

课程编号：1300167

课程性质：必修

误差理论与测量平差基础 课程设计报告

学院： 测绘学院

专业： 测绘工程

班级： 2017 级 3 班

学号： 2017301610095

姓名： 曹臻

2019-2-18 至 2018-3-1

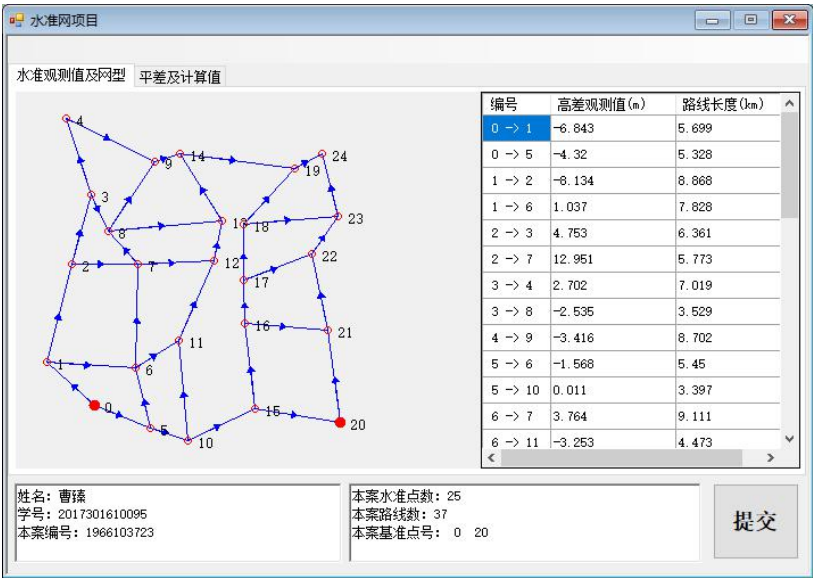
目录

一、题目内容.....	1
1. 水准网平差.....	1
2. 导线网平差.....	1
二、计算所需的数学模型.....	2
1. 基础方程及其解.....	2
2. 精度评定.....	3
三、水准网篇.....	3
1. 计算过程.....	3
2. 最后结果.....	4
3. 对结果的检核.....	6
4. 计算框图与程序说明.....	7
四、导线网篇.....	8
1. 计算过程.....	8
2. 最后结果.....	11
3. 对结果的检核.....	13
4. 计算框图与程序说明.....	14
五、实习中的心得体会.....	15
六、附录.....	17

一、题目内容

1. 水准网平差

如图所示的水准网及起算数据，进行平差。

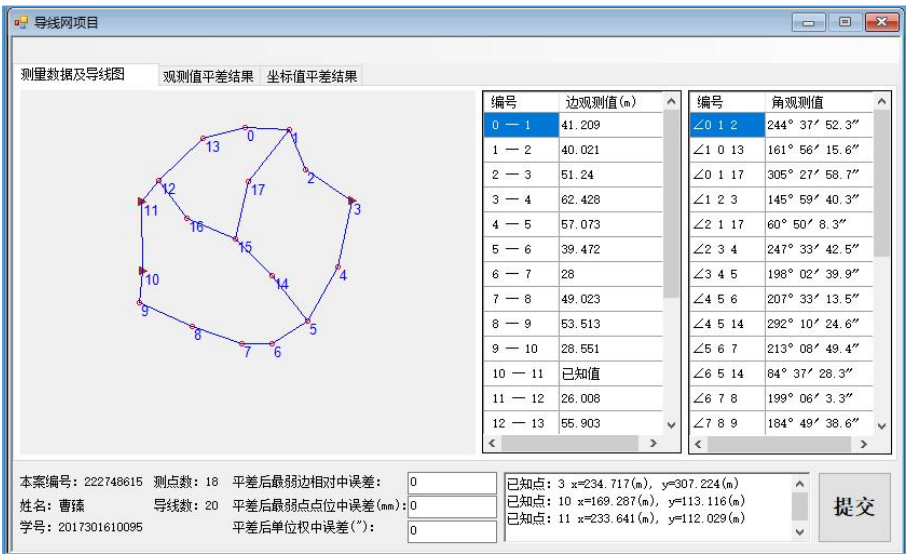


须完成以下三点要求：

- (1) 1 公里高差观测值的中误差；
- (2) 各待定点高程平差值及中误差；
- (3) 最弱点及其精度。

2. 导线网平差

如图所示的导线网及起算数据，进行平差。



注意导线网先验精度信息：导线网边长误差 1/2000，测角误差 12”。

对各导线网，须完成以下几点要求：

- (1) 平差后单位权中误差；
- (2) 各个待定点的坐标平差值及中误差；
- (3) 各观测值的平差值；
- (4) 平差后最弱边的边长相对中误差。

二、计算所需的数学模型

对于上述水准网和导线网的平差计算，均采用间接平差模型。

间接平差函数模型的一般形式为：

$$\underset{n \times 1}{\hat{L}} = \underset{n \times t}{B} \underset{t \times 1}{\hat{X}} + \underset{n \times 1}{d}$$

其误差方程为：

$$\underset{n \times 1}{V} = \underset{n \times t}{B} \underset{t \times 1}{\hat{x}} - \underset{n \times 1}{l}$$

其随机模型为：

$$\underset{n \times n}{D} = \sigma_0^2 \underset{n \times n}{Q} = \sigma_0^2 \underset{n \times n}{P}^{-1}$$

平差的准则为：

$$V^T P V = \min$$

即最小二乘准则。

1. 基础方程及其解

由上可知，方程的解满足方程：

$$B^T P V = 0$$

从而，若令：

$$N_{BB} = B^T P B, W = B^T P l$$

$t \times t$ $t \times 1$

则可求得参数平差值：

$$\hat{x} = N_{BB}^{-1} W$$

再将求出的 \hat{x} 代入误差方程求得 V ，从而平差结果为：

$$\hat{L} = L + V, \hat{X} = X^0 + \hat{x}$$

2. 精度评定

利用以下公式计算单位权方差 σ_0^2 的估值 $\hat{\sigma}_0^2$ ：

$$\hat{\sigma}_0^2 = \frac{V^T P V}{r} = \frac{V^T P V}{n - t}$$

中误差的估值为：

$$\hat{\sigma}_0 = \sqrt{\frac{V^T P V}{n - t}}$$

若设 $Q_{LL} = Q$ ，则由 $\hat{x} = N_{BB}^{-1} B^T P l$ 可知：

$$Q_{\hat{x}\hat{x}} = N_{BB}^{-1} B^T P Q P B N_{BB}^{-1} = N_{BB}^{-1}$$

故参数 $\hat{x}_i (1 \leq i \leq t)$ 的中误差为：

$$\hat{\sigma}_{\hat{x}_i} = \hat{\sigma}_0 \sqrt{Q_{\hat{x}_i \hat{x}_i}}$$

三、水准网篇

1. 计算过程

本题观测值 $n=37$ ，共有 25 个点，其中 2 个点（点 0 与点 20）已知，故必要观测数 $t=23$ ，于是取 23 个未知点的高程为参数。

(1) 先计算 37 个观测值的权阵 P ，取 1km 的观测高差为单位权观测，即 $P_i=1/S_i$ ，

($1 \leq i \leq 37$ ， S_i 为第 i 个观测值对应的路线长度)。

(2) 由于水准网不需线性化，故将 23 个参数的近似值均取 0 不会改变结果。

(3) 再计算系数矩阵 B 与常数项矩阵 1, 分析可知 B 矩阵中的数值只可能取 0, 1, -1, 于是对于每一个观测值 L_i , 设其为从点 j 到点 k 的高差, 于是便有:

$$L_i + v_i = x_k - x_j$$

分情况讨论:

i.若 k, j 均为未知点, 则矩阵 B 第 i 行中点 j 对应的列值为-1, 点 k 对应的列值为 1, 其余列为 0, 此时常数项 $l_i = L_i$ 。

ii.若 k 为已知点, j 为未知点, 则矩阵 B 第 i 行中点 j 对应的列值为-1, 其余列为 0, 此时常数项 $l_i = L_i - x_k$ 。

iii.若 j 为已知点, k 为未知点, 则矩阵 B 第 i 行中点 k 对应的列值为 1, 其余列为 0, 此时常数项 $l_i = L_i + x_j$ 。

iv.若 j, k 均为已知点则不存在误差方程。

(4) 然后通过上面“计算所需的数学模型”中的公式与步骤解算出 \hat{x} 与改正数 V , 其中由于近似值为 0, 所以算出的参数改正数可直接作为参数平差值。

2. 最后结果

(采用导出数据进行的计算)

(1) 平差后的一公里高差观测值的中误差为 15.6126949079299mm。

(2) 平差后最弱点为点 4, 中误差为 51.1988690413284mm。

(3) 观测值的平差值:

路线编号	路线长度 (km)	高差观测值 (m)	高差平差值 (m)
0 → 1	5.699	-6.842	-6.880880409
0 → 5	5.328	-4.319	-4.304452624
1 → 2	8.868	-8.133	-8.175727964
1 → 6	7.828	1.036	1.020311886
2 → 3	6.361	4.752	4.728408144
2 → 7	5.773	12.951	12.94459549
3 → 4	7.019	2.701	2.697112145

3 → 8	3.529	-2.535	-2.546133726
4 → 9	8.702	-3.416	-3.420820077
5 → 6	5.45	-1.568	-1.556115899
5 → 10	3.397	0.01	0.011867652
6 → 7	9.111	3.764	3.748555638
6 → 11	4.473	-3.252	-3.243628333
7 → 8	3.928	-10.768	-10.76232107
7 → 12	6.766	-3.374	-3.402757398
8 → 9	7.366	1.822	1.822425794
8 → 13	10.018	4.274	4.256298467
9 → 14	2.279	11.149	11.14786939
10 → 11	8.745	-4.839	-4.811611884
10 → 15	6.563	-7.468	-7.484946096
11 → 12	7.68	3.551	3.589426573
12 → 13	3.631	-3.106	-3.103265205
13 → 14	6.908	8.721	8.713996717
14 → 19	10.189	-11.16	-11.17538429
15 → 16	7.611	2.735	2.68264064
15 → 20	7.57	1.006	1.038531067
16 → 17	3.717	-6.614	-6.604361552
16 → 21	7.362	-1.01	-1.079736576
17 → 18	4.947	-0.298	-0.310614148
17 → 22	6.455	5.968	6.001197613
18 → 19	6.682	4.942	4.930443064
18 → 23	8.338	5.49	5.483160375
19 → 24	2.75	1.137	1.128091499
20 → 21	8.15	0.496	0.564372996
21 → 22	6.857	0.484	0.476572638
22 → 23	4.027	-0.845	-0.828651385
23 → 24	5.672	0.557	0.575374188

