1目的

为了保证所每个项目组编写出的程序都符合相同的规范，便于提高代码的可读性和可维护性，便于检查、减少出错概率，有助于项目组成员间交流，保证一致性、统一性而建立的C#程序编码规范。

2范围

该规范适用于所有基于C#开发的.Net平台项目，为详细设计，代码编写和代码审核提供参考和依据。

3规范内容

1、代码格式

1.1、要使所有的缩进为一个Tab，即4个空格，使用VS.NET的默认设置。

1.2、在代码中[垂直对齐](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=66636822&ss_c=ssc.citiao.link" \t "_blank)左括号和右括号。

if (x == 0)

{

Response.Write("用户编号必须输入！");

}

不允许以下情况：

if (x == 0) {

Response.Write("用户编号必须输入！");

}

或者：

if (x == 0){ Response.Write("用户编号必须输入！");}

1.3一行被分为几行时，将串联运算符放在每一行的末尾而不是开头，清楚地表示没有后面的行是不完整的。

1.5、每一行上放置的语句避免超过一条，避免写 int i = 0;int j = 1;

1.6、在大多数运算符[之前和之后](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=142255302&ss_c=ssc.citiao.link" \t "_blank)使用空格，这样做时不会改变代码的意图却可以使代码容易阅读。例：

int j = i + k;

而不应写为

    int j=i+k;

1.7、将大的复杂代码节分为较小的、易于理解的模块。

2、注释规范

注释是为了增加代码的清晰度，保持注释的简洁，不是任何代码都需要注释的，过多的注释反而会影响代码的可读性。

注释不要包括其他的特殊字符，建议先写注释，后写代码，注释和代码一起完成。

如果语句块（比如循环和条件分枝的代码块）代码太长，嵌套太多，则在其结束“｝”要加上注释，标志对应的开始语句。如果分支条件逻辑比较复杂，也要加上注释。

注释规范主要包括：模块（类）注释规范、类属性注释规范、方法注释规范、代码间注释规范。

2.1类属性注释规范

在类的属性必须以以下格式编写属性注释：

/// <summary>

///属性说明

/// </summary>

2.2方法注释规范

在类的方法声明前必须以以下格式编写注释：

/// </summary>

        ///函数的作用

/// <param name="<参数名称>"><参数说明></param>

/// <对方法[返回值](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=581547&ss_c=ssc.citiao.link" \t "_blank)的说明，该说明必须明确说明返回的值代表什么含义>

        ///<exception>异常类型、异常结果说明（为

#endregion

2.4、代码间注释规范

代码间注释分为[单行注释](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=63796783&ss_c=ssc.citiao.link)和多行注释。

  单行注释：

//<单行注释>（不提倡使用，尽量都使用/\*\*/）

  多行注释：

/\*多行注释1

 \*多行注释2

 \*多行注释3\*/

代码中遇到语句块（if，case……）或实现一个逻辑功能时，必须添加注释。添加的注释应该能够说明此语句块的作用和实现手段（所用算法等等）。

3、命名规范

3.1、类变量命名:类变量命名一律用下划线开头，多个单词组成，第一个字母小写的，其他单词第一个字母大写。

public class Hello

{

private string \_firstName;

private DateTime \_date;

}

3.2、类命名规范:

3.2.1、名字应该能够标识事物的特性；

3.2.2、首字母大写，缩略语可以全部字母大写；

3.2.3、名字可以有两个或三个单词组成，但通常不应多于三个；

3.2.4、在名字中，每个单词的第一个字母大写。例如：IsSuperUser；包含ID的，ID全部大写，如CustomerID；

3.2.5、使用名词或名词短语命名类；

3.2.6、少用缩写，除非是被广泛使用的；

3.2.7、不要使用下划线字符 (\_)。

3.3、局部变量命名：由小写字母开头，其他单词开头字母大写的各单词组成。

int  localVar;

3.4、函数命名：一律由大写字母开头的各个单词组成。

public void GetValue(int type)

{

//…

}

3.6[抽象类](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7766757&ss_c=ssc.citiao.link)的命名:一律由大写字母”A”开头,第一个字母大写的各单词组成。

