

《误差理论与测量平差基础》

课程设计报告

学 院： 测绘科学与空间信息学院

专业班级： 测绘工程xxx

姓 名： xxx

学 号： xxx

指导教师： xxx

完成日期： xxx

教务处制

课程设计任务书示例:

**课 程 设 计 任 务 书**

一、课程设计题目：

《误差理论与测量平差基础》课程设计

二、课程设计主要参考资料：

（1）宋力杰. 测量平差程序设计[M].第1版.

（2）张书毕. 测量平差[M].第1版.

（3）武汉大学测绘学院平差学科组误差理论与测量平差基础

（4）隋立芬，宋力杰. 误差理论与测量平差基础

（5）姚连璧，周小平.《基于Matlab的控制网平差程序设计》

三、课程设计应解决的主要问题：

（1）用计算机进行编程实现具有一定规模的高程控制网、平面控制网和GPS网的通用平差程序。

（2）编写课程设计说明书，详细阐述测量平差程序的设计思路、编写、检测方法、关键问题和解决方法、运行主要结果。

四、课程设计相关附件（如：图纸、软件等）：

（1）原始数据、解算结果

（2）《误差理论与测量平差基础》平差报告

（3）程序设计相关源代码及exe可执行程序

五、任务发出日期： xxx

课程设计完成日期： xxx

**目 录**

**第1章 课程设计的目的和任务 3**

1.1 课程目的 3

1. 2 课程要求 3

1.3 课程题目 4

**第2章 数据结构及算法（C++） 5**

2.1 间接平差原理 5

2.2 数据文件组成 5

2.3 程序数据结构 6

2.4 程序设计流程 7

2.5 关键代码 8

2.6 完整代码 15

2.7 程序运行结果 16

**第3章 心得体会 16**

**第一章 课程设计的目的和任务**

**1.1 课程目的**

《误差理论与测量平差基础课程设计》是完成《数字地形测量学》、《误差理论与测量平差基础》等测绘专业基础课程教学后进行的一门实践课程，是测量数据处理理论学习的重要的实践环节。其目的是增强学生对课堂所学误差理论与测量平差基础理论的理解，牢固掌握测量平差的基本原理和基本公式，熟悉测量数据处理的基本流程和计算方法，灵活准确地将其应用于解决测绘工程中相关数据处理的实际问题；能用所学的计算机高级编程语言，如C++、Matlab等，针对测量平差中常见的高程控制网、导线网、GPS网等编制通用的测量平差程序，锻炼学生的编程能力，培养学生解决生产实际问题的能力和对所学测绘专业基本知识的综合应用能力。

**1.2 课程任务**

主要任务有以下两个方面：

1. 用计算机进行编程实现具有一定规模的高程控制网、平面控制网和GPS网的通用平差程序；结合具体的平差实例，设计已知数据、观测数据文件的格式与输入方式，根据各控制点间的联结关系，设计误差方程、法方程的系数阵组成，根据观测数据，设计权阵；选取C++、Matlab等计算机编程语言，设计通用平差程序，实现高程控制网、平面控制网和GPS网的平差过程。要求能够正确地提炼和表达复杂测绘工程问题的技术要点，分析影响因素，获得有效结论，能够准确地理解和把握复杂测绘工程问题或项目的特定需求，合理确定方案设计的具体目标。

2. 编写课程设计说明书，详细阐述测量平差程序的设计思路、编写、检测方法、关键问题和解决方法、运行主要结果和程序的正确性、完整性分析。具备撰写报告、设计文稿、陈述发言、清晰表达等基本技能，能够就复杂测绘工程问题与业界同行及社会公众进行沟通和交流。

**1.3 课程要求**

1.基于Matlab或其他编程语言（如C++等）编程实现GPS网间接平差过程通用程序，包括误差方程、法方程的组成与解算。得出平差后各基线向量观测值的平差值及各待定点的坐标平差值；评定各待定点坐标平差值的精度，各基线向量观测值的平差值的精度。给出程序设计思路、流程图、程序代码和计算结果；计算误差椭圆三要素，画出最终的GPS控制网和各待定点误差椭圆示意图。

2.将书上计算结果与计算机计算结果进行比对，检核验证程序编写是否正确。

**2.4 课程题目**

如下图所示为一简单GPS网，用两台GPS接收机观测，测得5条基线向量，每一条基线向量中三个坐标差观测值相关，由于只用两台GPS接收机观测，所以各观测基线向量互相独立。观测基线向量信息见下表。假定1号点为起算点，坐标信息见下表。

