Задачи на сфере: обратная геодезическая задача

Обсудить в форуме Комментариев — 21

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/sphere-geodesic-invert-problem.html

Обратная геодезическая задача — это нахождение начального направления и расстояния между двумя точками с известными координатами.

Содержание

- 1 Общие положения
- 2 Постановка задачи
- 3 Алгоритм
  - 3.1 Преобразование сферических координат в декартовы
  - 3.2 Вращение вокруг оси
  - 3.3 Преобразование декартовых координат в сферические
- 4 Пример программной реализации
- 5 Решение обратной задачи средствами PROJ.4
- 6 Альтернативные методы
- 7 Ссылки

Общие положения

В качестве модели Земли принимается сфера с радиусом *R*, равным среднему радиусу земного эллипсоида. Аналогом прямой линии на плоскости является геодезическая линия на поверхности. На сфере геодезическая линия — дуга большого круга.

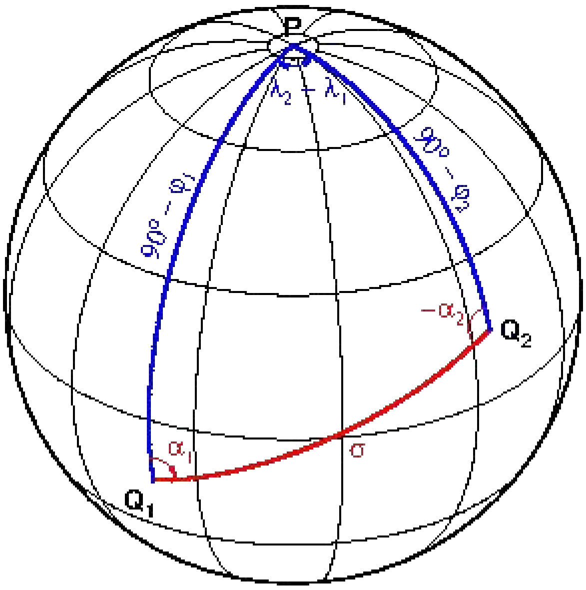
Введём следующие обозначения:

- φ — географическая широта,
- λ — географическая долгота,
- α — азимут дуги большого круга,
- σ — сферическое расстояние (длина дуги большого круга, выраженная в долях радиуса шара).

Линейное расстояние по дуге большого круга *s* связано со сферическим расстоянием σ формулой $s = R\,\sigma$.

Прямая и обратная геодезические задачи являются важными элементами более сложных геодезических задач.

Постановка задачи



Обратная геодезическая задача

Исходные данные координаты пунктов Q_1 и Q_2 на сфере — φ_1, λ_1 и φ_2, λ_2. Определяемые величины расстояние между пунктами и начальный азимут направления с точки Q_1 на пункт Q_2 — σ, α_1.

На рисунке синим цветом выделены заданные элементы сферического треугольника, красным цветом неизвестные.

Алгоритм

Существует великое множество подходов к решению поставленной задачи. Рассмотрим простой и надёжный векторный метод.

Последовательность решения:

1. преобразовать углы φ_2 и λ_2 в декартовы координаты,
2. развернуть координатные оси вокруг оси Z на угол λ_1,
3. развернуть координатные оси вокруг оси Y на угол $(90° - \varphi_1)$,
4. преобразовать декартовы координаты в сферические.

Можно устранить второй пункт, если в первом заменить долготу λ_2 на разность долгот $(\lambda_2 - \lambda_1)$.

Пример реализации алгоритма в виде функции языка Си:

```
/*
 * Решение обратной геодезической задачи
 *
 * Аргументы исходные:
 *     pt1  - {широта, долгота} точки Q1
 *     pt2  - {широта, долгота} точки Q2
 *
 * Аргументы определяемые:
 *     azi  - азимут начального направления
 *     dist - расстояние (сферическое)
 */
void SphereInverse(double pt1[], double pt2[], double *azi, double *dist)
{
  double x[3], pt[2];

  SpherToCart(pt2, x);                // сферические -> декартовы
  Rotate(x, pt1[1], 2);                       // первое вращение
  Rotate(x, M_PI_2 - pt1[0], 1);      // второе вращение
  CartToSpher(x, pt);                 // декартовы -> сферические
  *azi = M_PI - pt[1];
```

```
  *dist = M_PI_2 - pt[0];

  return;
}
```

Следует заметить, что прямая и обратная задача математически идентичны, и алгоритмы их решения зеркально отражают друг друга.