///

///抽象类样例

}

3.7、方法（Method）、属性（Atribute）命名规范

3.7.1、方法命名应采用首字母大写的方式；

3.7.2、返回值是Bool值的方法和属性以Is开头；

3.7.3、使用返回值是Bool值的方法（属性）的时候，不要将bool值方法（属性）与if语句放在同一行，如应该这样写：

bool valid = IsValid();

if (valid)

{

}

不应这样写：

if (IsValid())

{

}

3.14、类的实例命名规范：类的实例第一个字母小写，其他单词的第一个字母大写。

FileStream fileStream=new FileStream();

**5控件命名规则**：控件命名=控件缩写前缀+变量具体释义。

|  |  |  |
| --- | --- | --- |
| 控件名称 | 前缀 | 例子 |
| Button | btn | btnSubmit |
| CheckBox | chk | chkReadOnly |
| ComboBox | cbo | cboEnglish |
| Command | cmd | cmdDelete |
| [Connection](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=99672510&ss_c=ssc.citiao.link) | con | conOracle |
| [DataAdapter](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=59252236&ss_c=ssc.citiao.link) | dad | dadDiction |
| [DataColumn](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=69152893&ss_c=ssc.citiao.link) | dcol | dcolDiction |
| DataGrid | dgd | dgdTitles |
| [DataReader](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=9177376&ss_c=ssc.citiao.link) | dr | drDiction |
| [DataRelation](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=71493373&ss_c=ssc.citiao.link) | drel | drelDiction |
| DataRow | drow | drowDiction |
| DataSet | ds | dsDiction |
| [DataTable](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7835293&ss_c=ssc.citiao.link) | dt | dtDiction |
| DataView | dv | dvDiction |
| DateTimePicker | dtp | dtpPublished |
| [DropDownList](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8750484&ss_c=ssc.citiao.link) | drp | drpOffice |
| Form | frm | frmMain |
| ImageList | ils | ilsAllIcons |
| Label | lbl | lblHelpMessage |
| ListBox | lst | lstPolicyCodes |
| ListView | lv | lvHeadings |
| Menu | mnu | mnuFileOpen |
| MonthCalender | mcd | mcdPeriod |
| Parameter | parm | parmIn |
| PictureBox | pic | picVGA |
| [ProgressBar](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=57159109&ss_c=ssc.citiao.link) | prg | prgLoadFile |
| [RadioButton](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55808598&ss_c=ssc.citiao.link) | rad | radGender |
| RichTextBox | rtb | rtbReport |
| StatusBar | stab | stabDateTime |
| TabControl | tab | tabOptions |
| TextBox | txt | txtLastName |
| ToolBar | tlb | tlbActions |
| TreeView | trv | trvOrganization |
| [ScriptManager](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=2362711&ss_c=ssc.citiao.link) | smg | smgContent |
| UpdatePanel | upd | updQuery |
| Panel | pl | plQuery |
| GridView | gv | gvShow |
| UpdateProgress | upp | uppUpdate |
| MultiView | mulv | mulvContent |
| View | vie | vieAdd |
| FormView | fvi | fviAdd |
| HiddenField | hf | hfMian |
| ObjectDataSource | ods | odsData |
| DynamicPopulateExtender | dpe | dpeText |
| FilteredTextBoxExtender | ftbe | ftbeNum |

**6、调用规范**

6.1、类内变量和函数：类内函数和变量的调用一定要在前面加上”this.”；

6.2、类常量和类[静态变量](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=786565&ss_c=ssc.citiao.link" \t "_blank)，以及[静态方法](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10695808&ss_c=ssc.citiao.link)：类内静态元素(元素包括变量和方法)的调用，一定要在前面加上该类的名字；

6.3、系统类的使用：系统类型的使用，.Net中对于基本数值类型,都有相应的引用类型定义，在一般表示的情况下，我们使用小写字母开头的类型，而用到他们方法和数据的时候，则使用引用空间的名称。比如string 有 System.String对应，int 有System.Int32对应。

//注意string和String的使用地方

string strSample = String.Empty;

//注意int和Int32的使用地方

int intSample = 0;

System.Int32.Equals(intSample,3);

6.4、引用类型转换：我们不要使用类型加在前面的转换方式，而使用as，这样当转换不成功时，也不会有异常，但可以根据转换结果进行判断。

object obj;

不要：

ISample sam = (ISample)obj;

而要：

ISample sam = obj as ISample;

if(null == sam)