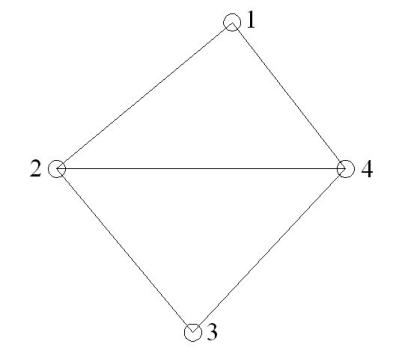


图 GPS网平差算例一

表 GPS网平差算例一观测数据

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编号 | 起点 | 终点 | △X | △Y | △Z | 基线方差阵 |
| 1 | LC02 | LC01 | -1218.561 | -1039.227 | 1737.720 |  |
| 2 | LC03 | LC02 | 1405.531 | -178.157 | 1171.380 |  |
| 3 | LC04 | LC03 | 83.497 | 714.153 | -1029.199 |  |
| 4 | LC04 | LC01 | 270.457 | -503.208 | 1879.923 |  |
| 5 | LC04 | LC02 | 1489.013 | 536.030 | 142.218 |  |

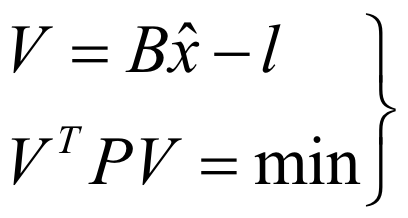
表 GPS网平差算例一的假定已知点数据

|  |  |  |  |
| --- | --- | --- | --- |
| 点号 | X | Y | Z |
| LC01 | -1974638.7340 | 4590014.8190 | 3953144.9235 |

**第二章 数据结构及算法（C++）**

**2.1 间接平差原理**

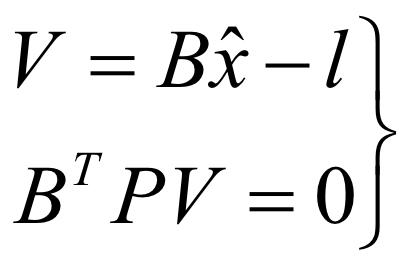
间接平差法是通过选定t个与观测值有一定关系的独立未知量作为参数，将每个观测值都分别表达成这t个参数的函数，建立函数模型，按最小二乘原理，用求函数极值的方法解出参数的最或然值，从而求得各观测值的平差值。

