

测量程序设计

C#-WPF

朱学军编著

xazhuxj@qq.com

西安科技大学

测绘科学与技术学院

二零一八年六月

本文档仅适用于西安科技大学测绘相关专业本科二年级以上学生教学使用。

也可用于其他专业人员参考。

内部文档，版权所有 © 2018，朱学军。

未经作者本人授权，严谨出版发行！

教学计划及课程目标

教学计划

教学课时

该课程标准学时为 64 学时。本学期为 60 学时，上课 15 次 30 学时，上机 15 次 30 学时。必修课，考试。笔试（60%）+ 上机实验与作业（40%）

学习内容

学习面向对象的程序设计与编写；
学习基本的测量算法程序编写，学习软件界面的编写。

编程预言的选择

注意：该课程不是再次学习某种编程语言，而是运用某种编程语言进行测量数据处理及测量算法的编写。

可能大家只学习了 C 语言，C 语言是面向过程的功能强大的语言，执行效率高，但界面程序编写较复杂。现代编程语言更多的是面向对象的编程语言，高效而且兼容 C 语言的是 C++，但界面的编写仍较复杂。因此我们选择更加现代化的编程语言 C#，它语法优美，界面编写方式有 WinForm 与 WPF 方式，虽然执行效率没有 C++ 高，但在 Windows7、Windows8/8.1、Windows10 中都几乎预装了运行库 .Net Frame，并且得到了 Microsoft 的大力推广，Microsoft 的许多软件都在使用 C# 开发，学习与开发成本相比与 C++ 显著减少，效率显著提升。

C# 的语法与 JAVA 的许多语法是极其相似的，现代软件工程的许多方法也可以用 C# 去实现。因此，在本课程中我们将学习 C# 编程语言，并且会学习如何运用 C# 语言进行测量程序设计及测量数据处理。

现代软件基本上都具有简洁易用的界面，我们还将学习 WPF 的界面编写技术，为我们的程序设计处美观简洁的界面，这些都包含在 C# 之中。

编程环境与工具

本课程的基本编程工具为 Visual Studio，版本理论上为 2010 及更新版，推荐大家用 Visual Studio 2015 或 2017 版（目前的最新版）。

参考教材

C# 语言及 WPF 界面编写参考马骏主编，人民邮电出版社出版，《C# 程序设计及应用教程》第 3 版。

测量算法参考本教程（一直更新中.....）。

目录

教学计划及课程目标	i	4.2.5 换带计算	44
目录	iii	4.3 图形界面程序编写	45
第一章 从 C 走向 C#	1	4.3.1 单点高斯投影正反算图 形界面编写	45
1.1 Client/Server 程序设计模式	1	4.3.2 界面程序的优化	47
1.2 从面向过程走向面向对象的程 序设计	1	4.3.3 点类的进一步优化	50
1.2.1 面向过程的 C 代码示例	2	4.4 更加实用的多点计算图形程序	51
1.2.2 对 C 代码的解释	2	4.4.1 程序的功能	51
1.2.3 与 C 相对应的 C# 代码	3	4.4.2 程序的面向对象分析与 实现	52
1.3 面向对象的 C# 代码	5	4.4.3 多点的高斯投影计算	55
1.4 作业	9	4.4.4 点坐标数据的读入与写出	56
第二章 测量基础函数设计	11	4.4.5 界面设计与实现	59
2.1 C# 知识点	11	4.4.6 换带功能的实现	63
2.2 常用测量计算公式	11	4.4.7 注意事项与功能扩展	64
2.2.1 角度弧度互换函数:	12	第五章 平面坐标系之间的转换	67
2.2.2 坐标方位角推算	13	5.1 原理和数学模型	67
2.2.3 平面坐标正反算	13	5.1.1 原理	67
第三章 附合导线近似的平差程序设计	15	5.1.2 相似变换法的数学模型	68
3.1 程序功能简单分析	15	5.2 程序设计与实现	69
3.2 程序初步功能	15	5.2.1 程序功能分析	69
3.3 实现代码	16	5.2.2 程序界面设计	69
第四章 高斯投影正反算与换带	35	5.2.3 数据文件和成果文件格式	72
4.1 高斯投影的数学模型	35	5.2.4 程序流程	73
4.2 程序功能分析与设计	38	5.2.5 主要功能设计	73
4.2.1 高斯投影的主要内容	38	第六章 线路要素计算程序设计	85
4.2.2 参考椭球类的设计	38	6.1 圆曲线与有缓和曲线的圆曲线	85
4.2.3 高斯投影正算功能的实现	41	6.1.1 圆曲线的数学模型	85
4.2.4 高斯投影反算功能的实现	42	6.1.2 圆曲线上点的坐标计算	85
		6.1.3 缓和曲线的数学模型	86

第一章 从 C 走向 C#

基本上我们都学习过 C 语言，有过一定的软件设计基础。C 语言是功能强大的高效程序设计语言，但在现代的软件开发中特别是 Window 界面程序中开发的效率太低了。Microsoft 推出的 .Net Framework 在新版 Windows 中越来越多的进行了预安装，C# 语言面向对象、语法优美、功能强大、开发效率高，应用也越来越广泛。因此我们将从 C 语言程序设计转向 C# 程序设计，学习更多的现代程序设计方法，满足现代测绘大数据的处理与算法编写需求。

1.1 Client/Server 程序设计模式

现代软件往往是多人协作开发的成果，单个人进行大型软件的开发是比较少的。在软件开发中需要遵循软件工程的组织原则，寻求代码的可复用性、可测试性与可阅读性。在学习编程时我们首先应该改掉以下三个不良编码习惯：

- 改变 C 语言中将代码写入 main 函数中的习惯；

- 改变 C# 语言中将代码写入 Main 函数中的习惯；

- 改变在 UI(WinForm 或 WPF 或其它的界面)中直接写入逻辑算法代码的习惯。

以上三种形式的代码书写处我们可以称为 Client，我们编写的逻辑算法代码可以称为 Server。试想如果我们去商场买东西，我们就是 Client，提供需求；商场就是 Server，提供服务。如果我们要买一只圆珠笔，也许我们只需简单的告诉商场人员，然后付钱拿笔走人就行了。但如果商场要让我们自己去仓库里找笔、查价钱、再在商场登记簿上登记库存等等一些动作，你想想会是怎么样的一个结果？买只笔都要把我们累死了。

这说明了什么呢？提供服务功能的商场应该对有功能需求的客户简化和屏蔽各种复杂的中间环节。在软件开发时同样如此，我们的 main 或 Main 函数或 UI 处的代码应该简洁，基本上只是调用各个功能算法而已。

如果我们像以上这三种方式组织代码，将会带来一系列的问题，尤其在团队开发与多人协作时代码不能复用，不能进行 unittest(单元测试)、不能用 git 工具（著名的源代码管理工具）进行源代码的自动合并。软件系统稍微复杂一点，我们的开发就会面临失败的危险。

万丈高楼从地起，因此，在学习编程时首先我们要有 Client/Server 模式意识，要遵循界面与算法相分离的原则进行程序设计。

1.2 从面向过程走向面向对象的程序设计

良好的面向过程设计程序设计程序是可以很好的转向面向对象的程序设计的，我们将从一个简单的 C 程序开始设计结构较为良好的 C 代码，再将其用面向对象的 C# 进行实现。从中体会面向过程与面向对象的程序设计方法的不同。

1.2.1 面向过程的 C 代码示例

在测绘专业中我们经常与测量点打交道, 因此我们定义一个点 Point (测点除了它的坐标 x,y 和高程 z 之外, 还需至少有点名), 另外我们定义一个简单的为大家所熟知的圆 Circle, 来使用点 Point 定义它的圆心, 并实现判断两圆是否相交, 计算圆的面积和周长等功能。代码如下所示:

```
1 // Ch01Ex01.cpp : Defines the entry point for the console application.
2 //
3
4 #include "stdafx.h"
5
6 #define _USE_MATH_DEFINES
7 #include <math.h>
8 #include <string.h>
9 #define PI M_PI
10
11 typedef struct __point {
12     char name[11];
13     double x, y, z;
14 }Point;
15
16 typedef struct __circle {
17     Point center;
18     double r;
19     double area;
20     double length;
21 }Circle;
22
23 // 计算圆的面积
24 void Area(Circle * c) {
25     c->area = PI * c->r * c->r;
26 }
27
28 // 计算两点的距离
29 double Distance(const Point * p1, const Point * p2) {
30     double dx = p2->x - p1->x;
31     double dy = p2->y - p1->y;
32     return sqrt(dx*dx + dy * dy);
33 }
34
35 // 判断两圆是否相交
36 bool IsIntersectWithCircle(const Circle * c1, const Circle * c2) {
37     double d = Distance(&c1->center, &c2->center);
38     return d <= (c1->r + c2->r);
39 }
```

1.2.2 对 C 代码的解释

代码中的 11 和 16 行定义结构体时使用了 typedef, 这样在标准 C 中再定义 Point 类型或 Circle 类型时可以避免在其前面加关键词 struct。

第 24-26 行计算圆的面积, 传入了圆的指针, 这是 C 及 C++ 中传递自定义数据类型的常用的高效方式。由于计算的圆面积会保存在结构体 Circle 中的 area 中, 所以函数不需要返回值, 返回值定义为 void。

第 29-33 行为计算两点的距离, 由于该函数不会修改 p1、p2 两个 Point 内的任何成员, 故应该将这两个指针定义为 const 类型, 防止函数内部无意或故意的修改。

同理第 36-39 行判断两圆是否相交，由于该函数不会修改 c1、c2 两个圆内的任何成员，也应将这两个指针定义为 const 类型。

第 37 行由于 Distance 函数的参数需要两个 Point 指针类型的参数，尽管 c1 与 c2 均为指针，但 c1->center 与 c2->center 不是指针，所以需要在其前用取地址符 &。

第 38 行的关系运算符本身的运算结果就为 bool 型，故不需要进行 if、else 类型的判断。相应的 main 函数测试代码如下：

```
1 int main()
2 {
3     Point pt1;
4     strcpy_s(pt1.name, 11, "pt1");
5     pt1.x = 100; pt1.y = 100; pt1.z = 425.324;
6
7     Point pt2;
8     strcpy_s(pt2.name, 11, "pt2");
9     pt2.x = 200; pt2.y = 200; pt2.z = 417.626;
10
11     Circle c1;
12     c1.center = pt1; c1.r = 80;
13
14     Circle c2;
15     c2.center = pt2; c2.r = 110;
16
17     Area(&c1) ; //计算圆c1的面积
18     printf("Circle1 的面积 = %lf \n", c1.area );
19
20     bool yes = IsIntersectWithCircle(&c1, &c2);
21     printf("Circle1 与 Circle2 是否相交 : %s\n", (yes ? "是" : "否") );
22
23     return 0;
24 }
```

程序的运行结果如下：

Circle1 的面积 = 20106.192983

Circle1 与 Circle2 是否相交 : 是

1.2.3 与 C 相对应的 C# 代码

将以上代码相对应的 C 代码直接翻译为 C# 代码为：

```
1 using System;
2
3 namespace Ch01Ex02
4 {
5     class Point
6     {
7         public string name;
8         public double x;
9         public double y;
10        public double z;
11
12        public double Distance(Point p2)
13        {
14            double dx = x - p2.x;
```

```
15         double dy = y - p2.y;
16         return Math.Sqrt(dx * dx + dy * dy);
17     }
18 }
19
20 class Circle
21 {
22     public Point center;
23     public double r;
24     public double area;
25     public double length;
26
27     public void Area()
28     {
29         area = Math.PI * r * r;
30     }
31
32     public bool IsIntersectWithCircle(Circle c2)
33     {
34         double d = this.center.Distance(c2.center);
35         return d <= (r + c2.r);
36     }
37 }
38 }
```

由以上 C# 代码可以看出与 C 代码的不同。这是因为 C# 语言是纯面向对象语言的缘故。

程序或软件的基本概念是：程序 = 数据结构 + 算法。在以上 C 或 C# 代码中，Point 和 Circle 都可以看作是数据结构，计算两点距离和圆的面积或判断两圆是否相交都可以看作是算法。在 C 语言中，数据结构和算法是分离的，尽管函数的参数是数据结构，但函数并不属于数据结构。在 C# 中则不同，数据结构用 class 定义，则数据与算法都属于同一个 class。

计算两点距离的函数设计为 Point 类的一个方法，可以看作是计算当前点与另一个点的距离，因此函数的参数就只有一个 Point p2，另一个自然是 this（当前类本身）了。

同样的道理圆的面积计算也不需要参数了（可以想象为计算知道自己的半径，计算自己的面积并存入自己的成员变量中了）。判断两圆是否相交也可以看作是判断当前圆与另一个圆 Circle c2 是否相交了，只需要一个参数。

相应 C# 的 Main 函数测试代码如下：

```
1 using System;
2
3 namespace Ch01Ex02
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Point p1 = new Point();
10            p1.name = "p1";
11            p1.x = 100;
12            p1.y = 100;
13            p1.z = 425.324;
14
15
16            Circle c1 = new Circle();
17            c1.center = p1;
18            c1.r = 80;
```

```
19         Point p2 = new Point();
20         p2.name = "p2";
21         p2.x = 200;
22         p2.y = 200;
23         p2.z = 417.626;
24
25         Circle c2 = new Circle();
26         c2.center = p2;
27         c2.r = 110;
28
29         c1.Area(); // 计算圆c1的面积
30         Console.WriteLine("Circle1的面积={0}", c1.area);
31
32         c2.Area(); // 计算圆c2的面积
33         Console.WriteLine("Circle2的面积={0}", c2.area);
34
35         bool yes = c1.IsIntersectWithCircle(c2);
36         Console.WriteLine("Circle1与Circle2是否相交:{0}",
37             yes ? "是" : "否");
38     }
39 }
40
41 }
```

程序的运行结果如下：

Circle1的面积=20106.1929829747

Circle2的面积=38013.2711084365

Circle1与Circle2是否相交:是

1.3 面向对象的 C# 代码

上面的 C# 不符合面向对象的软件设计原则（违背封装、继承与多态中的封装原则），且在 Main 函数中，如果由于疏漏忘记第 30 行或 33 行对圆的面积进行计算，圆 c1 或 c2 将不会有正确的面积。

因此，我们将其优化。首先对 Point 类进行封装，将其字段定义为 private，用属性方法将其向外暴露接口，相应的 C# 代码为：

```
1     private string name;
2     public string Name
3     {
4         get { return name; }
5         set { name = value; }
6     }
7
8     private double x;
9     public double X
10    {
11        get { return x; }
12        set { x = value; }
13    }
14    private double y;
15    public double Y
16    {
17        get { return y; }
```

```

18         set { y = value; }
19     }
20     private double z;
21     public double Z
22     {
23         get { return z; }
24         set { z = value; }
25     }

```

以上是用 C# 的属性对 private 字段的进行简单的封装，也可以写成如下形式：

```

1     public string Name { get ; set ; }
2     public double X { get ; set ;}
3     public double Y { get ; set ; }
4     public double Z {get ; set ;}

```

为了简化 Main 函数对 Point 类的调用及对其各字段值的初始化，我们定义构造函数如下：

```

1     public Point(double x, double y)
2     {
3         this.name = "";
4         this.x = x;
5         this.y = y;
6         this.z = 0;
7     }
8
9     public Point(string name, double x, double y, double z)
10    {
11        this.name = name;
12        this.x = x;
13        this.y = y;
14        this.z = z;
15    }

```

同样道理，也应对 Circle 进行封装，将其各个字段定义为 private，并用属性方法将其向外的简单的暴露接口，相应的 C# 代码为：

```

1     public Point Center { get; set; }
2
3     private double r;
4     public double R
5     {
6         get { return r; }
7         set
8         {
9             if(value != r && value >= 0)
10            {
11                r = value;
12                CalArea(); //r的值发生改变，重新计算圆的面积
13            }
14        }
15    }

```

圆心 Center 只是简单的封装，圆的半径则不同。由圆的特性知，当圆的半径确定，其面积与周长也就确定了。为了计算的高效，封装时，当圆的半径值发生改变时，就需要重新计算圆的面积，确保面积的正确性。

由于圆的面积只与圆的半径有关，我们不需要在其他的情况对圆的面积进行修改，因此封装的 C# 代码为：

```
1     private double area;
2     public double Area
3     {
4         get { return area; }
5     }
```

在此我们去掉了属性 Area 中的 set 语句，并将 area 成员定义为了 private，使外部无法修改 area 的值，来保证圆的面积 Area 的正确性。

还有一个地方可能会修改圆的半径，就是圆的初始化时，因此圆的构造函数应定义为：

```
1     public Circle(double x, double y, double r)
2     {
3         Center = new Point(x, y);
4
5         this.R = r; //赋值给R而不是this.r，确保计算圆的面积
6     }
```

为确保在半径 r 的值发生改变后能重新计算圆的面积和周长，此处赋值给半径属性 R。完整的优化后的 C# 代码如下：

```
1 using System;
2
3 namespace Ch01Ex03
4 {
5     class Point
6     {
7         private string name;
8         public string Name
9         {
10             get { return name; }
11             set { name = value; }
12         }
13
14         private double x;
15         public double X
16         {
17             get { return x; }
18             set { x = value; }
19         }
20         private double y;
21         public double Y
22         {
23             get { return y; }
24             set { y = value; }
25         }
26         private double z;
27         public double Z
28         {
29             get { return z; }
30             set { z = value; }
31         }
32
33         public Point(double x, double y)
34         {
35             this.name = "";
36             this.x = x;
```

```
37         this.y = y;
38         this.z = 0;
39     }
40
41     public Point(string name, double x, double y, double z)
42     {
43         this.name = name;
44         this.x = x;
45         this.y = y;
46         this.z = z;
47     }
48
49     public double Distance(Point p2)
50     {
51         double dx = X - p2.X;
52         double dy = Y - p2.Y;
53         return Math.Sqrt(dx * dx + dy * dy);
54     }
55 }
56
57 class Circle
58 {
59     public Point Center { get; set; }
60
61     private double r;
62     public double R
63     {
64         get { return r; }
65         set
66         {
67             if(value != r && value >= 0)
68             {
69                 r = value;
70                 CalArea(); //r的值发生改变，重新计算圆的面积
71             }
72         }
73     }
74
75     private double area;
76     public double Area
77     {
78         get { return area; }
79     }
80
81     private double length;
82
83     public Circle(double x, double y, double r)
84     {
85         Center = new Point(x, y);
86
87         this.R = r; //赋值给R而不是this.r，确保计算圆的面积
88     }
89
90     //计算圆的面积
91     private void CalArea()
92     {
```

```

93         area = Math.PI * r * r;
94     }
95
96     //判断两圆是否相交
97     public bool IsIntersectWithCircle(Circle c2)
98     {
99         double d = this.Center.Distance(c2.Center);
100         return d <= (r + c2.r);
101     }
102 }
103 }

```

代码中由于计算圆的面积将会在属性 R 中和构造函数中调用,我们将其定义为函数 CalArea 供多次复用,并将可访问性定义为 private 供内部使用。

相应 C# 的 Main 函数测试代码如下:

```

1 using System;
2
3 namespace Ch01Ex03
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Circle c1 = new Circle(100, 100, 80);
10            Circle c2 = new Circle(200, 200, 110);
11
12            Console.WriteLine("Circle1 的面积={0}", c1.Area );
13            Console.WriteLine("Circle2 的面积={0}", c2.Area);
14
15            bool yes = c1.IsIntersectWithCircle(c2);
16            Console.WriteLine("Circle1 与 Circle2 是否相交:{0}",
17                yes ? "是" : "否" );
18        }
19    }
20 }

```

由优化后的 C# 代码可以看出 Main 十分简洁。

1.4 作业

1. 请完成示例中的圆的周长计算功能;
2. 请试着完成多段线 Polyline 的面积与长度计算功能 (提示: 多段线是点序的集合);
3. 请试着完成多边形 Polygon 的面积与长度计算功能。

提示: 多边形面积计算公式为:

$$S = \frac{1}{2} \sum_{k=1}^6 (x_k y_{k+1} - x_{k+1} y_k)$$

如已知某多边形的顶点坐标依次为 $p_1(-1, 0), p_2(2, 3), p_3(4, 2), p_4(4, 4), p_5(6, 8), p_6(-2, 5)$, 则依上式计算该多边形面积为 28.

第二章 测量基础函数设计

2.1 C# 知识点

C# 中的类分为实例类与静态类。

C# 是纯面向对象语言，也就是说所有的常量与方法都需要以类 *class* 为载体，如同其他的数学软件包一样，我将其命名为 *SMath*(*SurveyMath*)，保存在 *SMath.cs* 文件中，为了引用方便，我们需将常用的一些常量及方法定义为静态成员。示例代码如下所示：

```
1 namespace ZXY
2 {
3     public static class SMath
4     {
5         public const double PI=Math.PI;
6         public const double TODEG=180.0/PI;
7         public const double TORAD=PI/180.0;
8         public const double TOSECOND=180.0*3600.0/PI;
9
10        public static double DMS2RAD(double dmsAngle)
11        {
12            .....
13        }
14
15        public static double RAD2DMS(double radAngle)
16        {
17            .....
18        }
19    }
20 }
```

以上示例代码为了与其他的函数或符号相区别，也为了与其他的代码一起合作使用，在此加入了自已的命名空间 ZXY（这是我用的名称空间，你当然也可以根据自己的习惯或爱好命名适当的名称空间）。

2.2 常用测量计算公式

角度、距离与高差是测量工程师工作的基本对象，度分秒形式的角度与弧度之间的转换是我们进行测量数据处理的基础。为了方便的进行测量数据处理，我们需要定义一些常量，如前述示例代码所示。

PI 表示 π , *PI2* 表示 2π ; *TODEG* 表示 $180/\pi$ ，用于将弧度化为度；*TORAD* 表示 $\pi/180$ ，用于将度化为弧度；*TOSECOND* 表示 $180 * 3600/\pi$ ，用于将弧度转换为秒。

2.2.1 角度弧度互换函数:

在测量工程中角度的常用习惯表示法是度分秒的形式, 在计算程序中测量工程人员也常将度分秒形式的角度用格式为 xxx.xxxxx 的形式表示, 即以小数点前的整数部分表示度, 小数点后两位数表示分, 从小数点后第三位起表示秒。在计算机编程时所用的角度要以弧度表示的, 因此需要设计函数相互转换。

1. 角度化弧度函数:

角度化弧度函数的逻辑非常简单, 许多测量编程人员将其写成如下的形式:

```
public static double DMS2RAD(double dmsAngle)
{
    int d = (int)dmsAngle;
    dmsAngle = (dmsAngle - d) * 100.0;
    int m = (int)dmsAngle;
    double s = (dmsAngle - m) * 100.0;
    return (d + m / 60.0 + s / 3600.0) * TORAD;
}
```

但由于计算机中浮点数的表示方法的原因, 以上函数并不能精确的将度分秒的角度转换为弧度。

如某一角度为 $1^{\circ}40'00''$, 我们以 1.4000 形式的浮点数输入, 计算机将表示为 1.3999999999999999 的形式。这在计算机中并没有什么错误, 但以上函数在提取角度的分秒时, 提取到的 m 值为 39, 提取到的 s 值为 99.999999999999, 即我们提取到的角度为 $1^{\circ}39'100''$, 有 $40''$ 的角度误差, 这是我们测绘工程人员不能接受的。

有的软件设计人员在软件中发现这个问题后的处理的办法是让用户在角度后加减一秒, 进行规避这种误差, 显然, 将软件人员的责任推给用户是极其不合适和不负责任的。还有许多的书籍中介绍了许多五花八门的处理方法, 但奏效的小, 不奏效的却很多, 甚至有的将这么简单的算法逻辑变得逻辑十分复杂。