(4) 各待定点高程平差值及中误差：

测点编号	待定点高程平差值(m)	中误差(mm)
0 -基准	216.118	0
1	209.2371196	30.28832045
2	201.0613916	40.53662958
3	205.7897998	43.8353071
4	208.4869119	51.19886904
5	211.8135474	27.66835203
6	210.2574315	32.31880445
7	214.0059871	38.97626245
8	203.243666	41.51604478

9	205.0660918	44.9007366
10	211.825415	31.53959756
11	207.0138031	36.6014645
12	210.6032297	41.32357728
13	207.4999645	43.44361801
14	216.2139612	44.03931884
15	204.3404689	30.56031136
16	207.0231096	37.23173731
17	200.418748	40.7900309
18	200.1081339	44.77168531
19	205.0385769	45.73215747
20 -基准	205.379	0
21	205.943373	34.98787441
22	206.4199456	41.6975629
23	205.5912942	44.8726939
24	206.1666684	47.71772422

3. 对结果的检核

(1) $B^T PV = 2.25657166\ 563753E-14$, 达到要求。

(2) 闭合环检核:

路线	闭合差
0→1→6→5→0	3.01981E-14
1→2→7→6→1	-4.13003E-14
2→3→8→7→2	0
3→4→9→8→3	1.02141E-14
5→6→11→10→5	-4.45997E-15
6→7→12→11→6	0
7→8→13→12→7	3.06422E-14
8→9→14→13→8	0
10→11→12→13→14→19→18→17→16→ 15→10	-4.08562E-14
15→16→21→20→15	4.21885E-15
16→17→22→21→16	6.21725E-15
17→18→23→22→17	0
18→19→24→23→18	0

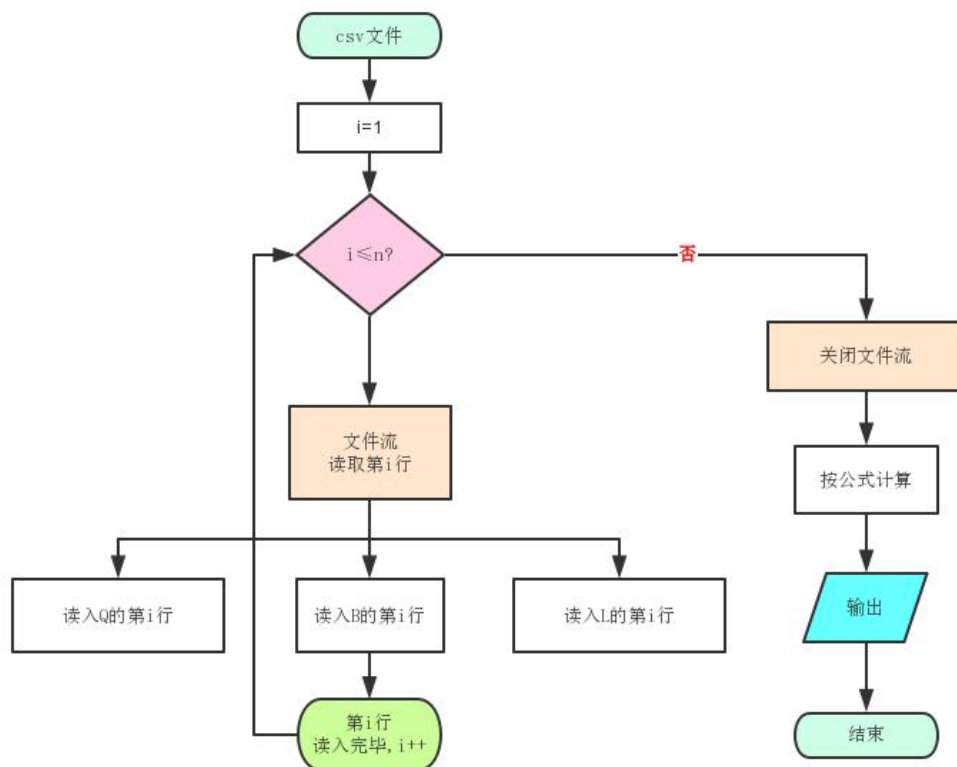
结果符合要求。

4. 计算框图与程序说明

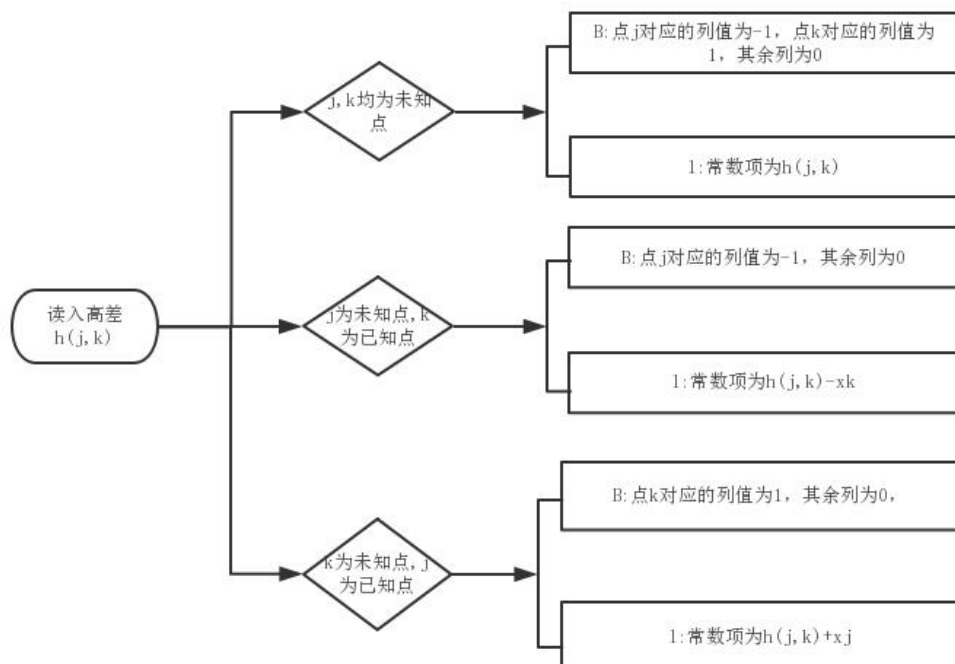
(1)程序说明：

本程序是用 **c#**语言写的，用了一个 **Matrix** 类来提供矩阵的基本运算，如乘法，求逆，转置等，主程序分成初始化模块，读取文件流模块，矩阵数值读取模块以及输出模块，其中为了方便，在读入矩阵的时候处理了一下点号问题。源代码见附录。

(1)程序运行框架：



(2)B, L 矩阵计算流程



四、导线网篇

1. 计算过程

本题共有 18 个测点，其中 3 个点已知，故必要观测数 $t=2*(18-3)=30$ ，共有 20 条边，其中一边已知，即观测了 19 条边的长度，同时观测了 26 个角的角度，故观测数 $n=19+26=45$ 。选取 15 个未知点的总共 30 个 x, y 坐标作为参数。

(1) 计算各个参数的近似值，以及每条观测边方位角的近似值。

i. 由于题中所给观测角度为左角，故对于观测角 $\angle h-j-k$ ，有：

$$\angle h-j-k = \alpha_{jk} - \alpha_{jh}, \text{if } (\alpha_{jk} - \alpha_{jh} > 0)$$

$$\angle h-j-k = \alpha_{jk} - \alpha_{jh} + 2\pi, \text{if } (\alpha_{jk} - \alpha_{jh} < 0)$$

ii. 坐标正算公式：

$$X_B = X_A + L_{AB} \cos(\alpha_{AB})$$

$$Y_B = Y_A + L_{AB} \sin(\alpha_{AB})$$

坐标反算公式：

设 $dY = Y_B - Y_A, dX = X_B - X_A, t = \arctan \frac{dY}{dX}$ ，则坐标方位角计算公式为：

①若 $dX > 0, dY > 0$, 则 $\alpha_{AB} = t$ ；

②若 $dX < 0, dY > 0$ 或 $dX < 0, dY < 0$ ，则 $\alpha_{AB} = t + \pi$ ；

③若 $dX > 0, dY < 0$ ，则 $\alpha_{AB} = t + 2\pi$ 。

iii. 从已知边 10-11 出发，依次先计算点 12, 13... 的近似坐标及其相关边的近似方位角。先算得 10 到 11 的方位角 $\alpha_{10,11}$ ，再依据上述公式 i 和 ii 依次计算近似坐标和相关边近似方位角。

(2) 取单位权中误差为测角先验误差，即 $\sigma_0^2 = \sigma_\beta^2 = 144''^2$ ，由此即得：

$$p_{S_i} = \frac{144}{(0.5S_i)^2} (\text{秒}^2/\text{mm}^2), \quad p_{\beta_i} = 1$$

其中 S_i 为这条边的观测值长度。

(3) 取前面 19 行为边观测值相关，后 26 行为角观测值相关，计算系数矩阵 B 与常数项矩阵 L。先明确矩阵 B 前 19 行单位为 1，后 26 行单位为秒/mm；矩阵 L 前 19 行单位为 mm，后 26 行单位为秒。

i. 先计算前 19 行的有关值：

假设第 i 个观测值为点 j 到点 k 的边长 L_{jk} ，则其满足方程：

$$\hat{L}_{jk} = \sqrt{(\hat{X}_j - \hat{X}_k)^2 + (\hat{Y}_j - \hat{Y}_k)^2}$$

将其按泰勒公式展开，线性化得：

$$L_{jk} + V_i = S_{jk}^0 + \frac{X_k - X_j}{S_{jk}^0} (\hat{x}_k - \hat{x}_j) + \frac{Y_k - Y_j}{S_{jk}^0} (\hat{y}_k - \hat{y}_j)$$

其中： $S_{jk}^0 = \sqrt{(X_j - X_k)^2 + (Y_j - Y_k)^2}$ ，再令： $l_i = L_i - S_{jk}^0$ ，可得测边的误差方

程为：

$$v_i = -\frac{X_k - X_j}{S_{jk}^0} \hat{x}_j - \frac{Y_k - Y_j}{S_{jk}^0} \hat{y}_j + \frac{X_k - X_j}{S_{jk}^0} \hat{x}_k + \frac{Y_k - Y_j}{S_{jk}^0} \hat{y}_k - l_i$$

然后可分情况讨论：

①若 j,k 均为未知点，则误差方程如上。

②若 j 为已知点，k 为未知点，则 $\hat{x}_j = \hat{y}_j = 0$ ，得：

$$v_i = \frac{X_k - X_j}{S_{jk}^0} \hat{x}_k + \frac{Y_k - Y_j}{S_{jk}^0} \hat{y}_k - l_i$$

