

The Programs for *Navigation on the Great Plane*

Introduction

These programs are written in Octave, which in my opinion is the best all-around program for doing mathematical computations. Octave is open-source and available as a free installation on almost all platforms; see octave.org for more information. Most Octave commands and functions -- and this includes those used in the accompanying programs -- are identical with those of the commercial Matlab®. The one exception is that where Octave ends if-loops with `endif` -- and similarly for other types of loops -- Matlab always uses `end`. Thus to use these programs in Matlab simply change `endif` to `end`, etc.

Those unfamiliar with Octave/Matlab can download my *A Guide to Matlab* and *An Introduction to Octave*, that I wrote for my students, at my mathematics web site:

<https://people.math.carleton.ca/~rhfischl/>

The programs described below use the powerful Octave tools for handling data stored in vectors and matrices and in particular conditional vectors. These are discussed in detail in: Sections II.5 ("Matrix Column Operations") and II.8 (Logical Operators).

Input to the Programs

Perhaps you know the coordinates in the degrees-minutes-seconds format for Edinburgh and Vancouver and you need to convert this information to the decimal degrees format via the `[dms2deg.m]` program. After running the latter program, type the following at the Octave prompt:

```
save edinburgh_vancouver_values.m
```

The data will then be available via the command:

```
load edinburgh_vancouver.m
```

The Output of the Programs

1. As described in connection with the program descriptions, some of the main programs automatically save certain variables in Octave's sixteen digit format (this can be modified; see `help save`). For example the program `[great_circle.m]` outputs to `[spherical_results_01.m]` and `[spherical_results_02.m]`. These should be renamed to match the cities, e.g. `[edinburgh_vancouver_results_01.m]` and `[edinburgh_vancouver_results_02.m]` so that they will not be overwritten and will remain available for other computations
2. Because the Octave high precision format can be difficult to read and edit it is suggested that you open a diary:

```
% a first diary for the Edinburgh -- Vancouver computations
diary edinburgh_vancouver_01.dia
% perform the computations
```

diary off % to toggle the diary

Note that in order to facilitate reading the display on the computer screen and in the diary the command `format bank` is invoked in the programs. This displays the answer to two decimal places. For other display possibilities, type `help format`.

Several of the programs were written while developing the package. In some cases part of their code has been incorporated into other programs. For the purposes of testing, I introduced the Octave pause command, as well as other commands. I have now commented them out via the percentage symbol: `%`. Remove the `%` if you want to follow the intermediate steps.

Main Programs

1. `[great_circle.m]`: This is the main program whose input variables are two points P_1 and P_2 . In addition to the computations described in the article it calls other programs, in particular `[bearing.m]` and `[decision.m]` and uses them to generate the various results. It saves the results, in Octave's precision format, in `[spherical_results_01.m]` and `[spherical_results_02.m]`. It automatically prints the graph of the equation of the great circle, i.e. ϕ in terms of θ .
2. `[bearing.m]`: Calculates the initial bearing when going from point P_1 to point P_2 . To find the bearing starting from P_2 , simply reverse the roles of the points in the function call.
3. `[decision.m]` Decides which of the possible bearings at p_1 and p_2 are the correct ones.
4. `[circle_path.m]` The function which calculates ϕ in terms of θ .
5. `[closest_point.m]`: The input is a point p_3 outside the path. The program finds the point on the path that is closest to the p_3 , and determines the distance and bearing from p_3 . It saves the results, in Octave's precision format, in `[coordinates+distance_closest_point.m]`
6. `[bearing_vector.m]`: Calculates the bearing at every degree (this can be modified) on the path between a starting point P_1 and a destination point P_2 . It saves the results in `[bearing_vector_results.m]` and automatically prints the graph of bearing vs. θ .
7. `[distance_vector.m]`: Calculates the distance at every degree (this can be modified) from a starting point P_1 on the path between P_1 and a destination point P_2 . It saves the results in `[distance_vector_results.m]` and automatically prints the graph of bearing vs. θ .

8. [latitude_crossing.m]: Calculates, to any desired degree of accuracy, where the great circle crosses a given latitude.

Angle Conversion Programs

N.B. To avoid having to convert back and forth between radians and degrees the programs employ the trigonometric functions (ending with a “d”) which use the degree format, e.g. sind(30), asind(.5).

9. [dms2deg.m]: Converts degrees, minutes, seconds to decimal degrees. Coordinates must be in the decimal degree format in the main programs.
10. [deg2dms.m]: Converts decimal degrees to degrees, minutes, seconds.
11. [radians2dms.m]: Converts radians to degrees, minutes, seconds.
12. [convert_spherical2cart.m]: Converts spherical coordinates to Cartesian coordinates.
13. [convert_cart2spherical.m]: Converts Cartesian coordinates to spherical coordinates.

Auxiliary Programs

14. [dot.m]: Octave function for the dot product.
15. [cross.m]: Octave function for the cross product.
16. [angle_vectors.m]: Finds the angle between two vectors, given in either two or three-dimensions.
17. [circle_dist.m]: Computes the great circle distance between two points P_1 and P_2 .
18. [integer_part.m]: Removes the decimal part of a number to produce an integer.
19. [fract.m, frac.m]: Finds the absolute value of the fractional part of the number.
20. [magn.m]: Calculates the magnitude of 1, 2 and 3-dimensional real-valued vectors.
21. [sqrt.m]: Finds the square root of a real number.