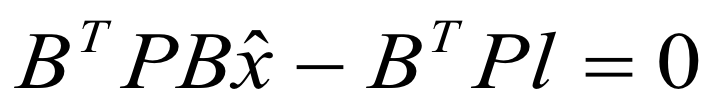


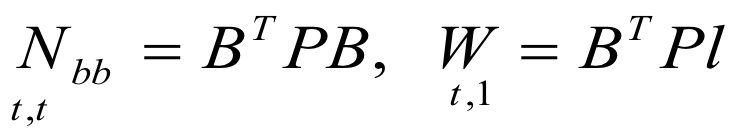


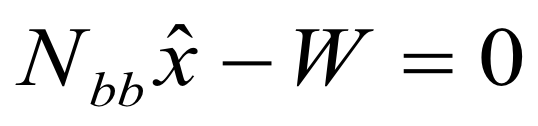


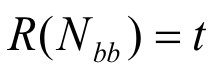


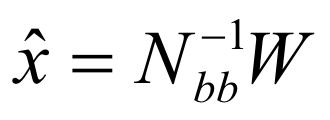
得基础方程



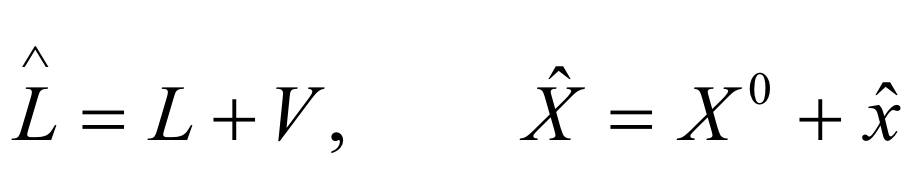


得法方程： 

为满秩矩阵，即,  有唯一解，得



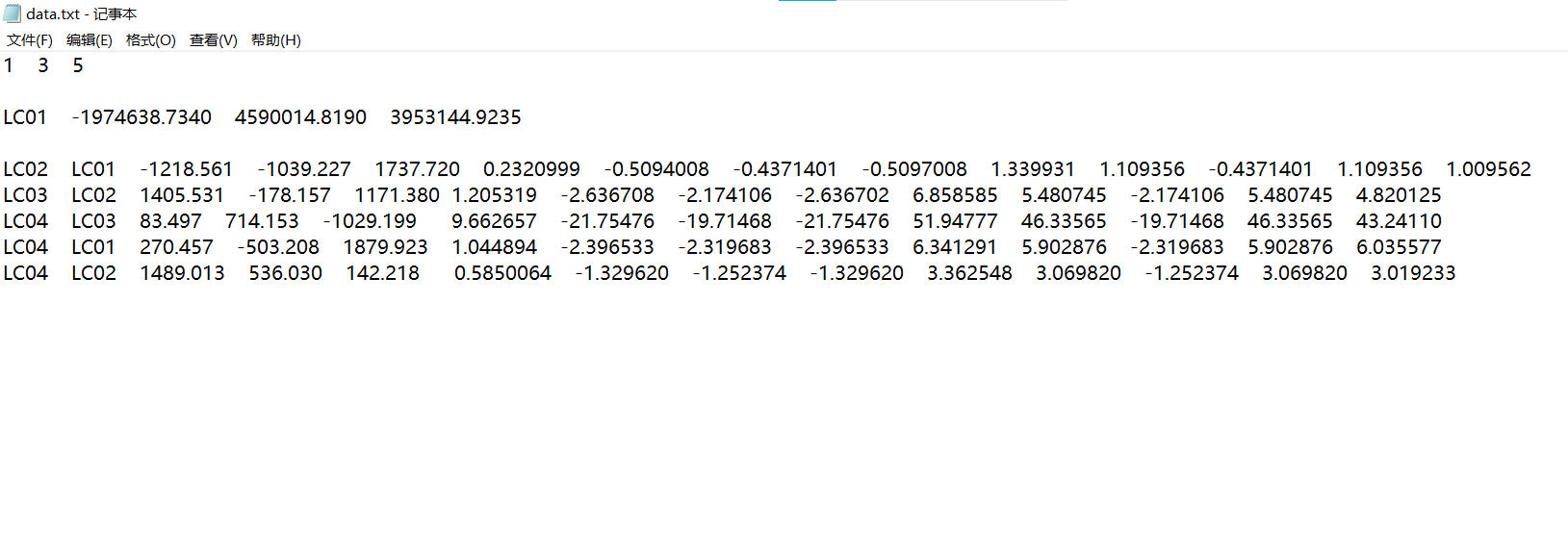
由，得改正数V

最终结果：

**2.2** **数据文件组成**

数据文件包括三个部分：文件头，已知点数据和基线数据。

首先，文件头只有一行，并以空格或制表符隔开，包括已知点数量、未知点数量、基线数量。然后是已知点数据行数与头文件中已知点数量相同，每一行存储一个点的点名，X坐标、Y坐标和Z坐标，以空格或制表符隔开。最后是基线数据，其行数与头文件中基线数量相同，包含起点点名、终点点名、X坐标差、Y坐标差、Z坐标差和基线方差阵，以空格或制表符隔开。如图所示：



**2.3 程序数据结构**

该程序设计了两个结构体Point、GPSline分别存储点数据和基线数据，此外还设计了一个GPSnet类，来存储GPS网络数据并提供相关解算函数。主要代码如下：

1. **struct** Point
2. {
3. string pointName;//点名
4. **int** pointNum = -1;//点号
5. **double** x = 0, y = 0, z = 0;//点的xyz坐标
6. **bool** known = **false**;//该点是否已知坐标或者近似坐标
7. };
9. **struct** GPSline
10. {
11. //基线结构体，包含起点、终点和基线的xyz测量坐标差
12. Point begin, end;
13. **double** dx, dy, dz;
14. };


18. //GPS网类
19. **class** GPSnet
20. {
21. **private**:
23. GPSline\* lines; //基线数组，存储每一条基线信息
24. Point\* knownPoints;//已知点数组，存储已知点信息
25. map<string, **int**>dic;//字典，key为点名，value为点号
26. MatrixXd B;//系数矩阵
27. MatrixXd l;//l
28. MatrixXd P;//权阵P
29. MatrixXd v;//改正数矩阵v
30. MatrixXd x;//
31. **int** known\_point\_num, nuknown\_point\_num, observation\_num;//已知点数目、未知点数目、基线数目

34. **public**:
36. GPSnet();//空白构造
37. **void** loadData(string filePath);//解算GPS网，需要传入数据文件路径
38. **void** print();//输出解算结果到控制台
39. **void** print(string filePath);//输出解算结果到文件
40. ~GPSnet();//析构函数
42. };

