//r1

[TestClass]

public class UnitTestPoint

{

[TestMethod]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void TestMethodCtor\_ThrowsArgumentOutOfRangeException()

{

Point p;

double a = -1000.0;

double b = -1000.0;

p = new Point(a, b);

}

}

public class Point

{

public Point(double lat, double longit)

{

}

}

//g1

using System;

namespace GeographicLibrary

{

public class Point

{

public Point(double lat, double longit)

{

throw new ArgumentOutOfRangeException();

}

}

}

//r2

[TestMethod]

public void TestMethodCtor\_CteatePoint()

{

Point p;

double a = 0.0;

double b = 0.0;

p = new Point(a, b);

Assert.IsNotNull(p);

}

//g2

double lat;

double longit;

public Point(double lat, double longit)

{

if (Math.Abs(lat) > 90 || Math.Abs(longit) > 180)

{

throw new ArgumentOutOfRangeException();

}

this.lat = lat;

this.longit = longit;

}

//ref2

using System;

namespace GeographicLibrary

{

public class Point

{

double latitude;

double longitude;

public double Longitude

{

get => longitude;

set

{

if (Math.Abs(value) > 180)

{

throw new ArgumentOutOfRangeException();

}

longitude = value;

}

}

public double Latitude

{

get => latitude;

set

{

if (Math.Abs(value) > 90)

{

throw new ArgumentOutOfRangeException();

}

latitude = value;

}

}

public Point(double latitude, double longitude)

{

if (Math.Abs(latitude) > 90 || Math.Abs(longitude) > 180)

{

throw new ArgumentOutOfRangeException();

}

this.Latitude = latitude;

this.Longitude = longitude;

}

}

}

//r3

using System;

using GeographicLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace GeographicLibraryTest

{

[TestClass]

public class UnitTestPoint

{

[TestMethod]

[DataRow(-1000, 1000)]

[DataRow(1000, 1000)]

[DataRow(91, 179)]

[DataRow(-91, 179)]

[DataRow(89, 181)]

[DataRow(-89, -181)]

[ExpectedException(typeof(ArgumentOutOfRangeException))]

public void TestMethodCtor\_ThrowsArgumentOutOfRangeException(double a, double b)

{

Point p;

p = new Point(a, b);

}

[TestMethod]

[DataRow(89, 179)]

[DataRow(-89.999999999999999999999999999999, 179)]

[DataRow(89, 179.9999999999999999999999999)]

[DataRow(-89, -179.9999999999999999999999999)]

[DataRow(0.0, 179.999999)]

[DataRow(-89.53465489347548957894, 179.48902378947238472389479823)]

[DataRow(89, 178)]

[DataRow(-89.6048923842389948923753452347, 0)]

public void TestMethodCtor\_CteatePoint(double a, double b)

{

Point p;

p = new Point(a, b);

Assert.IsNotNull(p);

}

}

}

//r4

using GeographicLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Collections.Generic;

namespace GeographicLibraryTest

{

[TestClass]

class AzimutTests

{

[TestMethod]

[DynamicData(nameof(GetPoints), DynamicDataSourceType.Method)]

public void Test\_CreateArc\_CreatedArcNotNull(Point a, Point b)

{

Arc arc = new Arc(a, b);

Assert.IsNotNull(arc);

}

public static IEnumerable<List<Point>> GetPoints()

{

yield return new List<Point>() { new Point(0.0, 180.0), new Point(0.0, 90.0) };

yield return new List<Point>() { new Point(0.0, -90.0), new Point(0.0, 90.0) };

}

}

}

//g4

namespace GeographicLibrary

{

public class Arc

{

public Arc(Point p1, Point p2)

{

}

}

}

//ref4

namespace GeographicLibrary

{

public class Arc

{

Point p1;

Point p2;

public Arc(Point p1, Point p2)

{

this.p1 = p1;

this.p2 = p2;

