测量程序设计（C#–WPF）

朱学军

2018 年 3 月

目录

第 1 章 教学计划及课程目标 **5**
1.1 教学计划 5
1.1.1 教学课时 5
1.1.2 学习内容 5
1.2 编程预言的选择 5
1.3 编程环境与工具 5
1.4 参考教材 6

第 2 章 从 C 走向 C# **7**
2.1 Client/Server 程序设计模式 7
2.2 从面向过程走向面向对象的程序设计 7
2.3 C# 程序的组织结构 11

第 3 章 C# 语言基础 **13**
3.1 111111 13

第 4 章 C# 面向对象特性-封装 **15**
4.1 111111 15

第 5 章 C# 面向对象特性-继承 **17**
5.1 111111 17

第 6 章 C# 面向对象特性-多态 **19**
6.1 111111 19

第 7 章 C# 图形界面-WPF **21**
7.1 111111 21

第 8 章 常用测量函数设计 **23**
8.1 常用测量计算公式 23
8.1.1 角度弧度互换函数： 24
8.1.2 坐标方位角推算 25

8.1.3 平面坐标正反算 26

第 9 章 高斯投影正反算与换带 29

9.1 数据模型 29
9.1.1 投影换带的目的 29
9.1.2 高斯投影的主要内容 29
9.1.3 高斯投影的数学模型 29
9.1.4 数学模型分析 32
9.2 高斯投影基本功能实现 32
9.3 * 面向对象的高斯坐标正反算程序设计 37
9.4 实现换带计算和文件读写 43
9.4.1 换带计算 43
9.4.2 多点计算和文件读写 43
9.5 小结 45

第 10 章 坐标转换程序设计 47

10.1 111111 47

第 11 章 线路要素计算程序设计 49

11.1 111111 49

第 12 章 单导线近似平差程序设计 51

12.1 111111 51

第 1 章　教学计划及课程目标

1.1　教学计划

1.1.1　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%）

1.1.2　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。

1.2　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。

1.3　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）。

1.4 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.....）。

第 2 章　从 C 走向 C#

2.1　Client/Server 程序设计模式

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 WinForm 中直接写入代码的习惯；

以上这些习惯将带来一系列的问题，在团队开发与多人协作尤其如此，如：不能进行 unittest(单元测试)、git（著名的源代码管理工具）代码合并时会引发大量的冲突。

2.2　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，如下代码所示：

```
// Ch01Ex1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#define _USE_MATH_DEFINES
#include <math.h>
#include <string.h>
#define PI M_PI

typedef struct _point {
    char name[11];
    double x, y, z;
}Point;

typedef struct _circle {
    Point center;
    double r;
    double area;
    double length;
}Circle;

//计算圆的面积
double Area(Circle * c) {
```

```
25      return PI * c->r * c->r;
26  }
27
28  //计算两点的距离
29  double Distance(Point * p1, 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(Circle * c1, Circle * c2) {
37      double d = Distance(&c1->center, &c2->center);
38    return d <= (c1->r + c2->r);
39  }
40
41  int main()
42  {
43      Point pt1;
```

相应的 main 函数测试代码如下：

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

程序的运行结果如下：

```
Circle1 的面积 = 20106.192983
Circle1 与 Circle2 是否相交 : 是
```

与以上代码相对应的 C# 代码为：

```
using System;

namespace Ch01Ex02
{
    class Point
    {
        private string name;
        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        private double x;
        public double X
        {
            get { return x; }
            set { x = value; }
        }
        private double y;
        public double Y
        {
            get { return y; }
            set { y = value; }
        }
        private double z;
        public double Z
        {
            get { return z; }
            set { z = value; }
        }

        public Point()
        {
            Name = "";
            x = y = z = 0;
        }
        //函数重载：函数名称一样，但参数（个数或类型）不一样
        public Point(string name, double x, double y, double z)
        {
            this.name = name;
            this.x = x;
            this.y = y;
            this.z = z;
        }