③若 k 为已知点，j 为未知点，则 $\hat{x}_k = \hat{y}_k = 0$ ，得：

$$v_i = -\frac{X_k - X_j}{S_{jk}^0} \hat{x}_j - \frac{Y_k - Y_j}{S_{jk}^0} \hat{y}_j - l_i$$

④若 k, j 均为已知点，则不需要列误差方程。

ii.再计算后 26 行的有关值：

假设第 i 个观测值为 $\angle h-j-k$ ，则其满足观测方程：

$$\hat{L}_i = \hat{\alpha}_{jk} - \hat{\alpha}_{jh} + \mu * 2\pi$$

其中若 $\hat{\alpha}_{jk} - \hat{\alpha}_{jh} > 0$ ，则 $\mu = 0$ ，否则 $\mu = 1$ 。又因：

$$\hat{\alpha}_{jk} = \arctan \frac{\hat{Y}_k - \hat{Y}_j}{\hat{X}_k - \hat{X}_j} + C * \pi$$

$$\hat{\alpha}_{jh} = \arctan \frac{\hat{Y}_h - \hat{Y}_j}{\hat{X}_h - \hat{X}_j} + C * \pi$$

其中 C 可取 0, 1 或 2，故将其泰勒展开线性化可以得到：

$$\begin{aligned} v_i = & \rho'' \left(\frac{Y_k - Y_j}{(S_{jk}^0)^2} - \frac{Y_h - Y_j}{(S_{jh}^0)^2} \right) \hat{x}_j - \rho'' \left(\frac{X_k - X_j}{(S_{jk}^0)^2} - \frac{X_h - X_j}{(S_{jh}^0)^2} \right) \hat{y}_j - \rho'' \frac{Y_k - Y_j}{(S_{jk}^0)^2} \hat{x}_k \\ & + \rho'' \frac{X_k - X_j}{(S_{jk}^0)^2} \hat{y}_k + \rho'' \frac{Y_h - Y_j}{(S_{jh}^0)^2} \hat{x}_h - \rho'' \frac{X_h - X_j}{(S_{jh}^0)^2} \hat{y}_h - l_i \end{aligned}$$

其中 $l_i = L_i - L_i^0$ ，其中 L_i^0 可由(1)中公式 i 以及方位角近似值算出。

然后分情况讨论，也即若 t 是已知点，则 $\hat{x}_t = \hat{y}_t = 0$ ，其中 t 可为 i, j, k。

(4)精度评定中，观测值平差值的协因数阵为：

$$Q_{LL}^{\wedge\wedge} = B N_{BB}^{-1} B^T$$

2. 最后结果

(1)平差后单位权中误差为 14.2739"。

(2)各个点的坐标平差值及中误差：

坐标点号	x 平差值(m)	y 平差值(m)	x 改正数(m)	y 改正数(m)	中误差(mm)
0	302.809130	208.572860	-0.000050	-0.000183	23.396784
1	300.420247	249.708052	-0.000074	0.000005	20.966504
2	263.330467	264.727142	-0.000367	0.000042	21.078239
3	234.717	307.224	0	0	0
4	173.583004	294.766519	0.000038	-0.000027	27.740874
5	123.956368	266.615144	-0.000127	-0.000128	21.088992
6	102.536612	233.483414	-0.000070	-0.000053	24.706074
7	102.665206	205.486482	-0.000065	-0.000029	26.260969
8	118.920361	159.231063	-0.000062	-0.000012	28.001030
9	140.850577	110.406324	-0.000053	-0.000008	14.722627
10	169.287	113.116	0	0	0
11	233.641	112.029	0	0	0
12	253.488198	128.851803	-0.000084	-0.000270	12.780173
13	292.399566	168.996774	-0.000081	-0.000298	22.930647
14	165.902184	233.537403	-0.000177	-0.000066	27.987840
15	199.427484	199.016769	-0.000183	-0.000040	22.022701
16	218.125103	154.587443	-0.000107	-0.000259	23.389458
17	252.865165	211.489424	-0.000174	-0.000046	28.376315

(3)各观测值的平差值：

表一边长观测值的平差值：

边编号	边观测值 (m)	边平差值 (m)	边改正数 (m)
0 -- 1	41.209	41.20449889	-0.004501108
1 -- 2	40.021	40.01530764	-0.005692359

2 -- 3	51.24	51.2319574	-0.008042597
3 -- 4	62.428	62.39033767	-0.037662327
4 -- 5	57.073	57.05526206	-0.017737941
5 -- 6	39.472	39.45272478	-0.01927522
6 -- 7	28	27.99722705	-0.00277295
7 -- 8	49.023	49.02850125	0.005501248
8 -- 9	53.513	53.52372802	0.010728025
9 -- 10	28.551	28.56523205	0.014232046
10 -- 11	已知值	64.36317957	0
11 -- 12	26.008	26.01764741	0.009647409
12 -- 13	55.903	55.9080786	0.005078602
13 -- 0	40.924	40.92218949	-0.001810507
5 -- 14	53.407	53.41898923	0.011989226
14 -- 15	48.107	48.12088854	0.013888543
15 -- 16	48.176	48.20338127	0.027381271
16 -- 12	43.726	43.73638864	0.010388639
15 -- 17	54.878	54.8739723	-0.0040277
17 -- 1	61.012	61.00942049	-0.002579515

表二角度观测值的平差值：

角编号	角观测值(rad)	角平差值(rad)	角改正数(")
∠0-1-2	4.26962017	4.26962352	0.691033622
∠1-0-13	2.826345842	2.826382936	7.651367897
∠0-1-17	5.331393242	5.331355386	-7.808340918
∠1-2-3	2.54808552	2.548139246	11.08173219
∠2-1-17	1.061782354	1.061731866	-10.41405628
∠2-3-4	4.320768916	4.3208239	11.34128158
∠3-4-5	3.456527302	3.456572081	9.236356826
∠4-5-6	3.622496589	3.622475592	-4.331076754
∠4-5-14	5.099389886	5.099444477	11.26021312
∠5-6-7	3.720118386	3.720098549	-4.091628551
∠6-5-14	1.47697671	1.476968885	-1.613949648
∠6-7-8	3.474966669	3.474940309	-5.437172321
∠7-8-9	3.225846633	3.22580871	-7.822217958
∠8-9-10	4.385284597	4.385234235	-10.38793053
∠9-10-11	3.029752373	3.029701249	-10.54496742
∠10-11-12	3.861621894	3.86159039	-6.498166221
∠11-12-13	3.23951165	3.239485272	-5.44079336
∠11-12-16	4.950972207	4.950965799	-1.321658865
∠12-13-0	3.654227027	3.654187032	-8.249539912
∠13-12-16	1.711463056	1.711480527	3.603715951

∠5-14-15	3.009251554	3.00930838	11.72116068
∠14-15-16	2.769111259	2.769162315	10.53123809
∠14-15-17	4.170903259	4.170917649	2.968197651
∠15-16-12	3.684955644	3.684936395	-3.970344959
∠16-15-17	1.40167914	1.401755334	15.71619327
∠15-17-1	3.589177657	3.589267362	18.50291323

(4)平差后最弱边为 3-4，其相对中误差为 0.000433011。

3. 对结果的检核

(1)角度闭合差检核：

路线	角度闭合差(")
1->2->3->4->5->14->15->17->1	1.98772E-08
1->17->15->16->12->13->0->1	1.48392E-08
10->9->8->7->6->5->14->15->16->12-> 11->10	-1.4656E-08

结果非常令人满意。

(2)程序计算得 $B^T PV = 0$ ，结果令人满意。

(3)用边长检核(即带入误差方程)：

边编号	边平差值 (m)	根据坐标计算值(m)	差值 m
0 -- 1	41.20449889	41.20449889	-1.67688E-12
1 -- 2	40.01530764	40.01530764	-7.1239E-11
2 -- 3	51.2319574	51.2319574	-7.70328E-10
3 -- 4	62.39033767	62.39033767	-9.26548E-12
4 -- 5	57.05526206	57.05526206	0
5 -- 6	39.45272478	39.45272478	-1.58451E-12
6 -- 7	27.99722705	27.99722705	-4.05009E-13
7 -- 8	49.02850125	49.02850125	-3.48166E-13
8 -- 9	53.52372802	53.52372802	-1.54188E-12
9 -- 10	28.56523205	28.56523205	-1.13687E-13
10 -- 11	64.36317957	64.36317957	0
11 -- 12	26.01764741	26.01764741	-4.43606E-10
12 -- 13	55.9080786	55.9080786	-3.76588E-12
13 -- 0	40.92218949	40.92218949	-6.03961E-13
5 -- 14	53.41898923	53.41898923	-2.8848E-12
14 -- 15	48.12088854	48.12088854	-1.59872E-12
15 -- 16	48.20338127	48.20338127	-3.11928E-12
16 -- 12	43.73638864	43.73638864	-9.23706E-14

15 -- 17	54.8739723	54.8739723	-3.76588E-13
17 -- 1	61.00942049	61.00942049	-4.41247E-12

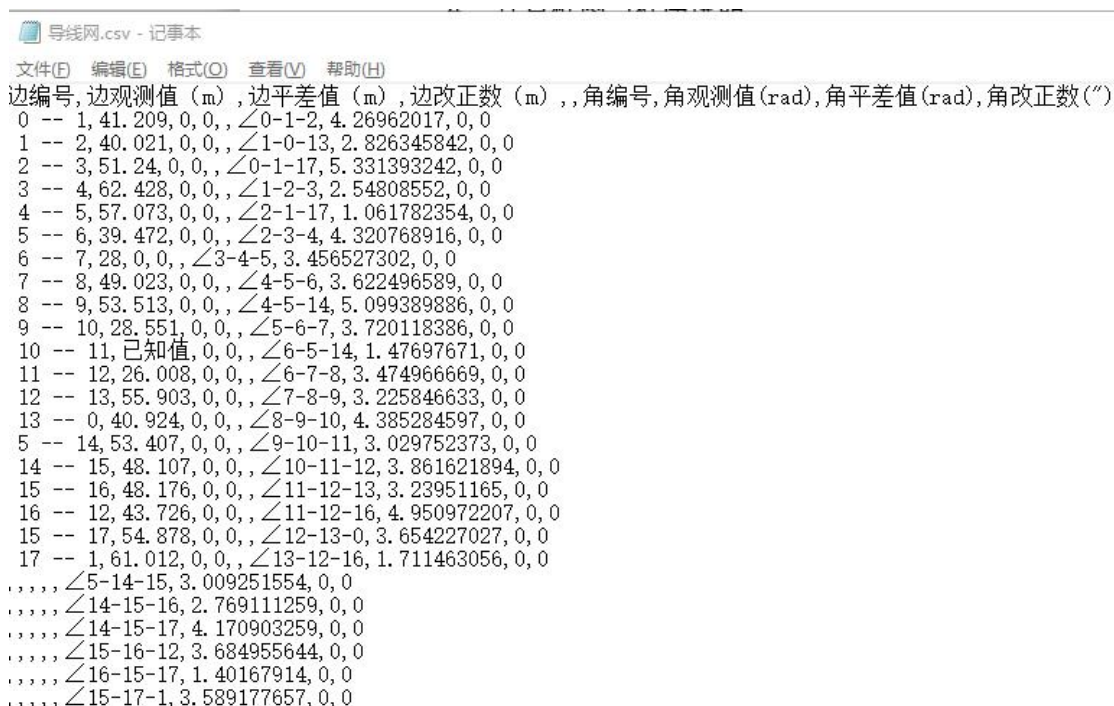
结果令人满意。

4. 计算框图与程序说明

(1) 程序说明：

本程序是用 **c#** 语言编写的。与水准网的程序共用了一个矩阵 **Matrix** 类，分为文件流部分，近似值计算部分，读入矩阵部分，计算部分，输出部分等，并且定义了一些小函数以实现一些特殊功能，人为定义了迭代次数为 100。源代码见附录。