**2.4 程序设计流程**

**2.4 关键代码**

1. //1.读取数据文件
2. ifstream ifs(filePath);
3. //读取已知点数目、未知点数目、基线数目
4. ifs >> known\_point\_num >> nuknown\_point\_num >> observation\_num;
5. //初始化类的属性
6. lines = **new** GPSline[observation\_num];
7. knownPoints = **new** Point[known\_point\_num];
8. B = MatrixXd::Zero(observation\_num \* 3, nuknown\_point\_num \* 3);
9. l = MatrixXd::Random(observation\_num \* 3, 1);
10. P = MatrixXd::Zero(observation\_num \* 3, observation\_num \* 3);
11. //读取已知点信息到knownPoints数组和dic字典
12. **for** (**int** i = 0; i < known\_point\_num; i++)
13. {
14. ifs >> knownPoints[i].pointName >> knownPoints[i].x >> knownPoints[i].y >> knownPoints[i].z;
15. knownPoints[i].known = **true**;
16. knownPoints[i].pointNum = i;
17. dic[knownPoints[i].pointName] = i;
18. }
19. //读取基线数据，并计算出权阵P
20. **for** (**int** i = 0; i < observation\_num; i++)
21. {
22. ifs >> lines[i].begin.pointName >> lines[i].end.pointName
23. >> lines[i].dx >> lines[i].dy >> lines[i].dz;
24. lines[i].begin.known = **false**;
25. lines[i].end.known = **false**;
26. ifs >> P(i \* 3 + 0, i \* 3 + 0) >> P(i \* 3 + 0, i \* 3 + 1) >> P(i \* 3 + 0, i \* 3 + 2)
27. >> P(i \* 3 + 1, i \* 3 + 0) >> P(i \* 3 + 1, i \* 3 + 1) >> P(i \* 3 + 1, i \* 3 + 2)
28. >> P(i \* 3 + 2, i \* 3 + 0) >> P(i \* 3 + 2, i \* 3 + 1) >> P(i \* 3 + 2, i \* 3 + 2);
29. }
30. P = P.inverse();
31. //读取基线数据，并计算出权阵P
32. **for** (**int** i = 0; i < observation\_num; i++)
33. {
34. ifs >> lines[i].begin.pointName >> lines[i].end.pointName
35. >> lines[i].dx >> lines[i].dy >> lines[i].dz;
36. lines[i].begin.known = **false**;
37. lines[i].end.known = **false**;
38. ifs >> P(i \* 3 + 0, i \* 3 + 0) >> P(i \* 3 + 0, i \* 3 + 1) >> P(i \* 3 + 0, i \* 3 + 2)
39. >> P(i \* 3 + 1, i \* 3 + 0) >> P(i \* 3 + 1, i \* 3 + 1) >> P(i \* 3 + 1, i \* 3 + 2)
40. >> P(i \* 3 + 2, i \* 3 + 0) >> P(i \* 3 + 2, i \* 3 + 1) >> P(i \* 3 + 2, i \* 3 + 2);
41. }
42. P = P.inverse();