```

```
47
48          /// <summary>
49          /// 计算当前点至p2点的距离
50          /// </summary>
51          /// <param name="p2">目标点</param>
52          /// <returns>两点的距离</returns>
53          public double Distance(Point p2)
54          {
55              double dx = X - p2.X;
56              double dy = Y - p2.Y;
57              return Math.Sqrt(dx * dx + dy * dy);
58          }
59      }
60
61      class Circle
62      {
63          private Point center;
64          private double r;
65
66          private double area;
67          public double Area
68          {
69              get { return area; }
70          }
71
72          private double length;
73
74          public Circle()
75          {
76
77          }
78
79          public Circle(string centerName, double x, double y, double z, double r)
80          {
81              center = new Point(centerName, x, y, z);
82              this.r = r;
83
84              CalArea();
85          }
86
87
88          /// <summary>
89          /// 计算圆的面积
90          /// </summary>
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 }
```

相应的 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              Circle c1 = new Circle("pt1", 100, 100, 425.324, 80);
10             Circle c2 = new Circle("pt2", 200, 200, 417.626, 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}", yes ? "是" : "否" );
17         }
18     }
19 }
```

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
```

2.3 C# 程序的组织结构

从以上代码可以看出，C# 是一纯面向对象语言。

第 3 章　C# 语言基础

3.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 4 章　C# 面向对象特性-封装

4.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 5 章　C# 面向对象特性-继承

5.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 6 章　C# 面向对象特性-多态

6.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 7 章　C# 图形界面-WPF

7.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 8 章　常用测量函数设计

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：

```
namespace ZXY
{
    public static class SMath
    {
        public const double PI=3.14159265358979323846264338327 95;
        public const double PI2=6.28318530717958647692528676655 9;
        public const double TODEG=57.29577951308232087679815481 4105;
        public const double TORAD=0.01745329251994329576923690768489;
        public const double TOSECOND=206264.80624709635515647335733078;

        public static double DMS2RAD(double dmsAngle)
        {
            ......
        }

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

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，在此加入了自己的命名空间 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）。

8.1　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。

8.1.1 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°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°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);
}
```