虽然浮点数的表达不够精确, 但我们知道在计算机中, 整数的表达与计算却是精确的, 因此在角度的度分秒值提取中, 我们采用整数的运算方式进行计算, 相应代码如下。

```
public static double DMS2RAD(double dmsAngle)
{
    dmsAngle *= 10000;
    int angle = (int)Math.Round(dmsAngle);
    int d = angle / 10000;
    angle -= d * 10000;
    int m = angle / 100;
    double s = dmsAngle - d * 10000 - m * 100;
    return (d + m / 60.0 + s / 3600.0) * TORAD;
}
```

以上算法对于负的角度值的提取同样有效。

2. 弧度化角度函数

同样的道理, 以下函数将不能正确的将一些弧度值转换为度分秒形式的角度值, 在某些情况下转换出的角度将出现 $59'60''$ 或 $59'59.9999''$ 的形式。

```
public static double RAD2DMS(double radAngle)
{

```

```

    radAngle *= TODEG;
    int d = (int)radAngle;
    radAngle = (radAngle - d) * 60;
    int m = (int)radAngle;
    double s = (radAngle - m) * 60;
    return (d + m / 100.0 + s / 10000.0);
}

```

同样，我们需要利用整数的精确表达能力与计算能力来进行转换，正确的代码如下：

```

public static double RAD2DMS(double radAngle)
{
    radAngle *= TOSECOND;
    int angle = (int)Math.Round(radAngle);
    int d = angle / 3600;
    angle -= d * 3600;
    int m = angle / 60;
    double s = radAngle - d * 3600 - m * 60;
    return (d + m / 100.0 + s / 10000.0);
}

```

2.2.2 坐标方位角推算

1. 已知 01 边的坐标方位角 α_0 和 01 边和 12 边间的水平角 β ，计算 12 边的坐标方位角。

```

double Azimuth(double azimuth0, double angle)
{
    return To0_2PI(azimuth0 + angle + _PI);
}

```

2. 将角度规划到 $0 \sim 2$ ，单位为弧度

```

double To0_2PI(double rad)
{
    int f = rad >= 0 ? 0 : 1;
    int n = (int)(rad / TWO_PI);

    return rad - n * TWO_PI + f * TWO_PI;
}

```

2.2.3 平面坐标正反算

1. 坐标正算：

根据 0->1 点的坐标方位角和水平边长，计算 0->1 点的坐标增量。

```

void double dxdy(double azimuth, double distance,
                 double& dx, double& dy)
{

```

```

    dx = cos(azimuth) * distance;
    dy = sin(azimuth) * distance;
}

```

根据 0 点的坐标和 0->i 点的坐标方位角和水平边长, 计算 i 点的坐标。

```

void Coordinate(double x0, double y0,
               double azimuth, double distance,
               double& xi, double& yi)
{
    xi = x0 + cos(azimuth) * distance;
    yi = y0 + sin(azimuth) * distance;
}

```

根据 1 点的坐标和后视边 (0->1) 点的坐标方位角, 水平角 (0-1-i), 水平边长 (1-i), 计算 i 点的坐标。

```

void Coordinate(double x0, double y0, double azimuth0,
               double angle, double distance,
               double& xi, double& yi)
{
    double azimuthi = Azimuth(azimuth0, angle);
    xi = x0 + cos(azimuthi) * distance;
    yi = y0 + sin(azimuthi) * distance;
}

```

2. 坐标反算:

计算 0 点至 1 点的坐标方位角, 返回值单位为弧度。

```

double Azimuth(double x0, double y0, double x1, double y1)
{
    double dx = x1 - x0;
    double dy = y1 - y0;
    return atan2(dy, dx) + (dy < 0 ? 1 : 0) * _2PI;
}

```

计算两点间 (0->1) 的平距

```

double Distance(double x0, double y0, double x1, double y1)
{
    double dx = x1 - x0;
    double dy = y1 - y0;
    return sqrt(dx * dx + dy * dy);
}

```

第三章 附和导线近似平差程序设计

虽然 GPS 的广泛应用使传统的控制测量趋于淘汰，但附和导线在一些工程中应用仍然非常广泛，如地铁、隧道或建筑物密集的城区中的控制测量。在这些数据处理中，近似平差法由于不会受边角权的影响，能最大程度的保持数据的真实性，仍然是数据检查的重要手段和数据处理的重要方法。

3.1 程序功能简单分析

3.2 程序初步功能

我们设计界面如图3.1所示：

控制点数据				
点名	X	Y	Z	已知点
D01	3805820.521	333150.649	0	1
D02	3805813.062	333067.961	0	1
D04	3805951.3	333151.196	0	0
D03	3805951.982	333063.347	0	0

坐标方位角数据		
起点	方向点	坐标方位角
D02	D01	1.48083325647482
D02	D01	1.48083325647482

观测值数据				
测站	起点	终点	观测角	平距
D01	D02	D04	95.234	130.779
D04	D01	D03	90.1217	87.851
D03	D04	D02	87.3918	138.998
D02	D03	D01	86.4453	-1

图 3.1 附和导线界面功能图

3.3 实现代码

```

1 <Window x:Class="SurApp.MainWindow"
2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4 xmlns:local="clr-namespace:SurApp.models"
5 Title="拟合导线近似平差" Height="350" Width="525"
6 WindowState="Maximized">
7
8 <DockPanel LastChildFill="True">
9 <Menu x:Name="mainmenu" DockPanel.Dock="Top" Background="AliceBlue">
10 <MenuItem Header="文件(F)">
11 <MenuItem Header="新建拟合导线..." Click="NewMenuItem_Click" />
12 <MenuItem Header="打开拟合导线文件..." Click="OpenMenuItem_Click" />
13 <MenuItem Header="保存拟合导线文件..." Click="SaveMenuItem_Click" />
14 <Separator />
15 <MenuItem Header="导入文本数据" Click="ImportSPointMenuItem_Click" />
16 <MenuItem Header="导出文本数据" Click="OutputSPointMenuItem_Click" />
17 <Separator />
18 <MenuItem Header="导出为DXF文件" />
19 </MenuItem>
20
21 <MenuItem Header="数据处理(D)">
22 <MenuItem Header="拟合导线平差" Click="AdjustMenuItem_Click" />
23 <MenuItem Header="平差成果" Click="AdjustResultMenuItem_Click" />
24 </MenuItem>
25
26 <MenuItem Header="测量(S)">
27 <MenuItem Header="角度弧度转换" Click="DMS2RADMenuItem_Click" />
28 <MenuItem Header="坐标方位角计算" Click="AzimuthMenuItem_Click" />
29 </MenuItem>
30 </Menu>
31 <StatusBar x:Name="statusBar" DockPanel.Dock="Bottom" Height="25" Background="
    AliceBlue" />
32 <Grid>
33 <Grid.ColumnDefinitions>
34 <ColumnDefinition Width="75*" />
35 <ColumnDefinition Width="5*" />
36 <ColumnDefinition Width="170*" />
37 </Grid.ColumnDefinitions>
38 <Border BorderThickness="2" Background="Goldenrod" Grid.Column="0">
39 <Grid x:Name="leftGrid" >
40 <Grid.RowDefinitions>
41 <RowDefinition Height="20" />
42 <RowDefinition Height="100*" />
43 <RowDefinition Height="20" />
44 <RowDefinition Height="40*" />
45 <RowDefinition Height="20" />
46 <RowDefinition Height="100*" />
47 </Grid.RowDefinitions>
48 <TextBlock Text="控制点数据" Grid.Row="0" TextAlignment="Center" Margin="2" />
49 <DataGrid x:Name="ctrPointDataGrid" Grid.Row="1" AutoGenerateColumns="False"
    Margin="2" ItemsSource="{Binding CtrPoints}" >
50 <DataGrid.Columns>
51 <DataGridTextColumn Header="点名" Binding="{Binding Name}" MinWidth="40" />

```

```

52 <DataGridTextColumn Header="X" Binding="{Binding X, StringFormat={}{0:##0.###}}"
    MinWidth="60" />
53 <DataGridTextColumn Header="Y" Binding="{Binding Y, StringFormat={}{0:##0.###}}"
    MinWidth="60" />
54 <DataGridTextColumn Header="Z" Binding="{Binding Z, StringFormat={}{0:##0.###}}"
    MinWidth="60" />
55 <DataGridTextColumn Header="已知点" Binding="{Binding IsKnown}" MinWidth="30" />
56 </DataGrid.Columns>
57 </DataGrid>
58
59 <TextBlock Text="坐标方位角数据" Grid.Row="2" TextAlignment="Center" Margin="2" /
    >
60 <DataGrid x:Name="azimuthDataGrid" Grid.Row="3" AutoGenerateColumns="False"
    Margin="2" ItemsSource="{Binding KnownAzimuthes}" >
61 <DataGrid.Columns>
62 <DataGridTextColumn Header="起点" Binding="{Binding StartPnt.Name}" MinWidth="60"
    />
63 <DataGridTextColumn Header="方向点" Binding="{Binding EndPnt.Name}" MinWidth="60"
    />
64 <DataGridTextColumn Header="坐标方位角" Binding="{Binding Azimuth}" MinWidth="100"
    />
65 </DataGrid.Columns>
66 </DataGrid>
67
68 <TextBlock Text="观测值数据" Grid.Row="4" TextAlignment="Center" Margin="2" />
69 <DataGrid x:Name="obsValueDataGrid" Grid.Row="5" AutoGenerateColumns="False"
    Margin="2" ItemsSource="{Binding ObsValues}">
70 <DataGrid.Columns>
71 <DataGridTextColumn Header="测站" Binding="{Binding StnPnt.Name}" MinWidth="40" />
72 <DataGridTextColumn Header="起点" Binding="{Binding StartPnt.Name}" MinWidth="40"
    />
73 <DataGridTextColumn Header="终点" Binding="{Binding EndPnt.Name}" MinWidth="40" /
    >
74 <DataGridTextColumn Header="观测角" Binding="{Binding DmsAngle}" MinWidth="60" />
75 <DataGridTextColumn Header="平距" Binding="{Binding Distance}" MinWidth="60" />
76 </DataGrid.Columns>
77 </DataGrid>
78
79 </Grid>
80 </Border>
81
82 <GridSplitter Background="Red" Width="5" Grid.Column="1" HorizontalAlignment="
    Stretch" />
83
84 <Border BorderThickness="2" Grid.Column="2">
85 <local:DrawingCanvas x:Name="figureCanvas" SizeChanged="figureCanvas_SizeChanged"
    >
86 <!--<Canvas.RenderTransform>
87 <TransformGroup>
88 <ScaleTransform x:Name="scaleTransform" ScaleX="1" ScaleY="-1" />
89 <TranslateTransform X="0" Y="{Binding ActualHeight, RelativeSource={
    RelativeSource AncestorType=Canvas}}" />
90 </TransformGroup>
91 </Canvas.RenderTransform-->
92

```

```

93 <!--<Rectangle Height="50" Width="50" Fill="Red" Stroke="Blue" StrokeThickness="2
    " Canvas.Left="50" Canvas.Top="50" />
94
95 <Rectangle Height="50" Width="50" Fill="#CCCCFF" Stroke="Blue" StrokeThickness=
    "2" Canvas.Left="50" Canvas.Top="50" >
96 <Rectangle.RenderTransform>
97 <TranslateTransform X="50" Y="50" />
98 </Rectangle.RenderTransform>
99 </Rectangle-->
100 </local:DrawingCanvas>
101 </Border>
102 </Grid>
103 </DockPanel>
104
105 </Window>
106

```

```

1 using System;
2
3 namespace SurApp.models
4 {
5     [Serializable]
6     public class CtrPoint : ZXY.SPoint
7     {
8         private int isKnown = 0;
9
10        /// <summary>
11        /// 是否已知点 (0: 待定点, 1: 已知点)
12        /// </summary>
13        public int IsKnown
14        {
15            get { return isKnown; }
16            set
17            {
18                isKnown = value;
19                this.RaisePropertyChanged("IsKnown");
20            }
21        }
22
23        public CtrPoint() { }
24
25        public CtrPoint(string name, double x, double y, double z, int isKnown) : base(
            name, x, y, z)
26        {
27            this.isKnown = isKnown;
28        }
29
30        public override string ToString()
31        {
32            return string.Format("{0},{1},{2},{3}", Name, X, Y, Z);
33        }
34    }
35 }
36
37 using System;
38
39 namespace SurApp.models

```



```
40 {
41 [Serializable]
42 public class ObsValue : ZXY.NotificationObject
43 {
44     private CtrPoint stnPnt;
45     public CtrPoint StnPnt
46     {
47         get { return stnPnt; }
48         set
49         {
50             stnPnt = value;
51             this.RaisePropertyChanged("StnPnt");
52         }
53     }
54
55     private CtrPoint startPnt;
56     public CtrPoint StartPnt
57     {
58         get { return startPnt; }
59         set
60         {
61             startPnt = value;
62             this.RaisePropertyChanged("StartPnt");
63         }
64     }
65
66     private CtrPoint endPnt;
67     public CtrPoint EndPnt
68     {
69         get { return endPnt; }
70         set
71         {
72             endPnt = value;
73             this.RaisePropertyChanged("EndPnt");
74         }
75     }
76
77     /// <summary>
78     /// 观测角度值(单位: 弧度)
79     /// </summary>
80     private double angleValue;
81
82     /// <summary>
83     /// 观测角度值(单位: 弧度)
84     /// </summary>
85     public double AngleValue
86     {
87         get { return angleValue; }
88         set
89         {
90             angleValue = value;
91             this.RaisePropertyChanged("DmsAngle");
92             this.RaisePropertyChanged("AngleValue");
93         }
94     }
95 }
```

```

96 /// <summary>
97 /// 观测角度值(单位: 度分秒)
98 /// </summary>
99 public double DmsAngle
100 {
101     get { return ZXY.SMath.RAD2DMS(angleValue); }
102     set
103     {
104         angleValue = ZXY.SMath.DMS2RAD(value);
105         this.RaisePropertyChanged("DmsAngle");
106         this.RaisePropertyChanged("AngleValue");
107     }
108 }
109
110
111 private double distance;
112 public double Distance
113 {
114     get { return distance; }
115     set
116     {
117         distance = value;
118         this.RaisePropertyChanged("Distance");
119     }
120 }
121
122 public ObsValue() { }
123
124 public ObsValue(CtrPoint stnPnt, CtrPoint startPnt, CtrPoint endPnt,
125 double angleValue, double distance)
126 {
127     this.stnPnt = stnPnt;
128     this.startPnt = startPnt;
129     this.endPnt = endPnt;
130     this.DmsAngle = angleValue;
131     this.distance = distance;
132 }
133
134 private double vB; //角度改正数
135 public double VB
136 {
137     get { return vB; }
138     set
139     {
140         vB = value;
141         this.RaisePropertyChanged("VB");
142     }
143 }
144
145 private double angleV; //改正后角度
146 public double AngleV
147 {
148     get { return angleV; }
149     set
150     {
151         angleV = value;

```

```
152         this.RaisePropertyChanged("AngleV");
153     }
154 }
155
156 private double azimuth;
157 public double Azimuth
158 {
159     get { return azimuth; }
160     set
161     {
162         azimuth = value;
163         this.RaisePropertyChanged("Azimuth");
164     }
165 }
166
167 private double dx;
168 public double DX
169 {
170     get { return dx; }
171     set
172     {
173         dx = value;
174         this.RaisePropertyChanged("DX");
175     }
176 }
177
178 private double dy;
179 public double DY
180 {
181     get { return dy; }
182     set
183     {
184         dy = value;
185         this.RaisePropertyChanged("DY");
186     }
187 }
188
189 private double vdx;
190 public double VDX
191 {
192     get { return vdx; }
193     set
194     {
195         vdx = value;
196         this.RaisePropertyChanged("VDX");
197     }
198 }
199
200 private double vdy;
201 public double VDY
202 {
203     get { return vdy; }
204     set
205     {
206         vdy = value;
207         this.RaisePropertyChanged("VDY");
```

```

208         }
209     }
210
211     public override string ToString()
212     {
213         return string.Format("{0},{1},{2},{3},{4}",
214             startPnt.Name, startPnt.Name, endPnt.Name,
215             DmsAngle, distance);
216     }
217 }
218 }
219
220 using System;
221 using System.Collections.Generic;
222 using System.Collections.ObjectModel;
223 using System.Windows.Media;
224 using ZXY;
225
226 namespace SurApp.models
227 {
228     /// <summary>
229     /// 用于已知边的坐标方位角信息
230     /// </summary>
231     [Serializable]
232     public class KnownAzimuth : NotificationObject
233     {
234         /// <summary>
235         /// 坐标方位角,单位: 弧度
236         /// </summary>
237         public double azimuth;
238         public double Azimuth
239         {
240             get { return azimuth; }
241             set
242             {
243                 azimuth = value;
244                 this.RaisePropertyChanged("Azimuth");
245             }
246         }
247
248         public CtrPoint startPnt; //坐标方位角的起点
249         public CtrPoint StartPnt
250         {
251             get { return startPnt; }
252             set
253             {
254                 startPnt = value;
255                 this.RaisePropertyChanged("StartPnt");
256             }
257         }
258
259         public CtrPoint endPnt; //坐标方位角的方向点
260         public CtrPoint EndPnt
261         {
262             get { return endPnt; }
263             set

```

```

264         {
265             endPnt = value;
266             this.RaisePropertyChanged("EndPnt");
267         }
268     }
269
270     public KnownAzimuth()
271     {
272         startPnt = null;
273         endPnt = null;
274         azimuth = 0;
275     }
276
277     public KnownAzimuth(CtrPoint startPnt, CtrPoint endPnt, double az)
278     {
279         this.startPnt = startPnt;
280         this.endPnt = endPnt;
281         this.azimuth = az;
282     }
283
284     public override string ToString()
285     {
286         if (startPnt == null || endPnt == null) return "~~~";
287         else
288             return string.Format("{0},{1},{2}", startPnt.Name, endPnt.Name, ZXY.SMath.RAD2DMS
                (azimuth));
289     }
290 }
291
292 [Serializable]
293 public class CtrNet : NotificationObject
294 {
295     private ObservableCollection<KnownAzimuth> knownAzimuthes = new
        ObservableCollection<KnownAzimuth>();
296     public ObservableCollection<KnownAzimuth> KnownAzimuthes
297     {
298         {
299             get { return knownAzimuthes; }
300         }
301     }
302     private double m0 = 10; // 中误差
303     private double fB;
304     private double FB; // 角度闭合差的限差值
305     private double fx;
306     private double fy;
307     private double fs;
308     private double sumD;
309     private double K; // 1/K
310
311     // 以下定义为绘图使用
312     private double minX; // 高斯坐标X的最小值xn
313     private double minY; // 高斯坐标Y的最小值yn
314     private double maxX; // 高斯坐标X的最大值xm
315     private double maxY; // 高斯坐标Y的最大值ym
316
317     private double maxVX; // 屏幕坐标X的最大值

```

```

318 private double maxVY; //屏幕坐标Y的最大值
319
320 private double k; //变换比例
321
322 private ObservableCollection<CtrPoint> ctrPoints =
323 new ObservableCollection<CtrPoint>();
324 public ObservableCollection<CtrPoint> CtrPoints
325 {
326     get { return ctrPoints; }
327 }
328
329 private ObservableCollection<ObsValue> obsValues =
330 new ObservableCollection<ObsValue>();
331 public ObservableCollection<ObsValue> ObsValues
332 {
333     get { return obsValues; }
334 }
335
336 /// <summary>
337 /// 正确的计算路线
338 /// </summary>
339 private List<ObsValue> route = new List<ObsValue>();
340
341 private bool isDirty = false;
342
343 public void ReadDataTextFile(string fileName)
344 {
345     using (System.IO.StreamReader sr = new System.IO.StreamReader(fileName))
346     {
347         string buffer;
348
349         //读入控制点数据
350         this.ctrPoints.Clear();
351         while (true)
352         {
353             buffer = sr.ReadLine();
354             if (string.IsNullOrEmpty(buffer)) break; //文件末尾或空行退出
355
356             if (buffer[0] == '#') continue;
357
358             string[] its = buffer.Split(new char[1] { ',' });
359             if (its.Length != 4) continue; //不为四项控制点数据的继续,直到空行退出
360
361             ctrPoints.Add(new CtrPoint(
362                 its[0].Trim(), // Name
363                 double.Parse(its[1]), //X
364                 double.Parse(its[2]), //Y
365                 double.Parse(its[3]), //H
366                 1)); //IsKnown
367         }
368
369         //读入已知方位角信息:该节有可能不存在,也有可能有一条边,最多两条边
370         knownAzimuthes.Clear();
371         while (true)
372         {
373             buffer = sr.ReadLine(); //由于是空行到此,所以继续往下读

```

```

374     if (buffer == null) break; //文件末尾退出
375
376     if (buffer == "" || buffer[0] == '#') continue; // 略过空行和注释行
377
378     string[] its = buffer.Split(new char[1] { ',', ' ' });
379     if (its.Length != 3) break; //数据项不为3，可能是角度观测值，退出当前
380
381     KnownAzimuth ka = new KnownAzimuth();
382
383     string ptName = its[0].Trim();
384     ka.startPnt = GetCtrPoint(ptName);
385     if (ka.startPnt == null)
386     {
387         ka.startPnt = new CtrPoint(ptName, 0, 0, 0, 0); //非已知点
388         this.ctrPoints.Add(ka.startPnt);
389     }
390
391     ptName = its[1].Trim();
392     ka.endPnt = GetCtrPoint(ptName);
393     if (ka.endPnt == null)
394     {
395         ka.endPnt = new CtrPoint(ptName, 0, 0, 0, 0); //非已知点
396         this.ctrPoints.Add(ka.endPnt);
397     }
398
399     ka.azimuth = ZXY.SMath.DMS2RAD(double.Parse(its[2]));
400     knownAzimuthes.Add(ka);
401 }
402
403 //读入观测值数据
404 this.obsValues.Clear();
405 while (true)
406 {
407     //此处可能由上不是3项数据的数据行退出，也有可能是文件末尾到此
408     //所以得先处理数据，后再读文本数据，否则，容易丢失数据
409     if (buffer == null) break; //文件末尾到此，继续退出
410     if (buffer == "" || buffer[0] == '#') //空行或正常的注释略过
411     {
412         buffer = sr.ReadLine();
413         continue;
414     }
415
416     string[] its = buffer.Split(new char[1] { ',', ' ' }); //进入正常的数据处理流程
417     if (its.Length == 5)
418     {
419         string ptName = its[0].Trim();
420         CtrPoint stnPnt = GetCtrPoint(ptName);
421         if (stnPnt == null)
422         {
423             stnPnt = new CtrPoint(ptName, 0, 0, 0, 0); //非已知点
424             this.ctrPoints.Add(stnPnt);
425         }
426
427         ptName = its[1].Trim();
428         CtrPoint startPnt = GetCtrPoint(ptName);
429         if (startPnt == null)

```

```

430     {
431         startPnt = new CtrPoint(ptName, 0, 0, 0, 0); // 非已知点
432         this.ctrPoints.Add(startPnt);
433     }
434
435     ptName = its[2].Trim();
436     CtrPoint endPnt = GetCtrPoint(ptName);
437     if (endPnt == null)
438     {
439         endPnt = new CtrPoint(ptName, 0, 0, 0, 0); // 非已知点
440         this.ctrPoints.Add(endPnt);
441     }
442
443     obsValues.Add(new ObsValue(
444         stnPnt, startPnt, endPnt,
445         double.Parse(its[3]), // AngleValue
446         double.Parse(its[4]))); // Distance
447 }
448
449 buffer = sr.ReadLine();
450 }
451 }
452 }
453
454 public void WriteDataTextFile(string fileName)
455 {
456     using (System.IO.StreamWriter sw = new System.IO.StreamWriter(fileName))
457     {
458         sw.WriteLine("# Name, X, Y, Z");
459         foreach (var pt in this.ctrPoints)
460         {
461             if (pt.IsKnown == 1) sw.WriteLine(pt);
462         }
463
464         sw.WriteLine();
465         sw.WriteLine("# StartPnt, EndPnt, Azimuth");
466         foreach (var az in this.knownAzimuths)
467         {
468             sw.WriteLine(az);
469         }
470
471         sw.WriteLine();
472         sw.WriteLine("# Station, StartPnt, EndPnt, Angle, Distance");
473         foreach (var obs in this.ObsValues)
474         {
475             sw.WriteLine(obs);
476         }
477     }
478 }
479
480
481 private ObsValue SearchStartObsValue()
482 {
483     /**
484     * 搜索起始边, 首先: 搜索直接给定的坐标方位角
485     * 其次: 上述搜索不成立的情况, 搜索: 测站点与后视点均为已知点的观测值