47. //为点编号
48. //向基线数组填入已知点数据
49. **for** (**int** i = 0; i < observation\_num; i++)
50. {
51. **if** (!(dic.find(lines[i].begin.pointName) == dic.end()))
52. {
53. lines[i].begin.known = **true**;
54. **for** (**int** j = 0; j < known\_point\_num; j++)
55. {
56. **if** (knownPoints[j].pointName == lines[i].begin.pointName)
57. {
58. lines[i].begin.x = knownPoints[j].x;
59. lines[i].begin.y = knownPoints[j].y;
60. lines[i].begin.z = knownPoints[j].z;
61. }
62. }
63. }
64. **if** (!(dic.find(lines[i].end.pointName) == dic.end()))
65. {
66. lines[i].end.known = **true**;
67. **for** (**int** j = 0; j < known\_point\_num; j++)
68. {
69. **if** (knownPoints[j].pointName == lines[i].end.pointName)
70. {
71. lines[i].end.x = knownPoints[j].x;
72. lines[i].end.y = knownPoints[j].y;
73. lines[i].end.z = knownPoints[j].z;
74. }
75. }
76. }
77. }
78. //然后为已知点、未知点编号（按数字由小到大，已知点在前，未知点在后）
79. **int** sortedNum = known\_point\_num - 1;
80. **for** (**int** i = 0; i < observation\_num; i++)
81. {
82. **if** (!(dic.find(lines[i].begin.pointName) == dic.end()))
83. lines[i].begin.pointNum = dic[lines[i].begin.pointName];
84. **else**
85. {
86. dic[lines[i].begin.pointName] = sortedNum + 1;
87. sortedNum++;
88. **for** (**int** i = 0; i < observation\_num; i++)
89. {
90. **if** (!(dic.find(lines[i].begin.pointName) == dic.end()))
91. lines[i].begin.pointNum = dic[lines[i].begin.pointName];
92. **if** (!(dic.find(lines[i].end.pointName) == dic.end()))
93. lines[i].end.pointNum = dic[lines[i].end.pointName];
94. }
95. }
96. **if** (!(dic.find(lines[i].end.pointName) == dic.end()))
97. lines[i].end.pointNum = dic[lines[i].end.pointName];
98. **else**
99. {
100. dic[lines[i].end.pointName] = sortedNum + 1;
101. sortedNum++;
102. **for** (**int** i = 0; i < observation\_num; i++)
103. {
104. **if** (!(dic.find(lines[i].begin.pointName) == dic.end()))
105. lines[i].begin.pointNum = dic[lines[i].begin.pointName];
106. **if** (!(dic.find(lines[i].end.pointName) == dic.end()))
107. lines[i].end.pointNum = dic[lines[i].end.pointName];
108. }
109. }
110. }
111. //计算近似坐标
112. **while** (**true**)
113. {
114. //循环结束标志，初值为true，若该次循环没有计算近似坐标，循环结束
115. **bool** success = **true**;
116. **for** (**int** i = 0; i < observation\_num; i++)
117. {
118. //第一种情况：起点坐标已知，终点未知，则终点坐标=起点坐标+Δ
119. **if** (lines[i].begin.known == **true** && lines[i].end.known == **false**)
120. {
121. lines[i].end.x = lines[i].begin.x + lines[i].dx;
122. lines[i].end.y = lines[i].begin.y + lines[i].dy;
123. lines[i].end.z = lines[i].begin.z + lines[i].dz;
124. //为GPS网络中，所有与该终点名称相同的点赋值
125. **for** (**int** j = 0; j < observation\_num; j++)
126. {
127. **if** (lines[j].end.pointName == lines[i].end.pointName)
128. {
129. lines[j].end.known = **true**;
130. lines[j].end.pointNum = lines[i].end.pointNum;
131. lines[j].end.x = lines[i].end.x;
132. lines[j].end.y = lines[i].end.y;
133. lines[j].end.z = lines[i].end.z;
134. }
135. **if** (lines[j].begin.pointName == lines[i].end.pointName)
136. {
137. lines[j].begin.known = **true**;
138. lines[j].begin.pointNum = lines[i].end.pointNum;
139. lines[j].begin.x = lines[i].end.x;
140. lines[j].begin.y = lines[i].end.y;
141. lines[j].begin.z = lines[i].end.z;
142. }
144. }
145. lines[i].end.known = **true**;
146. success = **false**;
147. }
148. //第二种情况：起点坐标未知，终点已知，则起点坐标=终点坐标-Δ
149. **if** (lines[i].begin.known == **false** && lines[i].end.known == **true**)
150. {
151. //为起点坐标赋值
152. lines[i].begin.x = lines[i].end.x - lines[i].dx;
153. lines[i].begin.y = lines[i].end.y - lines[i].dy;
154. lines[i].begin.z = lines[i].end.z - lines[i].dz;
155. //为GPS网络中，所有与该起点名称相同的点赋值
156. **for** (**int** j = 0; j < observation\_num; j++)
157. {
158. **if** (lines[j].begin.pointName == lines[i].begin.pointName)
159. {
160. lines[j].begin.known = **true**;
161. lines[j].begin.pointNum = lines[i].begin.pointNum;
162. lines[j].begin.x = lines[i].begin.x;
163. lines[j].begin.y = lines[i].begin.y;
164. lines[j].begin.z = lines[i].begin.z;
165. }
166. **if** (lines[j].end.pointName == lines[i].begin.pointName)
167. {
168. lines[j].end.known = **true**;
169. lines[j].end.pointNum = lines[i].begin.pointNum;
170. lines[j].end.x = lines[i].begin.x;
171. lines[j].end.y = lines[i].begin.y;
172. lines[j].end.z = lines[i].begin.z;
173. }
174. }
175. lines[i].begin.known = **true**;
176. success = **false**;
177. }
178. }
179. **if** (success == **true**)
180. **break**;
181. }