csv 文件如图所示：

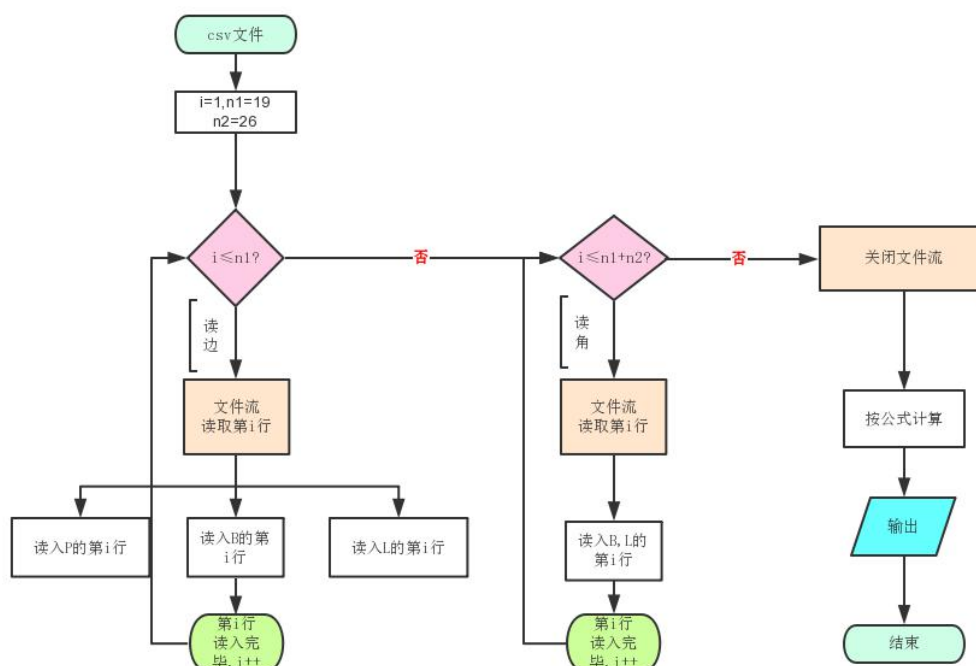


```

导线网.csv - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
边编号,边观测值(m),边平差值(m),边改正数(m),角编号,角观测值(rad),角平差值(rad),角改正数(度)
0 -- 1, 41.209, 0, 0, , 角0-1-2, 4.26962017, 0, 0
1 -- 2, 40.021, 0, 0, , 角1-0-13, 2.826345842, 0, 0
2 -- 3, 51.24, 0, 0, , 角0-1-17, 5.331393242, 0, 0
3 -- 4, 62.428, 0, 0, , 角1-2-3, 2.54808552, 0, 0
4 -- 5, 57.073, 0, 0, , 角2-1-17, 1.061782354, 0, 0
5 -- 6, 39.472, 0, 0, , 角2-3-4, 4.320768916, 0, 0
6 -- 7, 28, 0, 0, , 角3-4-5, 3.456527302, 0, 0
7 -- 8, 49.023, 0, 0, , 角4-5-6, 3.622496589, 0, 0
8 -- 9, 53.513, 0, 0, , 角4-5-14, 5.099389886, 0, 0
9 -- 10, 28.551, 0, 0, , 角5-6-7, 3.720118386, 0, 0
10 -- 11, 已知值, 0, 0, , 角6-5-14, 1.47697671, 0, 0
11 -- 12, 26.008, 0, 0, , 角6-7-8, 3.474966669, 0, 0
12 -- 13, 55.903, 0, 0, , 角7-8-9, 3.225846633, 0, 0
13 -- 0, 40.924, 0, 0, , 角8-9-10, 4.385284597, 0, 0
5 -- 14, 53.407, 0, 0, , 角9-10-11, 3.029752373, 0, 0
14 -- 15, 48.107, 0, 0, , 角10-11-12, 3.861621894, 0, 0
15 -- 16, 48.176, 0, 0, , 角11-12-13, 3.23951165, 0, 0
16 -- 12, 43.726, 0, 0, , 角11-12-16, 4.950972207, 0, 0
15 -- 17, 54.878, 0, 0, , 角12-13-0, 3.654227027, 0, 0
17 -- 1, 61.012, 0, 0, , 角13-12-16, 1.711463056, 0, 0
,,,,, 角5-14-15, 3.009251554, 0, 0
,,,,, 角14-15-16, 2.769111259, 0, 0
,,,,, 角14-15-17, 4.170903259, 0, 0
,,,,, 角15-16-12, 3.684955644, 0, 0
,,,,, 角16-15-17, 1.40167914, 0, 0
,,,,, 角15-17-1, 3.589177657, 0, 0

```


(2)程序框图：



五、实习中的心得体会

本次实习共做了水准网平差和导线网平差两个程序，加深了我对两种网型特别是对导线网计算的理解。

一开始我就打算用 c++ 或者 c# 来编程，但是考虑了一下还是选择了 c#，因为我并没有系统地学习过 c++ 特别是其中面向对象的部分，但是 c# 上个学期刚刚上过课，比较熟，之所以没打算用 matlab 是感觉 matlab 编程没有那种编程的感觉。我心里明确知道这些程序得引进一个矩阵 Matrix 类，并且至少要实现乘法，转置和求逆运算，于是我便去网上搜索 c# 编写的 Matrix 类的源代码，并将其稍作改动后在两个程序中都添加了这个 Matrix 类。

有了 Matrix 类后，下一步工作便是要能够读取 excel 文件。我百度了几种方

法，但是很多方法都是用了一种我并不熟悉的类，于是我选择了将 excel 文件转换成.csv 格式，并用读文本文件 txt 一样的方法来读取这个 csv 文件。这样的话我就能用上学期所学的流的有关知识来读取文件了。在读取过程中，我想要文件流的指针回到开头，我发现光用个 `sr.BaseStream.Seek(0, SeekOrigin.Begin)` 还不行，百度后知道还得加个 `sr.DiscardBufferedData()`，虽然我也不知道啥原理，感觉只是把缓冲区清空了，但起码问题得到了解决。在读取的时候，有一些奇怪的符号，比如这个“ \angle ”，我思考了下，直接用字符串的 `replace` 函数把这个替换成了空格，然后就可以用 `Trim` 函数直接把空格去掉，省去了麻烦。

然后便是先着手准备构建水准网的计算流程了。我最先就把最容易求的权阵 P 的计算方法给解决了，无非就是跟距离成反比。

经过思考我发现，由于水准网不需要线性化，故可以直接取参数近似值为 0，结果算得的参数改正数可直接作为参数平差值。同时每一个误差方程，对应在程序里面的时候，无非只是给系数矩阵和常数项矩阵这一行的某一些列赋上特定的值，而且由于每个未知点都有参数对应，所以对于每一个观测值都能用它的两个端点来列出误差方程。列了几个方程后发现，在这种思路下， B 矩阵的每一个值只能是 -1, 0, 1，并且 L 矩阵和 B 矩阵所需要讨论的情况是一样的，于是问题得到大大的简化，可以很轻松地用 if 语句来实现各种情况下 B, L 矩阵这些列的赋值。

有了计算出来的 B, P, L 矩阵之后，按照公式一步步算得改正数 x 和 V 就很轻松，然后根据精度评定的公式一步步求出单位权中误差，参数平差值的协因数阵等等都是水到渠成。

编写计算水准网的程序，利用间接平差是比较简单的。而且我把我的数据给同学计算了，跟我自己算出来的是一模一样的，于是我就放心了。但是水准网

给老师检核的分数只有 81，因为几个同学算我的结果都跟我自己的一模一样，而且不止我一个同学 80 分左右，所以我觉得可能是因为原始数据导出导入的时候出了误差，而评分软件评分标准又太严导致的。

导线网就没有水准网那么简单易编写了。首先因为我数测的时候导线网就没太学好，上平差的时候又不考导线网所以也没怎么注意，而且这个给的网型不是普通的一条线路或者是一个环，是一个自由网，所以一看到这个导线网的时候我是一点想法都没有。然后通过书上 117 页到书 151 页这些模型，例题，我大概花了一天多的时间才慢慢理清导线网计算的思路。就是先算近似值，再用测边测角网函数模型线性化来求 B, L ，同时还有个定权问题也困扰了我一会儿才理清楚， $1/2000$ 的测边误差想了想才知道应该是测量 1m 边长对应的误差是 5mm ，然后就算出各观测值的权。虽然近似值我没想出通用的程序，是手算出来的，也想办法把外面一圈的用程序读出来了，但是后面的程序还是通用的。

值得一提的是，这个导线网模型要特别注意单位，我特意在草稿纸上把各个矩阵的单位推了一遍，方便在读入 B 和 L 的时候注意了单位转换。

一段时间的实习，我感觉编程就是集模型演算，资料搜集，Bug 调试等于一身的体力活。

六、附录

附录一：矩阵类源代码

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```

namespace dxw
{
    class Matrix
    {
        public double[,] m;
        public int length, width;
        public Matrix(int l, int w)
        {
            length = l; width = w;
            m = new double[l, w];
        }
        public double this[int x, int y]
        {
            get
            {
                return m[x, y];
            }
            set
            {
                m[x, y] = value;
            }
        }
        public int Length
        {
            get { return length; }
        }
        public int Width
        {
            get { return width; }
        }
        public static Matrix operator +(Matrix a, Matrix b)
        {
            if (a.Width != b.Width || a.Length != b.Length)
            {
                return null;
            }
            Matrix c = new Matrix(a.Length, a.Width);
            for (int i = 0; i < a.Length; i++)
            {
                for (int j = 0; j < a.Width; j++)
                {
                    c[i, j] = a[i, j] + b[i, j];
                }
            }
        }
    }
}

```