```



```

486     * 再次：反向搜索：测站点与后视点均为已知点的观测值
487     * */
488     ObsValue obs = null;
489
490     if (this.knownAzimuthes.Count > 0)
491     {
492         foreach (var azi in this.knownAzimuthes)
493         {
494             if (azi.endPnt.IsKnown == 1)
495             {
496                 foreach (var it in obsValues)
497                 {
498                     if (it.StartPnt == azi.startPnt && it.
StnPnt == azi.endPnt)
499                     {
500                         obs = it;
501                         return obs;
502                     }
503                 }
504             }
505         }
506     }
507     else if (this.knownAzimuthes.Count == 0)
508     {
509         foreach (var it in obsValues)
510         {
511             double az = 0;
512
513             if (it.StartPnt.IsKnown == 1 && it.StnPnt.IsKnown == 1)
514             {
515                 az = ZXY.SMath.Azimuth(it.StartPnt.X, it.StartPnt
.Y, it.StnPnt.X, it.StnPnt.Y);
516                 this.knownAzimuthes.Add(new KnownAzimuth(it.
StartPnt, it.StnPnt, az));
517                 obs = it;
518             }
519
520             if (it.StnPnt.IsKnown == 1 && it.EndPnt.IsKnown == 1)
521             {
522                 az = ZXY.SMath.Azimuth(it.StnPnt.X, it.StnPnt.Y,
it.EndPnt.X, it.EndPnt.Y);
523                 this.knownAzimuthes.Add(new KnownAzimuth(it.
StnPnt, it.EndPnt, az));
524             }
525         }
526     }
527     return obs;
528 }
529
530
531 /// <summary>
532 /// 递归搜索观测值obs0的下一条边
533 /// </summary>
534 /// <param name="obs0">当前观测值</param>
535 /// <returns>1: 附和导线, -1: 不构成附和导线</returns>
536 private int SearchObsValue(ObsValue obs0)

```

```

537 {
538 //传进来的第一条边应为起始边
539 ObsValue obsi = null;
540
541 foreach (var it in obsValues)
542 {
543 if (it == obs0) continue;
544
545 if (obs0.StnPnt == it.StartPnt && obs0.EndPnt == it.StnPnt) //满足条件的下一条边
546 {
547 obsi = it;
548 break;
549 }
550 }
551
552 if (obsi == null) return -1; //没找到这样的边
553
554 this.route.Add(obsi);
555 if (obsi.StnPnt.IsKnown == 1 && obsi.EndPnt.IsKnown == 1) //附和到另一条已知边
    了, 退出
556 {
557 return 1;
558 }
559 else
560 SearchObsValue(obsi); //递归继续寻找下一条这样的边
561
562 return 0;
563 }
564
565
566 /// <summary>
567 /// 搜索正确的计算路线
568 /// </summary>
569 /// <returns>是否成功</returns>
570 public bool SearchCalRoute()
571 {
572     ObsValue obs0 = SearchStartObsValue();
573     if (obs0 == null) return false;
574
575     route.Clear(); //清空搜索线路
576
577     route.Add(obs0);
578     SearchObsValue(obs0);
579     return true;
580 }
581
582 /// <summary>
583 /// 简易平差
584 /// </summary>
585 /// <returns>0: 成功</returns>
586 public int Adjust()
587 {
588     /*
589     1. 求起始边: 后视->测站 的方位角
590     求末边: 测站->前视 的方位角
591     2. 计算角度闭合差fB, FB

```

```

592     3. 改正角度值
593     4. 推算各边坐标方位角
594     5. 计算各边的坐标增量
595     6. 计算fx, fy, fs, 1/K
596     7. 计算改正后的坐标增量
597     8. 计算各点的坐标值
598     */
599     if (this.obsValues.Count == 0) return -1; //观测值为空
600
601     if (SearchCalRoute() == false) return -2; //搜索不到正确的附和路线
602
603     double az0 = this.knownAzimuthes[0].azimuth;
604     CtrPoint startPnt0 = this.knownAzimuthes[0].startPnt;
605     CtrPoint stnPnt0 = this.knownAzimuthes[0].endPnt;
606
607     double azn = this.knownAzimuthes[1].azimuth;
608     CtrPoint stnPntn = this.knownAzimuthes[1].startPnt;
609     CtrPoint endPntn = this.knownAzimuthes[1].endPnt;
610
611     double azi = az0;
612     double n = route.Count;
613     foreach (var obs in route) //foreach (var obs in this.ObsValues)
614     {
615         obs.Azimuth = azi + obs.AngleValue + Math.PI;
616         if (obs.Azimuth >= 2 * Math.PI) obs.Azimuth -= 2 * Math.PI;
617         if (obs.Azimuth < 0) obs.Azimuth += 2 * Math.PI;
618
619         azi = obs.Azimuth;
620     }
621
622     fB = azi - azn; //单位: 弧度
623     FB = m0 * 2 * Math.Sqrt(n); //单位: 秒
624
625     //改正角度, 推算各边改正后的方位角
626     azi = az0;
627     double vi = -fB / n;
628     foreach (var obs in route) //foreach (var obs in this.ObsValues)
629     {
630         obs.VB = vi;
631         obs.AngleV = obs.AngleValue + obs.VB;
632         obs.Azimuth = azi + obs.AngleV + Math.PI;
633         if (obs.Azimuth >= 2 * Math.PI) obs.Azimuth -= 2 * Math.PI;
634         if (obs.Azimuth < 0) obs.Azimuth += 2 * Math.PI;
635         azi = obs.Azimuth;
636     }
637
638     //计算各边的坐标增量
639     double sumDX = 0, sumDY = 0;
640     sumD = 0;
641     foreach (var obs in route) //foreach (var obs in this.ObsValues)
642     {
643         if (obs.Distance <= 0) continue;
644
645         obs.DX = obs.Distance * Math.Cos(obs.Azimuth);
646         obs.DY = obs.Distance * Math.Sin(obs.Azimuth);
647         sumDX += obs.DX;

```

```

648         sumDY += obs.DY;
649         sumD += obs.Distance;
650     }
651     fx = stnPnt0.X + sumDX - stnPntn.X;
652     fy = stnPnt0.Y + sumDY - stnPntn.Y;
653     fs = Math.Sqrt(fx * fx + fy * fy);
654     K = sumD / fs;
655
656     //改正坐标增量, 计算各点坐标
657     foreach (var obs in route) //foreach (var obs in this.ObsValues)
658     {
659         if (obs.Distance <= 0) continue;
660
661         obs.VDX = -fx / sumD * obs.Distance;
662         obs.VDY = -fy / sumD * obs.Distance;
663
664         obs.EndPnt.X = obs.StnPnt.X + obs.DX + obs.VDX;
665         obs.EndPnt.Y = obs.StnPnt.Y + obs.DY + obs.VDY;
666     }
667
668     return 0;
669 }
670
671 private CtrPoint GetCtrPoint(string pointName)
672 {
673     foreach (var pt in this.ctrPoints)
674     {
675         if (pt.Name == pointName)
676             return pt;
677     }
678
679     return null;
680 }
681
682 public void OnDraw(DrawingCanvas canvas)
683 {
684     if (this.ctrPoints.Count < 1) return;
685
686     GetGaussXySize();
687
688     maxVX = canvas.ActualWidth - 20;
689     maxVY = canvas.ActualHeight;
690
691     double kx = maxVX / (maxY - minY);
692     double ky = maxVY / (maxX - minX);
693     k = kx <= ky ? kx : ky;
694
695     canvas.ClearAll(); //先清除屏幕
696
697     //画观测值
698     double x0, y0, x1, y1, x2, y2; //画直线的两个端点
699     foreach (var it in this.obsValues)
700     {
701         if (it.StnPnt.X <= 0 && it.StnPnt.Y <= 0) continue; //略过坐标为0
702         GaussXyToViewXy(it.StnPnt.X, it.StnPnt.Y, out x0, out y0);

```

```

703
704         if (it.StartPnt.X > 0 && it.StartPnt.Y > 0)
705         {
706             GaussXyToViewXy(it.StartPnt.X, it.StartPnt.Y, out x1, out
y1);
707             canvas.DrawLine(x1, y1, x0, y0, Brushes.Black, 1);
708         }
709
710         if (it.EndPnt.X > 0 && it.EndPnt.Y > 0)
711         {
712             GaussXyToViewXy(it.EndPnt.X, it.EndPnt.Y, out x2, out y2)
;
713             canvas.DrawLine(x0, y0, x2, y2, Brushes.Black, 1);
714         }
715     }
716
717     //再画控制点
718     foreach (var pt in this.ctrPoints)
719     {
720         if (pt.X <= 0 && pt.Y <= 0) continue; //排除坐标为0的点
721
722         GaussXyToViewXy(pt.X, pt.Y, out x0, out y0);
723         if (pt.IsKnown == 1)
724             canvas.DrawKnCtrPnt(x0, y0, Brushes.Black, 1);
725         else
726             canvas.DrawCtrPnt(x0, y0, Brushes.Black, 1);
727         canvas.DrawText(pt.Name, x0 + 10, y0 - 7);
728     }
729 }
730
731 private void GaussXyToViewXy(double xt, double yt, out double xp, out double yp)
732 {
733     //xp = x0 + kx(yt - yn);
734     //yp = y1 - (y0 + ky * (xt - xn));
735     // x0 = y0 = 0 且 kx = ky = k, 故以上公式简化为:
736
737     xp = 5 + k * (yt - minY); //x0 = 5;
738     yp = maxY - (5 + k * (xt - minX)); //y0=5;
739 }
740
741 private void GetGaussXySize()
742 {
743     minX = this.ctrPoints[0].X; minY = this.ctrPoints[0].Y;
744     maxX = this.ctrPoints[0].X; maxY = this.ctrPoints[0].Y;
745
746     for (int i = 1; i < this.ctrPoints.Count; i++) //如果只有一个点, 由循环条件知, 不
会执行循环体
747     {
748         if (this.ctrPoints[i].X <= 0 && this.ctrPoints[i].Y <= 0) continue;
749
750         if (this.ctrPoints[i].X < minX) minX = this.ctrPoints[i].X;
751         if (this.ctrPoints[i].Y < minY) minY = this.ctrPoints[i].Y;
752
753         if (this.ctrPoints[i].X > maxX) maxX = this.ctrPoints[i].X;
754         if (this.ctrPoints[i].Y > maxY) maxY = this.ctrPoints[i].Y;
755     }

```

```

756
757 // 针对一个点或点范围较小的情况，进行范围扩展
758 if (minX + 10 > maxX) { maxX = minX + 10; minX = maxX - 20; }
759 if (minY + 10 > maxY) { maxY = minY + 10; minY = maxY - 20; }
760 }
761 }
762 }
763
764 using System.Globalization;
765 using System.Windows;
766 using System.Windows.Media;
767
768 namespace SurApp.models
769 {
770     public class DrawingCanvas : System.Windows.Controls.Canvas
771     {
772         private VisualCollection visuals;
773
774         public DrawingCanvas()
775         {
776             visuals = new VisualCollection(this);
777         }
778
779         // 获取 Visual 的个数
780         protected override int VisualChildrenCount
781         {
782             get { return visuals.Count; }
783         }
784
785         // 获取 Visual
786         protected override Visual GetVisualChild(int index)
787         {
788             return visuals[index];
789         }
790
791         // 添加 Visual
792         public void AddVisual(Visual visualObject)
793         {
794             visuals.Add(visualObject);
795         }
796
797         // 删除 Visual
798         public void RemoveVisual(Visual visualObject)
799         {
800             base.RemoveLogicalChild(visualObject);
801         }
802
803         // 命中测试
804         public DrawingVisual GetVisual(System.Windows.Point point)
805         {
806             HitTestResult hitResult = VisualTreeHelper.HitTest(this,
point);
807             return hitResult.VisualHit as DrawingVisual;
808         }
809
810         public void ClearAll()

```

```

811         {
812             this.visuals.Clear();
813         }
814
815
816         //使用DrawVisual画Polyline
817         public void DrawLine(double x0, double y0, double x1, double y1,
Brush color, double thickness)
818         {
819             DrawingVisual visualLine = new DrawingVisual();
820             DrawingContext dc = visualLine.RenderOpen();
821             Pen pen = new Pen(color, thickness);
822             pen.Freeze(); //冻结画笔, 这样能加快绘图速度
823             dc.DrawLine(pen, new Point(x0, y0), new Point(x1, y1));
824
825             dc.Close();
826             visuals.Add(visualLine);
827         }
828
829         public void DrawText(string text, double x, double y)
830         {
831             DrawingVisual visualText = new DrawingVisual();
832             DrawingContext dc = visualText.RenderOpen();
833             Typeface tp = new Typeface(new FontFamily("宋体"),
FontStyles.Normal, FontWeights.Normal, FontStretches.Normal);
834             FormattedText ft = new FormattedText(text, CultureInfo.
CurrentCulture,
835             FlowDirection.LeftToRight, tp, 12, Brushes.Black);
836             dc.DrawText(ft, new Point(x, y));
837             dc.Close();
838             visuals.Add(visualText);
839         }
840
841         //使用DrawVisual画Circle, 用作控制点
842         public void DrawCtrPnt(double x, double y, Brush color, double
thickness)
843         {
844             DrawingVisual visualCircle = new DrawingVisual();
845             DrawingContext dc = visualCircle.RenderOpen();
846             Pen pen = new Pen(color, thickness);
847             pen.Freeze(); //冻结画笔, 这样能加快绘图速度
848             dc.DrawEllipse(Brushes.LemonChiffon, pen, new Point(x,
y), 5, 5);
849             dc.Close();
850             visuals.Add(visualCircle);
851         }
852
853         //使用DrawVisual画Circle, 用作控制点
854         public void DrawKnCtrPnt(double x, double y, Brush color, double
thickness)
855         {
856             DrawingVisual visualCircle = new DrawingVisual();
857             DrawingContext dc = visualCircle.RenderOpen();
858             Pen pen = new Pen(color, thickness);
859             pen.Freeze(); //冻结画笔, 这样能加快绘图速度
860             dc.DrawEllipse(Brushes.Black, pen, new Point(x, y), 1, 1)

```

```
861      ;  
      dc.DrawLine(pen, new Point(x-5, y+2.9), new Point(x+5, y  
862      +2.9));  
      dc.DrawLine(pen, new Point(x + 5, y + 2.9), new Point(x,  
863      y - 5.8));  
      dc.DrawLine(pen, new Point(x, y - 5.8), new Point(x - 5,  
864      y + 2.9));  
      dc.Close();  
865      visuals.Add(visualCircle);  
866      }  
867      }  
868      }  
869      }  
870      }
```


第四章 高斯投影正反算与换带

高斯投影是为了解决球面与平面之间的坐标映射问题，即大地坐标 (B, L) 与高斯平面直角坐标 (x, y) 之间的相互换算，以及不同带之间的高斯坐标的换算问题。

本章将运用 C# 编程语言编写一个通用的高斯投影程序，用于 1954 北京坐标系、1980 西安坐标系、WGS84 坐标系以及 CGCS2000 大地坐标系或自定义参考椭球的高斯投影正反算与换带计算。

4.1 高斯投影的数学模型

为了编制正确而且高效的高斯投影与换带程序，我们首先需要分析高斯投影的数学模型，也就是我们常说的算法。

本章所引用的公式来自：孔祥元, 郭际明, 刘宗泉. 大地测量学基础-2 版. 武汉: 武汉大学出版社, 2010.5, 以下将这本书简称为大地测量学基础或大地测量学。

高斯投影是在椭球的几何参数 (长半轴 a、短半轴 b、扁率 α) 确定的条件下，根据给定的数学模型来进行计算的，我们首先分析这些计算公式与数学模型。

1. 基本公式

(a) 扁率:

$$\alpha = \frac{a - b}{a}$$

(b) 第一偏心率:

$$e = \sqrt{\frac{a^2 - b^2}{a^2}}$$

(c) 第二偏心率:

$$e' = \sqrt{\frac{a^2 - b^2}{b^2}}$$

(d) 子午圈曲率半径:

$$M = \frac{a(1 - e^2)}{(1 - e^2 \sin^2 B)^{\frac{3}{2}}}$$

(e) 卯酉圈曲率半径:

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 B}}$$

(f) 辅助符号:

$$t = \tan B \quad \eta = e' \cos B$$

2. 椭球面梯形图幅面积计算

$$P = \frac{b^2}{2}(L_2 - L_1) \left| \frac{\sin B}{1 - e^2 \sin^2 B} + \frac{1}{2e} \ln \frac{1 + e \sin B}{1 - e \sin B} \right|_{B_1}^{B_2}$$

公式引用自《大地测量学基础》第 121 页 (4-140)。

$$P = b^2(L_2 - L_1) \left| \sin B + \frac{2}{3}e^2 \sin^3 B + \frac{3}{5}e^4 \sin^5 B + \frac{4}{7}e^6 \sin^7 B \right|_{B_1}^{B_2}$$

公式引用自《大地测量学基础》第 121 页 (4-142)，该公式为 (4-140) 的展开式。

3. 子午线弧长

$$X = a_0 B - \frac{a_2}{2} \sin 2B + \frac{a_4}{4} \sin 4B - \frac{a_6}{6} \sin 6B + \frac{a_8}{8} \sin 8B \quad (4.1)$$

式中：

$$\left. \begin{aligned} a_0 &= m_0 + \frac{m_2}{2} + \frac{3}{8}m_4 + \frac{5}{16}m_6 + \frac{35}{128}m_8 \\ a_2 &= \frac{m_2}{2} + \frac{m_4}{2} + \frac{15}{32}m_6 + \frac{7}{16}m_8 \\ a_4 &= \frac{m_4}{8} + \frac{3}{16}m_6 + \frac{7}{32}m_8 \\ a_6 &= \frac{m_6}{32} + \frac{m_8}{16} \\ a_8 &= \frac{m_8}{128} \end{aligned} \right\} \quad (4.2)$$

式中 m_0, m_2, m_4, m_6, m_8 的值为：

$$\left. \begin{aligned} m_0 &= a(1 - e^2) \\ m_2 &= \frac{3}{2}e^2 m_0 \\ m_4 &= \frac{5}{4}e^2 m_2 \\ m_6 &= \frac{7}{6}e^2 m_4 \\ m_8 &= \frac{9}{8}e^2 m_6 \end{aligned} \right\} \quad (4.3)$$

公式引用自《大地测量学基础》第 115 页 (4-101)、(4-100) 与 (4-72)。

或用很多书上都引用的公式：

$$X = c[\beta_0 B + (\beta_2 \cos B + \beta_4 \cos^3 B + \beta_6 \cos^5 B + \beta_8 \cos^7 B) \sin B] \quad (4.4)$$

式中 c 为极曲率半径，其值为： $c = a^2/b$ ， $\beta_0, \beta_2, \beta_4, \beta_6, \beta_8$ 的值为：

$$\left. \begin{aligned} \beta_0 &= 1 - \frac{3}{4}e'^2 + \frac{45}{64}e'^4 - \frac{175}{256}e'^6 + \frac{11025}{16384}e'^8 \\ \beta_2 &= \beta_0 - 1 \\ \beta_4 &= \frac{15}{32}e'^4 - \frac{175}{384}e'^6 + \frac{3675}{8192}e'^8 \\ \beta_6 &= -\frac{35}{96}e'^6 + \frac{735}{2048}e'^8 \\ \beta_8 &= \frac{315}{1024}e'^8 \end{aligned} \right\} \quad (4.5)$$

公式引用自《大地测量学基础》第 115 页 (4-107)、与 (4-108)。

或用以下公式：

$$X = d[\alpha_0 B - (\alpha_2 \sin B + \alpha_4 \sin^3 B + \alpha_6 \sin^5 B + \alpha_8 \sin^7 B) \cos B] \quad (4.6)$$

式中 d 为赤道子午曲率半径, 其值为 $d = b^2/a$, $\alpha_0, \alpha_2, \alpha_4, \alpha_6, \alpha_8$ 的值为:

$$\left. \begin{aligned} \alpha_0 &= 1 + \frac{3}{4}e^2 + \frac{45}{64}e^4 + \frac{175}{256}e^6 + \frac{11025}{16384}e^8 \\ \alpha_2 &= \alpha_0 - 1 \\ \alpha_4 &= \frac{15}{32}e^4 + \frac{175}{384}e^6 + \frac{3675}{8192}e^8 \\ \alpha_6 &= \frac{35}{96}e^6 + \frac{735}{2048}e^8 \\ \alpha_8 &= \frac{315}{1024}e^8 \end{aligned} \right\} \quad (4.7)$$

4. 坐标正算

$$\left. \begin{aligned} x &= X + \frac{N}{2} \sin B \cos B l^2 + \frac{N}{24} \sin B \cos^3 B (5 - t^2 + 9\eta^2 + 4\eta^4) l^4 \\ &\quad + \frac{N}{720} \sin B \cos^5 B (61 - 58t^2 + t^4) l^6 \\ y &= N \cos B l + \frac{N}{6} \cos^3 B (1 - t^2 + \eta^2) l^3 \\ &\quad + \frac{N}{120} \cos^5 B (5 - 18t^2 + t^4 + 14\eta^2 - 58\eta^2 t^2) l^5 \end{aligned} \right\} \quad (4.8)$$

公式引用自《大地测量学基础》第 169 页 (4-367)。

5. 坐标反算

$$\left. \begin{aligned} B &= B_f - \frac{t_f}{2M_f N_f} y^2 + \frac{t_f}{24M_f N_f^3} (5 + 3t_f^2 + \eta_f^2 - 9\eta_f^2 t_f^2) y^4 \\ &\quad - \frac{t_f}{720M_f N_f^5} (61 + 90t_f^2 + 45t_f^4) y^6 \\ l &= \frac{1}{N_f \cos B_f} y - \frac{1}{6N_f^3 \cos B_f} (1 + 2t_f^2 + \eta_f^2) y^3 \\ &\quad + \frac{1}{120N_f^5 \cos B_f} (5 + 28t_f^2 + 24t_f^4 + 6\eta_f^2 + 8\eta_f^2 t_f^2) y^5 \end{aligned} \right\} \quad (4.9)$$

公式引用自《大地测量学基础》第 171 页 (4-383)。

6. 平面子午线收敛角计算

利用 (B, l) 计算公式如下:

$$\gamma = \sin B \cdot l + \frac{1 + 3\eta^2 + 2\eta^4}{3} \sin B \cos^2 B \cdot l^3 + \frac{2 - t^2}{15} \sin B \cos^4 B \cdot l^5$$

公式引用自《大地测量学基础》第 181 页 (4-408)。

利用 (x, y) 计算公式如下:

$$\gamma = \frac{1}{N_f} t_f y - \frac{1 + t_f^2 - \eta_f^2}{3N_f^3} t_f y^3 + \frac{2 + 5t_f^2 + 3t_f^4}{15N_f^5} t_f y^5$$

公式引用自《大地测量学基础》第 182 页 (4-410)。

7. 长度比计算

利用 (B, l) 计算公式如下:

$$m = 1 + \frac{1}{2}l^2 \cos^2 B(1 + \eta^2) + \frac{1}{24}l^4 \cos^4 B(5 - 4t^2)$$