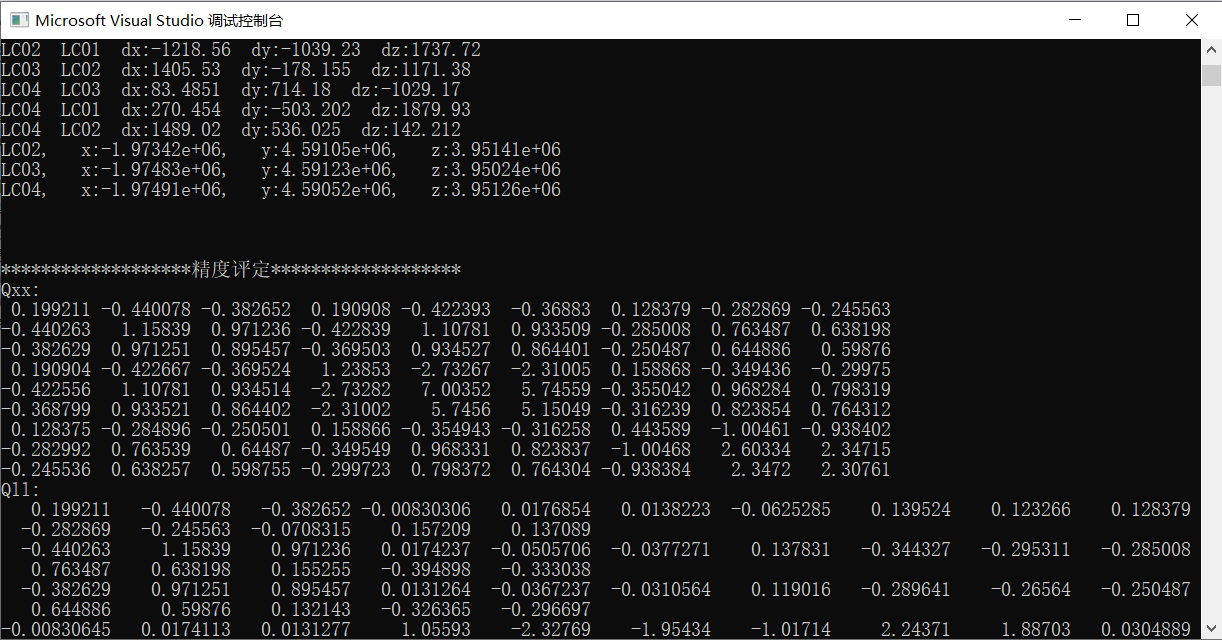

185. //计算B、l矩阵
186. **for** (**int** i = 0; i < observation\_num; i++)
187. {
188. **int** beginIndex = lines[i].begin.pointNum - (known\_point\_num - 1) - 1;
189. **int** endIndex = lines[i].end.pointNum - (known\_point\_num - 1) - 1;
190. **if** (beginIndex >= 0)
191. {
192. B(3 \* i + 0, 3 \* beginIndex + 0) = -1;
193. B(3 \* i + 1, 3 \* beginIndex + 1) = -1;
194. B(3 \* i + 2, 3 \* beginIndex + 2) = -1;
195. }
196. **if** (endIndex >= 0)
197. {
198. B(3 \* i + 0, 3 \* endIndex + 0) = 1;
199. B(3 \* i + 1, 3 \* endIndex + 1) = 1;
200. B(3 \* i + 2, 3 \* endIndex + 2) = 1;
201. }
202. l(3 \* i + 0, 0) = lines[i].dx - (lines[i].end.x - lines[i].begin.x);
203. l(3 \* i + 1, 0) = lines[i].dy - (lines[i].end.y - lines[i].begin.y);
204. l(3 \* i + 2, 0) = lines[i].dz - (lines[i].end.z - lines[i].begin.z);
205. }
206. //输出解算结果到控制台
207. **void** GPSnet::print()
208. {
209. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*平差结果\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
210. cout << "P:" << endl << P << endl;
211. cout << "B:" << endl << B << endl;
212. cout << "l:" << endl << l << endl;
213. cout << "x:" << endl << x << endl;
214. cout << "v:" << endl << v << endl;
215. **for** (**int** i = 0; i < observation\_num; i++)
216. {
217. cout << lines[i].begin.pointName << "  " << lines[i].end.pointName << "  ";
218. cout << "dx:" << lines[i].dx + v(i \* 3 + 0,0) << "  dy:" << lines[i].dy + v(i \* 3 + 1,0) << "  dz:" << lines[i].dz + v(i \* 3 + 2,0) << endl;
219. }
220. map<string, **int**>temp;
221. **int** index = 0;
222. **for** (**int** i = 0; i < known\_point\_num; i++)
223. {
224. temp[knownPoints[i].pointName] = 0;
225. }
226. **for** (**int** i = 0; i < observation\_num; i++)
227. {
228. **if** ((temp.find(lines[i].begin.pointName) == temp.end()))
229. {
230. cout << lines[i].begin.pointName << ",   x:" << lines[i].begin.x + x(index,0)
231. << ",   y:" << lines[i].begin.y + x(index + 1,0) << ",   z:" << lines[i].begin.z + x(index + 2,0) << endl;
232. index += 3;
233. temp[lines[i].begin.pointName] = 1;
234. }
235. **if** ((temp.find(lines[i].end.pointName) == temp.end()))
236. {
237. cout << lines[i].end.pointName << ",   x:" << lines[i].end.x + x(index,0)
238. << ",   y:" << lines[i].end.y + x(index + 1,0) << ",   z:" << lines[i].end.z + x(index + 2,0) << endl;
239. index += 3;
240. temp[lines[i].end.pointName] = 1;
241. }
242. }
243. cout << "\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*精度评定\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
244. cout << "Qxx:" << endl << (B.transpose()\*P\*B).inverse() << endl;
245. cout << "Qll:" << endl<<B\* (B.transpose()\*P\*B).inverse() \*B.transpose()<<endl;
246. }