}

}

}

//r5

public static IEnumerable<object[]> GetPointsForCalculating()

{

yield return new object[] { new Point(0.0, 90.0), new Point(0.0, 180.0), 90.0 };

}

[TestMethod]

[DynamicData(nameof(GetPointsForCalculating), DynamicDataSourceType.Method)]

public void Test\_ValidData\_AzimutCalculated(Point a, Point b, double ans)

{

Arc arc = new Arc(a, b);

double az = arc.GetAzimut();

Assert.AreEqual(ans, az);

}

}

//g5

namespace GeographicLibrary

{

public class Arc

{

Point p1;

Point p2;

public Arc(Point p1, Point p2)

{

this.p1 = new Point(p1.Latitude, p1.Longitude);

this.p2 = new Point(p2.Latitude, p2.Longitude);

}

public double GetAzimut()

{

return 90.0;

}

}

}

//r6

private const double Epsilon = 0.0001d;

public static IEnumerable<object[]> GetPointsForCalculating()

{

yield return new object[] { new Point(77.1539, -139.398), new Point(-77.1804, -139.55), 180.077867811 };

yield return new object[] { new Point(0, 90), new Point(-0, 180), 90.0 };

yield return new object[] { new Point(77.1539, 120.398), new Point(77.1804, 129.55), 84.7925159033 };

yield return new object[] { new Point(77.1539, -120.398), new Point(77.1804, 129.55), 324.384112704 };

}

[TestMethod]

[DynamicData(nameof(GetPointsForCalculating), DynamicDataSourceType.Method)]

public void Test\_ValidData\_AzimutCalculated(Point a, Point b, double ans)

{

Arc arc = new Arc(a, b);

double az = arc.GetAzimut();

Assert.IsTrue(Math.Abs(ans - az) < Epsilon);

}

//g6

public class Arc

{

Point p1;

Point p2;

const double ConvToRadiansCoef = Math.PI / 180.0;

private const double Radius = 6372795;

public Arc(Point p1, Point p2)

{

this.p1 = new Point(p1.Latitude, p1.Longitude);

this.p2 = new Point(p2.Latitude, p2.Longitude);

this.p1.Latitude \*= ConvToRadiansCoef;

this.p2.Latitude \*= ConvToRadiansCoef;

this.p1.Longitude \*= ConvToRadiansCoef;

this.p2.Longitude \*= ConvToRadiansCoef;

}

public double GetAzimut()

{

var cl1 = Math.Cos(this.p1.Latitude);

var cl2 = Math.Cos(this.p2.Latitude);

var sl1 = Math.Sin(this.p1.Latitude);

var sl2 = Math.Sin(this.p2.Latitude);

var delta = this.p2.Longitude - this.p1.Longitude;

var cdelta = Math.Cos(delta);

var sdelta = Math.Sin(delta);

var numerator = Math.Sqrt(

Math.Pow(cl2 \* sdelta, 2) +

Math.Pow(cl1 \* sl2 - sl1 \* cl2 \* cdelta, 2));

// Вычисление длины большого круга и расстояния

var denumerator = sl1 \* sl2 + cl1 \* cl2 \* cdelta;

var ad = Math.Atan2(numerator, denumerator);

var dist = ad \* Radius;

// Вычисление начального азимута

var x = cl1 \* sl2 - sl1 \* cl2 \* cdelta;

var y = sdelta \* cl2;

var z = Math.Atan(-y / x) \* (1 / ConvToRadiansCoef);

if (x < 0)

{

z += 180;

}

var z2 = (z + 180) % 360 - 180;

z2 = -z2 \* ConvToRadiansCoef;

var anglerad = z2 - 2 \* Math.PI \* Math.Floor(z2 / (2 \* Math.PI));

var angledeg = anglerad \* (1 / ConvToRadiansCoef);

return angledeg;