公式引用自《大地测量学基础》第 189 页 (4-447)。

利用 (x, y) 计算公式如下:

$$m = 1 + \frac{y^2}{2R^2} + \frac{y^4}{24R^4}$$

式中 $R = \sqrt{MN}$, 公式引用自《大地测量学基础》第 189 页 (4-451)。

4.2 程序功能分析与设计

4.2.1 高斯投影的主要内容

1. 坐标正算: 将点的大地坐标转换成高斯投影平面直角坐标。
2. 坐标反算: 将点的高斯投影平面直角坐标转换成大地坐标。
3. 换带计算: 将某带的点的高斯投影平面直角坐标转换成邻带或某中央子午线经度的高斯投影平面直角坐标。
4. 其他计算: 计算子午线收敛角、长度比等。

4.2.2 参考椭球类的设计

分析以上各个计算公式发现, 如果椭球长半轴与扁率确定, 参考椭球的第一偏心率 e 、第二偏心率 e' , 子午线弧长计算的辅助计算参数 $(m_0, m_2, m_4, m_6, m_8)$ 与 $(a_0, a_2, a_4, a_6, a_8)$ 也就确定了, 其中的 $(m_0, m_2, m_4, m_6, m_8)$ 作为计算 $(a_0, a_2, a_4, a_6, a_8)$ 的值使用过一次。也就是说这些参数对于某一种确定的参考椭球是常数。而 (M, N, t, η) 则是纬度 B 的函数。

我们新建一 WPF 项目, 命名为 GaussProj, 程序中我们将应用 WPF 技术编写图形界面。

在高斯投影中由于要处理角度与点等数据, 我们可以将前面的角度处理函数 DMS2RAD 与 RAD2DMS 与 SPoint 点类拷贝到当前项目中来, 也可以将其打包成一 Class Library(.NET Framework) 即类库文件引用到当前项目中, 在当前项目中尽量不修改前面的各个功能代码。

基于上一小节的分析, 我们新建一椭球类 (Spheroid), 在其中我们定义以上与椭球类型相关的元素。代码如下所示:

```

1 namespace GaussProj
2 {
3     /// <summary>
4     /// 参考椭球
5     /// </summary>
6     public class Spheroid
7     {
8         /// <summary>
9         /// 长半轴
10        /// </summary>
11        public double a { get; set; }
12
13        /// <summary>

```

```

14      /// 短半轴
15      /// </summary>
16      public double b { get; set; }
17
18      /// <summary>
19      /// 扁率分母
20      /// </summary>
21      public double f { get; set; }  //(a-b)/a = 1/f
22
23      /// <summary>
24      /// 第一偏心率的平方
25      /// </summary>
26      public double e2 { get; set; }
27
28      /// <summary>
29      /// 第二偏心率的平方
30      /// </summary>
31      public double eT2 { get; set; }
32
33      //计算子午线弧长时的各个系数项
34      private double a0;
35      private double a2;
36      private double a4;
37      private double a6;
38      private double a8;
39  }
40 }

```

在各项计算中，第一偏心率与第二偏心率的直接使用较少，其平方值用的较多，因此在 Spheroid 类我们直接用其平方值。

在 Spheroid 类中我们可以如下直接定义构造函数用于初始化其各个字段 (Field):

```

1  /// <summary>
2  /// 构造函数
3  /// </summary>
4  private Spheroid(double semimajor_axis, double inverse_flattening)
5  {
6      this.a = semimajor;
7      this.f = inverse_flattening;
8      .....
9  }

```

但这样我们可能无法为已知的一些参考椭球直接提供参数，所以我们将构造函数定义为 private，让用户无法通过 new 直接创建实例化对象。我们设计类的静态函数 Create**** 来创建类 Spheroid 的实例化对象，其代码如下：

```

1  /// <summary>
2  /// 构造函数
3  /// </summary>
4  private Spheroid() {}
5
6  /// <summary>
7  /// 初始化椭球参数的内部函数
8  /// </summary>
9  private void Init(double semimajor_axis, double inverse_flattening)
10 {
11     this.a = semimajor_axis; this.f = inverse_flattening;

```

```

12     b = a * (1 - 1 / f);
13
14     e2 = 1 - b / a * b / a;
15     eT2 = a / b * a / b - 1;
16
17     double m0 = a * (1 - e2);
18     double m2 = 3.0 / 2.0 * e2 * m0;
19     double m4 = 5.0 / 4.0 * e2 * m2;
20     double m6 = 7.0 / 6.0 * e2 * m4;
21     double m8 = 9.0 / 8.0 * e2 * m6;
22
23     a0 = m0 + m2 / 2.0 + 3.0 / 8.0 * m4 + 5.0 / 16.0 * m6
24           + 35.0 / 128.0 * m8;
25     a2 = m2 / 2.0 + m4 / 2.0 + 15.0 / 32.0 * m6 + 7.0 / 16.0 * m8;
26     a4 = m4 / 8.0 + 3.0 / 16.0 * m6 + 7.0 / 32.0 * m8;
27     a6 = m6 / 32.0 + m8 / 16.0;
28     a8 = m8 / 128.0;
29 }
30
31 public static Spheroid CreateBeiJing1954()
32 {
33     Spheroid spheroid = new Spheroid();
34     spheroid.Init(6378245, 298.3);
35     return spheroid;
36 }
37
38 public static Spheroid CreateXian1980()
39 {
40     Spheroid spheroid = new Spheroid();
41     spheroid.Init(6378140, 298.257);
42     return spheroid;
43 }
44
45 public static Spheroid CreateWGS1984()
46 {
47     Spheroid spheroid = new Spheroid();
48     spheroid.Init(6378137, 298.257223563);
49     return spheroid;
50 }
51
52 public static Spheroid CreateCGCS2000()
53 {
54     Spheroid spheroid = new Spheroid();
55     spheroid.Init(6378137, 298.257222101);
56     return spheroid;
57 }
58
59 public static Spheroid CreateCoordinateSystem(double semimajor_axis,
60     double inverse_flattening)
61 {
62     Spheroid spheroid = new Spheroid();
63     spheroid.Init(semimajor_axis, inverse_flattening);
64     return spheroid;
65 }

```

这种将构造函数设为 private，通过静态函数来创建实例化对象的技术在软件设计中会经常用到。同时在设计类的方法时，应注意访问权限的设置，如上面代码中 Init 函数，在类中用于

初始化椭球的各项几何参数，并不需要在类外来调用它，所以我们将其设置为 private 权限。

4.2.3 高斯投影正算功能的实现

有了类 Spheroid 的设计，我们先来完成高斯投影正算功能。为了避免在计算中频繁的进行度分秒与弧度之间的转换问题，我们设定除了特别声明之外，在类 Spheroid 中所用的角度均为弧度。

我们将高斯投影正算的函数名称定义为 BLtoXY，其设计如下：

```

1  /// <summary>
2  /// 高斯投影正算
3  /// </summary>
4  /// <param name="B">纬度,单位:弧度</param>
5  /// <param name="L">经度,单位:弧度</param>
6  /// <param name="L0">中央子午线经度,单位:弧度</param>
7  /// <param name="x">高斯平面x坐标</param>
8  /// <param name="y">高斯平面y坐标</param>
9  public void BLtoXY(double B, double L, double L0,
10     out double x, out double y)
11  {
12     .....
13 }
```

由于函数的返回值为两个值 x 与 y，无法以函数的返回值 return 的形式返回计算结果，所以我们用函数参数 out 的形式将计算结果返回。

由前面的高斯投影正算公式分析可知，高斯投影正算的计算较为简单，没有复杂的逻辑，先计算经差，然后计算子午线弧长后就可以直接写计算坐标 x, y 的算法了。但公式较为复杂，极容易写错，公式中具有大量的平方、四次方等变量。因此在编程时应将这些变量命名为与其相似的形式并提前计算。相关代码如下：

```

1  /// <summary>
2  /// 计算子午线弧长
3  /// </summary>
4  /// <param name="B">纬度(单位:弧度)</param>
5  /// <returns>子午线弧长</returns>
6  private double funX(double B)
7  {
8     return a0 * B - a2 / 2.0 * Math.Sin(2 * B)
9         + a4 / 4.0 * Math.Sin(4 * B)
10        - a6 / 6.0 * Math.Sin(6 * B)
11        + a8 / 8.0 * Math.Sin(8 * B);
12 }
13
14 private double funN(double sinB)
15 {
16     return a / Math.Sqrt(1 - e2 * sinB * sinB);
17 }
18
19 /// <summary>
20 /// 高斯投影正算
21 /// </summary>
22 /// <param name="B">纬度,单位:弧度</param>
23 /// <param name="L">经度,单位:弧度</param>
24 /// <param name="L0">中央子午线经度,单位:弧度</param>
25 /// <param name="x">高斯平面x坐标</param>
```

```

26 /// <param name="y">高斯平面y坐标</param>
27 public void BLtoXY(double B, double L, double L0,
28     out double x, out double y)
29 {
30     double l = L - L0; //计算经差
31
32     double sinB = Math.Sin(B);
33     double cosB = Math.Cos(B);
34     double cosB2 = cosB * cosB;
35     double cosB4 = cosB2 * cosB2;
36     double t = Math.Tan(B);
37     double t2 = t * t;
38     double t4 = t2 * t2;
39     double g2 = eT2 * cosB * cosB;
40     double g4 = g2 * g2;
41     double l2 = l * l;
42     double l4 = l2 * l2;
43
44     double X = funX(B); //计算子午线弧长
45     double N = funN(sinB);
46
47     x = X + 0.5 * N * sinB * cosB * l2 * (1
48         + cosB2 / 12.0 * (5 - t2 + 9 * g2 + 4 * g4) * l2
49         + cosB4 / 360.0 * (61 - 58 * t2 + t4) * l4);
50
51
52     y = N * cosB * l * (1
53         + cosB2 * (1 - t2 + g2) * l2 / 6.0
54         + cosB4 * (5 - 18 * t2 + t4 + 14 * g2 - 58 * g2 * t2)
55         * l4 / 120.0);
56 }

```

利用 BLtoXY 函数就可以进行高斯投影正算了，其算法流程为：

```

1 //创建克拉索夫斯基参考椭球
2 Spheroid spheroid = Spheroid.CreateBeiJing1954();
3
4 //传入点的纬度B、经度与中央子午线经度，单位为弧度
5 double B = ZXY.SMath.DMS2RAD(21.58470845);
6 double L = ZXY.SMath.DMS2RAD(113.25314880);
7 double L0 = ZXY.SMath.DMS2RAD(111);
8 double x, y;
9
10 spheroid.BLtoXY(B, L, L0, out x, out y);

```

点的纬度、经度为： $B = 21^{\circ}58'47.0845''$, $L = 113^{\circ}25'31.4880''$ ，中央子午线经度为： $L0 = 111^{\circ}$ ，计算出的坐标为： $x = 2433586.692$, $y = 250547.403$ ，坐标 y 为点的自然坐标，未加常数 500km 与带号。

4.2.4 高斯投影反算功能的实现

由高斯投影反算公式分析，在反算时需要首先计算底点纬度。底点纬度需要由子午线弧长公式进行反算，由该公式可以看出，已知 $X=x$ 时，这个函数在计算 B 值时它并不是一个线型函数，不能直接计算。解决这类问题的计算方法就是迭代计算，我们将公式进行变换，如下所示：

$$B = (X + \frac{a_2}{2} \sin 2B - \frac{a_4}{4} \sin 4B + \frac{a_6}{6} \sin 6B - \frac{a_8}{8} \sin 8B) / a_0$$

在该公式中，两边都有 B ，我们将公式右边的 B 赋初始值 $B_0 = X/a_0$ 代入可以计算出新的 B 值，循环进行计算，由于该迭代收敛，两值之差 $B - B_0$ 在一定范围内时我们认为其值 B 即为我们的解。

因此底点纬度计算函数设计为：

```

1  /// <summary>
2  /// 计算底点纬度
3  /// </summary>
4  /// <param name="x">高斯平面坐标x</param>
5  /// <returns>底点纬度</returns>
6  private double funBf(double x)
7  {
8      double sinB, sin2B, sin4B, sin6B, sin8B;
9      double Bf0 = x / a0, Bf = 0; // 子午线弧长的初值
10     int i = 0;
11     while (i < 10000) // 设定最大迭代次数
12     {
13         sinB = Math.Sin(Bf0);
14         sin2B = Math.Sin(2 * Bf0);
15         sin4B = Math.Sin(4 * Bf0);
16         sin6B = Math.Sin(6 * Bf0);
17         sin8B = Math.Sin(8 * Bf0);
18         Bf = (x
19             + a2 * sin2B / 2
20             - a4 * sin4B / 4
21             + a6 * sin6B / 6
22             - a8 * sin8B / 8) / a0;
23
24         if (Math.Abs(Bf - Bf0) < 1e-10) // 计算精度
25             return Bf;
26         else
27         {
28             Bf0 = Bf;
29             i++;
30         }
31     }
32     return -1e12;
33 }

```

在底点纬度计算出以后，高斯投影反算计算就没有难度了。我们将函数名命名为 XYtoBL，其函数设计为：

```

1  public void XYtoBL(double x, double y, double L0,
2      out double B, out double L)
3  {
4      double Bf = funBf(x);
5
6      double cosBf = Math.Cos(Bf);
7      double gf2 = eT2 * cosBf * cosBf;
8      double gf4 = gf2 * gf2;
9      double tf = Math.Tan(Bf);
10     double tf2 = tf * tf;
11     double tf4 = tf2 * tf2;
12     double sinB = Math.Sin(Bf);
13     double Nf = funN(sinB);
14     double Mf = funM(sinB);
15     double Nf2 = Nf * Nf;

```

```

16     double Nf4 = Nf2 * Nf2;
17     double y2 = y * y;
18     double y4 = y2 * y2;
19
20     B = Bf + tf * y2 / Mf / Nf * 0.5 * (
21         -1.0
22         + y2 * (5.0 + 3.0 * tf2 + gf2 - 9.0 * gf2 * tf2) / 12.0 / Nf2
23         - y4 * (61.0 + 90.0 * tf2 + 45.0 * tf4) / 360.0 / Nf4);
24
25     double l = y / Nf / cosBf * (
26         1.0
27         - y2 / 6.0 / Nf2 * (1.0 + 2.0 * tf2 + gf2)
28         + y4 / 120.0 / Nf4
29         * (5.0 + 28.0 * tf2 + 24.0 * tf4 + 6.0 * gf2 + 8.0 * gf2 * tf2));
30
31     L = L0 + l;
32 }

```

该函数中还有 Nf 与 Mf 需要计算, Nf 的计算同前面的 funN 函数, Mf 的计算函数为

```

1 private double funM(double sinB)
2 {
3     return a * (1 - e2) * Math.Pow(1 - e2 * sinB * sinB, -1.5);
4 }

```

有了高斯投影反算函数 XYtoBL, 就可以比较容易的写出其反算示例了, 如下代码所示:

```

1 //创建克拉索夫斯基参考椭球
2 Spheroid spheroid = Spheroid.CreateBeiJing1954();
3
4 //传入点的X、Y坐标与中央子午线经度(单位为弧度)
5 double x=2433586.692, y=250547.403;
6 double L0 = ZXY.SMath.DMS2RAD(111);
7
8 double B, L;
9 spheroid.XYtoBL(x, y, L0, out B, out L);
10 //B= ZXY.SMath.RAD2DMS(B);
11 //L= ZXY.SMath.RAD2DMS(L);

```

传入的 y 坐标应为真实坐标值, 应不包括 500km 与带号等。如果计算的经纬度需向界面展示, 还应像上述代码后两行所示将其值转换为度分秒形式。

计算出的点的纬度与经度为: $B = 21^{\circ}58'47.0845''$, $L = 113^{\circ}25'31.4880''$

计算点的子午线收敛角与计算某点处的长度变形值均比较简单, 可以在正算或反算时将其同时计算出, 大家可以自己练习完成, 在此就不再讲述了。

4.2.5 换带计算

换带计算其实质就是变换坐标系的中央子午线位置。因此首先应根据点的坐标反算出点的经纬度, 然后再根据新的坐标系中央子午线位置计算出点在新的坐标系中的高斯平面坐标。

也就是先反算, 再正算, 注意此处的反算与正算的中央子午线经度值是不一样的。

该函数我们命名为 XYtoXY, 其设计为如下代码:

```

1 /// <summary>
2 /// 高斯投影换带
3 /// </summary>
4 /// <param name="ox">点在源坐标系的x坐标</param>
5 /// <param name="oy">点在源坐标系的y坐标</param>

```

```

6  /// <param name="oL0">源坐标系的中央子午线经度, 单位:弧度</param>
7  /// <param name="nL0">目标坐标系的中央子午线经度, 单位:弧度</param>
8  /// <param name="nx">点在目标坐标系的x坐标</param>
9  /// <param name="ny">点在目标坐标系的y坐标</param>
10 public void XYtoXY(double ox, double oy,
11     double oL0, double nL0,
12     out double nx, out double ny)
13 {
14     double B, L;
15     XYtoBL(ox, oy, oL0, out B, out L); // 高斯投影反算
16     BLtoXY(B, L, nL0, out nx, out ny); // 高斯投影正算
17 }

```

高斯投影换带的外部调用可以按如下方式写:

```

1 double oldX = 3275110.535, oldY = 235437.233;
2
3 double oldL0 = ZXY.SMath.DMS2RAD(117);
4 double newL0 = ZXY.SMath.DMS2RAD(120);
5
6 double newX, newY;
7 Spheroid spheroid = Spheroid.CreateBeiJing1954();
8 spheroid.XYtoXY(oldX, oldY, oldL0, newL0, out newX, out newY);

```

计算出的点在新坐标系下的坐标为 (3272782.315, -55299.545)。

至此, 我们已经完成了高斯投影的全部计算功能了。需要注意的是以上的函数调用中的角度均使用了弧度的形式, 在外部调用时可以利用前面所讲的度分秒化弧度和弧度化度分秒函数先行转换。

4.3 图形界面程序编写

以上我们已经将主要的算法编写完毕, 下面我们将用 C# 的 WPF 技术编写图形界面, 让我们的程序变得更加实用与易用。

4.3.1 单点高斯投影正反算图形界面编写

我们先设计一简单的界面进行高斯投影正反算计算, 以验证程序功能。程序中我们默认的坐标系为 1954 北京坐标系, 界面设计如图 4.1 所示, 界面我们采用 Grid 布局, 将其划分成三行三列, 将我们的控件布置在相应的单元格中, Textblock 控件用来描述文字类的标题, TextBox 用来输入经纬度与坐标数据, 中间用 Button 控件响应事件。

图 4.1 简单的高斯投影正反算界面

相应的主要界面代码如下:

```

1 <TextBlock Text="大地坐标" HorizontalAlignment="Center"
2     VerticalAlignment="Center"/>
3 <TextBlock Text="B=" Grid.Row="1" Margin="5"/>
4 <TextBox x:Name="textBox_B" Grid.Row="1"
5     Text=""
6     VerticalAlignment="Center" Margin="25,0,0,0"/>
7 <TextBlock Text="L=" Margin="5" Grid.Row="2" />
8 <TextBox x:Name="textBox_L"
9     Text=""
10    VerticalAlignment="Center" Margin="25,0,0,0" Grid.Row="2" />
11
12 <TextBlock Text="L0=" Grid.Column="1"
13     Margin="5,0,0,0" VerticalAlignment="Center"/>
14 <TextBox x:Name="textBox_L0"
15     Text=""
16     Grid.Column="1"
17     Margin="30,0,3,0" VerticalAlignment="Center"/>
18 <Button Content="(B,L)→(x,y)" Grid.Column="1" Grid.Row="1"
19     Click="BLtoXYButton_Click" Margin="3,3"/>
20 <Button Content="(B,L)&lt;->(x,y)" Grid.Column="1" Grid.Row="2"
21     Click="XYtoBLButton_Click" Margin="3,3"/>
22
23 <TextBlock Text="高斯投影坐标" Grid.Column="2"
24     HorizontalAlignment="Center" VerticalAlignment="Center"/>
25 <TextBlock Text="x=" Margin="5" Grid.Row="1" Grid.Column="2"/>
26 <TextBox x:Name="textBox_x" Grid.Row="1" Grid.Column="2"
27     Text=""
28     Margin="25,0,0,0" VerticalAlignment="Center" />
29 <TextBlock Text="y=" Grid.Row="2" Grid.Column="2" Margin="5,0,0,0"/>
30 <TextBox x:Name="textBox_y"
31     Text=""
32     Grid.Row="2" Grid.Column="2"
33     Margin="25,0,0,0" VerticalAlignment="Center" />

```

请注意,在这段 xaml 界面代码中,由于我们的算法需要与界面上的 TextBox 控件进行数据交换,每个 TextBox 都需要命名(如上的每个 TextBox 控件都有一个 x:Name 属性)。

相应的正算与反算按钮响应事件代码如下:

```

1 private void BLtoXYButton_Click(object sender, RoutedEventArgs e)
2 {
3     double B, L, L0, x, y;
4     double.TryParse(this.textBox_B.Text, out B);
5     double.TryParse(this.textBox_L.Text, out L);
6     double.TryParse(this.textBox_L0.Text, out L0);
7
8     Spheroid proj = Spheroid.CreateBeiJing1954();
9     proj.BLtoXY(
10         ZXY.SMath.DMS2RAD(B),
11         ZXY.SMath.DMS2RAD(L),
12         ZXY.SMath.DMS2RAD(L0),
13         out x, out y);
14     this.textBox_x.Text = x.ToString();
15     this.textBox_y.Text = y.ToString();
16 }
17
18 private void XYtoBLButton_Click(object sender, RoutedEventArgs e)
19 {

```

```

20 double B, L, L0, x, y;
21 double.TryParse(this.textBox_x.Text, out x);
22 double.TryParse(this.textBox_y.Text, out y);
23 double.TryParse(this.textBox_L0.Text, out L0);
24
25 Spheroid proj = Spheroid.CreateBeiJing1954();
26 proj.XYtoBL(x, y,
27     ZXY.SMath.DMS2RAD(L0),
28     out B, out L);
29 this.textBox_B.Text = ZXY.SMath.RAD2DMS(B).ToString();
30 this.textBox_L.Text = ZXY.SMath.RAD2DMS(L).ToString();
31 }

```

在这个两个响应事件中，首先需要直接从界面上的 TextBox 控件中取值，由于 TextBox 的 Text 属性是文本（string 类型），需要用 double.TryParse 函数将其转换为 double 类型。

其次我们默认参考椭球为 1954 北京坐标系的参考椭球，需要将其创建，此处我们用类的静态函数可以方便创建，不必记忆其椭球的几何参数值。

然后我们就可以调用我们前边写的正算与反算函数进行高斯投影正反算了，算完后将值再赋值给相应的 TextBox 控件的 Text 属性就可以了。注意传入函数的值如果是度分秒形式的角度，应该先调用我们前边的写的函数将其转换为弧度，如果算出的值也是角度，也应该调用我们前面所写的函数将其转换为度分秒形式。

4.3.2 界面程序的优化

上面的简单界面程序存在着一个很大的问题，就是从界面取数据时无法判断数据的有效性等，也无法发挥 WPF 界面技术。WPF 界面技术里的数据绑定功能（binding）可以很好的简化这一过程。

我们仔细分析前边的界面，这个程序的实质就是一个点的两种形式的坐标之间的转换，因此我们可以定义一个点类 GeoPoint，其定义如下：

```

1 public class GeoPoint
2 {
3     public string Name { get; set; } //点名
4     public double B { get; set; } //纬度，单位：度分秒
5     public double L { get; set; } //经度，单位：度分秒
6     public double L0 { get; set; } //中央子午线经度，单位：度分秒
7     public double X { get; set; } //X坐标
8     public double Y { get; set; } //Y坐标
9 }