Преобразование сферических координат в декартовы

$$x = \cos\varphi \cos\lambda$$
$$y = \cos\varphi \sin\lambda$$
$$z = \sin\varphi$$

В данном случае в качестве сферических координат φ, λ подставим φ_2, λ_2.

Реализация на Си:

```
/*
 * Преобразование сферических координат в вектор
 *
 * Аргументы исходные:
 *     y - {широта, долгота}
 *
 * Аргументы определяемые:
 *     x - вектор {x, y, z}
 */
void SpherToCart(double y[], double x[])
{
  double p;

  p = cos(y[0]);
  x[2] = sin(y[0]);
  x[1] = p * sin(y[1]);
  x[0] = p * cos(y[1]);

  return;
}
```

Вращение вокруг оси

Представим оператор вращения вокруг оси *X* на угол *ϑ* в следующем виде:

$$x' = x$$
$$y' = y\cos\theta + z\sin\theta$$
$$z' = -y\sin\theta + z\cos\theta$$

Операторы вращения вокруг осей *Y* и *Z* получаются перестановкой символов.

Реализация вращения вокруг *i*-ой координатной оси на Си:

```
/*
 * Вращение вокруг координатной оси
 *
 * Аргументы:
 *     x - входной/выходной 3-вектор
 *     a - угол вращения
 *     i - номер координатной оси (0..2)
 */
void Rotate(double x[], double a, int i)
{
```

```
  double c, s, xj;
  int j, k;

  j = (i + 1) % 3;
  k = (i - 1) % 3;
  c = cos(a);
  s = sin(a);
  xj = x[j] * c + x[k] * s;
  x[k] = -x[j] * s + x[k] * c;
  x[j] = xj;

  return;
}
```

Преобразование декартовых координат в сферические

$$\lambda = \operatorname{arctg}\frac{y}{x}$$

$$\varphi = \operatorname{arctg}\frac{z}{\sqrt{x^2 + y^2}}$$

В данном случае в роли сферических координат φ, λ окажутся углы (90° − σ), (180° − α_1).

Реализация на Си:

```
/*
 * Преобразование вектора в сферические координаты
 *
 * Аргументы исходные:
 *     x - {x, y, z}
 *
 * Аргументы определяемые:
 *     y - {широта, долгота}
 *
 * Возвращает:
 *     длину вектора
 */
double CartToSpher(double x[], double y[])
{
  double p;

  p = hypot(x[0], x[1]);
  y[1] = atan2(x[1], x[0]);
  y[0] = atan2(x[2], p);

  return hypot(p, x[2]);
}
```

Пример программной реализации

Исходники вышеприведённых функций можно найти в архиве Sph.zip в файле **sph.c**. Кроме того, в файл **sph.h** включены следующие определения:

```
#define A_E 6371.0                              // радиус Земли в километрах
#define Degrees(x)  (x * 57.29577951308232)     // радианы -> градусы
#define Radians(x)  (x / 57.29577951308232)     // градусы -> радианы
```

Теперь напишем программу, которая обращается к функции SphereInverse для решения обратной задачи:

```
#include <stdio.h>
#include <stdlib.h>
#include "sph.h"

int main(int argc, char *argv[])
```

```
{
  char buf[1024];
  double pt1[2], pt2[2];
  double lat1, lon1, lat2, lon2, azi1, azi2, dist;

  while (fgets(buf, 1024, stdin) != NULL) {
    sscanf(buf, "%lf %lf %lf %lf", &lat1, &lon1, &lat2, &lon2);
    pt1[0] = Radians(lat1);
    pt1[1] = Radians(lon1);
    pt2[0] = Radians(lat2);
    pt2[1] = Radians(lon2);
    SphereInverse(pt2, pt1, &azi2, &dist);            // Решение обратной задачи
    SphereInverse(pt1, pt2, &azi1, &dist);            // Вычисление обратного азимута
    printf("%f\t%f\t%.4f\n", Degrees(azi1), Degrees(azi2), dist * A_E);
  }
  return 0;
}
```

В архиве Sph.zip этот код находится в файле **inv.c**. Создадим исполняемый модуль **inv** компилятором **gcc**:

```
$ gcc -o inv inv.c sph.c -lm
```

Впрочем, в архиве есть **Makefile**. Для MS Windows готовую программу **inv.exe** можно найти в архиве Sph-win32.zip.