}

}

//r7 (тут замена возвращаемого значения с числа на экземпляр класс, где есть None и Any) и проверка одной и той же точки, т.е. нужно созать класс со значенями None и Any

using GeographicLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

namespace GeographicLibraryTest

{

[TestClass]

public class AzimutTests

{

private const double Epsilon = 0.0001d;

[TestMethod]

[DynamicData(nameof(GetPoints), DynamicDataSourceType.Method)]

public void Test\_CreateArc\_CreatedArcNotNull(Point a, Point b)

{

Arc arc = new Arc(a, b);

Assert.IsNotNull(arc);

}

public static IEnumerable<object[]> GetPoints()

{

yield return new object[] { new Point(0.0, 90.0), new Point(0.0, 180.0)};

}

public static IEnumerable<object[]> GetPointsForCalculating()

{

yield return new object[] { new Point(77.1539, -139.398), new Point(-77.1804, -139.55), 180.077867811 };

yield return new object[] { new Point(0, 90), new Point(-0, 180), 90.0 };

yield return new object[] { new Point(77.1539, 120.398), new Point(77.1804, 129.55), 84.7925159033 };

yield return new object[] { new Point(77.1539, -120.398), new Point(77.1804, 129.55), 324.384112704 };

yield return new object[] { new Point(90, 0), new Point(-90, 0), 180.0 };

}

[TestMethod]

[DynamicData(nameof(GetPointsForCalculating), DynamicDataSourceType.Method)]

public void Test\_ValidData\_AzimutCalculated(Point a, Point b, double ans)

{

Arc arc = new Arc(a, b);

GeoInfo az = arc.GetAzimut();

Assert.IsTrue(Math.Abs(ans - az.AzimutValue) < Epsilon);

}

public static IEnumerable<object[]> GetSamePoints()

{

yield return new object[] { new Point(77.1539, -139.398), new Point(77.1539, -139.398)};

yield return new object[] { new Point(0, 90), new Point(0, 90) };

yield return new object[] { new Point(77.1539, 120.398), new Point(77.1539, 120.398) };

}

[TestMethod]

[DynamicData(nameof(GetSamePoints), DynamicDataSourceType.Method)]

public void Test\_SamePoints\_AzimutNone(Point a, Point b)

{

Arc arc = new Arc(a, b);

GeoInfo az = arc.GetAzimut();

Assert.IsTrue(az.AzimutStatus == AzimutStatus.None);

}

}

}

}

//g7

namespace GeographicLibrary

{

public enum AzimutStatus

{

Any = -2,

None = -1,

Defined = 0,

}

}

namespace GeographicLibrary

{

public class GeoInfo

{

public AzimutStatus AzimutStatus { get; private set; }

public double AzimutValue { get; private set; }

public GeoInfo(AzimutStatus azimutStatus, double azimutValue)

{

AzimutStatus = azimutStatus;

AzimutValue = azimutValue;

}

}

}

using System;

namespace GeographicLibrary

{

public class Arc

{

Point p1;

Point p2;

const double ConvToRadiansCoef = Math.PI / 180.0;

private const double Radius = 6372795;

private const double Epsilon = 0.0001d;

public Arc(Point p1, Point p2)

{

this.p1 = new Point(p1.Latitude, p1.Longitude);

this.p2 = new Point(p2.Latitude, p2.Longitude);

this.p1.Latitude \*= ConvToRadiansCoef;

this.p2.Latitude \*= ConvToRadiansCoef;

this.p1.Longitude \*= ConvToRadiansCoef;

this.p2.Longitude \*= ConvToRadiansCoef;

}

public GeoInfo GetAzimut()

{

if (IsEqual(p1.Latitude, p2.Latitude)

&& IsEqual(p2.Longitude, p1.Longitude))

{

return new GeoInfo(AzimutStatus.None, -1);