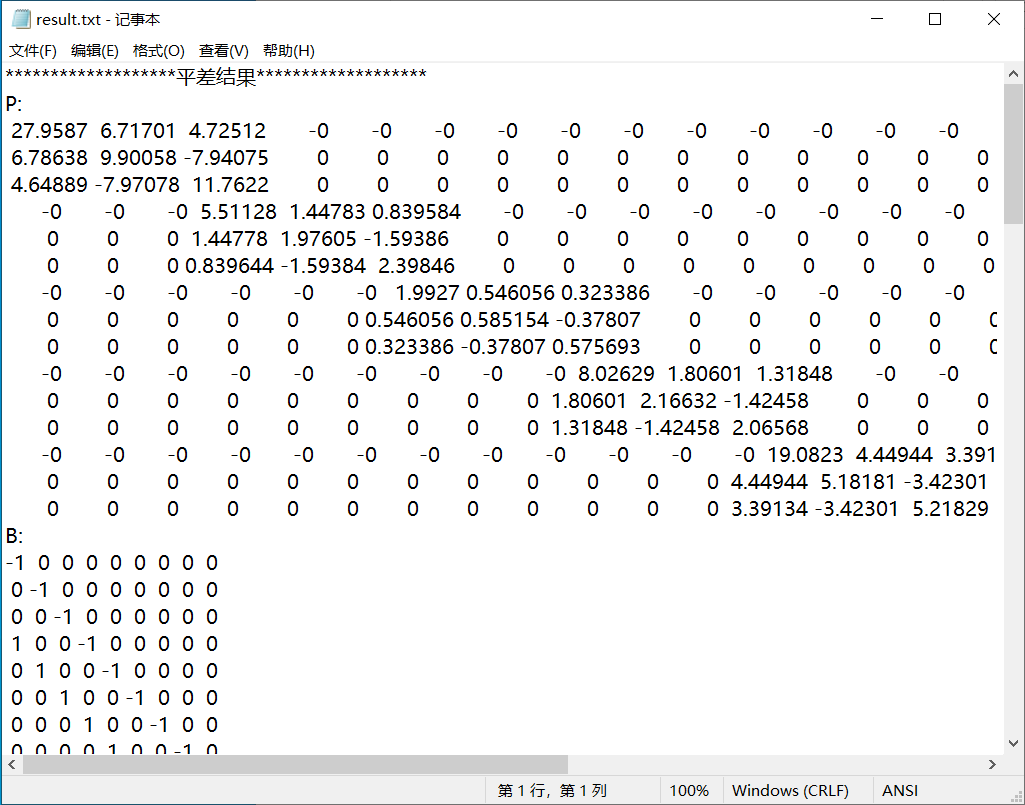
249. //输出解算结果到文件
250. **void** GPSnet::print(string filePath)
251. {
252. ofstream ofs(filePath, ios::trunc);
253. ofs << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*平差结果\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
254. ofs << "P:" << endl << P << endl;
255. ofs << "B:" << endl << B << endl;
256. ofs << "l:" << endl << l << endl;
257. ofs << "x:" << endl << x << endl;
258. ofs << "v:" << endl << v << endl;
259. **for** (**int** i = 0; i < observation\_num; i++)
260. {
261. ofs << lines[i].begin.pointName << "  " << lines[i].end.pointName << "  ";
262. ofs << "dx:" << lines[i].dx + v(i \* 3 + 0, 0) << "  dy:" << lines[i].dy + v(i \* 3 + 1, 0) << "  dz:" << lines[i].dz + v(i \* 3 + 2, 0) << endl;
263. }
264. map<string, **int**>temp;
265. **int** index = 0;
266. **for** (**int** i = 0; i < known\_point\_num; i++)
267. {
268. temp[knownPoints[i].pointName] = 0;
269. }
270. **for** (**int** i = 0; i < observation\_num; i++)
271. {
272. **if** ((temp.find(lines[i].begin.pointName) == temp.end()))
273. {
274. ofs << lines[i].begin.pointName << ",   x:" << lines[i].begin.x + x(index, 0)
275. << ",   y:" << lines[i].begin.y + x(index + 1, 0) << ",   z:" << lines[i].begin.z + x(index + 2, 0) << endl;
276. index += 3;
277. temp[lines[i].begin.pointName] = 1;
278. }
279. **if** ((temp.find(lines[i].end.pointName) == temp.end()))
280. {
281. ofs << lines[i].end.pointName << ",   x:" << lines[i].end.x + x(index, 0)
282. << ",   y:" << lines[i].end.y + x(index + 1, 0) << ",   z:" << lines[i].end.z + x(index + 2, 0) << endl;
283. index += 3;
284. temp[lines[i].end.pointName] = 1;
285. }
286. }
287. ofs << "\n\n\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*精度评定\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
288. ofs << "Qxx:" << endl << (B.transpose()\*P\*B).inverse() << endl<<endl;
289. ofs << "Qll:" << endl << B \* (B.transpose()\*P\*B).inverse() \*B.transpose() << endl;
290. ofs.close();
291. }
292. #include"GPSnet.h"
294. **int** main()
295. {
296. //类调用方式
297. GPSnet network;//创建GPS网对象
298. network.loadData("../data/data.txt");//导入数据文件
299. network.print();//控制台输出解算结果
300. network.print("../results/result.txt");//保存结算结果到文件中
301. **return** 0;
302. }