8.1.2 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;
}
```

8.1.3 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 9 章　高斯投影正反算与换带

9.1　数据模型

9.1.1　投影换带的目的

高斯投影是为解决球面与平面之间坐标映射问题的，即大地坐标 (B, L) 与高斯平面直角坐标 (x, y) 之间的换算，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。

9.1.2　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。

9.1.3　高斯投影的数学模型

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，以下简称为大地测量学基础。

高斯投影是在椭球参数 (长半轴 a、短半轴 b、扁率 α) 一定的条件下，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。

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：

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

(e) 卯酉圈曲率半径：

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

(f) 辅助符号：

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

2. 椭球面梯形图幅面积计算

$$P = \frac{b^2}{2}(L_2 - L_1)\left|\frac{\sin B}{1 - e^2 \sin^2 B} + \frac{1}{2e}\ln\frac{1 + e\sin B}{1 - e\sin B}\right|_{B1}^{B2}$$

公式引用自《大地测量学基础》第 121 页 (4-140)。

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

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

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

式中：

$$\begin{cases} a_0 = m_0 + \dfrac{m_2}{2} + \dfrac{3}{8}m_4 + \dfrac{5}{16}m_6 + \dfrac{35}{128}m_8 \\ a_2 = \dfrac{m_2}{2} + \dfrac{m_4}{2} + \dfrac{15}{32}m_6 + \dfrac{7}{16}m_8 \\ a_4 = \dfrac{m_4}{8} + \dfrac{3}{16}m_6 + \dfrac{7}{32}m_8 \\ a_6 = \dfrac{m_6}{32} + \dfrac{m_8}{16} \\ a_8 = \dfrac{m_8}{128} \end{cases}$$

式中 m_0, m_2, m_4, m_6, m_8 的值为：

$$\begin{cases} m_0 = a(1 - e^2) \\ m_2 = \dfrac{3}{2}e^2 m_0 \\ m_4 = \dfrac{5}{4}e^2 m_2 \\ m_6 = \dfrac{7}{6}e^2 m_4 \\ m_8 = \dfrac{9}{8}e^2 m_6 \end{cases}$$

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

4. 坐标正算

$$\begin{cases} x = X + \dfrac{N}{2} sinBcosBl^2 + \dfrac{N}{24} sinBcos^3B(5 - t^2 + 9\eta^2 + 4\eta^4)l^4 \\ \quad + \dfrac{N}{720} sinBcos^5B(61 - 58t^2 + t^4)l^6 \\ y = NcosBl + \dfrac{N}{6} cos^3B(1 - t^2 + \eta^2)l^3 \\ \quad + \dfrac{N}{120} cos^5B(5 - 18t^2 + t^4 + 14\eta^2 - 58\eta^2t^2)l^5 \end{cases}$$

公式引用自《大地测量学基础》第 169 页 (4-367)。

5. 坐标反算

$$\begin{cases} B = B_f - \dfrac{t_f}{2M_fN_f} y^2 + \dfrac{t_f}{24M_fN_f^3}(5 + 3t_f^2 + \eta_f^2 - 9\eta_f^2t_f^2)y^4 \\ \quad - \dfrac{t_f}{720M_fN_f^5}(61 + 90t_f^2 + 45t_f^4)y^6 \\ l = \dfrac{1}{N_f cosB_f} y - \dfrac{1}{6N_f^3 cosB_f}(1 + 2t_f^2 + \eta_f^2)y^3 \\ \quad + \dfrac{1}{120N_f^5 cosB_f}(5 + 28t_f^2 + 24t_f^4 + 6\eta_f^2 + 8\eta_f^2t_f^2)y^5 \end{cases}$$

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)。

9.1.4 数学模型分析

分析以上各个计算公式发现，如果长半轴与扁率确定, 参考椭球的第一偏心率 e、第二偏心率 e'，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。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。

9.2 高斯投影基本功能实现

为了让我们的算法能被其他项目使用，我们在此新建一 Class Library(.NET Framework) 项目，在项目中将以前所写的 SMath.cs 文件加入，将命名空间改为 ZXY 。

基于以上分析，我们新建一椭球类 (*Ellipsoid*)，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：

```
//File Ellipsoid.cs
using System;

namespace ZXY
{
    public class Ellipsoid
    {
        private double a;
        private double b;
        private double f; //(a-b)/a = 1/f

        private double e2;
        private double eT2;

        private double a0;
        private double a2;
        private double a4;
        private double a6;
        private double a8;

        private double m0;
        private double m2;
        private double m4;
        private double m6;
        private double m8;

        ..............................
```

```
    }
}
```

C 语言是典型的面向过程设计语言，小巧、灵活，功能强大。下面我们将从一个小功能开始构造满足以上全部功能的程序。

在此我们先完成正算功能，我们设定正算中所用的角度均为弧度，从而避免在正算中进行度分秒与弧度的转换问题，我们也先把读数据文件、多点等问题去掉，从只计算一个点开始考虑问题。我们现在只考虑 54 北京坐标系的问题，别的暂且不管。假如我们的主函数的调用形式如下：

```
void main()
{
    //克拉索夫斯基参考椭球
    double a = 6378245.0,  f = 298.3;
    //正算点的纬度和经差
    double B = 0.3836189311, l = 0.0423314484;
    // B = 21°58 47.0845  l = 2°25 31.4880
    double x, y;
    CalEcd(a, f);
    GaussZs(B, l, &x, &y);
    printf("x = %lf, y = %lf \n", x, y);
    //算出的x=2433586.692,  y=250547.403
}
```

以上程序的流程为：

(1) 给定一个类型的参考椭球：如长半轴 a，扁率 f

(2) 计算相关的参数：计算偏心率，子午线弧长系数等，这些相对于一个给定的椭球来说，它是一个定值；

(3) 根据给定的 B，l 计算相应的高斯平面坐标；

(4) 输出计算值。

从流程上可看出，GaussZs 函数的原型如下：