}

var cl1 = Math.Cos(this.p1.Latitude);

var cl2 = Math.Cos(this.p2.Latitude);

var sl1 = Math.Sin(this.p1.Latitude);

var sl2 = Math.Sin(this.p2.Latitude);

var delta = this.p2.Longitude - this.p1.Longitude;

var cdelta = Math.Cos(delta);

var sdelta = Math.Sin(delta);

var numerator = Math.Sqrt(

Math.Pow(cl2 \* sdelta, 2) +

Math.Pow(cl1 \* sl2 - sl1 \* cl2 \* cdelta, 2));

// Вычисление длины большого круга и расстояния

var denumerator = sl1 \* sl2 + cl1 \* cl2 \* cdelta;

var ad = Math.Atan2(numerator, denumerator);

var dist = ad \* Radius;

// Вычисление начального азимута

var x = cl1 \* sl2 - sl1 \* cl2 \* cdelta;

var y = sdelta \* cl2;

var z = Math.Atan(-y / x) \* (1 / ConvToRadiansCoef);

if (x < 0)

{

z += 180;

}

var z2 = (z + 180) % 360 - 180;

z2 = -z2 \* ConvToRadiansCoef;

var anglerad = z2 - 2 \* Math.PI \* Math.Floor(z2 / (2 \* Math.PI));

var angledeg = anglerad \* (1 / ConvToRadiansCoef);

var geoInfo = new GeoInfo(AzimutStatus.Defined, angledeg);

return geoInfo;

}

private bool IsEqual(double a, double b)

{

return Math.Abs(a - b) < Epsilon;

}

}

}

//r8

public static IEnumerable<object[]> GetPointsTwoOnPolar()

{

yield return new object[] { new Point(90, 90), new Point(90, 180)};

yield return new object[] { new Point(-90, 0), new Point(-90, 10) };

}

[TestMethod]

[DynamicData(nameof(GetPointsTwoOnPolar), DynamicDataSourceType.Method)]

public void Test\_PointsTwoOnPolar\_AzimutNone(Point a, Point b)

{

Arc arc = new Arc(a, b);

GeoInfo az = arc.GetAzimut();

Assert.IsTrue(az.AzimutStatus == AzimutStatus.None);

}

//g8

public GeoInfo GetAzimut()

{

if (IsEqual(p1.Latitude, p2.Latitude)

&& IsEqual(p2.Longitude, p1.Longitude))

{

return new GeoInfo(AzimutStatus.None, -1);

}

var polarDegree = 90 \* ConvToRadiansCoef;

if (IsEqual(p1.Latitude, p2.Latitude) &&

IsEqual(Math.Abs(p1.Latitude), polarDegree)

&& IsEqual(Math.Abs(p2.Latitude), polarDegree))

{

return new GeoInfo(AzimutStatus.None, -1);

}

var cl1 = Math.Cos(this.p1.Latitude);

var cl2 = Math.Cos(this.p2.Latitude);

var sl1 = Math.Sin(this.p1.Latitude);

var sl2 = Math.Sin(this.p2.Latitude);

var delta = this.p2.Longitude - this.p1.Longitude;

var cdelta = Math.Cos(delta);

var sdelta = Math.Sin(delta);

var numerator = Math.Sqrt(

Math.Pow(cl2 \* sdelta, 2) +

Math.Pow(cl1 \* sl2 - sl1 \* cl2 \* cdelta, 2));

// Вычисление длины большого круга и расстояния

var denumerator = sl1 \* sl2 + cl1 \* cl2 \* cdelta;

var ad = Math.Atan2(numerator, denumerator);

var dist = ad \* Radius;

// Вычисление начального азимута

var x = cl1 \* sl2 - sl1 \* cl2 \* cdelta;

var y = sdelta \* cl2;

var z = Math.Atan(-y / x) \* (1 / ConvToRadiansCoef);

if (x < 0)

{

z += 180;

}

var z2 = (z + 180) % 360 - 180;

z2 = -z2 \* ConvToRadiansCoef;

var anglerad = z2 - 2 \* Math.PI \* Math.Floor(z2 / (2 \* Math.PI));

var angledeg = anglerad \* (1 / ConvToRadiansCoef);

var geoInfo = new GeoInfo(AzimutStatus.Defined, angledeg);

return geoInfo;

