GeodesicSphere

Finding shortest path on a 2D sphere

Sangwook Ryu

I present `GeodesicSphere` class, which calculates the shortest path between two points — origin (i) and destination (f) — on a 2-dimensional sphere with unit radius. Each point on a sphere can be specified by latitude ϕ and longitude λ. Distance ds between two infinitesimally close points becomes

$$ds^2 = d\phi^2 + \cos^2\phi\, d\lambda^2 \tag{1}$$

where $d\phi$ and $d\lambda$ are difference in latitude and longitude, respectively. The task on hand is to find a trajectory which minimizes

$$S_{\mathrm{i}\to\mathrm{f}}[\phi,\, \lambda] = \int_{\mathrm{i}}^{\mathrm{f}} \sqrt{d\phi^2 + \cos^2\phi\, d\lambda^2} \tag{2}$$

The equation (2) can be written in terms of a parametric variable ξ, such that a trajectory can be specified by $\phi\,(\xi)$ and $\lambda\,(\xi)$ as functions in ξ with boundary conditions.

$$u\, S_{\mathrm{i}\to\mathrm{f}}[\phi,\, \lambda] \;=\; \int_0^u d\xi \left[\left(\frac{d\phi}{d\xi}\right)^2 + \cos^2\phi\left(\frac{d\lambda}{d\xi}\right)^2\right]^{1/2} \tag{3}$$

$$\text{where} \qquad \begin{array}{l} \phi\,(\xi = 0,\, 1) = \phi_{\mathrm{i,f}} \\ \lambda\,(\xi = 0,\, 1) = \lambda_{\mathrm{i,f}} \end{array} . \tag{4}$$

Let us consider small variations in ϕ and λ and see how much deviation in S we have.

$$\begin{aligned} \delta S_{\mathrm{i}\to\mathrm{f}} &= S_{\mathrm{i}\to\mathrm{f}}[\phi + \delta\phi, \lambda + \delta\lambda] - S_{\mathrm{i}\to\mathrm{f}}[\phi, \lambda] && (5)\\ &= -\frac{1}{S_{\mathrm{i}\to\mathrm{f}}[\phi,\, \lambda]}\int_0^1 d\xi\left\{\left[\frac{d^2\phi}{d\xi^2} + \cos\phi\,\sin\phi\left(\frac{d\lambda}{d\xi}\right)^2\right]\delta\phi\,(\xi) + \frac{d}{d\xi}\left(\cos^2\phi\,\frac{d\lambda}{d\xi}\right)\delta\lambda\,(\xi)\right\} && (6) \end{aligned}$$

If a trajectory is the shortest path, we have $\delta S_{\mathrm{i}\to\mathrm{f}} = 0$ for any arbitrary infinitesimal $\delta\phi$ and $\delta\lambda$. Equation (6) implies that $\phi\,(\xi)$ and $\lambda\,(\xi)$ meet the following set of differential equations, which is also called *geodesic equation*.

$$\frac{d\phi}{d\xi} = \dot\phi \tag{7}$$

$$\frac{d\lambda}{d\xi} = \dot\lambda \tag{8}$$

$$\frac{d\dot\phi}{d\xi} = -\cos\phi\,\sin\phi\,\dot\lambda^2 \tag{9}$$

$$\frac{d\dot\lambda}{d\xi} = 2\tan\phi\,\dot\phi\,\dot\lambda \tag{10}$$

`GeodesicSphere` class implements relaxation methods[1] to solve the geodesic equation with boundary conditions. There are aforementioned 4 functions of interest — `y[1]` $= \phi$, `y[2]` $= \lambda$, `y[3]` $= \dot\phi$ and `y[4]` $= \dot\lambda$, and two boundary conditions (4) at each of boundaries (origin and destination).