```
void GaussZs(double B, double l,
             double * x, double * y)
```

这里 B，l 为给定值，x，y 设定为指针类型，用于返回计算后的值。下面我们来实现这个函数，它的计算流程如下：

(1) 计算子午线弧长；

(2) 计算 x 和 y 坐标；

我们设计该函数如下：

```
void GaussZs(double B, double l,
             double * x, double * y)
```

```
{
    double sinB = sin( B );
    double cosB = cos( B );
    double cosB2 = cosB * cosB;
    double cosB4 = cosB2 * cosB2;
    double l2 = l * l;
    double l4 = l2 * l2;

    double g = _eT * cosB;
    double g2 = g * g;
    double g4 = g2 * g2;
    double t = tan(B);
    double t2 = t * t;
    double t4 = t2 * t2;

    //计算子午线弧长
    double X = MeridianArcLength( B );
    double N = _c / sqrt(1.0 + g2);

    *x = X + N * sinB * cosB * l2 *
    ( 0.5
      + cosB2 * l2 * (5.0 - t2 + 9.0 * g2 + 4.0 * g4) / 24.0
      + cosB4 * l4 * (61.0 - 58.0 * t2
          + t4 + 270.0 * g2 - 330.0 * g2 * t2)  / 720.0
    );
    *y = N * cosB * l *
    ( 1.0
      + cosB2 * l2 * (1.0 - t2 + g2) / 6.0
      + cosB4 * l4 * (5.0 - 18.0 * t2 + t4
          + 14.0 * g2 - 58.0 * t2 * g2) / 120.0
    );
}
```

从函数的实现看，前面的变量只是为了简化后面的计算式，整个函数中没有逻辑判断和循环，因此较为简单。但子午线弧长的计算函数 MeridianArcLength(double) 我们还没实现，根据算法，它的实现如下：

```
double MeridianArcLength(double B)
{
    double sinB = sin(B);
```

```
    double cosB = cos(B);
    double sinB3 = sinB * sinB * sinB;
    double sinB5 = sinB * sinB * sinB3;
    double sinB7 = sinB * sinB * sinB5;
    return _A0 * B - cosB * (_B0 * sinB + _C0 * sinB3
           + _D0 * sinB5 + _E0 * sinB7);
}
```

程序中的_A0、_B0、_C0、_D0、_E0 为子午线弧长计算的系数，在此我把它们设为全局变量，在 CalEcd 函数中进行计算。如：

```
double _a, _b, _c, _d, _e, _eT;
double _A0, _B0, _C0 , _D0, _E0;
double _e2, _e4, _e6, _e8;
```

由于这些相关的系数只与椭球的参数有关，我们只在函数 CalEcd 计算一次。其实现如下：

```
void CalEcd(double a, double f)
{
    _a = a; _b = _a * (1.0 - 1.0 / f);

    _e = sqrt(_a * _a - _b * _b) / _a;
    _eT = sqrt(_a * _a - _b * _b) / _b;
    _c = _a / _b * _a;
    _d = _b / _a * _b;

    _e2 = _e * _e;
    _e4 = _e2 * _e2;
    _e6 = _e2 * _e4;
    _e8 = _e2 * _e6;

    _A0 = _d * (  1    + 3.0 /4.0 * _e2
                  + 45.0 / 64.0 * _e4
              + 175.0 / 256.0 * _e6
              + 11025.0 / 16384.0 * _e8 );
    _B0 = _d * (         3.0 /4.0 * _e2
                  + 45.0 / 64.0 * _e4
              + 175.0 / 256.0 * _e6
              + 11025.0 / 16384.0 * _e8);
    _C0 = _d * (         15.0 / 32.0 * _e4
```