}

//r9

public static IEnumerable<object[]> GetPointsOneOnPolar()

{

yield return new object[] { new Point(90, 90), new Point(70, 180) };

yield return new object[] { new Point(-90, 0), new Point(-70, 10) };

yield return new object[] { new Point(70, 180), new Point(90, 90) };

yield return new object[] { new Point(-70, 10), new Point(-90, 0) };

}

[TestMethod]

[DynamicData(nameof(GetPointsOneOnPolar), DynamicDataSourceType.Method)]

public void Test\_PointsOneOnPolar\_AzimutNone(Point a, Point b)

{

var ans = 180;

Arc arc = new Arc(a, b);

GeoInfo az = arc.GetAzimut();

Assert.IsTrue(Math.Abs(ans - az.AzimutValue) < Epsilon);

}

//g9

public GeoInfo GetAzimut()

{

if (IsEqual(p1.Latitude, p2.Latitude)

&& IsEqual(p2.Longitude, p1.Longitude))

{

return new GeoInfo(AzimutStatus.None, -1);

}

var polarDegree = 90 \* ConvToRadiansCoef;

if (IsEqual(p1.Latitude, p2.Latitude) &&

IsEqual(Math.Abs(p1.Latitude), polarDegree)

&& IsEqual(Math.Abs(p2.Latitude), polarDegree))

{

return new GeoInfo(AzimutStatus.None, -1);

}

if (!IsEqual(Math.Abs(p1.Latitude), Math.Abs(p2.Latitude)) && (

IsEqual(Math.Abs(p1.Latitude), polarDegree)

|| IsEqual(Math.Abs(p2.Latitude), polarDegree)

))

{

return new GeoInfo(AzimutStatus.Defined, 180);

}

var cl1 = Math.Cos(this.p1.Latitude);

var cl2 = Math.Cos(this.p2.Latitude);

var sl1 = Math.Sin(this.p1.Latitude);

var sl2 = Math.Sin(this.p2.Latitude);

var delta = this.p2.Longitude - this.p1.Longitude;

var cdelta = Math.Cos(delta);

var sdelta = Math.Sin(delta);

var numerator = Math.Sqrt(

Math.Pow(cl2 \* sdelta, 2) +

Math.Pow(cl1 \* sl2 - sl1 \* cl2 \* cdelta, 2));

// Вычисление длины большого круга и расстояния

var denumerator = sl1 \* sl2 + cl1 \* cl2 \* cdelta;

var ad = Math.Atan2(numerator, denumerator);

var dist = ad \* Radius;

// Вычисление начального азимута

var x = cl1 \* sl2 - sl1 \* cl2 \* cdelta;

var y = sdelta \* cl2;

var z = Math.Atan(-y / x) \* (1 / ConvToRadiansCoef);

if (x < 0)

{

z += 180;

}

var z2 = (z + 180) % 360 - 180;

z2 = -z2 \* ConvToRadiansCoef;

var anglerad = z2 - 2 \* Math.PI \* Math.Floor(z2 / (2 \* Math.PI));

var angledeg = anglerad \* (1 / ConvToRadiansCoef);

var geoInfo = new GeoInfo(AzimutStatus.Defined, angledeg);

return geoInfo;

}

//r10 добавил два теста для пункта 10 в критериях. Зелёной фазы не надо, т.к. по формула рассчитывается согласно ожиданиями в п. 10.

public static IEnumerable<object[]> GetPointsForCalculating()