`GeodesicSphere` class is shipped with `GeoIATA` (`Python`) module and `WrapGeoIATA` (`C++`) class, such that one can specify the origin and destination in terms of IATA code to find the shortest path between two airports. `WrapGeoIATA` class can take IATA codes as input to return latitude and longitude of the airport. One has to install `airportsdata` (`Python`) module to run the main program (`main_proj_geodesic.cpp`) for `GeodesicSphere`. In addition, `plotly` module needs to be installed to display geodesics on the world map. For instance, in `Linux/UNIX` system with `bash/zsh` shell, one can install `plotly` and `airportsdata` in a virtual environment (venv) by running the following commands.

[1] The mathematical algorithm is described in *Numerical Recipes in C* (2nd edition).

```
$ python3 -m venv [directory for venv]
$ source [directory for venv]/bin/activate
$ python3 -m pip install plotly
$ python3 -m pip install airportsdata
```

One can use an environmental variable `LIBODE_PATH_PYTHON` to specify the location of installed `airportsdata` module. The value of `LIBODE_PATH_PYTHON` is used to set `PYTHONPATH` via `setenv` function inside the main program, while this separate environmental variable is involved to prevent interference with the existing value of `PYTHONPATH`.

```
$ export LIBODE_PATH_PYTHON=[directory for venv]/lib/python3.13/site-packages
```

Note that the path varies depending on the `Python` version and operating system, so users are advised to check for their own path to the directory. One can run `proj_GeodesicSphere.exec` executable, which takes IATA code for the origin and destination as input. For instance, the shortest path (geodesic) from San Francisco to Seoul would be obtained by putting SFO and ICN, respectively.

```
$ ./proj_GeodesicSphere.exec
IATA code for the origin : SFO
IATA code for the destination : ICN

Origin :
  San Francisco, US
    lat (deg) = 3.761881e+01
    lon (deg) = -1.223754e+02
Destination :
  Seoul, KR
    lat (deg) = 3.746910e+01
    lon (deg) = 1.264510e+02

    n_iteration = 1
      err =     0.082540, fac =    0.605769

    n_iteration = 2
      err =     0.036594, fac =    1.000000

    n_iteration = 3
      err =     0.001655, fac =    1.000000

    n_iteration = 4
      err =     0.000013, fac =    1.000000

    n_iteration = 5
      err =     0.000000, fac =    1.000000
```

Alternatively, the origin and destination can be specified in the command-line arguments. The above example corresponds to running the following command.

```
$ ./proj_GeodesicSphere.exec SFO ICN
```

The main program produces a text file `geodesic_[origin]_[destination].txt`, which contains discretized path (i.e., list of latitude ϕ, longitude λ as functions of ξ). The output file also has an additional column for distance from the origin divided by the radius of sphere. One can multiply with the radius of Earth to get physical distance.

Alternatively, `GeoFigure` module can be used to directly visualize geodesics between airports. `GeoFigure.py` script file will be automatically copied to a directory where `LibODESolve` library is built. One has to set the environmental variable (`LIBODE_PATH_PYTHON`) and activate venv, if has not done so already.

```
$ export LIBODE_PATH_PYTHON=[directory for venv]/lib/python3.13/site-packages
$ source [directory for venv]/bin/activate
$ python3
```

First of all, one needs to import `GeoFigure` module. Then, `set_origin` and `add_destination` functions can be called to specify the origin and destination, respectively. Note that those functions take IATA codes in string as input parameters. `add_destination` function can be called more than once for a multi-city itinerary. Lastly, once the origin and destination are set, one can call `present` function to visualize geodesics on a world map. As an example, the shortest path from Boston to San Francisco, and then to Seoul can be obtained by running the following set of `Python` statments.

```
>>> import GeoFigure
>>> GeoFigure.set_origin('BOS')       # set the IATA code of the origin.
>>> GeoFigure.add_destination('SFO')  # set the IATA code of the first destination
>>> GeoFigure.add_destination('ICN')  # and the second one.
>>> GeoFigure.present()
```

Figure 1 shows the visualized result for the BOS-SFO-ICN itinerary.

Figure 1: The shortest path from Boston (BOS) to San Francisco (SFO) and then to Seoul (ICN).