Universal (CC0 licensed software)	CORALTECH SURVEY	Inverse Geodetic Problem White Paper A1
Client	**Survey Contractor**	**Document**

INVERSE GEODETIC PROBLEM – White Paper

2022 04 02	A1	Initial Issue	CJA		
Date	**Revision**	**Description of Revision**	**Prepared by**	**Checked**	**Approved**

Universal (CC0 licensed software)	CORALTECH SURVEY	Inverse Geodetic Problem White Paper A1
Client	**Survey Contractor**	**Document**

EXECUTIVE SUMMARY

The inverse geodetic problem is a well-studied problem, and one which has an established solution across both plane and geodetic distances, in which the distance between two known points is obtained (Smith, 2008). Despite this, the ability of the everyday user to batch convert solutions to the inverse geodetic problem is sorely lacking, invariably requiring a high degree of familiarity with mapping software such a QGIS, ArcGIS, or similar. The cost to the user in both time and data can be prohibitive. Further, add-on scripts to these programs generally require skill with coding languages such as Python. The author has produced a batch conversion solution to the inverse geodetic problem, that can be used by the everyday user with little to no introduction to surveying or geodesy, which is provided under CC0 license with additional caveats.

The software developed in this paper can be found online at *https://github.com/cjaddison82/Survey-Resources*

1 Solution - Established

The following equations are used in the solution of the inverse geodetic problem (Allan, 2004).

1.) $L = \lambda = \lambda_2 - \lambda_1$

2.) $tanU1 = (1 - \mathrm{f})\tan\varphi_1$

3.) $tanU2 = (1 - \mathrm{f})\tan\varphi_2$

4.) $S = bA(\sigma - \Delta\sigma)$

Which requires iterations of 4.) until the difference between σ and delta σ becomes negligible. For the solution calculated here, a novel method to solve the same problem is established.

2 Solution – coded for stand-alone batch conversion

The following program is coded in Python 3.6. Firstly, the math module is imported

```
from math import *
```

An output file is created or, if one exists, wiped clean

```
output1 = open("C:/temp/output1.txt", "w+")
```

The input file is opened in read-only mode

```
csv = open("C:/temp/input1.txt", "r")
```

The input file lines are read to a variable called “splitline” and split on comma, so that individual variables are assigned Latitude and Longitude values

```
for line in csv:
```

splitline = line.split(",") #split on comma to array

Lat1 = float(splitline[0])

Long1 = float(splitline[1])

Lat2 = float(splitline[2])

Long2 = float(splitline[3])

The next section will calculate the distance in metres of 1.0000000° of LONGITUDE at the given point of interest. First we calculate the ratio $\lim_{0-1}$ which will be used to multiply by the distance of 1.0000000° at the equator. This will correct for the decreasing distance between each degree of latitude moving from the equator to the pole. Latitude must be in Radians.

d2 = float(cos(radians(float(Lat1))))

We assign to a variable, the length in metres of one degree at the equator

d3 = float(111319.490793)

We calculate the length in metres of one degree at the given latitude. Multiply by the LONG decimal degree difference to get actual metre length

*d4 = d2*d3*

This section finds the distance in metres of LAT at given Degree Latitude. We find the earth's radius at the given Latitude using the WGS84 Spheroid

*d5 = 6378137-((1-d2)*float(21384.68579))*

We find the length in metres of one degree at given latitude

*d6 = float(2*pi*d5)/int(360)*

We take the above variables and calculate the distance in metres between the two points

dx = float(d4(Long1-Long2))*

dy = float(d6(Lat1-Lat2))*

*dxy = float(((dx**2)+(dy**2))**0.5)*

We format the float to a string, truncated to 2 decimal places

dxyf = '{:.2f}'.format(dxy) #formats dxy to 2dp

This step writes to the output file, the original input data with the resultant dxy value appended to the comma separated values

output1.write(str(Lat1)+","+str(Long1)+","+str(Lat2)+","+str(Long2)+","+str(dxyf)+"\r") #write output to output file

Finally, we close input and output files on the original indentation as the opening sentences.