{ //转换失败

}

B.1.1  数据类型简写规则

程序中定义常量、变量或方法等内容时，常常需要指定类型。下面介绍一种常见的数据类型简写规则，如表B.1所示。

表B.1  数据类型简写规则

|  |  |  |  |
| --- | --- | --- | --- |
| 数 据 类 型 | 简    写 | 数 据 类 型 | 简    写 |
| 整型 | int | [长整型](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10636914&ss_c=ssc.citiao.link) | lint |
| 字符串 | str | 单精度浮点型 | flt |
| [布尔型](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=54150574&ss_c=ssc.citiao.link) | bl | 双精度浮点型 | dbl |
| 短整型 | sint | 字节型 | bt |

B.1.3  类命名规则

使用完整的单词，避免使用缩写。

一般使用名词。类通常表示一种对象，而对象一般使用具有名词性质的名称。

不要使用C或下划线作为前缀。尽管在Visual Basic和Visual C++等其他语言中，这是非常普遍的，但在.[NET框架](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=584747&ss_c=ssc.citiao.link)中不提倡使用，.NET框架提供的类中没有以C为前缀的类名。

异常类命名通常以Exception结尾，但应尽量避免自定义异常类。

避免将多个类放在一个文件中。

初始化类的实例时，除非十分必要，否则不要赋值为null。

一个类文件里应该有且仅有一个命名空间，避免在一个类文件里包含多个不同的命名空间。

避免在一个类文件里代码超过500行（除去自动生成的代码）。

B.1.7  控件命名规则

所有的控件名称都应该为自然名称的拼音简写，出现冲突可采用不同的简写规则。另外，在编码过程中涉及不到编码的控件，其名称可以取默认名称。控件命名规则如表B.3所示。

表B.3  控件命名规则

|  |  |
| --- | --- |
| 控    件 | 缩 写 形 式 |
| Form | frm |
| TextBox | txt |
| Button | btn |
| ComboBox | cbox |
| Label | lab |
| [DataGridView](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=29490967&ss_c=ssc.citiao.link) | dgv |
| ListBox | lb |
| Timer | tmr |
| CheckBox | ckbox |
| RichTextBox | rtbox |
| CheckedListBox | clbox |
| RadioButton | rbtn |
| NumericUpDown | nudown |
| Panel | pl |
| GroupBox | gbox |
| ListView | lv |
| TreeView | tv |
| PictrueBox | pbox |
| …… | …… |

B.2  代码版式

B.2.1  统一代码缩进

每个项目都应该根据实际情况建立统一的代码缩进标准，比如统一缩进两个字符，这样代码看起来非常美观，而且层次感很清楚，方便代码的阅读。

B.2.2  合理使用空行

在自动生成的命名空间和手动添加的命名空间之间要加空行分隔。

在每个类声明之后、每个方法声明之后都要加空行。

在一个方法体内，逻辑上密切相关的语句之间不加空行，其他地方应该加空行分隔。

B.2.3  代码换行

一行代码只做一件事情，如只定义一个变量，或只编写一条语句。

if、switch、for、foreach、while及do…while等语句独占一行，执行语句不能紧跟其后；而且，不论执行语句有多少都应该加{}。

代码行最大长度应该控制在70～80个字符以内，代码不要过长，否则不便于查看。

长表达式要在低优先级运算符处拆分成新行，运算符放在新行之首（以便突出运算符）。拆分的新行要适当缩进，以便使排版整齐。

B.2.4  空格的使用

关键字后面应该加空格，否则无法解析关键字。

if、for、while等控制语句关键字后面应该加空格，然后再跟左括号"("，以便突出关键字。

方法名后面不要加空格，而应该紧跟"("，以便与关键字进行区分。

"("向后紧跟，")"、","、";"向前紧跟，紧跟处不应该留空格。

[赋值运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55257294&ss_c=ssc.citiao.link)、[比较运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8011293&ss_c=ssc.citiao.link)、[算术运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=56555579&ss_c=ssc.citiao.link)、[逻辑运算符](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=54086220&ss_c=ssc.citiao.link)和位运算符等二元运算符的前后应该加空格。