```
                + 175.0 / 384.0 * _e6
                        + 3675.0 /  8192.0 * _e8);
    _D0 = _d * (          35.0 /  96.0 * _e6
                    + 735.0 /  2048.0 * _e8);
    _E0 = _d * (     315.0 /  1024.0 * _e8 );
}
```

至此，我们完成了高斯平面直角坐标的正算功能的程序设计。为了利用 C++ 语言增强了的 C 功能，我们把文件的扩展名由 c 改为 cpp，以利用 C++ 编译器进行严格检查。

为了进行源代码一级的复用，我们在工程中新加入一个头文件（.h），将上面除主函数 main() 之外的所有代码剪切到头文件中（如头文件为 Gauss.h），在主函数所在的文件中将头文件包含在内，即在文件顶端加入：`#include "Gauss.h"`，在别的地方我们就可继续使用这些函数。

为了直接利用纬度、经度和中央子午线的经度计算高斯平面坐标，我们可以利用函数重载的功能继续实现坐标正算：

```
void GaussZs(double B, double L, double L0,
             double * x, double * y)
{
    double l = L - L0;
    GaussZs(B, l, x, y);
}
```

这里的实现很简单，只是计算了经差，调用了原来的正算函数。我们就可在主函数中作如下形式的调用了：

```
void main()
{
    //21°58 47.0845 的弧度形式
    double B = 0.3836189311;
    // 113°25 31.4880 的弧度形式
    double L = 1.9796469181;
    // 111°的弧度形式
    double L0 = 1.9373154697;
    //54//x:2433586.692, y:250547.403
    GaussZs(B, L, L0, &x, &y);
    printf("x = %lf, y = %lf \n", x, y);
}
```

至此，我们已经完成了高斯坐标正算的全部计算了。以上的函数调用中的角度均使用了弧度的形式，利用前面所讲的角度化弧度的函数，我们可以以下形式的调用：

```
void main()
{
    //21°58 47.0845
    double B = DMStoRAD(21.58470845);
    //113°25 31.4880
    double L = DMStoRAD (113.25314880);
    double L0 = DMStoRAD(111.0); //111°
    //54//x:2433586.692, y:250547.403
    GaussZs(B, L, L0, &x, &y);
    printf("x = %lf, y = %lf \n", x, y);
}
```

同样，坐标反算的程序设计的实现方法也基本一样。由于面向过程的程序设计不是我们的重点，就将这部分的实现放入面向对象中。

9.3 * 面向对象的高斯坐标正反算程序设计

在前面的程序设计中，我们将椭球参数用全局变量表示，为了消除这些全局变量的影响，我们采用类的形式对其进行封装，封装的形式如下：

```
class CEarth
{
private:
    double _a, _b, _c, _d, _e, _eT;
    double _A0, _B0, _C0 , _D0, _E0;
    double _e2, _e4, _e6, _e8, _e10;
};
```

将它们设为 private，防止类以外的代码随意访问它们。再将上面实现的正算等函数作为其成员函数，则形式为：

```
class CEarth
{
private:
    double _a, _b, _c, _d, _e, _eT;
    double _A0, _B0, _C0 , _D0, _E0;
    double _e2, _e4, _e6, _e8, _e10;
public:
    CEarth();//构造函数
    virtual ~CEarth();  //析构函数
```