{

yield return new object[] { new Point(77.1539, -139.398), new Point(-77.1804, -139.55), 180.077867811 };

yield return new object[] { new Point(0, 90), new Point(0, 180), 90.0 };

yield return new object[] { new Point(77.1539, 120.398), new Point(77.1804, 129.55), 84.7925159033 };

yield return new object[] { new Point(77.1539, -120.398), new Point(77.1804, 129.55), 324.384112704 };

yield return new object[] { new Point(90, 0), new Point(-90, 0), 180.0 };

yield return new object[] { new Point(0, 90), new Point(10, 90), 0.0 };

yield return new object[] { new Point(10, 90), new Point(0, 90), 180.0 };

}

//r11 Точки на противоположный полюсах

public static IEnumerable<object[]> GetPointsOppositeOnPolar()

{

yield return new object[] { new Point(90, 90), new Point(-90, 180) };

yield return new object[] { new Point(-90, 0), new Point(90, 10) };

}

[TestMethod]

[DynamicData(nameof(GetPointsOppositeOnPolar), DynamicDataSourceType.Method)]

public void Test\_PointsOppositeOnPolar\_AzimutNone(Point a, Point b)

{

Arc arc = new Arc(a, b);

GeoInfo az = arc.GetAzimut();

Assert.IsTrue(az.AzimutStatus == AzimutStatus.Any);

}

//g11

public GeoInfo GetAzimut()

{

if (IsEqual(p1.Latitude, p2.Latitude)

&& IsEqual(p2.Longitude, p1.Longitude))

{

return new GeoInfo(AzimutStatus.None, -1);

}

var polarDegree = 90 \* ConvToRadiansCoef;

if (IsEqual(p1.Latitude, p2.Latitude) &&

IsEqual(Math.Abs(p1.Latitude), polarDegree)

&& IsEqual(Math.Abs(p2.Latitude), polarDegree))

{

return new GeoInfo(AzimutStatus.None, -1);

}

if (!IsEqual(Math.Abs(p1.Latitude), Math.Abs(p2.Latitude)) && (

IsEqual(Math.Abs(p1.Latitude), polarDegree)

|| IsEqual(Math.Abs(p2.Latitude), polarDegree)

))

{

return new GeoInfo(AzimutStatus.Defined, 180);

}

if (!IsEqual(p1.Longitude, p2.Longitude) && (

IsEqual(Math.Abs(p1.Latitude), polarDegree)

&& IsEqual(Math.Abs(p2.Latitude), polarDegree)

))

{

return new GeoInfo(AzimutStatus.Any, -1);

}

var cl1 = Math.Cos(this.p1.Latitude);

var cl2 = Math.Cos(this.p2.Latitude);

var sl1 = Math.Sin(this.p1.Latitude);

var sl2 = Math.Sin(this.p2.Latitude);

var delta = this.p2.Longitude - this.p1.Longitude;

var cdelta = Math.Cos(delta);

var sdelta = Math.Sin(delta);

var numerator = Math.Sqrt(

Math.Pow(cl2 \* sdelta, 2) +

Math.Pow(cl1 \* sl2 - sl1 \* cl2 \* cdelta, 2));

// Вычисление длины большого круга и расстояния

var denumerator = sl1 \* sl2 + cl1 \* cl2 \* cdelta;

var ad = Math.Atan2(numerator, denumerator);

var dist = ad \* Radius;

// Вычисление начального азимута

var x = cl1 \* sl2 - sl1 \* cl2 \* cdelta;

var y = sdelta \* cl2;

var z = Math.Atan(-y / x) \* (1 / ConvToRadiansCoef);

if (x < 0)

{

z += 180;

}

var z2 = (z + 180) % 360 - 180;

z2 = -z2 \* ConvToRadiansCoef;

var anglerad = z2 - 2 \* Math.PI \* Math.Floor(z2 / (2 \* Math.PI));

var angledeg = anglerad \* (1 / ConvToRadiansCoef);

var geoInfo = new GeoInfo(AzimutStatus.Defined, angledeg);

return geoInfo;

}

//r12 Расстояние

public static IEnumerable<object[]> GetPointsForCalculatingDistanse()