一元运算符如"!"、"++"、"--"、"&"等的前后不加空格。

类似"[]"、"."、"->"这类符号的前后不加空格。

对于表达式比较长的if语句和for语句，为了看上去更加紧凑，可以适当地去掉一些空格。

B.3  代码注释

B.3.1  注释的目的

文字说明代码的作用（即为什么要编写该代码，而不是如何编写）。

指出该代码的编写思路和逻辑方法。

代码中的重要转折点。

使代码的阅读者不必在他们的头脑中仿真运行代码的执行方法。

B.3.2  代码注释规范

方法体内使用类似"//"的形式进行注释，前面需要使用空格进行对齐时，使用Tab键替代空格键。

对类文件进行属性注释说明。

避免对浅显易懂的语句进行注释说明。

代码应具有解释其自身作用的功能。好的代码可以通过可读的变量和方法名了解该段代码的作用，而不需要过多的注释。

逻辑性较强的地方加入注释，说明这段程序的逻辑是怎样的，以方便其他人理解，并且还可以在一定程度上排除Bug。

使用注释来说明何时可能出错和为什么出错。

在编写代码前进行注释。

纯色字符注释行只用于主要注释。

避免形成注释框。用星号围成的注释框，右边的星号看起来很好，但却给编写或编辑注释的人增加了许多工作。

增强注释的可读性。注释是供人阅读的，而不是让计算机阅读，所以应该使用完整的语句。

对注释进行缩进，使之与后随的语句对齐。注释通常位于它们要说明的代码的前面，为了从视觉上突出注释与代码之间的关系，可将注释缩进，使之与代码处于同一个层次上。

在每个if语句的前面加上注释。

在每个switch语句的前面加上注释。

在每个循环语句的前面加上注释。

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.OleDb;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

namespace \_20170643\_贾金长4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void toolStripMenuItem1\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void excel文件ToolStripMenuItem\_Click(object sender, EventArgs e)

{

}

private void 数据导入ToolStripMenuItem\_Click(object sender, EventArgs e)

{

dataGridView1.DataSource = null;

dataGridView1.Rows.Clear();

dataGridView1.Columns.Clear();

OpenFileDialog file = new OpenFileDialog();

file.Filter = "Excel 文件 |\*.xls|Excel 文件 |\*.xlsx";

if (file.ShowDialog() == DialogResult.OK)

{

string fname = file.FileName;

string strSource = "provider=Microsoft.ACE.OLEDB.12.0;" + "Data Source=" + fname + ";Extended Properties='Excel 8.0;HDR=Yes;IMEX=1'";

OleDbConnection conn = new OleDbConnection(strSource);

string sqlstring = "SELECT \* FROM [Sheet1$]";

OleDbDataAdapter adapter = new OleDbDataAdapter(sqlstring, conn);

DataSet da = new DataSet();

adapter.Fill(da);

dataGridView1.DataSource = da.Tables[0];

}

else

return;

}

//坐标方位角推算

public double fangweijiao(double[] sdr, double[] cr)

{

double sum = 0;

for (int i = 1; i < sdr.Length; i++) // 从第二行开始循环计算坐标方位角、观测角度累加值

{

cr[i] = cr[i - 1] + sdr[i] - Math.PI; // 计算坐标方位角/ / 左角

if (cr[i] >= Math.PI \* 2) // 判断坐标方位角是否在 0 到 2PI 之间

cr[i] -= Math.PI \* 2;

else if (cr[i] < 0.0)

cr[i] += Math.PI \* 2;

sum += sdr[i];

}

return sum;

}

//角度转弧度

public double dmstorad(String s)

{

String[] ss = s.Split(new char[3] { '°', '′', '″' }, StringSplitOptions.RemoveEmptyEntries);

double[] d = new double[ss.Length];

for (int i = 0; i < d.Length; i++)

d[i] = Convert.ToDouble(ss[i]);

double sign = d[0] >= 0.0 ? 1.0 : -1.0;

double rad = 0;

if (d.Length == 1)

rad = (Math.Abs(d[0])) \* Math.PI / 180;

else if (d.Length == 2)

rad = ((Math.Abs(d[0])) + d[1] / 60) \* Math.PI / 180;

else

rad = ((Math.Abs(d[0])) + d[1] / 60 + d[2] / 60 / 60) \* Math.PI / 180;

rad = sign \* rad;

return rad;

}

//弧度转角度

public String radtodms(double rad)

{

double sign = rad >= 0.0 ? 1.0 : -1.0;

rad = Math.Abs(rad) \* 180 / Math.PI;

double[] d = new double[3];

d[0] = (int)rad;

d[1] = (int)((rad - d[0]) \* 60);

d[2] = (rad - d[0] - d[1] / 60) \* 60 \* 60;

d[2] = Math.Round(d[2], 4);

if (d[2] == 60)