```
    //计算相关系数
    void CalEcd(double a, double alf);
    //正算
    void GsZs(double l, double B,
              double& x, double& y);
    void GsZs(double L, double B, double L0,
              double& x, double& y);
    //反算
    void GsFs(double x, double y,
              double& l, double& B);
    void GsFs(double x, double y, double L0,
              double & L, double & B);
    //子午线收敛角
    double MeridianConvergentAngleByBl(
              double l, double B);
    double MeridianConvergentAngleByxy(
              double x, double y);
    //子午线弧长
    double MeridianArcLength(double B);
     //底点纬度
    double Bf(double x);
};
```

在类的成员函数定义中，将传进的参数用引用的形式加 const 进行限定，同时用引用取代指针进行函数返回值，如 GsZs，GsFs 函数。

在这里，我们看看反算的实现。在反算的计算流程中，经差 l 的计算很简单，直接计算即可，但纬度的计算需要先计算底点纬度，由于我们的程序是针对多种椭球的，底点纬度的数值计算公式是不能使用的。我们用子午线弧长计算公式进行迭代计算，在开始迭代时，取弧长的初值为`x / _A0`，其具体实现如下：

```
double CEarth::Bf(double x)
{
    double Bf0 = x / _A0; //子午线弧长的初值
    int i = 0;
    while( i < 10000 )//设定迭代次数
    {
        double sinBf = sin(Bf0);
        double cosBf = cos(Bf0);
        double sinBf3 = sinBf * sinBf * sinBf;
```

```
        double sinBf5 = sinBf * sinBf * sinBf3;
        double sinBf7 = sinBf * sinBf * sinBf5;

        double Bf = (  x
                 + cosBf * ( _B0 * sinBf
                                 + _C0 * sinBf3
                    + _D0 * sinBf5
                    + _E0 * sinBf7)
            ) / _A0;
        if( fabs(Bf - Bf0) < 1e-10) //计算精度
            return Bf;
        else
        {
            Bf0 = Bf;
            i++;
        }
    }
    return -1e12;
}
```

反算的实现如下：

```
void CEarth::GsFs(double x, double y,
                  double & l, double & B)
{
    double Bf0 = Bf( x );
    double cosBf = cos( Bf0 );

    double gf = _eT * cosBf;
    double gf2 = gf * gf;
    double gf4 = gf2 * gf2;

    double tf = tan(Bf0);
    double tf2 = tf * tf;
    double tf4 = tf2 * tf2;

    double Nf = _c / sqrt(1.0 + gf2);
    double Nf2 = Nf * Nf;
    double Nf4 = Nf2 * Nf2;
```

```
    double y2 = y * y;
    double y4 = y2 * y2;

    l = y / (Nf * cosBf) *
    ( 1.0
      - y2 / (6.0 * Nf2) * (1.0 + 2.0 * tf2 + gf2)
          + y4 / (120.0 * Nf4 ) * (5.0
      + 28.0 * tf2
          + 24.0 * tf4
      + 6.0 * gf2
      + 8.0 * gf2 * tf2)
    );
    B = Bf0 - y2 / Nf2 * tf * 0.5 *
      ( (1.0 + gf2)
            - y2 * (5.0
                    + 3.0 * tf2
                    + 6.0 * gf2
                    - 6.0 * tf2 * gf2
                    - 3.0 * gf4
                    + 9.0 * gf4 * tf4
                   ) / (12.0 * Nf2)
            + y4 * (61.0
                    + 90.0 * tf2
                    + 45.0 * tf4
                    + 107.0 * gf2
                    + 162.0 * gf2 * tf2
                    + 45.0 * gf2 * tf4
                   ) / ( 360 * Nf4)
     );
}

//L: 经度（弧度）, B: 纬度（弧度）， L0: 中央子午线经度
void CEarth::GsFs(double x, double y, double L0,
                  double& L, double& B)
{
    double l;
    GsFs(x, y, l, B);
```

```
    L = L0 + l;
}
```

在以上实现中，由于我们的成员变量均为 private 类型，而构造函数为默认的形式，无法将椭球的参数值传递进去。由于要实现 54、80 和自定义椭球的坐标系，相应的解决方法有：

1. 将默认的构造函数改为 CEarth (double a, double alf) 形式，将不同的椭球参数传给类。比如要建立 54 坐标系，只需要如下作即可：