```

则在界面代码中可对 TextBox 的 Text 做如下绑定：

```

1 <TextBlock Text="大地坐标" HorizontalAlignment="Center"
2     VerticalAlignment="Center"/>
3 <TextBlock Text="B=" Grid.Row="1" Margin="5"/>
4 <TextBox x:Name="textBox_B" Grid.Row="1"
5     Text="{Binding B, StringFormat={}{0:#0.###}} "
6     VerticalAlignment="Center" Margin="25,0,0,0"/>
7 <TextBlock Text="L=" Margin="5" Grid.Row="2" />
8 <TextBox x:Name="textBox_L"
9     Text="{Binding L, StringFormat={}{0:#0.###}} "
10    VerticalAlignment="Center" Margin="25,0,0,0" Grid.Row="2" />
11
12 <TextBlock Text="L0=" Grid.Column="1"
13    Margin="5,0,0,0" VerticalAlignment="Center"/>

```

```

14 <TextBox x:Name="textBox_L0"
15     Text="{Binding L0, StringFormat={}{0:##0.###}}"
16     Grid.Column="1"
17     Margin="30,0,3,0" VerticalAlignment="Center"/>
18 <Button Content="(B,L)→(x,y)" Grid.Column="1" Grid.Row="1"
19     Click="BLtoXYButton_Click" Margin="3,3"/>
20 <Button Content="(B,L)&lt;->(x,y)" Grid.Column="1" Grid.Row="2"
21     Click="XYtoBLButton_Click" Margin="3,3"/>
22
23 <TextBlock Text="高斯投影坐标" Grid.Column="2"
24     HorizontalAlignment="Center" VerticalAlignment="Center"/>
25 <TextBlock Text="x=" Margin="5" Grid.Row="1" Grid.Column="2"/>
26 <TextBox x:Name="textBox_x" Grid.Row="1" Grid.Column="2"
27     Text="{Binding X, StringFormat={}{0:##0.###} }"
28     Margin="25,0,0,0" VerticalAlignment="Center" />
29 <TextBlock Text="y=" Grid.Row="2" Grid.Column="2" Margin="5,0,0,0"/>
30 <TextBox x:Name="textBox_y"
31     Text="{Binding Y, StringFormat={}{0:##0.###} }"
32     Grid.Row="2" Grid.Column="2"
33     Margin="25,0,0,0" VerticalAlignment="Center" />

```

以上代码中的 TextBox 控件中的 x:Name 属性甚至都可以省略。由于这些控件都是以这个窗体 (Window) 作为容器的，他们的数据源都可用这个窗体的 DataContext 一次性设置，让系统以冒泡的形式自动为属性绑定寻找数据源。窗体为之设定数据源的代码如下：

```

1 public partial class MainWindow : Window
2 {
3     private GeoPoint geoPoint;
4     private Spheroid proj = Spheroid.CreateBeiJing1954();
5
6     public MainWindow()
7     {
8         InitializeComponent();
9         geoPoint = new GeoPoint() { B= 21.58470845, L= 113.25314880 };
10        this.DataContext = geoPoint;
11    }
12    // ..... 省略了其他代码 .....
13 }

```

程序中由于正反算都是基于相同的椭球基准，所以在类 MainWindow 中定义了 geoPoint 实例字段与 proj 实例字段。在构造函数中为其赋了初值以简化每次在界面输入数据，为这个窗体的 DataContext 指定点的各项属性绑定的数据源。

相应的正反算按钮的响应事件修改为：

```

1 private void BLtoXYButton_Click(object sender, RoutedEventArgs e)
2 {
3     double x, y;
4     proj.BLtoXY(
5         ZXY.SMath.DMS2RAD(geoPoint.B),
6         ZXY.SMath.DMS2RAD(geoPoint.L),
7         ZXY.SMath.DMS2RAD(geoPoint.L0),
8         out x, out y);
9     geoPoint.X = x; geoPoint.Y = y;
10 }
11
12 private void XYtoBLButton_Click(object sender, RoutedEventArgs e)
13 {

```

```

14     double B, L;
15     proj.XYtoBL(geoPoint.X, geoPoint.Y,
16         ZXY.SMath.DMS2RAD(geoPoint.L0),
17         out B, out L);
18     geoPoint.B = ZXY.SMath.RAD2DMS(B);
19     geoPoint.L = ZXY.SMath.RAD2DMS(L);
20 }

```

从响应事件可以看出，代码简洁了很多。运行程序时，TextBox 框中都有默认数值，而且非数值数据也输入不进去了，也不需要将文本框中的 Text 属性转换为 double 类型了。一切看似都好，但你发现在点击正算或反算按钮时，界面上的数据没有变化，好像功能没有实现一样，问题出现在什么地方呢？

我们回过头再看 GeoPoint 的定义，发现其属性定义过于简单。根据 WPF 知识可知，在对象属性发生改变时（如我们的计算中正算改变了 X 与 Y，反算改变了 B 与 L），还需要一种机制通知系统需要刷新界面，这就需要类 GeoPoint 从接口 INotifyPropertyChanged 继承并实现它（该接口所在的命名空间为 System.ComponentModel）。修改后的 GeoPoint 类如下：

```

1 public class GeoPoint : INotifyPropertyChanged
2 {
3     public event PropertyChangedEventHandler PropertyChanged;
4
5     public void RaisePropertyChange(string propertyName)
6     {
7         if (this.PropertyChanged != null)
8         {
9             this.PropertyChanged.Invoke(this,
10                 new PropertyChangedEventArgs(propertyName));
11         }
12     }
13
14     private string _name;
15     public string Name //点名
16     {
17         get { return _name; }
18         set
19         {
20             _name = value;
21             RaisePropertyChange("Name");
22         }
23     }
24
25     private double _B;
26     public double B //纬度，单位：度分秒
27     {
28         get { return _B; }
29         set
30         {
31             _B = value;
32             RaisePropertyChange("B");
33         }
34     }
35
36     private double _L;
37     public double L //经度，单位：度分秒
38     {
39         get { return _L; }

```

```

40         set
41         {
42             _L = value;
43             RaisePropertyChanged("L");
44         }
45     }
46
47     private double _L0;
48     public double L0 //中央子午线经度, 单位: 度分秒
49     {
50         get { return _L0; }
51         set
52         {
53             _L0 = value;
54             RaisePropertyChanged("L0");
55         }
56     }
57
58     private double _X;
59     public double X //X坐标
60     {
61         get { return _X; }
62         set
63         {
64             _X = value;
65             RaisePropertyChanged("X");
66         }
67     }
68
69     private double _Y;
70     public double Y //Y坐标
71     {
72         get { return _Y; }
73         set
74         {
75             _Y = value;
76             RaisePropertyChanged("Y");
77         }
78     }
79 }

```

与前面的 GeoPoint 类相比较, 现在的这个类从 INotifyPropertyChanged 继承并实现了接口成员 PropertyChanged, 在属性值发生改变时利用该接口成员通知界面属性值发生了变化。

运行程序, 功能一切正常。

4.3.3 点类的进一步优化

我们再次审阅界面程序后的正反算代码, 发现事实上的正反算都是基于点类的, 也就是说只与 GeoPoint 类相关, 因此我们将正反算功能移到 GeoPoint 类中, 以进一步简化界面的方法调用。其代码如下:

```

1 public class GeoPoint : INotifyPropertyChanged
2 {
3     // ..... 其它代码 .....
4     public void BLtoXY(Spheroid spheroid)
5     {

```



```

6      double x, y;
7      spheroid.BLtoXY(
8          ZXY.SMath.DMS2RAD(this.B),
9          ZXY.SMath.DMS2RAD(this.L),
10         ZXY.SMath.DMS2RAD(this.L0),
11         out x, out y);
12     this.X = x; this.Y = y;
13 }
14
15 public void XYtoBL(Spheroid spheroid)
16 {
17     double B, L;
18     spheroid.XYtoBL(X, Y, ZXY.SMath.DMS2RAD(this.L0),
19         out B, out L);
20     this.B = ZXY.SMath.RAD2DMS(B);
21     this.L = ZXY.SMath.RAD2DMS(L);
22 }
23 }

```

界面正反算代码可以进一步简化为如下形式：

```

1 private void BLtoXYButton_Click(object sender, RoutedEventArgs e)
2 {
3     geoPoint.BLtoXY(proj);
4 }
5
6 private void XYtoBLButton_Click(object sender, RoutedEventArgs e)
7 {
8     geoPoint.XYtoBL(proj);
9 }

```

至此，我们的算法与界面几乎是完全分离了，这符合我们第一章所讲的界面与算法相分离的原则，也为我们的算法进行单元测试和进一步优化迭代打下了基础。

4.4 更加实用的多点计算图形程序

上面的程序从编程的角度讲比较完美了，但从实用的角度来说还有很多缺点，比如不能选择椭球基准，不能进行多点的正反算，不能读取和写出文本文件数据。这一节我们将从算法和界面两个方面来构造这个更加实用的多点高斯投影计算的图形程序。

4.4.1 程序的功能

实用的高斯投影程序功能如下：

1. 椭球基准的选择：能够自由的选择参考椭球基准，或者自定义参考椭球；
2. 能实现高斯投影正反算与换带功能；
3. 高斯投影坐标的定义：能自动去除或添加点的 Y 坐标前的常数 500km 和带号；
4. 数据的界面录入：能利用程序界面组织输入数据；
5. 能导入导出文本数据：能将外部文本数据导入到程序中，能将程序中的数据导出为文本文件；
6. 能实现多个点的批量计算。

软件运行时的界面如图4.2所示，该界面基本能满足以上功能。



图 4.2 实用高斯投影程序界面

4.4.2 程序的面向对象分析与实现

从界面上我们可以看出，程序中应该包含三部分内容：椭球基准、坐标系、点集。为了满足以上功能，我们需要对我们的程序进行重构。

分析我们前边的 GeoPoint 类会发现，一个点有中央子午线经度 L0 这个属性会很奇怪，而在实际应用中点集也是基于坐标系的点集，一个点集的 y 坐标甚至 x 坐标前的加常数也是基本相同。因此坐标系应该有中央子午线经度 L0 属性，带号 N 与 Y 坐标前的加常数 YKM，由于坐标系是依赖于椭球基准的，也应有椭球基准属性 ProjSpheroid。坐标系的定义如下代码所示：

```
1 using System.Collections.ObjectModel;
2
3 /// <summary>
4 /// 坐标系
5 /// </summary>
6 public class CoordinateSystem : INotifyPropertyChanged
7 {
8     // .... 省略与接口 INotifyPropertyChanged 有关的代码 ....
9
10    /// <summary>
11    /// 中央子午线经度
12    /// </summary>
13    private double _L0;
14
```

```
15  /// <summary>
16  /// 中央子午线经度
17  /// </summary>
18  public double L0
19  {
20      get { return _L0; }
21      set
22      {
23          _L0 = value;
24          RaisePropertyChanged("L0");
25      }
26  }
27
28  /// <summary>
29  /// 带号
30  /// </summary>
31  private int _N;
32
33  /// <summary>
34  /// 带号
35  /// </summary>
36  public int N
37  {
38      get { return _N; }
39      set
40      {
41          _N = value;
42          RaisePropertyChanged("N");
43      }
44  }
45
46  /// <summary>
47  /// Y坐标的加常数
48  /// </summary>
49  private double _YKM;
50
51  /// <summary>
52  /// Y坐标的加常数
53  /// </summary>
54  public double YKM
55  {
56      get { return _YKM; }
57      set
58      {
59          _YKM = value;
60          RaisePropertyChanged("YKM");
61      }
62  }
63
64  /// <summary>
65  /// 坐标点集
66  /// </summary>
67  private ObservableCollection<GeoPoint>geoPointList =
68      new ObservableCollection<GeoPoint>();
69
70  /// <summary>
```

```

71     /// 坐标点集
72     /// </summary>
73     public ObservableCollection<GeoPoint> GeoPointList
74     {
75         get { return geoPointList; }
76     }
77
78     /// <summary>
79     /// 投影椭球基准
80     /// </summary>
81     private Spheroid spheroid = Spheroid.CreateBeiJing1954();
82
83     /// <summary>
84     /// 投影椭球基准
85     /// </summary>
86     public Spheroid ProjSpheroid
87     {
88         get { return spheroid; }
89         set
90         {
91             spheroid = value;
92             RaisePropertyChanged("ProjSpheroid");
93         }
94     }
95
96     public CoordinateSystem() { }
97 }

```

在类 GeoPoint 中将属性 L0 的定义删除。在 CoordinateSystem 类中, 由于 N, L0, YKM 需与界面交互, 故须从接口 INotifyPropertyChanged 继承。

为了与 WPF 界面中的 DataGrid 控件交互, 点集需要用 ObservableCollection<GeoPoint> 表达, 不能用 List<GeoPoint>, 而且默认状态下就应该生成其实例对象。注意二者的命名空间也不一样, 前者为 System.Collections.ObjectModel, 用于界面交互较多, 后者为 System.Collections.Generic, 用于不需要界面的算法较多。

为了与界面的初始状态一致, 对投影的参考椭球我们默认生成北京 54 坐标系的参考椭球。

在我们的程序中 GeoPoint 类与 CoordinateSystem 类为了处理与界面交互的问题, 都需要实现接口 INotifyPropertyChanged, 实现接口的代码重复。本着相同或相似的代码在程序中只写一次的原则, 我们将这部分的代码独立到类 NotificationObject 中, 让 GeoPoint 类与 CoordinateSystem 类从 NotificationObject 类继承。相应的实现代码如下:

```

1 using System.ComponentModel;
2
3 namespace GaussProj
4 {
5     public class NotificationObject : INotifyPropertyChanged
6     {
7         public event PropertyChangedEventHandler PropertyChanged;
8
9         public void RaisePropertyChanged(string propertyName)
10        {
11            if (this.PropertyChanged != null)
12            {
13                this.PropertyChanged.Invoke(this, new PropertyChangedEventArgs(
14                    propertyName));
15            }
16        }
17    }
18 }

```

```

15     }
16 }
17 }
18
19 public class GeoPoint : NotificationObject
20 {
21     //删除与NotificationObject类中相同的代码
22     //省略GeoPoint类中的内容
23 }
24
25 public class CoordinateSystem : NotificationObject
26 {
27     //删除与NotificationObject类中相同的代码
28     //省略CoordinateSystem类中的内容
29 }

```

4.4.3 多点的高斯投影计算

如此，在类 `CoordinateSystem` 中就有了用于高斯投影的椭球基准，有了坐标系的中央子午线经度，有了带号及 y 坐标前的加常数与点集，多点的高斯投影计算就万事俱备，只欠实现了。其实现代码如下：

```

1 public class CoordinateSystem : NotificationObject
2 {
3     //省略其他代码
4
5     /// <summary>
6     /// 多点高斯投影正算
7     /// </summary>
8     public void BLtoXY()
9     {
10         foreach (var pnt in this.geoPointList)
11         {
12             pnt.BLtoXY(spheroid, this);
13         }
14     }
15
16     /// <summary>
17     /// 多点高斯投影反算
18     /// </summary>
19     public void XYtoBL()
20     {
21         foreach (var pnt in this.geoPointList)
22         {
23             pnt.XYtoBL(spheroid, this);
24         }
25     }
26 }

```

从代码中可以看出，在类 `CoordinateSystem` 中并没有真正的进行高斯投影正算与反算，而是通过循环将其委托给每个点的实例了。

点类 `GeoPoint` 中的高斯投影正反算实现如下：

```

1 public class GeoPoint : NotificationObject
2 {
3     //省略其他代码

```

```

4  /// <summary>
5  /// 高斯投影正算
6  /// </summary>
7  /// <param name="spheroid">投影椭球</param>
8  /// <param name="cs">坐标系</param>
9  public void BLtoXY(Spheroid spheroid, CoordinateSystem cs)
10 {
11     double x, y;
12     spheroid.BLtoXY(
13         ZXY.SMath.DMS2RAD(this.B),
14         ZXY.SMath.DMS2RAD(this.L),
15         ZXY.SMath.DMS2RAD(cs.L0),
16         out x, out y);
17     this.X = x; this.Y = y + cs.YKM * 1000 + cs.N * 1000000;
18 }
19 /// <summary>
20 /// 高斯投影反算
21 /// </summary>
22 /// <param name="spheroid">投影椭球</param>
23 /// <param name="cs">坐标系</param>
24 public void XYtoBL(Spheroid spheroid, CoordinateSystem cs)
25 {
26     double tB, tL;
27     double y = Y - cs.YKM * 1000 - cs.N * 1000000;
28     spheroid.XYtoBL(X, y, ZXY.SMath.DMS2RAD(cs.L0),
29         out tB, out tL);
30     this.B = ZXY.SMath.RAD2DMS(tB);
31     this.L = ZXY.SMath.RAD2DMS(tL);
32 }
33 }

```

从如上的代码中可以看出，真正的高斯投影正反算还是在我们前面写的 Spheroid 类中。请注意在 Spheroid 类中，所有的与角度有关的单位是弧度，y 坐标也是点的真实坐标，在此我们需要根据坐标系中的信息对其做相应的预处理。

还应注意，在 BLtoXY 中，为了与界面交互，要赋值给 this.X 与 this.Y，而不是赋值给其变量 this._X 与 this._Y。在 XYtoBL 中也同样如此，当然还需要将计算出的弧度值转换为度分秒形式。

4.4.4 点坐标数据的读入与写出

利用我们的界面可以手工输入点的坐标数据，但导入与导出文本数据对于一个程序来讲是必不可少的功能。

为了避免我们的教学程序过于复杂，我们对数据文件的格式进行适当简化。

在高斯投影正算时，所需的数据应该是：点名，纬度 B，经度 L，设计我们的文本文件内容如下：

```

#点名,   B,   L
GP01,34.2154,109.112
GP02,34.215,109.1119
GP03,34.2145,109.1119
GP04,34.2147,109.1114
GP05,34.2151,109.1114

```

文件中的每一行第一个字符以 # 开头的我们视为注释行，予以忽略。

在高斯投影反算时，所需的数据应该是：点名，X，Y，设计我们的文本文件内容如下：

```
#点名,      X,      Y,      H
GP01, 3805709.2106, 19333388.3123, 466.419
GP02, 3805595.1034, 19333360.1973, 470.94
GP03, 3805440.0738, 19333360.1727, 478.728
GP04, 3805481.9494, 19333237.0999, 475.975
GP05, 3805598.4201, 19333235.7343, 469.738
```

文件中的数据项可以多余三项，我们只读取第 1、2、3 项，其余忽略。

在写出数据时，我们将点的五项数据：点名，B，L，X，Y 全部写出，如下所示：

```
# 点名,      B,      L,      X,      Y
GP01, 34.2154, 109.112 , 3804863.095, 36609369.106
GP02, 34.215,  109.1119, 3804748.229, 36609344.381
GP03, 34.2145, 109.1119, 3804593.296, 36609348.937
GP04, 34.2147, 109.1114, 3804631.509, 36609224.705
GP05, 34.2151, 109.1114, 3804747.866, 36609219.899
```

用户在使用时可以将数据拷贝到 Word 中按分隔符 “,” 生成表格进行编辑处理，或按分隔符 “;” 导入到 Excel 中进行编辑排版处理。

读入的点坐标数据应存储在我们的程序中，很显然应该在类 CoordinateSystem 中实现读入文本文件数据功能，写出数据的功能也如此，其实现代码如下：

```
1 public class CoordinateSystem : NotificationObject
2 {
3     //省略其他代码
4
5     /// <summary>
6     /// 读入点集坐标数据
7     /// </summary>
8     /// <param name="fileName">文件名</param>
9     /// <param name="format">点的坐标格式：BL-Name, B, L
10    ///                               XY-Name, X, Y
11    /// </param>
12    public void ReadGeoPointData(string fileName, string format)
13    {
14        using (System.IO.StreamReader sr = new System.IO.StreamReader(fileName) )
15        {
16            string buffer;
17            //读入点的坐标数据
18            this.GeoPointList.Clear();
19            while (true)
20            {
21                buffer = sr.ReadLine();
22                if (string.IsNullOrEmpty(buffer)) break; //文件末尾或空行退出
23                if (buffer[0] == '#') continue;
24                string[] its = buffer.Split(new char[1] { ',', ' ' });
25                if (its.Length < 3) continue; //少于三项数据，不是点的坐标数据，
                忽略
26                GeoPoint pnt = new GeoPoint();
27                pnt.Name = its[0].Trim();
```

```

28         if (format == "XY")
29         {
30             pnt.X = double.Parse(its[1]);
31             pnt.Y = double.Parse(its[2]);
32             pnt.B = 0; pnt.L = 0;
33         }
34         else if (format == "BL")
35         {
36             pnt.B = double.Parse(its[1]);
37             pnt.L = double.Parse(its[2]);
38             pnt.X = 0; pnt.Y = 0;
39         }
40
41         this.GeoPointList.Add(pnt);
42     }
43 }
44 }
45
46 /// <summary>
47 /// 写出点集坐标数据
48 /// </summary>
49 /// <param name="fileName">文件名</param>
50 public void WriteGeoPointData(string fileName)
51 {
52     using (System.IO.StreamWriter sr = new System.IO.StreamWriter(fileName))
53     {
54         sr.WriteLine("#点名, B, L, X, Y");
55         foreach (var pnt in this.geoPointList)
56         {
57             sr.WriteLine(pnt);
58         }
59     }
60 }
61 }

```

写数据比较简单，按要求写出即可，需要注意的是第 57 行，在函数 WriteLine 中我们直接写出了 GeoPoint 的实例对象 pnt，这需要在 GeoPoint 中将 ToString() 进行 override，实现代码如下：

```

1 public class GeoPoint : NotificationObject
2 {
3     public override string ToString()
4     {
5         return string.Format("{0},{1:0.0000},{2:0.0000},{3:0.000},{4:0.000}",
6             Name, B, L, X, Y);
7     }
8 }

```

在占位符中我们加入了输出浮点数据的格式控制，保证输出的角度小数位数不超过四位，输出的坐标不超过三位。

读入数据相对于写出数据要由于需要解码数据，所以要复杂一些。小于三个数据项的行我们直接略过，同时我们能加入格式控制，如果格式控制 BL，意味着数据文件是经纬度数据，其他属性相应置零。

读入数据的这段代码我们实现的方式较为粗略，只是通过逗号分隔的数据项个数进行了判断，这在真正的程序开发中是不可靠的，可以通过正则表达式对每行数据进行检验，对符合要求的文本数据加以处理，以此来提高读取文本数据功能的容错能力。

清除 (X, Y) 与清除 (B, L) 功能实质上是点的这些属性置零, 比较简单, 相应代码如下:

```

1 public class CoordinateSystem : NotificationObject
2 {
3     //省略其他代码
4
5     public void ClearXY()
6     {
7         foreach (var pnt in this.geoPointList)
8         {
9             pnt.X = pnt.Y = 0;
10        }
11    }
12    public void ClearBL()
13    {
14        foreach (var pnt in this.geoPointList)
15        {
16            pnt.B = pnt.L = 0;
17        }
18    }
19 }

```

4.4.5 界面设计与实现

按图4.2设计的界面代码如下:

```

1 <Window x:Class="GaussProj.GaussProjWin"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6     xmlns:local="clr-namespace:GaussProj"
7     mc:Ignorable="d"
8     Title="高斯投影换带" Height="360" Width="540">
9     <Grid>
10         <Grid.RowDefinitions>
11             <RowDefinition Height="45"/>
12             <RowDefinition Height="5"/>
13             <RowDefinition Height="45"/>
14             <RowDefinition Height="5"/>
15             <RowDefinition Height="Auto"/>
16         </Grid.RowDefinitions>
17         <GroupBox x:Name="groupBox_spheroid" Header="参考椭球定义"
18             DataContext="ProjSpheroid" Margin="1">
19             <Grid>
20                 <Grid.ColumnDefinitions>
21                     <ColumnDefinition Width="200"/>
22                     <ColumnDefinition Width="70"/>
23                     <ColumnDefinition Width="110"/>
24                     <ColumnDefinition Width="60"/>
25                     <ColumnDefinition Width="110"/>
26                 </Grid.ColumnDefinitions>
27                 <ComboBox Grid.Column="0" x:Name="comboBox_Spheroid"
28                     SelectionChanged="comboBox_Spheroid_SelectionChanged">
29                     <ComboBoxItem Content="1954北京坐标系" Tag="BJ1954"/>
30                     <ComboBoxItem Content="1980西安坐标系" Tag="XA1980"/>
31                     <ComboBoxItem Content="CGCS2000大地坐标系" Tag="CGCS2000"/>