```

        return c;
    }

    public static Matrix operator -(Matrix a, Matrix b)
    {
        if (a.Width != b.Width || a.Length != b.Length)
        {
            return null;
        }
        Matrix c = new Matrix(a.Length, a.Width);
        for (int i = 0; i < a.Length; i++)
        {
            for (int j = 0; j < a.Width; j++)
            {
                c[i, j] = a[i, j] - b[i, j];
            }
        }
        return c;
    }

    public static Matrix operator *(Matrix a, Matrix b)
    {
        //Console.WriteLine(" a:{0}X{1}  b:{2}X{3}", a.Length, a.Width, b.Length,
b.Width);

        if (a.Width != b.Length)
        {
            // Console.WriteLine("error a:{0}X{1}  b:{2}X{3}", a.Length, a.Width,
b.Length, b.Width);
            return null;
        }
        Matrix c = new Matrix(a.Length, b.Width);
        for (int i = 0; i < c.Length; i++)
        {
            for (int j = 0; j < c.Width; j++)
            {
                c[i, j] = 0;
                for (int k = 0; k < a.Width; k++)
                {
                    c[i, j] += a[i, k] * b[k, j];
                }
            }
        }
        return c;
    }

```

```

}

//求逆
public static Matrix inv(Matrix m)
{
    if (m.Length != m.Width)
    {
        return null;
    }
    //clone
    Matrix a = new Matrix(m.Length, m.Width);
    for (int i = 0; i < a.Length; i++)
    {
        for (int j = 0; j < a.Width; j++)
        {
            a[i, j] = m[i, j];
        }
    }
    Matrix c = new Matrix(a.Length, a.Width);
    for (int i = 0; i < a.Length; i++)
    {
        for (int j = 0; j < a.Width; j++)
        {
            if (i == j) { c[i, j] = 1; }
            else { c[i, j] = 0; }
        }
    }

    //i 表示第几行, j 表示第几列
    for (int j = 0; j < a.Length; j++)
    {
        bool flag = false;
        for (int i = j; i < a.Length; i++)
        {
            if (a[i, j] != 0)
            {
                flag = true;
                double temp;
                //交换 i, j, 两行
                if (i != j)
                {
                    for (int k = 0; k < a.Length; k++)
                    {
                        temp = a[j, k];

```

```

        a[j, k] = a[i, k];
        a[i, k] = temp;

        temp = c[j, k];
        c[j, k] = c[i, k];
        c[i, k] = temp;
    }
}

//第 j 行标准化
double d = a[j, j];
for (int k = 0; k < a.Length; k++)
{
    a[j, k] = a[j, k] / d;
    c[j, k] = c[j, k] / d;
}

//消去其他行的第 j 列
d = a[j, j];
for (int k = 0; k < a.Length; k++)
{
    if (k != j)
    {
        double t = a[k, j];
        for (int n = 0; n < a.Length; n++)
        {
            a[k, n] -= (t / d) * a[j, n];
            c[k, n] -= (t / d) * c[j, n];
        }
    }
}

}

if (!flag) return null;
}

return c;
}

public void print()
{
    for (int i = 0; i < length; i++)
    {
        for (int j = 0; j < width; j++)
        {
            Console.Write(m[i, j].ToString("0.00") + " ");
        }
    }
}

```

```

        Console.WriteLine("");
    }
    Console.WriteLine();
}
public static Matrix transposs(Matrix A)
{
    int i = 0;
    int j = 0;
    Matrix B = new Matrix(A.width, A.length);
    //运算
    for (i = 0; i < B.length; i++)
    {
        for (j = 0; j < B.width; j++)
        {
            B.m[i, j] = A.m[j, i];
        }
    }

    return B;
}
}
}

```

附录二：水准网源代码

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace szw
{
    class Program
    {
        static void Main(string[] args)
        {
            //定义观测值、必要观测值、多余观测量个数等各种数

```



```

const int n = 37; const int t = 23; const int r = n - t;
double m0;//单位权中误差
const int N1 = 0; const int N2 = 20;//基准点
double H1 = 216.118; double H2 = 205.379;//基准点高程, 单位 m
int minPointIndex;//精度最弱点号
double mMinPoint;//精度最弱点中误差
double[] height = new double[n];//高差

//初始化各类矩阵
Matrix B = new Matrix(n, t);
Matrix V = new Matrix(n, 1);
Matrix x = new Matrix(t, 1);
Matrix L = new Matrix(n, 1);
Matrix P = new Matrix(n, n);
Matrix Q = new Matrix(n, n);
Matrix NBB = new Matrix(t, t);
Matrix W = new Matrix(t, 1);
Matrix Qxx = new Matrix(t, t);

//导入数据至矩阵
FileStream fs = new FileStream("E:\\曹臻个人\\2017301610095\\水准网 2. csv",
FileMode.Open, FileAccess.Read, FileShare.None);
StreamReader sr = new StreamReader(fs,
System.Text.Encoding.GetEncoding(936));
string str = "";
sr.ReadLine();
for (int i = 0; i < n; i++)
{
    str = sr.ReadLine();
    string[] PrimaryStr = new String[100];
    PrimaryStr = str.Split(',');
    //协因数阵 Q 的构造
    Q.m[i, i] = Convert.ToDouble(PrimaryStr[1]);
    //处理参数点号并构造 B
    //同时构造 L, 其中 t 个参数近似值均取 0
    string[] num = PrimaryStr[0].Split(new char[2] { '-',
'>' }, StringSplitOptions.RemoveEmptyEntries);
    height[i] = Convert.ToDouble(PrimaryStr[2]);//读取高差观测值
    int num1 = int.Parse(num[0].Trim());
    int num2 = int.Parse(num[1].Trim());//读取文件中的点号
    L.m[i, 0] = height[i];
    //B 矩阵中每一行点号在上面的为-1, 后面的为 1, 其余对应的数值为 0
    //已知点点号除外
    //若两点均未知, L 矩阵对应行中的值即为高差, 否则需要根据点的前后做调整

```

```

        if (num1 != N1 && num1 != N2)
        {
            if (num1 > N1 && num1 < N2) num1 -= 1; else if (num1 > N2) num1 -= 2;
            B.m[i, num1] = -1;
        }
        else if (num1 == N1) L.m[i, 0] = height[i] + H1;
        else if (num1 == N2) L.m[i, 0] = height[i] + H2;
        if (num2 != N1 && num2 != N2)
        {
            if (num2 > N1 && num2 < N2) num2 -= 1; else if (num2 > N2) num2 -= 2;
            B.m[i, num2] = 1;
        }
        else if (num2 == N1) L.m[i, 0] = height[i] - H1;
        else if (num2 == N2) L.m[i, 0] = height[i] - H2;
    }
    sr.Close();

    //进行计算
    P = Matrix.inv(Q);
    NBB = Matrix.transposs(B) * P * B;
    W = Matrix.transposs(B) * P * L;
    x = Matrix.inv(NBB) * W;
    V = B * x - L;
    m0 = Math.Sqrt(((Matrix.transposs(V) * P * V).m[0, 0]) / r);
    Qxx = Matrix.inv(NBB);
    //找出精度最低点的点号
    double max = Qxx.m[0, 0];
    minPointIndex = 0;
    for (int i = 0; i < t; i++)
    {
        if(Qxx.m[i, i]>max)
        {
            max = Qxx.m[i, i]; minPointIndex = i;
        }
    }
    mMinPoint = m0 * Math.Sqrt(Qxx.m[minPointIndex, minPointIndex]); //精度最弱点

    中误差
    Console.WriteLine("1 公里高差观测值的中误差为: {0}mm", 1000*m0);
    Console.WriteLine("最弱点中误差: {0}mm", 1000 * mMinPoint);
    Console.WriteLine("各未知点的高程平差值及中误差为: ");
    for (int i = 0; i < t; i++)
    {

```

```

        Console.WriteLine("{0}m {1}mm", x.m[i, 0], 1000 * m0 * Math.Sqrt(Qxx.m[i,
i]));
    }
    Console.WriteLine("高差平差值为:");
    for (int i = 0; i < n; i++)
    {
        Console.WriteLine("{0}m", V.m[i, 0]+height[i]);
    }
    Console.WriteLine("{0}", (Matrix.transposs(B) * P * V).m[0, 0]); //检核
    Console.ReadKey();
}
}
}

```

附录三：导线网源代码

```

namespace dxw
{
    class Program
    {
        const int N1 = 3, N2 = 10, N3 = 11; //三个已知点标号
        static int[] NN = new int[] { N1, N2, N3 };
        static int PointDeal(int k)
        {
            if (k < N2 && k > N1) k -= 1;
            else if (k < N3 && k > N2) k -= 2;
            else if (k > N3) k -= 3;
            return k;
        }
        static double ComputeL(double a1, double a2)
        {
            if (a1 - a2 >= 0) return (a1 - a2);
            else return (a1 - a2 + 2 * Math.PI);
        }

        static double InvAzimuth(double ang) //反方位角计算
        {
            if (ang < Math.PI) ang += Math.PI;
            else if (ang >= Math.PI) ang -= Math.PI;
            return ang;
        }

        static double ComputeAzimuth(double x1, double y1, double x2, double y2) //计算方位
        角
    }
}