**2.5 完整代码**

完整代码已上传至GitHub个人仓库。（链接：[jiulieqingshen/GPSnet: GPS网间接平差计算程序 (github.com)](https://github.com/jiulieqingshen/GPSnet)）

**2.6 程序运行结果**

****

****

**第四章 心得体会**

1.在本学期的课程中，我们学到了间接平差函数模型、条件平差函数模型、误差椭圆和各种网型误差方程的列立，学的并不是很透彻，无法理解学这个东西到底是拿来干什么。但是，直到本次课程设计完成以后，我领悟到了书中最小二乘平差的思想和用途，知道了学这个到底可以拿来干什么。

2.测绘学科与程序设计息息相关。在传统的测绘方面，我们全站仪测出的数据可以用C++、Matlab来简化任务难度，减少工作量。在新测绘技术方面，如GNSS测绘数据处理、点云数据处理、导航与姿态数据处理，面对如此海量的数据处理，人工计算会花费大量时间进行重复劳动，这就更离不开测绘数据处理程序。除此之外，我们测绘的优势不在于用户操作界面的设计，也不在于花里胡哨的网站特效，而是在于测绘算法！我们编写的日用小工具、网站，其他专业学生也可以很简单的进行实现。但是在测绘算法，如最小二乘平差、GNSS数据处理方面，我们测绘占有绝对的领先地位，这是其他专业学生无法做到的。

3.学习要善于交流，经常与同学交流或浏览开源网站（如GitHub）有时会灵光乍现，一个难以理解的算法突然间就明白了。

参考文献

1. 张书毕.测量平差［M］.江苏：中国矿业大学出版社，2013
2. 武汉大学测绘学院测量平差学科组. 误差理论与测量平差基础习题集［M］.武汉：武汉大学出版社，2005
3. 陶秋香，翟敏. 基于Matlab 的GPS网间接平差教学设计与实现［J］. 测绘与空间地理信息,2016,39(3):10-12

|  |  |
| --- | --- |
| 教师对课程设计的评语： | |
| 考核等级（五级计分制）：  □优秀 □良好 □中等  □及格 □不及格 | 指导教师（签名）：  （电子签名）  年 月 日 |