Программа читает данные из стандартного ввода консоли и отправляет результаты на стандартный вывод. Для чтения и записи файлов используются символы перенаправления потока «>» и «<» соответственно. Из каждой строки ввода программа считывает координаты двух точек $\varphi_1, \lambda_1, \varphi_2, \lambda_2$, которые должны быть в градусах, решает обратную задачу и записывает в строку вывода α_1, α_2, s (азимуты прямого и обратного направлений в градусах; расстояние между пунктами в километрах, а точнее, в единицах, определённых константой A_E).

Создадим файл **inv.dat**, содержащий одну строку данных:

```
30 0 52 54
```

После запуска программы

```
$ inv < inv.dat
```

получим α_1, α_2, s:

```
44.804060 262.415109 5001.1309
```

В архиве Sph-py.zip находятся скрипты на языке Питон. Выполнение скрипта в командной консоли:

```
$ python inv.py inv.dat
```

Решение обратной задачи средствами PROJ.4

В пакет PROJ.4 входит программа **geod**, предназначенная для решения прямых и обратных геодезических задач на сфере. Так выглядит команда обработки файла **inv.dat**:

```
$ geod +a=6371000 -I -f "%f" -F "%.4f" +units=km inv.dat
```

Параметр +a определяет радиус сферы, -I — решение обратных задач, -f — формат вывода угловых величин, -F — формат вывода длин линий, +units — единица измерения расстояний. В результате получим идентичный вывод:

```
44.804060 -97.584891 5001.1309
```

Различие значений α_2 на 360° объясняется тем, что **inv** выводит азимуты в диапазоне от 0° до 360°, а **geod** от −180° до +180°.

Альтернативные методы

Некоторые элементы альтернативных методов решения обратной задачи представлены в статье Вычисление расстояния и начального азимута между двумя точками на сфере.

В большинстве своём другие методы основаны на сферической тригонометрии. Многие из них используют вычисление σ или α_1 по таким функциям, как синус, косинус или гаверсинус. Это приводит к неоднозначности результатов вблизи особых значений, когда производная функции равна нулю. Такие методы не могут считаться универсальными.

К наиболее надёжным относится следующий способ:

$$\xi = \sin\varphi_2 \cos\varphi_1 - \cos\varphi_2 \sin\varphi_1 \cos(\lambda_2 - \lambda_1)$$
$$\eta = \cos\varphi_2 \sin(\lambda_2 - \lambda_1)$$
$$\operatorname{tg}\alpha_1 = \frac{\eta}{\xi}$$
$$\operatorname{tg}\sigma = \frac{\xi\cos\alpha_1 + \eta\sin\alpha_1}{\sin\varphi_1 \sin\varphi_2 + \cos\varphi_1 \cos\varphi_2 \cos(\lambda_2 - \lambda_1)}$$

В сферической тригонометрии углы и стороны должны быть в диапазоне 0, 180°. Алгоритмизация формул требует анализа и обработки случаев, когда входные величины не попадают в эти рамки.

Ссылки

- Вычисление расстояния и начального азимута между двумя точками на сфере
- Задачи на сфере: прямая геодезическая задача
- Задачи на сфере: угловая засечка
- Задачи на сфере: линейная засечка
- Краткий справочник по сферической тригонометрии
- man_geod – PROJ.4
- Earth radius
- Степанов Н. Н. Сферическая тригонометрия

Обсудить в форуме Комментариев — 21

Последнее обновление: 2014-06-21 10:20

Дата создания: 11.03.2014
Автор(ы): ErnieBoyd