```

```

32         <ComboBoxItem Content="自定义参考椭球" Tag="CS0000" />
33     </ComboBox>
34     <TextBlock Text="长半轴: a=" Grid.Column="1"
35         VerticalAlignment="Center" HorizontalAlignment="Right" />
36     <TextBox x:Name="textBox_a" Grid.Column="2" Text="{Binding a}" />
37     <TextBlock Text="扁率: 1/" Grid.Column="3"
38         VerticalAlignment="Center" HorizontalAlignment="Right" />
39     <TextBox x:Name="textBox_f" Grid.Column="4" Text="{Binding f}" />
40 </Grid>
41 </GroupBox>
42
43 <Border Grid.Row="2" BorderBrush="Yellow" BorderThickness="2">
44     <GroupBox x:Name="groupBox_CoordinateSystem" Header="坐标系定义">
45         <StackPanel Orientation="Horizontal">
46             <TextBlock Text="中央子午线经度L0=" />
47             <TextBox x:Name="textBox_L0" Text="{Binding L0}" Width="50" />
48             <TextBlock Text="Y坐标加常数: " Margin="5,0,0,0" />
49             <TextBox x:Name="textBox_YKM" Text="{Binding YKM}" Width="50" />
50             <TextBlock Text="km" />
51             <TextBlock Text="带号: " Margin="5,0,0,0" />
52             <TextBox x:Name="textBox_N" Text="{Binding N}" Width="50" />
53         </StackPanel>
54     </GroupBox>
55 </Border>
56
57 <Border Grid.Row="4" BorderBrush="Yellow" BorderThickness="2">
58     <Grid>
59         <Grid.ColumnDefinitions>
60             <ColumnDefinition Width="200*" />
61             <ColumnDefinition Width="90" />
62         </Grid.ColumnDefinitions>
63         <DataGrid x:Name="dataGrid_ctrPnt"
64             AutoGenerateColumns="False" Margin="2"
65             ItemsSource="{Binding GeoPointList}" >
66             <DataGrid.Columns>
67                 <DataGridTextColumn Header="点名"
68                     Binding="{Binding Name}"
69                     MinWidth="40" />
70                 <DataGridTextColumn Header="B"
71                     Binding="{Binding B, StringFormat={}{0:##0.####}}"
72                     MinWidth="60" />
73                 <DataGridTextColumn Header="L"
74                     Binding="{Binding L, StringFormat={}{0:##0.####}}"
75                     MinWidth="60" />
76                 <DataGridTextColumn Header="X"
77                     Binding="{Binding X, StringFormat={}{0:##0.####}}"
78                     MinWidth="60" />
79                 <DataGridTextColumn Header="Y"
80                     Binding="{Binding Y, StringFormat={}{0:##0.####}}"
81                     MinWidth="60" />
82             </DataGrid.Columns>
83         </DataGrid>
84
85         <StackPanel Grid.Column="1" Orientation="Vertical">
86             <RadioButton x:Name="radioButton_BLtoXY" Content="正算"
87                 Height="25" Width="80" Margin="5" />

```

```

88         <RadioButton x:Name="radioButton_XYtoBL" Content="反算"
89             IsChecked="True" Height="25" Width="80" Margin="5" />
90
91         <Button x:Name="Button_ReadGaussProjData" Content="读入数据"
92             Height="25" Width="80" Margin="5"
93             Click="Button_ReadGaussProjData_Click" />
94         <Button x:Name="Button_WriteGaussProjData" Content="写出数据"
95             Height="25" Width="80" Margin="5"
96             Click="Button_WriteGaussProjData_Click" />
97
98         <Button x:Name="Button_ClearXY" Content="清除(X,Y)"
99             Height="25" Width="80" Margin="5"
100             Click="Button_ClearXY_Click" />
101         <Button x:Name="Button_ClearBL" Content="清除(B, L)"
102             Height="25" Width="80" Margin="5"
103             Click="Button_ClearBL_Click" />
104
105         <Button x:Name="Button_CalGaussProj" Content="计算"
106             Height="25" Width="80" Margin="5"
107             Click="Button_CalGaussProj_Click" />
108         <Button x:Name="Button_Close" Content="关闭"
109             Height="25" Width="80" Margin="5"
110             Click="Button_Close_Click" />
111     </StackPanel>
112 </Grid>
113 </Border>
114 </Grid>
115 </Window>

```

在这段 xaml 代码中，除了功能按钮区外，实际上分为了三部分：

第一部分为参考椭球定义，如第 18 行代码所示，我们将其属性布局到 groupBox_spheroid 中，数据绑定 a 与 f，数据源设置在 groupBox_spheroid 中，采用冒泡搜寻类 CoordinateSystem 中的 ProjSpheroid 属性。

ComboBox 中的参考椭球类型采用了硬编码方式直接写在其中了。通过 SelectionChange 事件对不同选择进行响应。

第二部分为坐标系定义，同样采用数据绑定的形式与界面交互数据，数据源来自类 CoordinateSystem。

第三部分为点集部分，数据源为类 CoordinateSystem 中的 GeoPointList 属性，如第 65 行代码所示。

界面后台代码如下：

```

1 public partial class GaussProjWin : Window
2 {
3     private CoordinateSystem myCoordinateSystem;
4
5     public GaussProjWin()
6     {
7         InitializeComponent();
8
9         myCoordinateSystem = new CoordinateSystem() {
10             L0 = 111, N = 19, YKM = 500};
11         this.DataContext = myCoordinateSystem;
12
13         this.comboBox_Spheroid.SelectedIndex = 0;
14     }

```

```

15
16 //省略其他代码
17 }

```

在窗体后台代码中,我们设置了窗体类的类实例变量 myCoordinateSystem 完成各项功能。在其窗体类的构造函数中实例化并对其属性赋了初始值,同时将窗体的 DataContext 属性设为 myCoordinateSystem,完成数据绑定的数据源设置。同时为了维持与界面属性的一致性,将 comboBox_Spheroid 的 SelectedIndex 设为 0。

界面的其它功能按钮的实现比较简单,实现代码如下:

```

1 public partial class GaussProjWin : Window
2 {
3     //省略其他代码
4
5     private void comboBox_Spheroid_SelectionChanged(object sender,
6         SelectionChangedEventArgs e)
7     {
8         if (comboBox_Spheroid.SelectedIndex == 0)
9             myCoordinateSystem.ProjSpheroid = Spheroid.CreateBeiJing1954();
10        else if (comboBox_Spheroid.SelectedIndex == 1)
11            myCoordinateSystem.ProjSpheroid = Spheroid.CreateXian1980();
12        else if (comboBox_Spheroid.SelectedIndex == 2)
13            myCoordinateSystem.ProjSpheroid = Spheroid.CreateCGCS2000();
14        else if (comboBox_Spheroid.SelectedIndex == 3)
15            myCoordinateSystem.ProjSpheroid =
16                Spheroid.CreateCoordinateSystem(6378137, 298.257222101);
17        this.groupBox_spheroid.DataContext = myCoordinateSystem.ProjSpheroid;
18    }
19
20    private void Button_ReadGaussProjData_Click(object sender,
21        RoutedEventArgs e)
22    {
23        OpenFileDialog dlg = new OpenFileDialog();
24        dlg.DefaultExt = ".txt";
25        dlg.Filter = "高斯投影坐标数据|*.txt|All File (*.*)|*.*";
26        if (dlg.ShowDialog() != true) return;
27
28        if (this.radioButton_BLtoXY.IsChecked == true)
29            myCoordinateSystem.ReadGeoPointData(dlg.FileName, "BL");
30        else if (this.radioButton_XYtoBL.IsChecked == true)
31            myCoordinateSystem.ReadGeoPointData(dlg.FileName, "XY");
32    }
33
34    private void Button_WriteGaussProjData_Click(object sender,
35        RoutedEventArgs e)
36    {
37        SaveFileDialog dlg = new SaveFileDialog();
38        dlg.DefaultExt = ".txt";
39        dlg.Filter = "高斯投影坐标数据|*.txt|All File (*.*)|*.*";
40        if (dlg.ShowDialog() != true) return;
41
42        myCoordinateSystem.WriteGeoPointData(dlg.FileName);
43    }
44
45    private void Button_ClearXY_Click(object sender, RoutedEventArgs e)
46    {

```

```
47     myCoordinateSystem.ClearXY();
48 }
49
50 private void Button_ClearBL_Click(object sender, RoutedEventArgs e)
51 {
52     myCoordinateSystem.ClearBL();
53 }
54
55 private void Button_CalGaussProj_Click(object sender, RoutedEventArgs e)
56 {
57     if (radioButton_BLtoXY.IsChecked == true)
58         myCoordinateSystem.BLtoXY();
59     else if (radioButton_XYtoBL.IsChecked == true)
60         myCoordinateSystem.XYtoBL();
61 }
62
63 private void Button_Close_Click(object sender, RoutedEventArgs e)
64 {
65     this.Close();
66 }
67 }
```

4.4.6 换带功能的实现

在我们以上的设计中好像没有换带计算，虽然没有直接实现，但确实是实现了。在前边的论述中，我们讲过，换带计算的实质是变换坐标系的中央子午线位置，过程为：先进行坐标反算，然后变换中央子午线经度，再反算新的坐标即可。

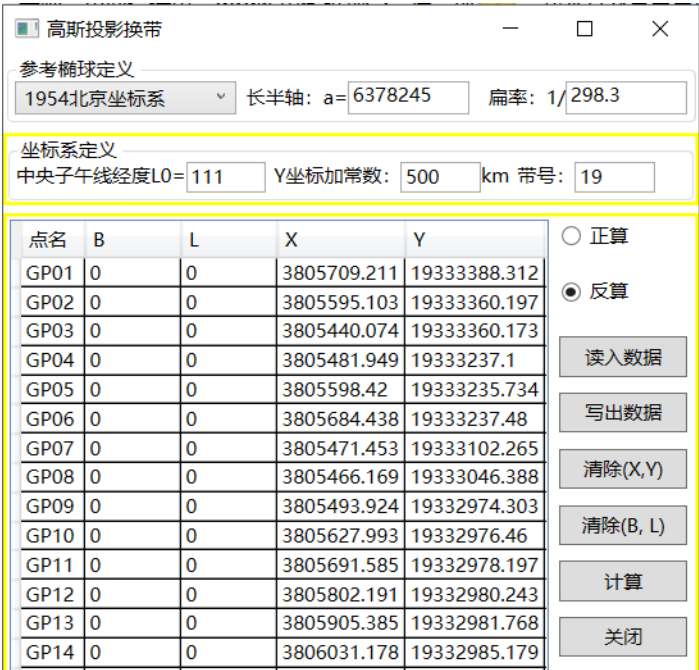


图 4.3 6° 带第 19 带数据界面

启动程序，在反算的模式下读入数据，如图4.3所示，这是一 6 度带第 19 带的北京 54 坐标：点击计算，反算各点的大地坐标经纬度，如图4.4所示。



图 4.4 6° 带第 19 带反算数据界面

现在我们准备将其换带到 3 度带的 38 带坐标系去，设置中央子午线经度 L0 为 108，带号设为 36，选择正算，点击清除 (x,y) 按钮将 X，Y 坐标设置为 0，如图4.5所示：
点击计算按钮，计算出新坐标系下各点的坐标，如图4.6所示：
到此就完成了换带功能，可以点击写出数据按钮将计算成果输出。

4.4.7 注意事项与功能扩展

以上过程也可以看成是我们这个软件的教程。但应注意，换带计算只限于相同椭球基准的情况才能应用，不能在不同椭球基准间应该该功能。
以上换带功能也可以设计一个界面，在新的界面上将新旧坐标系的中央子午线经度都设置出，从而完成一键式换带计算功能。
我们经常在一些资料中见到椭球膨胀法建立独立坐标系或施工坐标系的方法，椭球膨胀法是依据某种给定的参考椭球，根据测区的平均高程和平距纬度将椭球面扩大而维持扁率不变，也可以应用换带的方法将椭球膨胀后的坐标与原坐标系的坐标进行相互转换。但应注意椭球膨胀后，各个点的纬度值也会发生变化，在这种换带过程中需要修正各个点的纬度值，大家可以参看这方面的资料。

随着我国现在对 CGCS2000 大地坐标系的推广应用，大家还可以将大地坐标 (B, L) 与空间直角坐标 (X, Y, Z) 进行相互转换，这样就可以实现空间直角坐标 (X, Y, Z) 到高斯平面坐标之间的计算了。进而在桌面端电脑或移动设备上编程实现更多的功能应用。
大地坐标 (B, L, H) 与空间直角坐标 (X, Y, Z) 的相互转换公式如下：
已知大地坐标计算空间直角坐标的公式：

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

已知空间直角坐标计算大地坐标的公式如下：

高斯投影换带

参考椭球定义

1954北京坐标系

长半轴: a=6378245

扁率: 1/298.3

坐标系定义

中央子午线经度L0=108

Y坐标加常数: 500 km

带号: 36

点名	B	L	X	Y
GP01	34.2154	109.112	0	0
GP02	34.215	109.1119	0	0
GP03	34.2145	109.1119	0	0
GP04	34.2147	109.1114	0	0
GP05	34.2151	109.1114	0	0
GP06	34.2153	109.1114	0	0
GP07	34.2146	109.1109	0	0
GP08	34.2146	109.1107	0	0
GP09	34.2147	109.1104	0	0
GP10	34.2151	109.1104	0	0
GP11	34.2153	109.1104	0	0
GP12	34.2157	109.1104	0	0
GP13	34.22	109.1104	0	0
GP14	34.2204	109.1104	0	0

☒ 正算

☐ 反算

读入数据

写出数据

清除(X,Y)

清除(B, L)

计算

关闭

图 4.5 设置 3° 带第 36 带数据界面

高斯投影换带

参考椭球定义

1954北京坐标系

长半轴: a=6378245

扁率: 1/298.3

坐标系定义

中央子午线经度L0=108

Y坐标加常数: 500 km

带号: 36

点名	B	L	X	Y
GP01	34.2154	109.112	3804863.095	36609369.106
GP02	34.215	109.1119	3804748.229	36609344.381
GP03	34.2145	109.1119	3804593.296	36609348.937
GP04	34.2147	109.1114	3804631.509	36609224.705
GP05	34.2151	109.1114	3804747.866	36609219.899
GP06	34.2153	109.1114	3804833.881	36609219.101
GP07	34.2146	109.1109	3804617.034	36609090.265
GP08	34.2146	109.1107	3804610.103	36609034.58
GP09	34.2147	109.1104	3804635.711	36608961.72
GP10	34.2151	109.1104	3804769.759	36608959.915
GP11	34.2153	109.1104	3804833.362	36608959.772
GP12	34.2157	109.1104	3804943.958	36608958.548
GP13	34.22	109.1104	3805047.131	36608957.022
GP14	34.2204	109.1104	3805172.945	36608956.714

☒ 正算

☐ 反算

读入数据

写出数据

清除(X,Y)

清除(B, L)

计算

关闭

图 4.6 3° 带第 36 带正算数据界面

$$\left\{ \begin{array}{l} L = \arctan \frac{Y}{X} \\ \tan B = \frac{Z + Ne^2 \sin B}{\sqrt{X^2 + Y^2}} \\ H = \frac{Z}{\sin B} - N(1 - e^2) \end{array} \right.$$

本处只是列出了计算公式，大家可以查找资料寻找更加高效的计算方法。

小结

我们从一个较为简单的单点高斯投影正反算程序开始，运用 C# 知识与 WPF 界面技术实现了一个较为实用的高斯投影正反算与换带程序。

在程序过程中，我们遵循了界面与算法相分离的原则，运用了 WPF 的事件绑定等技术。

这个程序从功能上讲还是有许多不足的，从易用性上也还有许多值得改进的地方。希望随着我们的 C# 知识与 WPF 技术的积累，能在以后将它优化得更好！

第五章 平面坐标系统之间的转换

在某些工程中，由于不知道新旧两种坐标系的建立方法或参数，因此无法用换带计算的方法进行坐标转换。如果知道某些点在两个坐标系中的坐标值，我们就可以采用一些近似的转换方法将其它的点也转换到新坐标系中，求出其坐标值。尤其对于较低等级的大量控制点来说，采用这些近似方法，能够快速得到转换结果。

5.1 原理和数学模型

5.1.1 原理

这些方法的实质是根据新旧网的重合点（又称为公共点）的坐标值之差，按一定的规律修正旧网的各点坐标值，使旧网最恰当的配合到新网内。修正时因不合观测值改正数平方和为最小的原则，故为近似方法。

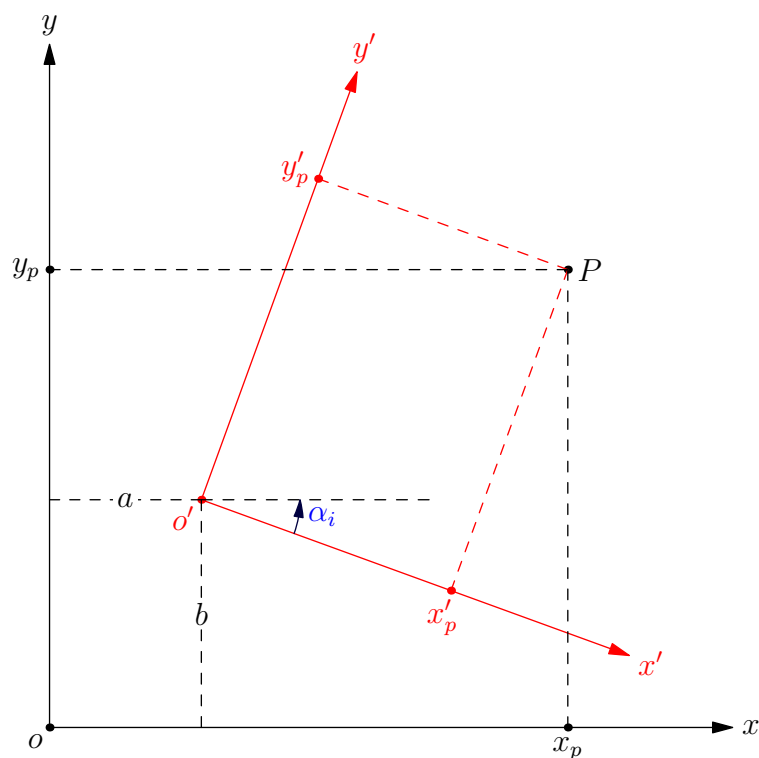


图 5.1 坐标相似变换示意图

常用的方法有简单变换方法（又称赫尔默特法或相似变换法）、仿射变换法、正形变换法等。在这里我们主要讲解简单变换法。

5.1.2 相似变换法的数学模型

实质是使旧网坐标系平移、旋转和进行尺度因子改正, 将旧网配合到新网上。因旧网形状保持不变, 故称为平面相似变换法。

变换方程为:

$$\left. \begin{aligned} x &= a + k(x' \cos \alpha + y' \sin \alpha) \\ y &= b + k(-x' \sin \alpha + y' \cos \alpha) \end{aligned} \right\}$$

式中 a, b 表示平移, α 是旧网 x' 轴逆转至新网 x 轴的转角, k 为尺度因子。这些变换参数是未知的, 要根据新旧网公共点上的已知坐标 x, y 和 x', y' 求解确定。

因此必须至少有两个公共点, 列出四个方程式, 解算出这四个未知参数值。如果具有两个以上的公共点时, 就应该应用最小二乘平差方法, 求解最或是参数值。

为解算出这些参数, 我们引入参数 c, d :

$$c = k \cos \alpha, \quad d = k \sin \alpha$$

将公式转换为:

$$\left. \begin{aligned} x &= a + x'c + y'd \\ y &= b + y'c - x'd \end{aligned} \right\}$$

由于新旧网都存在测量误差, 设新旧坐标 x, y 和 x', y' 的误差分别为 v_x, v_y 和 $v_{x'}, v_{y'}$, 因此上式改写为:

$$\left. \begin{aligned} x + v_x &= a + (x' + v_{x'})c + (y' + v_{y'})d \\ y + v_y &= b + (y' + v_{y'})c - (x' + v_{x'})d \end{aligned} \right\}$$

设:

$$\left. \begin{aligned} n_x &= -v_x + cv_{x'} + dv_{y'} \\ n_y &= -v_y - dv_{x'} + cv_{y'} \end{aligned} \right\}$$

则有:

$$\left. \begin{aligned} -n_x &= a + x'c + y'd - x \\ -n_y &= b + y'c - x'd - y \end{aligned} \right\}$$

若有 r 个新旧网的公共点, 则可组成 r 对方程:

$$V = BX - l$$

上式即为参数平差时的方程, l 代表观测向量, V 代表改正数向量, B 代表系数矩阵, X 是参数向量。它们的值为:

$$\mathbf{V} = \begin{pmatrix} -n_{x1} \\ -n_{y1} \\ \vdots \\ -n_{xr} \\ -n_{yr} \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 1 & 0 & x'_1 & y'_1 \\ 0 & 1 & y'_1 & -x'_1 \\ \dots & \dots & \dots & \dots \\ 1 & 0 & x'_r & y'_r \\ 0 & 1 & y'_r & -x'_r \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \quad \mathbf{l} = \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ x_r \\ y_r \end{pmatrix}$$

根据最小二乘原理 $V^T V = \min$ 可得到法方程:

$$B^T B X - B^T l = 0$$

解法方程可求得 a, b, c, d 的值:

$$X = (B^T B)^{-1} (B^T l)$$

旋转角 α 和尺度比 k 为:

$$\alpha = \arctan \frac{d}{c}$$

$$k = \sqrt{c^2 + d^2}$$

之后，就可计算旧网中所有待转换点的新坐标。

请注意，以上图及公式推导是按数学坐标系进行的，在用测量坐标代入计算时应将测量坐标 (x, y) 以 (y, x) 形式代入，否则应对以上图形与公式进行变换。

5.2 程序设计与实现

程序整体功能比较单一，从数学模型分析可看出，程序的关键是在于如何组成系数矩阵 B 与常数项 l 。

5.2.1 程序功能分析

在程序中，我们需要能够有录入数据界面手工输入数据，能够导入与导出文本文件形式组织的数据，能够输出计算成果，能够计算转换系数或根据已有的转换系数也能计算待计算点在新坐标系中的坐标。

