "\tZs =" << Zs << endl;
    cout << "\tΦ =" << t << "\tω = " << w << "\tK =" << k << endl<<endl;
} while (!CheckPrecison(XX));
```

4. 精度评定

原理：当参加空间后方交会的控制点有 n 个时，则单位权中误差可按下式计算：

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

第 i 个未知数的中误差为：

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

式中 Q_{ii}为 $(A^TA)^{-1}$第i个主对角线上元素

```
void SpaceResection::AccuracyEvaluation(Matrix ATA,Matrix A,Matrix XX,Matrix L)
{
    // 精度评定
    vector<vector<double>> Q(6, vector<double>(1));
    for (int i = 0; i < 6; i++) {
        Q[i][0] = ATA(i, i);
    }


    Matrix V = A * XX - L;          // 当有 N 个控制点时：V = A × X - L

    for (int i = 0; i < 8; i++) {
        vv = vv + V(i, 0) * V(i, 0);
    }
    m0 = sqrt(vv / (2 * pointnum - 6));          // 单位全中误差 m0

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