```

```

{
    double detY = y2 - y1, detX = x2 - x1;
    double t = Math.Atan(detY / detX);
    if (detY > 0 && detX > 0) return t;
    else if (detX > 0 && detY < 0) return (t + 2 * Math.PI);
    else return (t + Math.PI);
}

static void Main(string[] args)
{
    //定义观测值、必要观测值、多余观测量个数等各种数
    const int n1 = 19; const int n2 = 26; //n1 个边观测值, n2 个角观测值
    const int n3 = 18; //点个数
    const int n = 45; const int t = 30; const int r = n - t;
    double m0; //后验单位权中误差
    double mB = 12; //测角先验误差 12"
    double mD = 0.5; //测边先验误差 1/2000m, 即 1m 误差为 0.5mm
    const double x1 = 234.717, y1 = 307.224, x2 = 169.287, y2 = 113.116, x3 = 233.641,
    y3 = 112.029; //三个已知点坐标(单位: m), 其中 N3=N2+1, 即边 N2N3 为已知边
    double[] length=new double[n1]; //存储 n1 个边观测值
    double[, ] Length = new double[n3, n3]; //存储边观测值方便计算近似值
    int[, ] number1 = new int[n1, 2]; //储存 n1 个边的编号(起码有一个点为未知点)
    double[] ang=new double[n2]; //存储 n2 个角观测值
    double[, , ] Ang = new double[n3, n3, n3]; //存储角观测值方便计算近似值
    int[, ] number2 = new int[n2, 3]; //存储 n2 个角的名称编号, 如角 123
    double[] Xm = new double[t /
2] { 302.7957, 300.4014, 263.3049, 173.5139, 123.8708, 102.4403, 102.5686, 118.8243, 140.7522, 25
3.4806, 292.3873, 165.8046, 199.3172, 217.9976, 252.7601 };
    double[] Ym = new double[t / 2] { 208.5665,
249.7059, 264.7234, 294.7579, 266.5995, 233.4518, 205.4521, 159.2027, 110.3886, 128.8458, 168.9
883, 233.5258, 199.0122, 154.6054, 211.4802 }; //存储 t 个参数的近似值, 这里 Xm=[X0 X1 X2... ]T,
Ym=[Y0 Y1 Y2... ]T
    double[, ] azimuth = new double[t/2+3, t/2+3]; //方位角近似值
    int minSideIndex; //精度最弱点编号
    double mMinSide; //精度最弱边中误差
    const double p = 180 * 60 * 60 / Math.PI;
    //初始化各类矩阵
    Matrix B = new Matrix(n, t); //前面 t/2 列为坐标参数 x, 后面为坐标参数 y
    Matrix V = new Matrix(n, 1);
    Matrix x = new Matrix(t, 1);
    Matrix L = new Matrix(n, 1);
    Matrix P = new Matrix(n, n);
    Matrix Q = new Matrix(n, n);

```

```

Matrix NBB = new Matrix(t, t);
Matrix W = new Matrix(t, 1);
Matrix Qxx = new Matrix(t, t);
Matrix QLL = new Matrix(n, n);

//读取原始数据
{
    //先读边长

    FileStream fs = new FileStream("E:\\曹臻个人\\2017301610095\\导线网.csv",
    FileMode.Open, FileAccess.Read, FileShare.None);
    StreamReader sr = new StreamReader(fs,
    System.Text.Encoding.GetEncoding(936));
    string str = "";
    sr.ReadLine();
    for (int i = 0; i < n1; i++)
    {
        str = sr.ReadLine();
        string[] PrimiryStr = new String[100];
        PrimiryStr = str.Split(',');
        if (PrimiryStr[1] == "已知值")
        {
            i--;
            continue;
        };//读到已知边则跳出本层循环
        length[i] = Convert.ToDouble(PrimiryStr[1]);//读取边长(单位 m)
        string[] num = PrimiryStr[0].Split(new char[2] { '-', '-' },
    StringSplitOptions.RemoveEmptyEntries);
        number1[i, 0] = int.Parse(num[0].Trim());
        number1[i, 1] = int.Parse(num[1].Trim());//读取边编号
        Length[number1[i, 0], number1[i, 1]] = length[i];
        Length[number1[i, 1], number1[i, 0]] = length[i];
    }
    //读边完成

    //再读取角度及其对应的角编号
    sr.BaseStream.Seek(0, SeekOrigin.Begin);
    sr.DiscardBufferedData();//返回第一行
    sr.ReadLine();
    for (int i = 0; i < n2; i++)
    {
        str = sr.ReadLine();
        string[] PrimiryStr = new String[100];

```

```

        PrimiryStr = str.Split(',');
        ang[i] = Convert.ToDouble(PrimiryStr[6]); //读取角度(单位 rad)
        string[] num = PrimiryStr[5].Split('-');
        number2[i, 0] = int.Parse(num[0].Replace(num[0][0], ' ').Trim()); //
去除角编号前的'∠', 这样能把数字给导出到 number 中
        number2[i, 1] = int.Parse(num[1]);
        number2[i, 2] = int.Parse(num[2]); //读取角编号
        Ang[number2[i, 0], number2[i, 1], number2[i, 2]] = ang[i];
        Ang[number2[i, 2], number2[i, 1], number2[i, 0]] = 2*Math.PI - ang[i];
    }
    //读取完成
    sr.Close();
}

//求坐标参数近似值和方位角近似值
{
    azimuth[N2, N3] = ComputeAzimuth(x2, y2, x3, y3);
    if (InvAzimuth(azimuth[N3 - 1, N3]) + Ang[N3 - 1, N3, N3 + 1] < 2 * Math.PI)
        azimuth[N3, N3 + 1] = Ang[N3 - 1, N3, N3 + 1] + InvAzimuth(azimuth[N3
- 1, N3]);
    else azimuth[N3, N3 + 1] = Ang[N3 - 1, N3, N3 + 1] + InvAzimuth(azimuth[N3
- 1, N3]) - 2 * Math.PI;
    Xm[PointDeal(N3 + 1)] = x3 + Length[N3, N3 + 1] * Math.Cos(azimuth[N3, N3
+ 1]);
    Ym[PointDeal(N3 + 1)] = y3 + Length[N3, N3 + 1] * Math.Sin(azimuth[N3, N3
+ 1]);

    for (int i = N3 + 1; i != N1 - 1; i++) //从 N2, N3 边开始求, 先求到点 N1 的
导线
    {
        int j = i + 1, k = i - 1;
        if (i > 13) i -= 14;
        if (j > 13) j -= 14;
        if (InvAzimuth(azimuth[k, i]) + Ang[k, i, j] < 2 * Math.PI)
            azimuth[i, j] = Ang[k, i, j] + InvAzimuth(azimuth[k, i]);
        else azimuth[i, j] = Ang[k, i, j] + InvAzimuth(azimuth[k, i]) - 2 *
Math.PI;
        Xm[PointDeal(j)] = Xm[PointDeal(i)] + Length[i, j] *
Math.Cos(azimuth[i, j]);
        Ym[PointDeal(j)] = Ym[PointDeal(i)] + Length[i, j] *
Math.Sin(azimuth[i, j]);
    }
    if (InvAzimuth(azimuth[N1 - 2, N1 - 1]) + Ang[N1 - 2, N1 - 1, N1] < 2 * Math.PI)
        azimuth[N1 - 1, N1] = Ang[N1 - 2, N1 - 1, N1] + InvAzimuth(azimuth[N1
- 2, N1 - 1]);
}

```

```

        else azimuth[N1 - 1, N1] = Ang[N1 - 2, N1 - 1, N1] + InvAzimuth(azimuth[N1
- 2, N1 - 1]) - 2 * Math.PI;
        for (int i = N1; i != N2 - 1; i++)//再从点 N1 开始求, 求到点 N2 的导线
        {
            int j = i + 1, k = i - 1;
            if (i > 13) i -= 14;
            if (j > 13) j -= 14;
            if (InvAzimuth(azimuth[k, i]) + Ang[k, i, j] < 2 * Math.PI)
                azimuth[i, j] = Ang[k, i, j] + InvAzimuth(azimuth[k, i]);
            else azimuth[i, j] = Ang[k, i, j] + InvAzimuth(azimuth[k, i]) - 2 *
Math.PI;

            Xm[PointDeal(j)] = Xm[PointDeal(i)] + Length[i, j] *
Math.Cos(azimuth[i, j]);
            Ym[PointDeal(j)] = Ym[PointDeal(i)] + Length[i, j] *
Math.Sin(azimuth[i, j]);
        }
        if (InvAzimuth(azimuth[N2 - 2, N2 - 1]) + Ang[N2 - 2, N2 - 1, N2] < 2 * Math.PI)
            azimuth[N2 - 1, N2] = Ang[N2 - 2, N2 - 1, N2] + InvAzimuth(azimuth[N2
- 2, N2 - 1]);
        else azimuth[N2 - 1, N2] = Ang[N2 - 2, N2 - 1, N2] + InvAzimuth(azimuth[N2
- 2, N2 - 1]) - 2 * Math.PI;
        //再求中间边的方位角与坐标近似值, 此处为手算
        azimuth[5, 14] = 5.6154;
        azimuth[14, 15] = 5.4831;
        azimuth[15, 16] = 5.1106;
        azimuth[16, 12] = 5.6552;
        azimuth[15, 17] = 0.2292;
        azimuth[17, 1] = 0.6762;
    }

    //读入反方位角
    for (int i = 0; i < t / 2 + 3; i++)
    {
        for (int j = 0; j < t / 2 + 3; j++)
        {
            while (azimuth[i, j] > 2 * Math.PI) azimuth[i, j] -= 2 * Math.PI;
            if (azimuth[i, j] != 0) azimuth[j, i] = InvAzimuth(azimuth[i, j]);
        }
    }

    for (int i = 0; i < t / 2 + 3; i++)
    {
        for (int j = 0; j < t / 2 + 3; j++)
        {

```

```

        Console.WriteLine("{0} {1} {2}", i, j, azimuth[i, j]);
    }
}

//读入 P 矩阵
for (int i = 0; i < n1; i++)
{
    P.m[i, i] = (mB * mB) / (mD * length[i] * mD * length[i]);
} //边观测值在前面的行, 单位: 秒2/mm2
for (int i = n1; i < n; i++)
{
    P.m[i, i] = 1;
} //角观测值在后面的行
//P 矩阵读入完成

//迭代开始
for (int count = 0; count < 100; count++)
{

    //读入 B 与 L 矩阵
    //先读前面 n1 条边观测值形成的数值
    for (int i = 0; i < n1; i++)
    {
        int j = number1[i, 0]; int k = number1[i, 1];
        double S = new double(); //近似边长
        //处理点号问题, 并在不同的情况下读入 B, L 矩阵
        if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, k) == -1) //j, k
        两点均为未知点时
        {
            j = PointDeal(j);
            k = PointDeal(k);
            S = Math.Sqrt((Xm[j] - Xm[k]) * (Xm[j] - Xm[k]) + (Ym[j] - Ym[k])
            * (Ym[j] - Ym[k])); //近似距离
            B.m[i, j] = (Xm[j] - Xm[k]) / S; B.m[i, k] = -(Xm[j] - Xm[k]) / S;
            B.m[i, j + t / 2] = (Ym[j] - Ym[k]) / S; B.m[i, k + t / 2] = -(Ym[j]
            - Ym[k]) / S; //单位 1
            L.m[i, 0] = 1000 * (length[i] - S); //单位 mm
        }
        else if (Array.IndexOf(NN, j) != -1 && Array.IndexOf(NN, k) == -1) //j
        已知而 k 未知时
        {
            k = PointDeal(k);
            if (j == N1)