由此，我们设计出程序界面如图5.2所示：

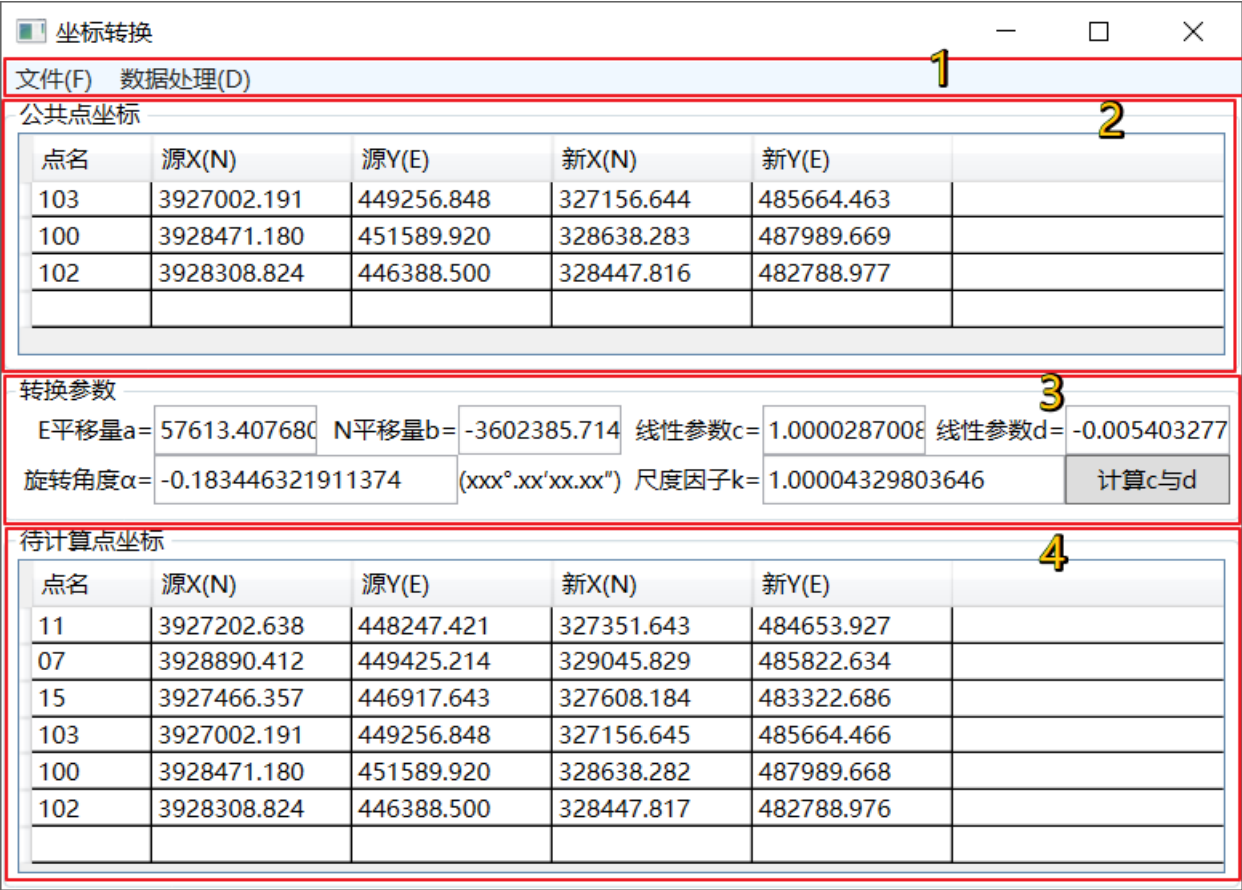


图 5.2 坐标转换程序界面

5.2.2 程序界面设计

由图5.2分析可知，界面分为菜单、公共点坐标、转换参数与待计算点坐标四个部分。

整个界面采用 DockPanel 布局,以加入菜单项,使用菜单项进行功能计算可以有效节省界面的布局空间。余下的部分采用 Grid 布局,将其划分为三行,即三个部分。第一行中加入 GroupBox 控件作为标题,在其中加入 DataGrid 控件;第二部分中加入 Grid 布局,将其二行八列进行布局;第三部分同第一部分一样。

由于需要根据输入的 a 、 b 、 α 、 k 计算各点在新坐标系中的坐标,为了输入旋转角度的便捷性(直接输入度分秒),因此加入了计算 c 与 d 的按钮。程序中计算各待计算点坐标时实际使用的参数是 a, b, c, d 。

整个界面布局 xaml 代码如下:

```

1 <DockPanel LastChildFill="True">
2     <Menu x:Name="mainmenu" DockPanel.Dock="Top" Background="AliceBlue">
3         <MenuItem Header="文件(F)">
4             <MenuItem Header="打开文本数据"
5                 Click="menuItem_OpenTextFileData_Click" />
6             <MenuItem Header="保存文本数据"
7                 Click="menuItem_SaveTextFileData_Click" />
8             <Separator />
9             <MenuItem Header="退出" Click="menuItem_Exit_Click" />
10        </MenuItem>
11        <MenuItem Header="数据处理(D)">
12            <MenuItem Header="计算转换参数"
13                Click="menuItem_CalCoefficient_Click" />
14            <MenuItem Header="计算待计算点坐标"
15                Click="menuItem_Cal_UnKnw_XY_Click" />
16            <Separator />
17            <MenuItem Header="写出计算成果"
18                Click="menuItem_Write_Result_Click" />
19        </MenuItem>
20    </Menu>
21
22    <Grid>
23        <Grid.RowDefinitions>
24            <RowDefinition Height="150*" />
25            <RowDefinition Height="75" />
26            <RowDefinition Height="200*" />
27        </Grid.RowDefinitions>
28        <GroupBox Header="公共点坐标" Grid.Row="0">
29            <DataGrid AutoGenerateColumns="False" Margin="2"
30                ItemsSource="{Binding KnwPointList}">
31                <DataGrid.Columns>
32                    <DataGridTextColumn Header="点名" MinWidth="60"
33                        Binding="{Binding Name}" />
34                    <DataGridTextColumn Header="源X(N)" MinWidth="100"
35                        Binding="{Binding OX, StringFormat={}{0:0.000}}" />
36                    <DataGridTextColumn Header="源Y(E)" MinWidth="100"
37                        Binding="{Binding OY, StringFormat={}{0:0.000}}" />
38                    <DataGridTextColumn Header="新X(N)" MinWidth="100"
39                        Binding="{Binding NX, StringFormat={}{0:0.000}}" />
40                    <DataGridTextColumn Header="新Y(E)" MinWidth="100"
41                        Binding="{Binding NY, StringFormat={}{0:0.000}}" />
42                </DataGrid.Columns>
43            </DataGrid>
44        </GroupBox>
45
46        <GroupBox Header="转换参数" Grid.Row="1">

```

```

47         <Grid>
48             <Grid.ColumnDefinitions>
49                 <ColumnDefinition Width="70" />
50                 <ColumnDefinition Width="120*" />
51                 <ColumnDefinition Width="70" />
52                 <ColumnDefinition Width="120*" />
53                 <ColumnDefinition Width="70" />
54                 <ColumnDefinition Width="120*" />
55                 <ColumnDefinition Width="70" />
56                 <ColumnDefinition Width="120*" />
57             </Grid.ColumnDefinitions>
58             <Grid.RowDefinitions>
59                 <RowDefinition Height="25" />
60                 <RowDefinition Height="25" />
61             </Grid.RowDefinitions>
62             <TextBlock Text="E平移量a=" Grid.Row="0" Grid.Column="0"
63                 VerticalAlignment="Center" HorizontalAlignment="Right" />
64             <TextBox x:Name="textBox_a" Grid.Row="0" Grid.Column="1"
65                 Text="{Binding a}"
66                 VerticalContentAlignment="Center" />
67             <TextBlock Text="N平移量b=" Grid.Row="0" Grid.Column="2"
68                 VerticalAlignment="Center" HorizontalAlignment="Right" />
69             <TextBox x:Name="textBox_b" Grid.Row="0" Grid.Column="3"
70                 Text="{Binding b}"
71                 VerticalContentAlignment="Center" />
72             <TextBlock Text="线性参数c=" Grid.Row="0" Grid.Column="4"
73                 VerticalAlignment="Center" HorizontalAlignment="Right" />
74             <TextBox x:Name="textBox_c" Grid.Row="0" Grid.Column="5"
75                 Text="{Binding c}"
76                 VerticalContentAlignment="Center" />
77             <TextBlock Text="线性参数d=" Grid.Row="0" Grid.Column="6"
78                 VerticalAlignment="Center" HorizontalAlignment="Right" />
79             <TextBox x:Name="textBox_d" Grid.Row="0" Grid.Column="7"
80                 Text="{Binding d}"
81                 VerticalContentAlignment="Center" />
82             <TextBlock Text="旋转角度 =" Grid.Row="1" Grid.Column="0"
83                 VerticalAlignment="Center" HorizontalAlignment="Right" />
84             <TextBox x:Name="textBox_alpha" Grid.Row="1" Grid.Column="1"
85                 Text="{Binding alpha}"
86                 Grid.ColumnSpan="2"
87                 VerticalContentAlignment="Center" />
88             <TextBlock Text="(xxxř.xx xx.xx )" Grid.Row="1" Grid.Column="3"
89                 VerticalAlignment="Center" HorizontalAlignment="Right" />
90             <TextBlock Text="尺度因子k=" Grid.Row="1" Grid.Column="4"
91                 VerticalAlignment="Center" HorizontalAlignment="Right" />
92             <TextBox x:Name="textBox_k" Grid.Row="1" Grid.Column="5"
93                 Text="{Binding k}"
94                 Grid.ColumnSpan="2"
95                 VerticalContentAlignment="Center" />
96             <Button x:Name="button_Cal_cd" Content="计算c与d"
97                 Grid.Row="1" Grid.Column="7"
98                 Click="button_Cal_cd_Click" />
99         </Grid>
100     </GroupBox>
101

```

```

102         <GroupBox Header="待计算点坐标" Grid.Row="2">
103             <DataGrid AutoGenerateColumns="False" Margin="2"
104                 ItemsSource="{Binding UnKnwPointList}">
105                 <DataGrid.Columns>
106                     <DataGridTextColumn Header="点名" MinWidth="60"
107                         Binding="{Binding Name}" />
108                     <DataGridTextColumn Header="源X(N)" MinWidth="100"
109                         Binding="{Binding OX, StringFormat={}{0:0.000}}" />
110                     <DataGridTextColumn Header="源Y(E)" MinWidth="100"
111                         Binding="{Binding OY, StringFormat={}{0:0.000}}" />
112                     <DataGridTextColumn Header="新X(N)" MinWidth="100"
113                         Binding="{Binding NX, StringFormat={}{0:0.000}}" />
114                     <DataGridTextColumn Header="新Y(E)" MinWidth="100"
115                         Binding="{Binding NY, StringFormat={}{0:0.000}}" />
116                 </DataGrid.Columns>
117             </DataGrid>
118         </GroupBox>
119     </Grid>
120 </DockPanel>

```

5.2.3 数据文件和成果文件格式

由于程序的功能较为单一，数据文件的格式也较为简单。我们设计格式如下：

```

#赫尔默特四参数转换法数据文件
#每行以“#”开头的行均被认为是注释行
#公共点在源坐标系中的坐标：点名，X(N)，Y(E)
103, 3927002.191, 449256.848
100, 3928471.180, 451589.920
102, 3928308.824, 446388.500

#公共点在目标坐标系中的坐标：点名，X(N)，Y(E)
102, 328447.816, 482788.977
103, 327156.644, 485664.463
100, 328638.283, 487989.669

#待转换点在源坐标系中的坐标：点名，X(N)，Y(E)
11, 3927202.638, 448247.421
07, 3928890.412, 449425.214
15, 3927466.357, 446917.643
103, 3927002.191, 449256.848
100, 3928471.180, 451589.920
102, 3928308.824, 446388.500

```

我们设计成果文件的格式如下：

```

#赫尔默特四参数转换法计算成果数据文件
# 公共点坐标
# 点名，源X(N)，源Y(E)，新X(N)，新Y(E)
103,3927002.191,449256.848,327156.644,485664.463

```

```
100,3928471.180,451589.920,328638.283,487989.669
102,3928308.824,446388.500,328447.816,482788.977
```

```
# 转换参数
```

```
a=57613.4076806228, b=-3602385.71435623, c=1.00002870085641, d=-0.00540327780963763
=-0.183446321911374, k=1.00004329803646
```

```
# 待计算点的坐标
```

```
# 点名, 源X(N), 源Y(E), 新X(N), 新Y(E)
```

```
11,3927202.638,448247.421,327351.643,484653.927
07,3928890.412,449425.214,329045.829,485822.634
15,3927466.357,446917.643,327608.184,483322.686
103,3927002.191,449256.848,327156.645,485664.466
100,3928471.180,451589.920,328638.282,487989.668
102,3928308.824,446388.500,328447.817,482788.976
```

5.2.4 程序流程

根据以上分析, 程序流程如下:

1. 读取公共点旧坐标
2. 读取公共点新坐标
3. 组成误差方程式
4. 解算参数向量
5. 解算待定点的坐标
6. 将计算成果写入文件

5.2.5 主要功能设计

为了实现以上功能, 我们需要设计一个类 (或结构) 用于表示点, 设计如下:

```
1 namespace CoordinateTransform
2 {
3     public class GeoPoint : NotificationObject
4     {
5         private string _name;
6         public string Name // 点名
7         {
8             get { return _name; }
9             set
10            {
11                _name = value;
12                RaisePropertyChanged("Name");
13            }
14        }
15
16        private double _oX;
17        public double OX // 源X坐标
```

```

18     {
19         get { return _oX; }
20         set
21         {
22             _oX = value;
23             RaisePropertyChanged("OX");
24         }
25     }
26
27     private double _oY;
28     public double OY // 源Y坐标
29     {
30         get { return _oY; }
31         set
32         {
33             _oY = value;
34             RaisePropertyChanged("OY");
35         }
36     }
37
38
39     private double _nX;
40     public double NX // 新X坐标
41     {
42         get { return _nX; }
43         set
44         {
45             _nX = value;
46             RaisePropertyChanged("NX");
47         }
48     }
49
50     private double _nY;
51     public double NY // 新Y坐标
52     {
53         get { return _nY; }
54         set
55         {
56             _nY = value;
57             RaisePropertyChanged("NY");
58         }
59     }
60
61
62     public override string ToString()
63     {
64         return string.Format("{0},{1:0.000},{2:0.000},{3:0.000},{4:0.000}",
65                                Name, OX, OY, NX, NY);
66     }
67 }

```

类 NotificationObject 的设计见前一章内容。

同时，我们设计另一个类 CoordinateSystem 来完成相应的其它功能，这个类相当于一个容器一样，它包括点集（已知公共点集和待计算点集）、转换参数等，具体代码如下：

```
1 using System;
```



```
2 using System.Collections.ObjectModel;
3
4 namespace CoordniateTransform
5 {
6     /// <summary>
7     /// 坐标系
8     /// </summary>
9     public class CoordinateSystem : NotificationObject
10    {
11        /// <summary>
12        /// 公共点集
13        /// </summary>
14        private ObservableCollection<GeoPoint> knwPointList =
15            new ObservableCollection<GeoPoint>();
16
17        /// <summary>
18        /// 公共点集
19        /// </summary>
20        public ObservableCollection<GeoPoint> KnwPointList
21        {
22            get { return knwPointList; }
23        }
24
25        /// <summary>
26        /// 待计算点集
27        /// </summary>
28        private ObservableCollection<GeoPoint> unKnwPointList =
29            new ObservableCollection<GeoPoint>();
30
31        /// <summary>
32        /// 待计算点集
33        /// </summary>
34        public ObservableCollection<GeoPoint> UnKnwPointList
35        {
36            get { return unKnwPointList; }
37        }
38
39        /// <summary>
40        /// X方向平移量
41        /// </summary>
42        private double _a;
43
44        /// <summary>
45        /// X方向平移量
46        /// </summary>
47        public double a
48        {
49            get { return _a; }
50            set
51            {
52                _a = value;
53                RaisePropertyChange("a");
54            }
55        }
56
57        /// <summary>
```

```
58     /// Y方向平移量
59     /// </summary>
60     private double _b;
61
62     /// <summary>
63     /// Y方向平移量
64     /// </summary>
65     public double b
66     {
67         get { return _b; }
68         set
69         {
70             _b = value;
71             RaisePropertyChanged("b");
72         }
73     }
74
75     /// <summary>
76     /// 线性方程计算系数c
77     /// </summary>
78     public double _c;
79
80     /// <summary>
81     /// 线性方程计算系数c
82     /// </summary>
83     public double c
84     {
85         get { return _c; }
86         set
87         {
88             _c = value;
89             RaisePropertyChanged("c");
90         }
91     }
92
93     /// <summary>
94     /// 线性方程计算系数d
95     /// </summary>
96     public double _d;
97
98     /// <summary>
99     /// 线性方程计算系数d
100    /// </summary>
101    public double d
102    {
103        get { return _d; }
104        set
105        {
106            _d = value;
107            RaisePropertyChanged("d");
108        }
109    }
110
111    /// <summary>
112    /// 旋转角度
113    /// </summary>
```

```

114     public double _alpha;
115
116     /// <summary>
117     /// 旋转角度
118     /// </summary>
119     public double alpha
120     {
121         get { return _alpha; }
122         set
123         {
124             _alpha = value;
125             RaisePropertyChanged("alpha");
126         }
127     }
128
129     /// <summary>
130     /// 尺度比因子k
131     /// </summary>
132     public double _k;
133
134     /// <summary>
135     /// 尺度比因子k
136     /// </summary>
137     public double k
138     {
139         get { return _k; }
140         set
141         {
142             _k = value;
143             RaisePropertyChanged("k");
144         }
145     }
146
147     public CoordinateSystem() { }
148
149     /// <summary>
150     /// 读入坐标转换数据文件
151     /// </summary>
152     /// <param name="fileName">文件名</param>
153     public void ReadTextFileData(string fileName)
154     {
155         using (System.IO.StreamReader sr = new System.IO.StreamReader(
156             fileName))
157         {
158             string buffer;
159
160             //读入点的坐标数据
161             this.KnwPointList.Clear();
162             while (true)//读入公共点源坐标系坐标数据,至空行退出
163             {
164                 buffer = sr.ReadLine();
165                 if (string.IsNullOrEmpty(buffer)) break; //文件末尾或空行退出
166
167                 if (buffer[0] == '#') continue;
168
169                 string[] its = buffer.Split(new char[1] { ',', ' ' });

```

```

169         if (its.Length == 3)
170         {
171             GeoPoint pnt = new GeoPoint();
172             pnt.Name = its[0].Trim();
173             pnt.OX = double.Parse(its[1]);
174             pnt.OY = double.Parse(its[2]);
175             this.KnwPointList.Add(pnt);
176         }
177     }
178
179     while (true) // 读入公共点新坐标系坐标数据,至空行退出
180     {
181         buffer = sr.ReadLine();
182         if (string.IsNullOrEmpty(buffer)) break; // 文件末尾或空行退出
183
184         if (buffer[0] == '#') continue;
185
186         string[] its = buffer.Split(new char[1] { ',', ' ' });
187         if (its.Length == 3)
188         {
189             string name = its[0].Trim();
190             GeoPoint pnt = GetGeoPoint(name);
191             pnt.NX = double.Parse(its[1]);
192             pnt.NY = double.Parse(its[2]);
193         }
194     }
195
196     this.UnKnwPointList.Clear();
197     while (true) // 读入待计算点源坐标系坐标数据,至空行退出
198     {
199         buffer = sr.ReadLine();
200         if (string.IsNullOrEmpty(buffer)) break; // 文件末尾或空行退出
201
202         if (buffer[0] == '#') continue;
203
204         string[] its = buffer.Split(new char[1] { ',', ' ' });
205         if (its.Length == 3)
206         {
207             GeoPoint pnt = new GeoPoint();
208             pnt.Name = its[0].Trim();
209             pnt.OX = double.Parse(its[1]);
210             pnt.OY = double.Parse(its[2]);
211
212             this.UnKnwPointList.Add(pnt);
213         }
214     }
215 }
216
217
218 /// <summary>
219 /// 根据点名获取点对象
220 /// </summary>
221 /// <param name="name">点名</param>
222 /// <returns>点对象</returns>
223 private GeoPoint GetGeoPoint(string name)
224 {

```

```

225         foreach (var it in this.KnwPointList)
226         {
227             if (it.Name == name)
228                 return it;
229         }
230
231         return null;
232     }
233
234     /// <summary>
235     /// 写坐标转换数据文件
236     /// </summary>
237     /// <param name="fileName">文件名</param>
238     public void WriteTextFileData(string fileName)
239     {
240         using (System.IO.StreamWriter sr = new System.IO.StreamWriter(
fileName))
241         {
242             sr.WriteLine("#赫尔默特四参数转换法数据文件");
243             sr.WriteLine("#每行以“#”开头的行均被认为是注释行");
244             sr.WriteLine("#公共点在源坐标系中的坐标: 点名, X(N), Y(E)");
245             foreach (var pnt in this.knwPointList)
246             {
247                 sr.WriteLine("{0}, {1}, {2}", pnt.Name, pnt.OX, pnt.OY);
248             }
249
250             sr.WriteLine();
251             sr.WriteLine("#公共点在新坐标系中的坐标: 点名, X(N), Y(E)");
252             foreach (var pnt in this.knwPointList)
253             {
254                 sr.WriteLine("{0}, {1}, {2}", pnt.Name, pnt.NX, pnt.NY);
255             }
256
257             sr.WriteLine();
258             sr.WriteLine("#待转换点在源坐标系中的坐标: 点名, X(N), Y(E)");
259             foreach (var pnt in this.unKnwPointList)
260             {
261                 sr.WriteLine("{0}, {1}, {2}", pnt.Name, pnt.OX, pnt.OY);
262             }
263         }
264     }
265
266     /// <summary>
267     /// 写计算成果数据文件
268     /// </summary>
269     /// <param name="fileName">文件名</param>
270     public void WriteResultTextFileData(string fileName)
271     {
272         using (System.IO.StreamWriter sr = new System.IO.StreamWriter(
fileName))
273         {
274             sr.WriteLine("#赫尔默特四参数转换法计算成果数据文件");
275             sr.WriteLine("# 公共点坐标");
276             sr.WriteLine("# 点名, 源X(N), 源Y(E), 新X(N), 新Y(E)");
277             foreach (var pnt in this.knwPointList)
278             {
279                 sr.WriteLine(pnt);

```

```

279         }
280
281         sr.WriteLine();
282         sr.WriteLine("# 转换参数");
283         sr.WriteLine("a={0}, b={1}, c={2}, d={3}\r\n={4}, k={5}",
284             this.a, this.b, this.c, this.d, this.alpha, this.k);
285
286         sr.WriteLine();
287         sr.WriteLine("# 待计算点的坐标");
288         sr.WriteLine("# 点名, 源X(N), 源Y(E), 新X(N), 新Y(E)");
289         foreach (var pnt in this.unKnwPointList)
290         {
291             sr.WriteLine(pnt);
292         }
293     }
294 }
295
296 /// <summary>
297 /// 根据尺度因子d与旋转角度 计算线性计算量c和d
298 /// </summary>
299 public void CalCd()
300 {
301     this.c = this.k * Math.Cos(ZXY.SMath.DMS2RAD(this.alpha));
302     this.d = this.k * Math.Sin(ZXY.SMath.DMS2RAD(this.alpha));
303 }
304
305 /// <summary>
306 /// 赫尔默特法计算转换系数
307 /// </summary>
308 public void CalCoefficient()
309 {
310     int n0 = this.knwPointList.Count;
311     if (n0 < 2) return; //少于两个公共点, 无法计算
312
313     double[,] B = new double[2 * n0, 4];
314     double[] l = new double[2 * n0];
315     double[,] N = new double[4, 4];
316     double[] U = new double[4];
317
318     //组成系数阵B与l, 此处应注意读入的坐标是测量坐标,
319     //应将测量坐标转换为数学坐标
320     double x, y, xT, yT;
321     for (int i = 0; i < n0; i++)
322     {
323         x = knwPointList[i].OY; //数学上的x, 测量上的y
324         y = knwPointList[i].OX; //数学上的y, 测量上的x
325         xT = knwPointList[i].NY;
326         yT = knwPointList[i].NX;
327
328         B[(2 * i), 0] = 1.0;
329         B[(2 * i), 1] = 0.0;
330         B[(2 * i), 2] = x;
331         B[(2 * i), 3] = y;
332         l[2 * i] = xT;
333
334         B[(2 * i + 1), 0] = 0.0;