```
CEarth earth54(6378245.0, 298.3);
CEarth * pEarth54 = new CEarth(6378245.0, 298.3);
```

这种形式很直观，但缺点也很明显，对于已知的常用椭球，编程人员每次均要提供其具体的参数值，很不方便，也容易出错。

2. 利用类厂的方法，将生成已知的常用椭球定义为静态成员函数，同时将构造函数声明为 private 类型，防止象方法 1 那样直接访问。这种方法的优点是显而易见的，它的实现如下：

在类 CEarth 中将默认的构造函数的访问域改为：

```
private:
    CEarth();
    void CalEcd(double a, double alf);
```

同时 CalEcd 函数计算椭球内部的基本系数，不需要外部程序访问，也将其定义为 private。

同时在类中声明以静态成员指针变量：

```
private:
    static CEarth * _pEarth;
```

在类的实现文件前面进行初始化：

```
CEarth* CEarth::_pEarth = NULL;
```

再在类中定义用于创建椭球的静态成员函数：

```
public:
    static CEarth * CreateEarth54();
    static CEarth * CreateEarth80();
    static CEarth * CreateEarthCustomize(double a, double alf);
```

它们的实现为：

```
CEarth * CEarth::CreateEarth54()
{
    if(_pEarth == NULL) _pEarth = new CEarth();
    _pEarth->CalEcd(6378245.0, 298.3);
```

```
    return _pEarth;
}
CEarth * CEarth::CreateEarth80()
{
    if(_pEarth == NULL) _pEarth = new CEarth();
    _pEarth->CalEcd(6378140.0, 298.257);
    return _pEarth;
}
CEarth * CEarth::CreateEarthCustomize(double a, double alf)
{
    if(_pEarth == NULL) _pEarth = new CEarth();
    _pEarth = new CEarth();
    _pEarth->CalEcd(a, alf);
    return _pEarth;
}
```

从它们的实现看，是用 new 的方式生成了椭球对象，为了避免内存泄漏，类的析构函数中，负责将申请的内存进行释放，它的实现如下：

```
CEarth::~CEarth()
{
    if(_pEarth != NULL)
    {
        delete _pEarth;
        _pEarth = NULL;
    }
}
```

主函数调用的形式为：

```
void main()
{
    double x, y;
CEarth * pEarth = CEarth::CreateEarth54();
    double B = zx::CSurMath::DmsToRad(21.58470845);
    double l = zx::CSurMath::DmsToRad(113.25314880)
                    - zx::CSurMath::DmsToRad(111);
    pEarth->GsZs( l, B, x, y);
printf("x = %lf, y = %lf \n", x, y);
}
```

现在我们基本上用面向对象的方法完成了高斯投影正反算的问题了。

9.4 实现换带计算和文件读写

9.4.1 换带计算

换带计算的基本方法：已知某一带（中央子午线经度已知 oldL0）的点坐标（oldX, oldY），利用高斯反算计算出大地坐标（B，L），再根据新带的中央子午线经度（newL0），高斯正算计算新带中的高斯平面坐标（newX, newY）。用算法描述如下：

1. $B\ L\ = GsFs(oldX, oldY, oldL0)$
2. $(newX, newY) = GsZs(B, L, newL0)$

用一函数描述它为：

```
void CEarth::GsHd(double oldX, double oldY,
                  double oldL0,  double newL0,
                  double& newX, double& newY)
{
    double B, L;
    GsFs(oldX, oldY, oldL0, L, B);
    GsZs(L, B, newL0, newX, newY);
}
```

注意：高斯换带计算一般是指同一参考椭球的不同带之间的换算，不同的参考椭球的坐标系是不能采用这种方法的。

计算示例如下：

```
void main()
{
    double oldX = 3275110.535;
    double oldY = 235437.233;
    double oldL0 = zx::CSurMath::DmsToRad(117);
    double newL0 = zx::CSurMath::DmsToRad(120);
    double newX, newY;