```



```

{
    S = Math.Sqrt((x1 - Xm[k]) * (x1 - Xm[k]) + (y1 - Ym[k]) * (y1
- Ym[k]));

    B.m[i, k] = -(x1 - Xm[k]) / S;
}
else if (j == N2)
{
    S = Math.Sqrt((x2 - Xm[k]) * (x2 - Xm[k]) + (y2 - Ym[k]) * (y2
- Ym[k]));

    B.m[i, k] = -(x2 - Xm[k]) / S;
}
else if (j == N3)
{
    S = Math.Sqrt((x3 - Xm[k]) * (x3 - Xm[k]) + (y3 - Ym[k]) * (y3
- Ym[k]));

    B.m[i, k] = -(x3 - Xm[k]) / S;
}
L.m[i, 0] = 1000 * (length[i] - S); //单位 mm
}
else if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, k) != -1) //j
未知而 k 已知时
{

    j = PointDeal(j);
    if (k == N1)
    {
        S = Math.Sqrt((x1 - Xm[j]) * (x1 - Xm[j]) + (y1 - Ym[j]) * (y1
- Ym[j]));

        B.m[i, j] = -(x1 - Xm[j]) / S;
    }
    else if (k == N2)
    {
        S = Math.Sqrt((x2 - Xm[j]) * (x2 - Xm[j]) + (y2 - Ym[j]) * (y2
- Ym[j]));

        B.m[i, j] = -(x2 - Xm[j]) / S;
    }
    else if (k == N3)
    {
        S = Math.Sqrt((x3 - Xm[j]) * (x3 - Xm[j]) + (y3 - Ym[j]) * (y3
- Ym[j]));

        B.m[i, j] = -(x3 - Xm[j]) / S;
    }
    L.m[i, 0] = 1000 * (length[i] - S); //单位 mm
}
}

```

```

    }
    //前 n1 行读取结束
    //再读取后面 n2 个角观测值形成的数值
    for (int i = n1; i < n; i++)
    {
        int h, j, k;
        h = number2[i - n1, 0]; j = number2[i - n1, 1]; k = number2[i - n1, 2];
        double S1, S2 = new double(); //S1 为 hj 距离的平方, S2 为 jk 距离的平方
        if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, k) == -1 &&
Array.IndexOf(NN, h) == -1) //当 h, j, k 均为未知点时
        {
            int h1 = h, k1 = k, j1 = j;
            j = PointDeal(j);
            k = PointDeal(k);
            h = PointDeal(h);
            S1 = (Xm[j] - Xm[h]) * (Xm[j] - Xm[h]) + (Ym[j] - Ym[h]) * (Ym[j]
- Ym[h]);
            S2 = (Xm[j] - Xm[k]) * (Xm[j] - Xm[k]) + (Ym[j] - Ym[k]) * (Ym[j]
- Ym[k]);

            //B 将角度系数的单位化成" /mm;
            B.m[i, j] = p * ((Ym[k] - Ym[j]) / S2 - (Ym[h] - Ym[j]) / S1) /
1000; //Xj 前面系数
            B.m[i, j + t / 2] = -p * ((Xm[k] - Xm[j]) / S2 - (Xm[h] - Xm[j])
/ S1) / 1000; //Yj 前面系数
            B.m[i, k] = -p * (Ym[k] - Ym[j]) / S2 / 1000; //Xk 前面系数
            B.m[i, k + t / 2] = p * (Xm[k] - Xm[j]) / S2 / 1000; //Yk 前面系
数
            B.m[i, h] = p * (Ym[h] - Ym[j]) / S1 / 1000; //Xh 前面系数
            B.m[i, h + t / 2] = -p * (Xm[h] - Xm[j]) / S1 / 1000; //Yh 前面系
数
            L.m[i, 0] = p * (ang[i - n1] - ComputeL(azimuth[j1, k1], azimuth[j1,
h1])); //L, 单位: "
        }
        else if (Array.IndexOf(NN, j) != -1 && Array.IndexOf(NN, k) == -1 &&
Array.IndexOf(NN, h) == -1) //当 j 已知而 h, k 未知时
        {
            int h1 = h, k1 = k, j1 = j;
            k = PointDeal(k);
            h = PointDeal(h);
            L.m[i, 0] = p * (ang[i - n1] - ComputeL(azimuth[j1, k1], azimuth[j1,
h1])); //L, 单位: "

            if (j == N1)
            {
                S1 = (x1 - Xm[h]) * (x1 - Xm[h]) + (y1 - Ym[h]) * (y1 - Ym[h]);

```

```

S2 = (x1 - Xm[k]) * (x1 - Xm[k]) + (y1 - Ym[k]) * (y1 - Ym[k]);
B.m[i, k] = -p * (Ym[k] - y1) / S2 / 1000; //Xk 前面系数
B.m[i, k + t / 2] = p * (Xm[k] - x1) / S2 / 1000; //Yk 前面系
数

B.m[i, h] = p * (Ym[h] - y1) / S1 / 1000; //Xh 前面系数
B.m[i, h + t / 2] = -p * (Xm[h] - x1) / S1 / 1000; //Xh 前面
系数

}
else if (j == N2)
{
S1 = (x2 - Xm[h]) * (x2 - Xm[h]) + (y2 - Ym[h]) * (y2 - Ym[h]);
S2 = (x2 - Xm[k]) * (x2 - Xm[k]) + (y2 - Ym[k]) * (y2 - Ym[k]);
B.m[i, k] = -p * (Ym[k] - y2) / S2 / 1000; //Xk 前面系数
B.m[i, k + t / 2] = p * (Xm[k] - x2) / S2 / 1000; //Yk 前面系
数

B.m[i, h] = p * (Ym[h] - y2) / S1 / 1000; //Xh 前面系数
B.m[i, h + t / 2] = -p * (Xm[h] - x2) / S1 / 1000; //Xh 前面
系数

}
else if (j == N3)
{
S1 = (x3 - Xm[h]) * (x3 - Xm[h]) + (y3 - Ym[h]) * (y3 - Ym[h]);
S2 = (x3 - Xm[k]) * (x3 - Xm[k]) + (y3 - Ym[k]) * (y3 - Ym[k]);
B.m[i, k] = -p * (Ym[k] - y3) / S2 / 1000; //Xk 前面系数
B.m[i, k + t / 2] = p * (Xm[k] - x3) / S2 / 1000; //Yk 前面系
数

B.m[i, h] = p * (Ym[h] - y3) / S1 / 1000; //Xh 前面系数
B.m[i, h + t / 2] = -p * (Xm[h] - x3) / S1 / 1000; //Xh 前面
系数

}
}
else if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, k) != -1 &&
Array.IndexOf(NN, h) == -1) //当 k 已知而 h, j 未知时
{
int h1 = h, k1 = k, j1 = j;
h = PointDeal(h);
j = PointDeal(j);
L.m[i, 0] = p * (ang[i - n1] - ComputeL(azimuth[j1, k1], azimuth[j1,
h1])); //L, 单位: "

if (k == N1)
{
S1 = (Xm[j] - Xm[h]) * (Xm[j] - Xm[h]) + (Ym[j] - Ym[h]) * (Ym[j]
- Ym[h]);

S2 = (Xm[j] - x1) * (Xm[j] - x1) + (Ym[j] - y1) * (Ym[j] - y1);

```

```

        B.m[i, j] = p * ((y1 - Ym[j]) / S2 - (Ym[h] - Ym[j]) / S1) /
1000;//Xj 前面系数
        B.m[i, j + t / 2] = -p * ((x1 - Xm[j]) / S2 - (Xm[h] - Xm[j])
/ S1) / 1000;//Yj 前面系数
        B.m[i, h] = p * (Ym[h] - Ym[j]) / S1 / 1000;//Xh 前面系数
        B.m[i, h + t / 2] = -p * (Xm[h] - Xm[j]) / S1 / 1000;//Xh 前
面系数
    }
    else if (k == N2)
    {
        S1 = (Xm[j] - Xm[h]) * (Xm[j] - Xm[h]) + (Ym[j] - Ym[h]) * (Ym[j]
- Ym[h]);
        S2 = (Xm[j] - x2) * (Xm[j] - x2) + (Ym[j] - y2) * (Ym[j] - y2);
        B.m[i, j] = p * ((y2 - Ym[j]) / S2 - (Ym[h] - Ym[j]) / S1) /
1000;//Xj 前面系数
        B.m[i, j + t / 2] = -p * ((x2 - Xm[j]) / S2 - (Xm[h] - Xm[j])
/ S1) / 1000;//Yj 前面系数
        B.m[i, h] = p * (Ym[h] - Ym[j]) / S1 / 1000;//Xh 前面系数
        B.m[i, h + t / 2] = -p * (Xm[h] - Xm[j]) / S1 / 1000;//Xh 前
面系数
    }
    else if (k == N3)
    {
        S1 = (Xm[j] - Xm[h]) * (Xm[j] - Xm[h]) + (Ym[j] - Ym[h]) * (Ym[j]
- Ym[h]);
        S2 = (Xm[j] - x3) * (Xm[j] - x3) + (Ym[j] - y3) * (Ym[j] - y3);
        B.m[i, j] = p * ((y3 - Ym[j]) / S2 - (Ym[h] - Ym[j]) / S1) /
1000;//Xj 前面系数
        B.m[i, j + t / 2] = -p * ((x3 - Xm[j]) / S2 - (Xm[h] - Xm[j])
/ S1) / 1000;//Yj 前面系数
        B.m[i, h] = p * (Ym[h] - Ym[j]) / S1 / 1000;//Xh 前面系数
        B.m[i, h + t / 2] = -p * (Xm[h] - Xm[j]) / S1 / 1000;//Xh 前
面系数
    }
}
else if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, k) == -1 &&
Array.IndexOf(NN, h) != -1)//当 h 已知而 k, j 未知时
{
    int h1 = h, k1 = k, j1 = j;
    k = PointDeal(k);
    j = PointDeal(j);
    L.m[i, 0] = p * (ang[i - n1] - ComputeL(azimuth[j1, k1], azimuth[j1,
h1])); //L, 单位: "
    if (h == N1)