```

```

335         B[(2 * i + 1), 1] = 1.0;
336         B[(2 * i + 1), 2] = y;
337         B[(2 * i + 1), 3] = -x;
338         l[2 * i + 1] = yT;
339     }
340
341     for (int k = 0; k < 4; k++)
342     {
343         for (int j = 0; j < 4; j++)
344         {
345             N[k, j] = 0.0;
346             for (int i = 0; i < 2 * n0; i++)
347             {
348                 N[k, j] += B[i, k] * B[i, j];
349             }
350         }
351
352         U[k] = 0.0;
353         for (int i = 0; i < 2 * n0; i++)
354             U[k] += B[i, k] * l[i];
355     }
356
357     NegativeMatrix(N, U, 4);
358
359     this.a = U[0];
360     this.b = U[1];
361     this.c = U[2];
362     this.d = U[3];
363     this.alpha = ZXY.SMath.RAD2DMS(Math.Atan2(d, c));
364     this.k = Math.Sqrt(d * d + c * c);
365 }
366
367 /// <summary>
368 /// 计算点在新坐标系中的坐标
369 /// </summary>
370 public void CalUnKwXY()
371 {
372     // 应将测量坐标转换为数学坐标
373     double x, y, xT, yT;
374     foreach (var it in this.unKwPointList)
375     {
376         x = it.OY; y = it.OX;
377
378         xT = this.a + this.c * x + this.d * y;
379         yT = this.b + this.c * y - this.d * x;
380
381         it.NY = xT; it.NX = yT;
382     }
383 }
384
385 /// <summary>
386 /// 高斯约化法解方程 AX = B中的X值, 结果存B中
387 /// </summary>
388 /// <param name="A">A: nxn</param>
389 /// <param name="B">B: nx1</param>
390 /// <param name="n">n</param>

```

```

391     private void NegativeMatrix(double[,] A, double[] B, int n)
392     {
393         for (int k = 0; k < n - 1; k++)
394         {
395             for (int i = k + 1; i < n; i++)
396             {
397                 A[i, k] /= A[k, k];
398                 for (int j = k + 1; j < n; j++)
399                 {
400                     A[i, j] -= A[i, k] * A[k, j];
401                 }
402                 B[i] -= A[i, k] * B[k];
403             }
404         }
405         B[n - 1] /= A[(n - 1), (n - 1)];
406         for (int i = n - 2; i >= 0; i--)
407         {
408             double s = 0.0;
409             for (int j = i + 1; j < n; j++)
410             {
411                 s += A[i, j] * B[j];
412             }
413             B[i] = (B[i] - s) / A[i, i];
414         }
415     }
416 }
417 }

```

界面响应代码如下：

```

1  using Microsoft.Win32;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using System.Windows;
8  using System.Windows.Controls;
9  using System.Windows.Data;
10 using System.Windows.Documents;
11 using System.Windows.Input;
12 using System.Windows.Media;
13 using System.Windows.Media.Imaging;
14 using System.Windows.Navigation;
15 using System.Windows.Shapes;
16
17 namespace CoordniateTransform
18 {
19     /// <summary>
20     /// MainWindow.xaml 的交互逻辑
21     /// </summary>
22     public partial class MainWindow : Window
23     {
24         private CoordinateSystem cs;
25         public MainWindow()
26         {
27             InitializeComponent();
28

```



```
29         cs = new CoordinateSystem();
30         this.DataContext = cs;
31     }
32
33     private void menuItem_OpenTextFileData_Click(object sender,
RoutedEventArgs e)
34     {
35         OpenFileDialog dlg = new OpenFileDialog();
36         dlg.DefaultExt = ".txt";
37         dlg.Filter = "平面坐标相似变换数据文件|*.txt|All File (*.*)|*.*";
38         if (dlg.ShowDialog() != true) return;
39
40         cs.ReadTextFileData(dlg.FileName);
41     }
42
43     private void menuItem_SaveTextFileData_Click(object sender,
RoutedEventArgs e)
44     {
45         SaveFileDialog dlg = new SaveFileDialog();
46         dlg.DefaultExt = ".txt";
47         dlg.Filter = "平面坐标相似变换数据文件|*.txt|All File (*.*)|*.*";
48         if (dlg.ShowDialog() != true) return;
49
50         cs.WriteTextFileData(dlg.FileName);
51     }
52
53     private void menuItem_CalCoefficient_Click(object sender, RoutedEventArgs
e)
54     {
55         cs.CalCoefficient();
56     }
57
58     private void menuItem_Cal_UnKnw_XY_Click(object sender, RoutedEventArgs e
)
59     {
60         cs.CalUnKnwXY();
61     }
62
63     private void menuItem_Write_Result_Click(object sender, RoutedEventArgs e
)
64     {
65         SaveFileDialog dlg = new SaveFileDialog();
66         dlg.DefaultExt = ".txt";
67         dlg.Filter = "平面坐标相似变换成果数据文件|*.txt|All File (*.*)|*.*";
68         if (dlg.ShowDialog() != true) return;
69
70         cs.WriteResultTextFileData(dlg.FileName);
71     }
72
73     private void menuItem_Exit_Click(object sender, RoutedEventArgs e)
74     {
75         this.Close();
76     }
77
78     private void button_Cal_cd_Click(object sender, RoutedEventArgs e)
79     {
```

```
80         cs.CalCd();  
81     }  
82 }  
83 }
```

第六章 线路要素计算程序设计

6.1 圆曲线与有缓和曲线的圆曲线

6.1.1 圆曲线的数学模型

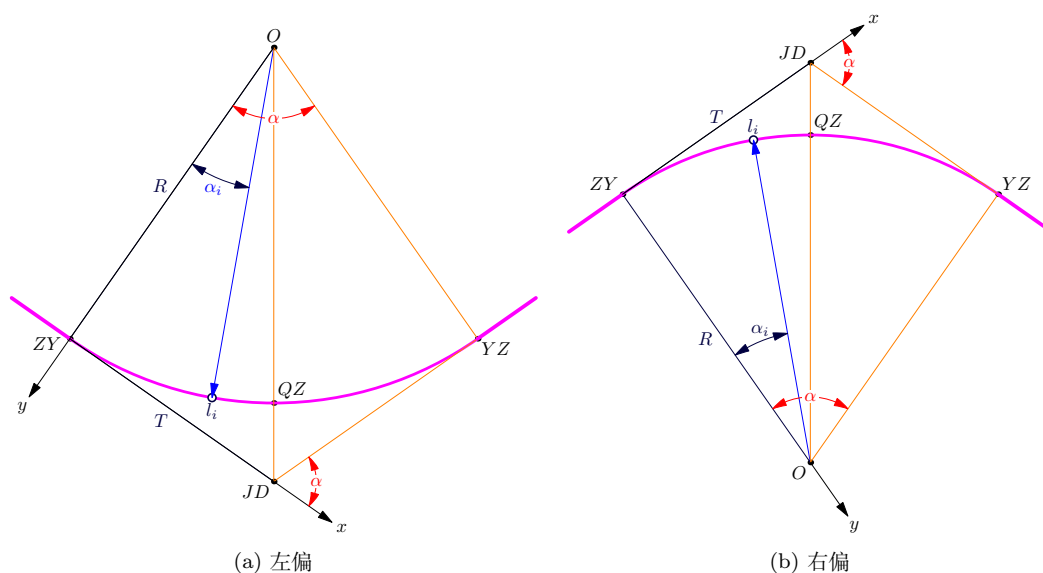


图 6.1 圆曲线要素图

切线长: $T = R \cdot \tan(\alpha/2)$

曲线长: $L = R \cdot \alpha$

外矢距: $E = R \cdot (\sec(\alpha/2) - 1)$

切曲差: $q = 2T - L$

6.1.2 圆曲线上点的坐标计算

全站仪的坐标放样模式与 GPS RTK 可以十分方便的进行圆曲线上点的坐标放样。

如图6.1所示,以ZY点为原点,以ZY至JD切线方向为x轴,以ZY至O点方向为y轴建立ZY切线测量坐标系。从(a)与(b)两图可以看出无论圆曲线是左偏还是右偏的,其坐标系是一致的。在ZY切线坐标系中用极坐标法按如下公式可以计算出圆曲线上任意一点 l_i 的坐标。

圆曲线偏左:

$$\left. \begin{aligned} x_i &= R \sin \alpha_i \\ y_i &= -R(1 - \cos \alpha_i) \end{aligned} \right\} \quad (6.1)$$

圆曲线偏右:

$$\left. \begin{aligned} x_i &= R \sin \alpha_i \\ y_i &= R(1 - \cos \alpha_i) \end{aligned} \right\} \quad (6.2)$$

式中: $\alpha_i = l_i/R$, $\alpha_i \leq \alpha$, l_i 可用圆曲线上任意一点的里程桩号减去 ZY 点的里程桩号。

已知 JD 的坐标与里程桩号, 根据圆曲线的结合几何要素 R, α 即可计算出圆曲线上特征点的里程桩号:

$$KNo_{ZY} = KNo_{JD} - T$$

$$KNo_{QZ} = KNo_{ZY} + T/2$$

$$KNo_{YZ} = KNo_{QZ} + T/2$$

在 ZY 切线坐标系中计算出圆曲线上各点的坐标之后, 还需将其转换为测量坐标系 (或更正式的称为大地坐标系或独立施工坐标系)。在前一章我们已经做过坐标系的转换了, 在这个线路转换中, 我们将 ZY-JD 边定义为 x 轴, 因此两坐标系的夹角即为 ZY-JD 边的坐标方位角, 其转换关系如图 6.2 所示:

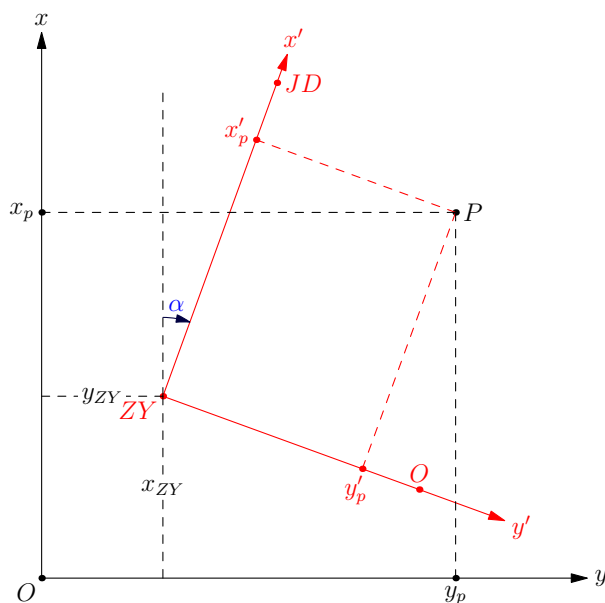


图 6.2 ZY 坐标系转测量坐标系

转换公式如下:

$$\left. \begin{aligned} x_P &= x_{ZY} + x'_P \cos \alpha - y'_P \sin \alpha \\ y_P &= y_{ZY} + x'_P \sin \alpha + y'_P \cos \alpha \end{aligned} \right\} \quad (6.3)$$

6.1.3 缓和曲线的数学模型

缓和曲线参数计算公式为:

$$\beta_0 = \frac{l_0}{2R}$$

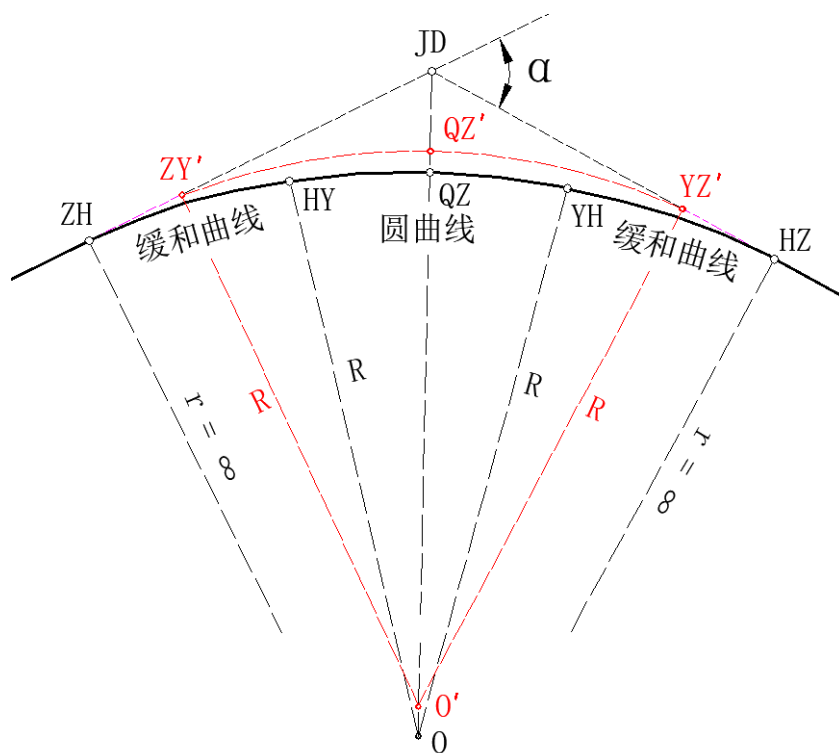


图 6.3 缓和曲线的定义

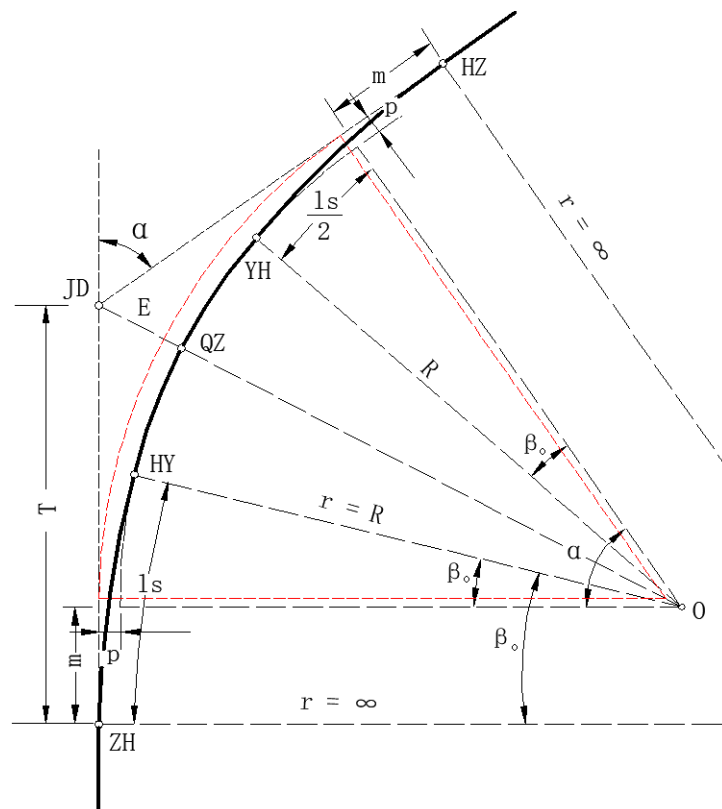


图 6.4 缓和曲线的内移距和切线增长

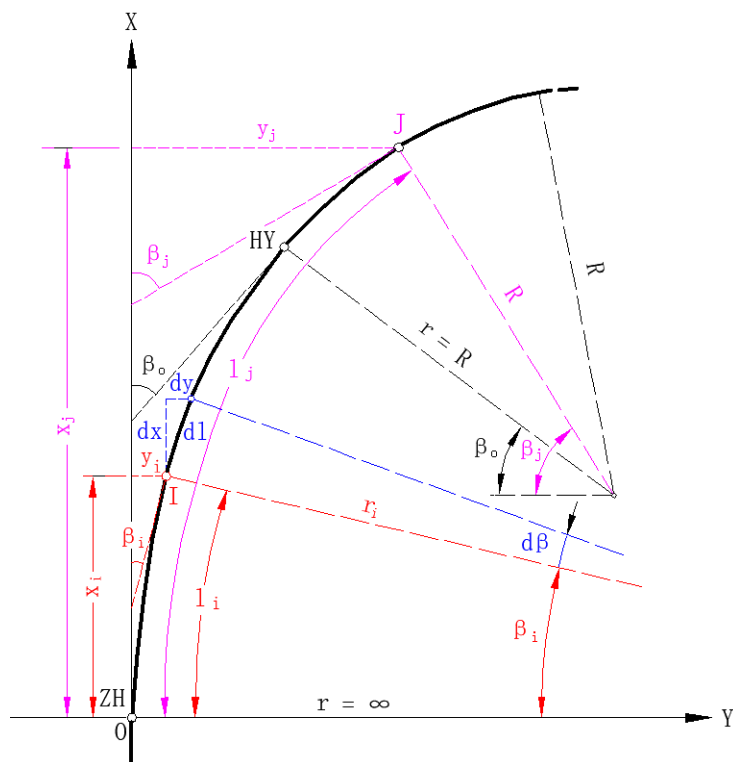


图 6.5 缓和曲线的坐标计算

切垂距:

$$m = \frac{l_0}{2} - \frac{l_0^3}{240R^2}$$

内移距:

$$p = \frac{l_0^2}{24R}$$

切线长:

$$T = m + (R + p) \cdot \tan \frac{\alpha}{2}$$

曲线全长:

$$L = R \cdot (\alpha - 2\beta_0) + 2l_0$$

圆曲线长:

$$L_C = R \cdot (\alpha - 2\beta_0)$$

外矢距:

$$E = (R + p) \cdot (\sec \frac{\alpha}{2} - 1)$$

1. 缓和曲线在 ZH 切线直角坐标系中的坐标计算 (ZH 段)

以 ZH 为坐标系原点, ZH 至 JD 方向为 x 轴, 过 ZH 点垂直于 ZH-JD 方向为 y 轴建立坐标, 如图6.4所示。则 ZH 部分缓和曲线上各点的坐标为:

$$\left. \begin{aligned} x_i &= l_i - \frac{l_i^5}{40R^2l_0^2} + \frac{l_i^9}{3456R^4l_0^4} - \frac{l_i^{13}}{599040R^6l_0^6} + \dots \\ y_i &= \frac{l_i^3}{6Rl_0} - \frac{l_i^7}{336R^3l_0^3} + \frac{l_i^{11}}{42240R^5l_0^5} - \frac{l_i^{15}}{9676800R^7l_0^7} + \dots \end{aligned} \right\} \quad (6.4)$$

以上公式在计算中取前两项或前三项即可。

2. 缓圆点 (HY) 坐标计算公式为:

$$\left. \begin{aligned} x_{HY} &= l_0 - \frac{l_0^3}{40R^2} + \frac{l_0^5}{3456R^4} - \frac{l_0^7}{599040R^6} + \dots \\ y_{HY} &= \frac{l_0^2}{6R} - \frac{l_0^4}{336R^3} + \frac{l_0^6}{42240R^5} - \frac{l_0^8}{9676800R^7} + \dots \end{aligned} \right\} \quad (6.5)$$

3. 圆曲线上点的坐标计算公式为:

$$\left. \begin{aligned} x_j &= R \sin \beta_j + m \\ y_j &= R(1 - \cos \beta_j) + p \end{aligned} \right\} \quad (6.6)$$

4. 缓和曲线在 HZ 切线直角坐标系中的坐标计算 (HZ 段)

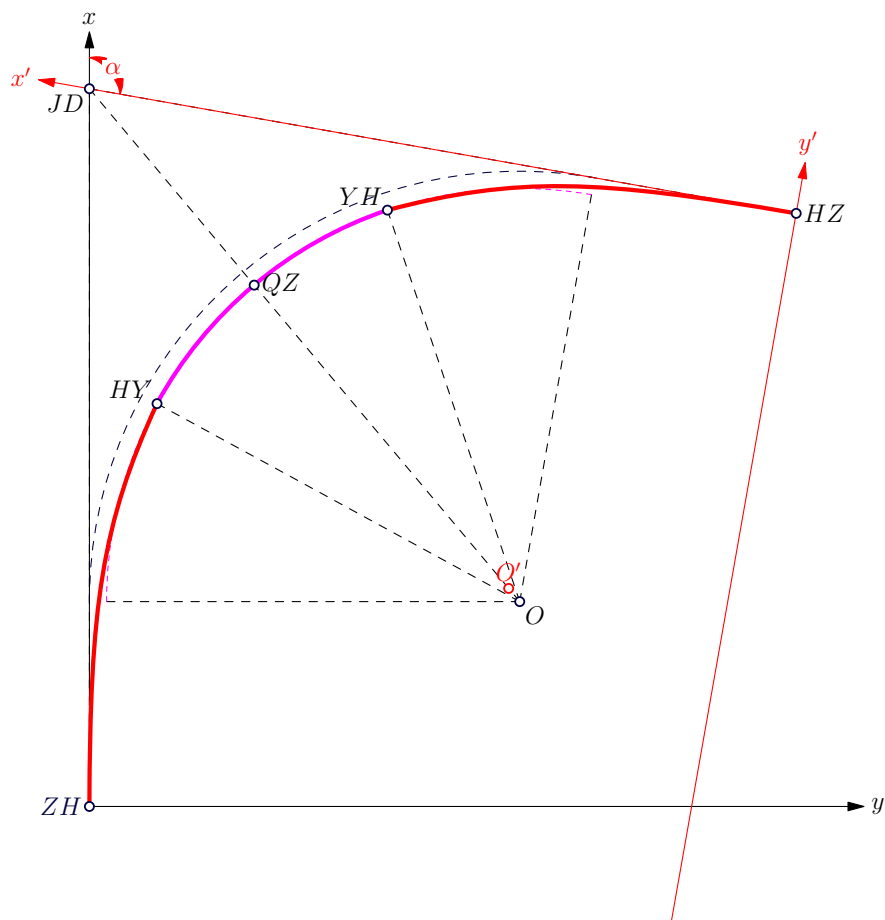


图 6.6 缓和曲线的 HZ 部分坐标计算

如图6.6所示建立以缓直点 HZ 为原点, 过 HZ 至 JD 方向为 x 轴, 过 HZ 点的缓和曲线切线为 y 轴的直角坐标系, 计算另一半曲线任意一点的坐标 (x'_i, y'_i) 。然后, 将坐标转换为以 ZH 点为原点的直角坐标系中。

缓和曲线在 HZ 坐标系中的坐标可以先继续使用公式 6.4 计算, 然后由图6.6 可知将其转换为 HZ 坐标系中的坐标计算公式为:

$$\left. \begin{aligned} x'_i &= x_i \\ y'_i &= -y_i \end{aligned} \right\} \quad (6.7)$$

为了将 HZ 坐标系中的点坐标转换为 ZH 坐标中,我们引用公式6.3来计算,请注意公式6.3中的 α 为 x' 轴的方位角 (即 x 轴到 x' 轴的水平夹角), 因此图6.6应用到公式6.3中的 α 应为 $\alpha + 180^\circ$ 。如果将其代入公式6.3, 则有转换公式为:

$$\left. \begin{aligned} x_i &= x_{HZ} - x'_i \cos \alpha + y'_i \sin \alpha \\ y_i &= y_{HZ} - x'_i \sin \alpha - y'_i \cos \alpha \end{aligned} \right\} \quad (6.8)$$

公式6.8中的 (x_{HZ}, y_{HZ}) 为 HZ 点在 ZH 坐标, 其值为:

$$\left. \begin{aligned} x_{HZ} &= T(1 + \cos \alpha) \\ x_{HZ} &= T \sin \alpha \end{aligned} \right\} \quad (6.9)$$

也可以不用公式6.8, 直接将 $\alpha + 180^\circ$ 代入到公式6.3中进行计算。

在以上计算中, 我们以曲线右偏为例的, 如果曲线左偏, 同样的方法建立坐标系, x 坐标是相同的, y 坐标乘以 -1 即可。

5. 曲线上点坐标转换为大地坐标的计算公式为:

由于已经将曲线上的点坐标统一到 ZH 坐标系中了, 我们继续引用公式6.3 将 ZH 切线坐标系坐标转换到大地坐标系 (测量坐标系) 中, 公式中的 α 为 $ZH \rightarrow JD$ 的坐标方位角。