csv.close() #close input file

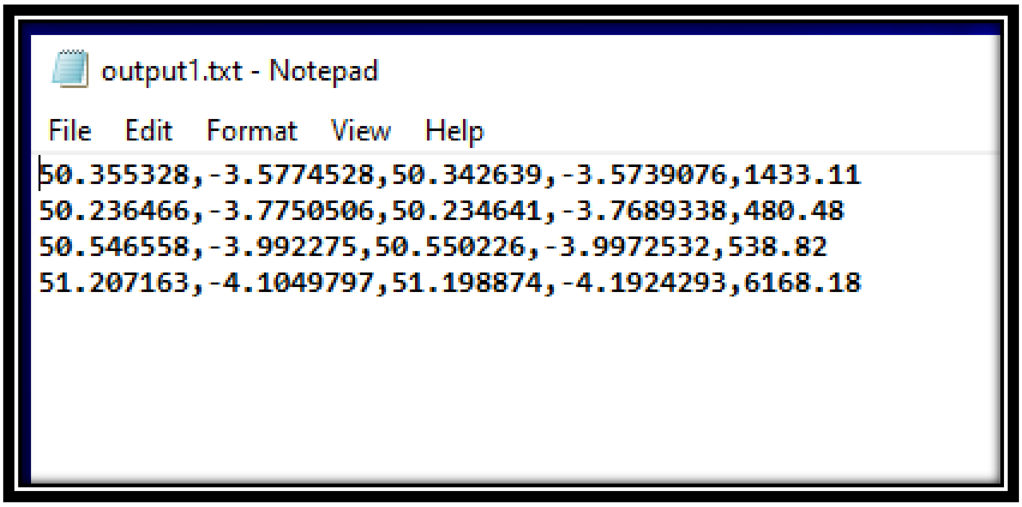
output1.close() #close output file

The .Py has been compiled as a stand-alone .exe, one that simply requires the user to format the data according to README file and place in an appropriate folder prior to running the program (“C:/temp/”).

The program has undergone quality control by way of firstly running the program on dummy data and checking the output file for sense (expected LAT1, LONG1, LAT2, LONG2, output distance between two points) as shown in figure 1.1. Further, the dummy data points were imported into mapping software and the inverse geodetic problem solved on the plane (figure 1.2).

Note that the solution coded here is calculated as geodetic distance, rather than plane, which explains the ~19cm deviation between 1433.11m shown in line 1 of output1.txt, and the 1433.30m shown in the mapping software.

The solution is calculated on the WGS84 spheroid, applicable area -180° to +180° LONG and -90° to +90° LAT (European Petroleum Survey Group, 2007).



output1.txt - Notepad

File Edit Format View Help

```
50.355328,-3.5774528,50.342639,-3.5739076,1433.11
50.236466,-3.7750506,50.234641,-3.7689338,480.48
50.546558,-3.992275,50.550226,-3.9972532,538.82
51.207163,-4.1049797,51.198874,-4.1924293,6168.18
```

Figure 1.1 – QC of program (1)

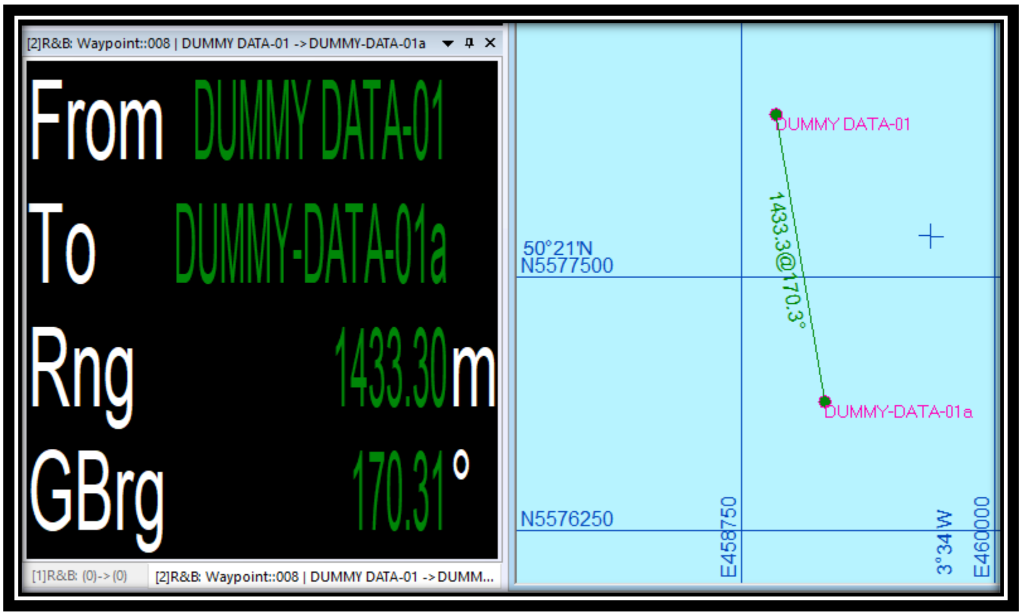


Figure 1.2 – QC of program (2)

3 Geodetic Parameters

SPHEROID	
Spheroid	GRS80
Semi-major axis	6 378 137.298
Inverse flattening	298.257 223 563
EPSG Code	4326

Table 2 – Geodetic Parameters

4 References

Allan A, 2004. *Maths for Map Makers*. Whittles Publishing, 2 Ed.

European Petroleum Survey Group, 2007. *WGS 84 - World Geodetic System 1984.* Online, URL: https://epsg.io/4326. Date accessed 04 Feb 2022.

Smith J, 2008. *Introduction to Geodesy: The History and Concepts of Modern Geodesy.* Wiley Interscience.