{

d[1] += 1;

d[2] -= 60;

if (d[1] == 60)

{

d[0] += 1;

d[1] -= 60;

}

}

d[0] = sign \* d[0];

String s = Convert.ToString(d[0]) + "°" + Convert.ToString(d[1]) + "′" + Convert.ToString(d[2]) + "″";

return s;

}

private void button1\_Click(object sender, EventArgs e)

{

string[] sd = new string[dataGridView1.RowCount - 5];

double[] sdr = new double[sd.Length];

double[] cr = new double[sd.Length];

double sum = 0;

cr[0] = dmstorad(Convert.ToString(dataGridView1.Rows[0].Cells[4].Value));

double acd = dmstorad(Convert.ToString(dataGridView1.Rows[dataGridView1.RowCount - 6].Cells[4].Value));

for (int i = 1; i < sd.Length; i++)

{

sd[i] = Convert.ToString(dataGridView1.Rows[i].Cells[1].Value);

sdr[i] = dmstorad(sd[i]);

}

sum = fangweijiao(sdr, cr);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[1].Value = radtodms(sum);

double fd, fdx;

fd = cr[cr.Length - 1] - acd;

fdx = 60 \* Math.Sqrt(sd.Length - 1);//计算角度闭合差限差，单位秒

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[1].Value = Convert.ToString(Math.Round(fd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度闭合差存入表格中

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[1].Value = Convert.ToString(Math.Round(fdx, 2)) + "″";//将角度

if (Math.Abs(fd \* 180 / Math.PI \* 3600) > fdx)

MessageBox.Show("角度闭合差超限！");

else

{

double vd = -fd / (sd.Length - 1);

double sumvd = 0;

for (int i = 1; i < sdr.Length; i++)

{

sdr[i] += vd;

sumvd += vd;

dataGridView1.Rows[i].Cells[2].Value = Convert.ToString(Math.Round(vd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数存入表格中

dataGridView1.Rows[i].Cells[3].Value = radtodms(sdr[i]);

}

if (Math.Round(sumvd, 8) != Math.Round(-fd, 8))

MessageBox.Show("角度改正数分配有误！");

else

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[2].Value = Convert.ToString(Math.Round(sumvd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数总和存入表格中

sum = fangweijiao(sdr, cr);//推算改正后的坐标方位角

if (Math.Round(cr[cr.Length - 1], 8) != Math.Round(acd, 8))

MessageBox.Show("坐标方位角推算有误！");

else

{

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[3].Value =

radtodms(sum); //将改正后观测角度总和放入表格中

for (int i = 1; i < cr.Length - 1; i++)

dataGridView1.Rows[i].Cells[4].Value = radtodms(cr[i]);

}

}

double x2, y2, x3, y3; //存放已知两个点的x，y坐标

x2 = Convert.ToDouble(dataGridView1.Rows[1].Cells[12].Value);

y2 = Convert.ToDouble(dataGridView1.Rows[1].Cells[13].Value);

x3 = Convert.ToDouble(dataGridView1.Rows[sd.Length - 1].Cells[12].Value);

y3 = Convert.ToDouble(dataGridView1.Rows[sd.Length - 1].Cells[13].Value);

double[] sl = new double[sd.Length - 1]; //存放观测距离

double[] dx = new double[sl.Length]; //存放坐标增量

double[] dy = new double[sl.Length];

double suml = 0, sumdx = 0, sumdy = 0;

for (int i = 1; i < sl.Length; i++)

{

sl[i] = Convert.ToDouble(dataGridView1.Rows[i].Cells[5].Value);

//将观测距离放到sl数组中

suml += sl[i]; //计算距离总和

dx[i] = sl[i] \* Math.Cos(cr[i]); //利用距离和坐标方位角计算坐标增量

dy[i] = sl[i] \* Math.Sin(cr[i]);

sumdx += dx[i]; //计算坐标增量总和

sumdy += dy[i];