{

yield return new object[] { new Point(77.1539, -139.398), new Point(-77.1804, -139.55), 17166029 };

yield return new object[] { new Point(77.1539, 120.398), new Point(77.1804, 129.55), 225883 };

yield return new object[] { new Point(77.1539, -120.398), new Point(77.1804, 129.55), 2332669 };

}

[TestMethod]

[DynamicData(nameof(GetPointsOppositeOnPolar), DynamicDataSourceType.Method)]

public void Test\_ValidPoints\_DistanceCalculated(Point a, Point b, int dist)

{

Arc arc = new Arc(a, b);

GeoInfo az = arc.GetAzimut();

Assert.IsTrue(Math.Abs(dist - az.Distance) < Epsilon);

}

//g12

namespace GeographicLibrary

{

public class GeoInfo

{

public AzimutStatus AzimutStatus { get; private set; }

public double AzimutValue { get; private set; }

public int Distance { get; private set; }

public GeoInfo(AzimutStatus azimutStatus, double azimutValue, int distance)

{

AzimutStatus = azimutStatus;

AzimutValue = azimutValue;

Distance = distance;

}

}

}

public GeoInfo GetAzimut()

{

if (IsEqual(p1.Latitude, p2.Latitude)

&& IsEqual(p2.Longitude, p1.Longitude))

{

return new GeoInfo(AzimutStatus.None, -1, -1);

}

var polarDegree = 90 \* ConvToRadiansCoef;

if (IsEqual(p1.Latitude, p2.Latitude) &&

IsEqual(Math.Abs(p1.Latitude), polarDegree)

&& IsEqual(Math.Abs(p2.Latitude), polarDegree))

{

return new GeoInfo(AzimutStatus.None, -1, -1);

}

var cl1 = Math.Cos(this.p1.Latitude);

var cl2 = Math.Cos(this.p2.Latitude);

var sl1 = Math.Sin(this.p1.Latitude);

var sl2 = Math.Sin(this.p2.Latitude);

var delta = this.p2.Longitude - this.p1.Longitude;

var cdelta = Math.Cos(delta);

var sdelta = Math.Sin(delta);

var numerator = Math.Sqrt(

Math.Pow(cl2 \* sdelta, 2) +

Math.Pow(cl1 \* sl2 - sl1 \* cl2 \* cdelta, 2));

// Вычисление длины большого круга и расстояния

var denumerator = sl1 \* sl2 + cl1 \* cl2 \* cdelta;

var ad = Math.Atan2(numerator, denumerator);

var dist = ad \* Radius;

// Вычисление начального азимута

var x = cl1 \* sl2 - sl1 \* cl2 \* cdelta;

var y = sdelta \* cl2;

var z = Math.Atan(-y / x) \* (1 / ConvToRadiansCoef);

if (x < 0)

{

z += 180;

}

var z2 = (z + 180) % 360 - 180;

z2 = -z2 \* ConvToRadiansCoef;

var anglerad = z2 - 2 \* Math.PI \* Math.Floor(z2 / (2 \* Math.PI));

var angledeg = anglerad \* (1 / ConvToRadiansCoef);

if (!IsEqual(Math.Abs(p1.Latitude), Math.Abs(p2.Latitude)) && (

IsEqual(Math.Abs(p1.Latitude), polarDegree)

|| IsEqual(Math.Abs(p2.Latitude), polarDegree)

))

{

return new GeoInfo(AzimutStatus.Defined, 180, (int)Math.Round(dist));

}

if (!IsEqual(p1.Longitude, p2.Longitude) && (

IsEqual(Math.Abs(p1.Latitude), polarDegree)

&& IsEqual(Math.Abs(p2.Latitude), polarDegree)

))

{

return new GeoInfo(AzimutStatus.Any, -1, (int)Math.Round(dist));

}

var geoInfo = new GeoInfo(AzimutStatus.Defined, angledeg, (int) Math.Round(dist));

return geoInfo;

}