```

```

        {
            S1 = (Xm[j] - x1) * (Xm[j] - x1) + (Ym[j] - y1) * (Ym[j] - y1);
            S2 = (Xm[j] - Xm[k]) * (Xm[j] - Xm[k]) + (Ym[j] - Ym[k]) * (Ym[j]
- Ym[k]);

            B.m[i, j] = p * ((Ym[k] - Ym[j]) / S2 - (y1 - Ym[j]) / S1) /
1000;//Xj 前面系数

            B.m[i, j + t / 2] = -p * ((Xm[k] - Xm[j]) / S2 - (x1 - Xm[j])
/ S1) / 1000;//Yj 前面系数

            B.m[i, k] = -p * (Ym[k] - Ym[j]) / S2 / 1000;//Xk 前面系数
            B.m[i, k + t / 2] = p * (Xm[k] - Xm[j]) / S2 / 1000;//Yk 前
面系数

        }
        else if (h == N2)
        {
            S1 = (Xm[j] - x2) * (Xm[j] - x2) + (Ym[j] - y2) * (Ym[j] - y2);
            S2 = (Xm[j] - Xm[k]) * (Xm[j] - Xm[k]) + (Ym[j] - Ym[k]) * (Ym[j]
- Ym[k]);

            B.m[i, j] = p * ((Ym[k] - Ym[j]) / S2 - (y2 - Ym[j]) / S1) /
1000;//Xj 前面系数

            B.m[i, j + t / 2] = -p * ((Xm[k] - Xm[j]) / S2 - (x2 - Xm[j])
/ S1) / 1000;//Yj 前面系数

            B.m[i, k] = -p * (Ym[k] - Ym[j]) / S2 / 1000;//Xk 前面系数
            B.m[i, k + t / 2] = p * (Xm[k] - Xm[j]) / S2 / 1000;//Yk 前
面系数

        }
        else if (h == N3)
        {
            S1 = (Xm[j] - x2) * (Xm[j] - x2) + (Ym[j] - y2) * (Ym[j] - y2);
            S2 = (Xm[j] - Xm[k]) * (Xm[j] - Xm[k]) + (Ym[j] - Ym[k]) * (Ym[j]
- Ym[k]);

            B.m[i, j] = p * ((Ym[k] - Ym[j]) / S2 - (y2 - Ym[j]) / S1) /
1000;//Xj 前面系数

            B.m[i, j + t / 2] = -p * ((Xm[k] - Xm[j]) / S2 - (x2 - Xm[j])
/ S1) / 1000;//Yj 前面系数

            B.m[i, k] = -p * (Ym[k] - Ym[j]) / S2 / 1000;//Xk 前面系数
            B.m[i, k + t / 2] = p * (Xm[k] - Xm[j]) / S2 / 1000;//Yk 前
面系数

        } }
        else if (Array.IndexOf(NN, j) != -1 && Array.IndexOf(NN, k) != -1 &&
Array.IndexOf(NN, h) == -1)//当 j,k 已知而 h 未知时
        {
            int h1 = h, k1 = k, j1 = j;
            h = PointDeal(h);
            L.m[i, 0] = p * (ang[i - n1] - ComputeL(azimuth[j1, k1], azimuth[j1,

```

```

h1]));//L,单位: "

if (j == N1)
{
    S1 = (x1 - Xm[h]) * (x1 - Xm[h]) + (y1 - Ym[h]) * (y1 - Ym[h]);
    B.m[i, h] = p * (Ym[h] - y1) / S1 / 1000;//Xh 前面系数
    B.m[i, h + t / 2] = -p * (Xm[h] - x1) / S1 / 1000;//Xh 前面
系数
}
else if (j == N2)
{
    S1 = (x2 - Xm[h]) * (x2 - Xm[h]) + (y2 - Ym[h]) * (y2 - Ym[h]);
    B.m[i, h] = p * (Ym[h] - y2) / S1 / 1000;//Xh 前面系数
    B.m[i, h + t / 2] = -p * (Xm[h] - x2) / S1 / 1000;//Xh 前面
系数
}
else if (j == N3)
{
    S1 = (x3 - Xm[h]) * (x3 - Xm[h]) + (y3 - Ym[h]) * (y3 - Ym[h]);
    B.m[i, h] = p * (Ym[h] - y3) / S1 / 1000;//Xh 前面系数
    B.m[i, h + t / 2] = -p * (Xm[h] - x3) / S1 / 1000;//Xh 前面
系数
}
}
else if (Array.IndexOf(NN, j) != -1 && Array.IndexOf(NN, k) == -1 &&
Array.IndexOf(NN, h) != -1)//当 j,h 已知而 k 未知时
{
    int h1 = h, k1 = k, j1 = j;
    k = PointDeal(k);
    L.m[i, 0] = p * (ang[i - n1] - ComputeL(azimuth[j1, k1], azimuth[j1,
h1]));//L,单位: "

if (j == N1)
{
    S2 = (x1 - Xm[k]) * (x1 - Xm[k]) + (y1 - Ym[k]) * (y1 - Ym[k]);
    B.m[i, k] = -p * (Ym[k] - y1) / S2 / 1000;//Xk 前面系数
    B.m[i, k + t / 2] = p * (Xm[k] - x1) / S2 / 1000;//Yk 前面系
数
}
else if (j == N2)
{
    S2 = (x2 - Xm[k]) * (x2 - Xm[k]) + (y2 - Ym[k]) * (y2 - Ym[k]);
    B.m[i, k] = -p * (Ym[k] - y2) / S2 / 1000;//Xk 前面系数
    B.m[i, k + t / 2] = p * (Xm[k] - x2) / S2 / 1000;//Yk 前面系
数
}
}

```

```

else if (j == N3)
{
    S2 = (x3 - Xm[k]) * (x3 - Xm[k]) + (y3 - Ym[k]) * (y3 - Ym[k]);
    B.m[i, k] = -p * (Ym[k] - y3) / S2 / 1000; //Xk 前面系数
    B.m[i, k + t / 2] = p * (Xm[k] - x3) / S2 / 1000; //Yk 前面系
数
}
}
else if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, k) != -1 &&
Array.IndexOf(NN, h) != -1) //当 h, k 已知而 j 未知时
{
    int h1 = h, k1 = k, j1 = j;
    h = PointDeal(h);
    k = PointDeal(k);
}
}
//B, L 矩阵读取完成

//进行计算
NBB = Matrix.transp(B) * P * B;
W = Matrix.transp(B) * P * L;
x = Matrix.inv(NBB) * W;
for (int i = 0; i < t/2; i++) //近似值获取
{
    Xm[i] = Xm[i] + x.m[i, 0] / 1000;
    Ym[i] = Ym[i] + x.m[i + t / 2, 0] / 1000;
}
for (int i = 0; i < t / 2 + 3; i++) //方位角近似值获取
{
    for (int j = 0; j < t / 2 + 3; j++)
    {
        if (azimuth[i, j] != 0)
        {
            if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, i) ==
-1) //i, j 未知
            {
                int i1 = i, j1 = j;
                int i2 = PointDeal(i), j2 = PointDeal(j);
                azimuth[i1, j1] = ComputeAzimuth(Xm[i2], Ym[i2], Xm[j2],
Ym[j2]);
            }
            else if (Array.IndexOf(NN, j) == -1 && Array.IndexOf(NN, i) !=
-1) //j 未知, i 已知
            {

```

```

        int i1 = i, j1 = j;
        int j2 = PointDeal(j);
        if(i==N1) azimuth[i1, j1] = ComputeAzimuth(x1, y1, Xm[j2],
Ym[j2]);

        else if(i==N2) azimuth[i1, j1] = ComputeAzimuth(x2, y2,
Xm[j2], Ym[j2]);

        else if(i==N3) azimuth[i1, j1] = ComputeAzimuth(x3, y3,
Xm[j2], Ym[j2]);
    }
    else if (Array.IndexOf(NN, j) != -1 && Array.IndexOf(NN, i) ==
-1)//i 未知, j 已知
    {
        int i1 = i, j1 = j;
        int i2 = PointDeal(i);
        if (j== N1) azimuth[i1, j1] = ComputeAzimuth(Xm[i2],
Ym[i2], x1,y1);

        else if (j == N2) azimuth[i1, j1] = ComputeAzimuth(Xm[i2],
Ym[i2], x2,y2);

        else if (j == N3) azimuth[i1, j1] = ComputeAzimuth(Xm[i2],
Ym[i2], x3,y3);
    }
}
}
}
}
//迭代完成

Q = Matrix.inv(P);
V = B * x - L;
m0 = Math.Sqrt(((Matrix.transpos(V) * P * V).m[0, 0]) / r);
Qxx = Matrix.inv(NBB);
QLL = B * Matrix.inv(NBB) * Matrix.transpos(B);
//找出边长精度最低点的点号, 也即边长最大点点号
double max = QLL.m[0, 0];
minSideIndex = 0;
for (int i = 0; i < n1; i++)
{
    if (QLL.m[i, i] > max)
    {
        max = QLL.m[i, i]; minSideIndex = i;
    }
}
mMinSide = m0 * Math.Sqrt(QLL.m[minSideIndex, minSideIndex]); //精度最弱边中
误差

```



```

Console.WriteLine("单位权中误差为: {0} ", m0);
Console.WriteLine("各个待定点坐标平差值和中误差为: ");
for (int i = 0; i < t/2; i++)
{
    double mx, my, mp;
    mx = m0 * Math.Sqrt(Qxx.m[i, i]); my = m0 * Math.Sqrt(Qxx.m[i + t / 2, i
+ t / 2]);
    mp = Math.Sqrt(mx * mx + my * my);
    Console.WriteLine("{0}m {1}m {2}mm
", Xm[i]+x.m[i, 0]/1000, Ym[i]+x.m[i+t/2, 0]/1000, mp);
}
Console.WriteLine("各观测值的平差值为: ");
Console.WriteLine("{0}条边观测值平差值: ", n1);
for (int i = 0; i < n1; i++)
{
    Console.WriteLine("{0}m", V.m[i, 0]/1000 + length[i]);
}
Console.WriteLine("{0}个角观测值平差值: ", n2);
for (int i = n1; i < n; i++)
{
    double an = V.m[i, 0] + p * ang[i - n1]; //单位秒
    Console.WriteLine("{0}° {1}' {2}" 即 {3} rad ",
(int) (an/3600), (int) (an/60-60*(int) (an/3600)), an-3600*(int) (an / 3600)-60 * (int) (an /
60 - 60 * (int) (an / 3600)), an/p);
}
Console.WriteLine("最弱边边长相对中误差为: {0}",
mMinSide/(1000*length[minSideIndex]));
//Console.WriteLine( "{0}", (Matrix.transposs(B) * P * V).m[0, 0]);
Console.WriteLine("观测值改正数: ");
Console.WriteLine("边长观测值: ");
for (int i = 0; i < n1; i++)
{
    Console.WriteLine("{0}mm", V.m[i, 0] );
}
Console.WriteLine("角观测值: ", n2);
for (int i = n1; i < n; i++)
{
    Console.WriteLine("{0} " ", V.m[i, 0]);
}
Console.WriteLine("坐标改正数: ");
Console.WriteLine("x 坐标");
for (int i = 0; i < t/2; i++)
{
    Console.WriteLine("{0}mm", x.m[i, 0]);
}

```

```

    }
    Console.WriteLine("y 坐标");
    for (int i = t/2; i < t; i++)
    {
        Console.WriteLine("{0}mm", x.m[i, 0]);
    }
    Console.WriteLine("{0}", (Matrix.transposs(B)*P*V).m[0, 0]);
    Console.ReadKey();
}
}

```

综合评语：

设计题目 1		所占比例	30%
设计题目 2		所占比例	30%
报 告 成 绩		所占比例	40%
总 评 成 绩			
指导教师：			
2019 年 月 日			