}

double fx, fy, fxy, k1;

fx = sumdx - (x3 - x2); //计算坐标增量闭合差

fy = sumdy - (y3 - y2);

fxy = Math.Sqrt(fx \* fx + fy \* fy); //计算导线全长闭合差

k1 = suml / fxy; //计算导线全长相对闭合差分母

double[] vx = new double[sl.Length]; //定义数组用于存放坐标增量的改正数及总和

double[] vy = new double[sl.Length];

double sumvx = 0, sumvy = 0;

double[] cx = new double[sl.Length]; //定义数组用于存放改正后的坐标增量及总和

double[] cy = new double[sl.Length];

double sumcx = 0, sumcy = 0;

double[] x = new double[sl.Length + 1]; //定义数组用于存放x，y坐标

double[] y = new double[sl.Length + 1];

x[1] = x2;

y[1] = y2;

if (k1 < 2000) //判断导线全长相对闭合差是否超限

MessageBox.Show("导线全长相对闭合差超限！");

else

{

for (int i = 1; i < vx.Length; i++)

{

vx[i] = -fx \* sl[i] / suml; //计算坐标增量改正数

vy[i] = -fy \* sl[i] / suml;

sumvx += vx[i]; //计算坐标增量改正数总和

sumvy += vy[i];

}

if (Math.Round(sumvx, 4) != Math.Round(-fx, 4)

|| Math.Round(sumvy, 4) != Math.Round(-fy, 4))

MessageBox.Show("坐标增量分配有误！");

else

{

for (int i = 1; i < vx.Length; i++)

{

cx[i] = dx[i] + vx[i]; //计算改正后坐标增量

cy[i] = dy[i] + vy[i];

sumcx += cx[i]; //计算改正后坐标增量总和

sumcy += cy[i];

}

if (Math.Round(sumcx, 4) != Math.Round(x3 - x2, 4)

|| Math.Round(sumcy, 4) != Math.Round(y3 - y2, 4))

MessageBox.Show("改正后的坐标增量计算有误");

else

{

for (int i = 2; i < x.Length; i++)

{

x[i] = x[i - 1] + cx[i - 1]; //计算x,y坐标

y[i] = y[i - 1] + cy[i - 1];

}

if (Math.Round(x[x.Length - 1], 4) != Math.Round(x3, 4)

|| Math.Round(y[y.Length - 1], 4) != Math.Round(y3, 4))

MessageBox.Show("坐标计算有误！");

else

{

for (int i = 1; i < sl.Length; i++)

{

dataGridView1.Rows[i].Cells[6].Value = Convert.ToString(Math.Round(dx[i], 4));

//将坐标增量放入表格

dataGridView1.Rows[i].Cells[7].Value = Convert.ToString(Math.Round(dy[i], 4));

dataGridView1.Rows[i].Cells[8].Value = Convert.ToString(Math.Round(vx[i], 4));

//将坐标增量改正数放入表格

dataGridView1.Rows[i].Cells[9].Value = Convert.ToString(Math.Round(vy[i], 4));

dataGridView1.Rows[i].Cells[10].Value = Convert.ToString(Math.Round(cx[i], 4));

dataGridView1.Rows[i].Cells[11].Value = Convert.ToString(Math.Round(cy[i], 4));

//将改正后坐标增量放入表格

dataGridView1.Rows[i].Cells[12].Value = Convert.ToString(Math.Round(x[i], 3));

dataGridView1.Rows[i].Cells[13].Value = Convert.ToString(Math.Round(y[i], 3));

//将x,y坐标放入表格

}

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[5].Value =

Convert.ToString(Math.Round(suml, 4));

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[6].Value =

Convert.ToString(Math.Round(sumdx, 4));

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[7].Value =

Convert.ToString(Math.Round(sumdy, 4));

//将距离总和、坐标增量总和放入表格中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[8].Value =

Convert.ToString(Math.Round(sumvx, 4));

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[9].Value =

Convert.ToString(Math.Round(sumvy, 4));

//将坐标增量改正数总和、改正后坐标增量总和放入表格中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[10].Value =

Convert.ToString(Math.Round(sumcx, 4));

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[11].Value =

Convert.ToString(Math.Round(sumcy, 4));

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[7].Value =

Convert.ToString(Math.Round(fx, 4));

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[7].Value =

Convert.ToString(Math.Round(fy, 4));

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[10].Value =

Convert.ToString(Math.Round(fxy, 4));

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[11].Value =

Convert.ToString((int)k1); //导线全长相对闭合差分母取整

}

}

}

}

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

}

}