   CEarth * pEarth = CEarth::CreateEarth54();
   pEarth->GsHd(oldX, oldY, oldL0, newL0, newX, newY);
   printf("newX = %lf,  newY = %lf \n", newX,  newY);
   // 3272782.315, -55299.545
}
```

9.4.2 多点计算和文件读写

以上我们的算法和测试验证程序均是针对一个点而言的，如果我们在一个文本形式的数据文件里存放有多个点，怎么计算呢？

1、我们设计正算的数据文件格式为：

成果文件名

转换点个数 ~ 中央子午线经度

点名 ~ 大地纬度 ~ 大地经度

示例数据文件为：

```
BLXY.txt
3  111
p1  21.58470845  113.25314880
p2  31.04416832  111.47248974
p3  30.45254425  111.17583596
```

则设计函数为：

```
struct PntInfo
{
    char name[10];
    double B, L;
    double x, y;
};
void Zs()
{
    double L0;//所在带的中央子午线经度
    int n;//转换点的个数
    char cg[256];//成果文件名
    FILE *in;
    //读取文本文件的数据
    in = fopen("BLtoXY.txt", "r"); //打开已知文件
    fscanf(in,"%s",cg); //首先读入成果文件名称
    fscanf(in,"%d %lf", &n, &L0); //转换点的个数  中央子午线经度
    PntInfo * pnts = new PntInfo[n];//动态数组
    for(int i = 0; i < n; i++)//循环读入点名B、L
        fscanf(in, "%s %lf %lf", pnts[i].name, &pnts[i].B, &pnts[i].L);
    fclose(in);
    //以下进行正算计算
    CEarth * pEarth = CEarth::CreateEarth54();
    for(i = 0; i < n; i++)
```

```
        pEarth->GsZs(zx::CSurMath::DmsToRad(pnts[i].L),
         zx::CSurMath::DmsToRad(pnts[i].B),
             zx::CSurMath::DmsToRad(L0),
         pnts[i].x, pnts[i].y);
    //将计算后的数据写到成果文件中
    FILE * out;
    out = fopen(cg, "w");
    fprintf(out, "大地坐标(B,L)=====>国家坐标(X,Y)\n ");
    fprintf(out,"中央子午线经度: L0 = %lf\n", L0);
    fprintf(out,"序号 点名 B      L ===> X坐标(m)  Y坐标(m)\n");
    for(i = 0; i < n; i++)
    {
    fprintf(out," %3d %s %lf %lf %11.3f %11.3f\n",
        i+1, pnts[i].name, pnts[i].B, pnts[i].L,
            pnts[i].x, pnts[i].y );
    }
    fclose(out);
    delete[] pnts;//释放申请的内存
}
```

同样也可实现多点的数据文件反算和换带计算

反算的数据文件格式设为：XYBL.txt

```
3   111
p1     2433586.692   250547.403
p2     3439978.970    75412.872
p3     3404139.839    28680.571
```

换带计算的数据文件格式设为：

```
XYBL.txt
3   111 112
p1     2433586.692   250547.403
p2     3439978.970    75412.872
p3     3404139.839    28680.571
```

相应的实现请参照正算实现。

9.5 小结

我们从一个逻辑结构不太明显的程序开始，将其改为一个结构较好的面向过程的程序。并从一个算法开始逐步实现了整个程序。最后用面向对象的方法将其改写并实现了全部功能。最后的程序从结

构上看明显要比第一个程序要好，且更易维护和扩充。当然高斯换带计算较为简单，我们的程序实现同样也较为简单。从这个例子，我们知道了怎样设计结构良好的程序。

第 10 章　坐标转换程序设计

10.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 11 章　线路要素计算程序设计

11.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。

第 12 章　单导线近似平差程序设计

12.1　111111

本章主要讲述 C 语言与 C++ 的不同之处，学习完本章后，应该能够在 Visual C++ 编译器下基本能够正确的编译 C 语言程序和简单的具有 C++ 特征的程序。