Алексеев Глеб
Добрилко Михаил
Черножуков Джордано

Работа посвящена нахождению местоположения грузового судна по имеющимся данным от маяков

Contents

1 Модель **2**

2 Точки и вектора **2**

3 Решение **3**

4 Формулы **4**

4.1 Расстояние между точками на сфере . . . 4
4.2 Пересечение трех сфер . . . 5
4.3 Пересечение сферы и прямой . . . 6
4.4 Конкретизация одной из двух точек . . . 6

5 Тестирование **6**

5.1 Генератор . . . 6
5.2 Bash . . . 7
5.3 Результаты тестирования . . . 7

1 Модель

Мы предполагаем, что Земля является сферой с радиусом 1. Каждая точка на сфере задается двумя углами α, ϕ, обозначающие широту и долготу соответственно. Для конкретизации координатам были выбраны следующие ограничения

- координаты в радианах
- $\alpha \in [\frac{-\pi}{2}, \frac{\pi}{2}]$
- $\phi \in [0; 2\pi)$

Точки, обозначающие полюса, могут быть представлены с $\alpha = \pm\frac{\pi}{2}$ и произвольным ϕ.

Маяк представляет из себя пару

- точка
- длина до корабля

Длина представляет из себя кратчайшее расстояние на сфере между точками, то есть ортодрома.

2 Точки и вектора

Чтобы удобно жонглировать координатами, мы будем представлять все в вектором стиле за исключением особых случаев, которые обговариваются отдельно. Для этого у нас будет класс $Point$, который представляет из себя

Listing 1: Point

```
const double EPS = ...;

struct Point {
   double x, y, z;

   Point();
   Point(double x, double y, double z);
   Point(double alpha, double phi);

   Point operator+(const Point& T) const;
   Point operator-(const Point& T) const;
   Point operator*(double T) const;

   ...
};
```

Он содержит всю основную векторную геометрию, необходимую для свободного манипулирования векторами. Мы будем использовать EPS как значение абсолютной погрешности во всех сравнениях, возникающих в вычислениях. Имея векторный арсенал, мы реализуем следующие функции

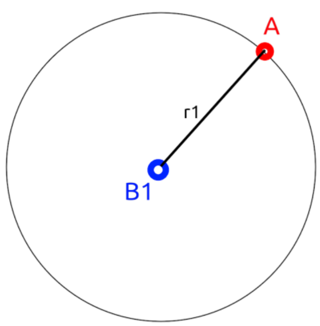
Listing 2: Functions

```
// находит угол между векторами [0; PI]
double angle(const Point& L, const Point& R);

// пересекает сферу с центром (0, 0, 0) и радиусом R с прямой L
vector<Point> interSphereLine(double R, const Line& L);

// находит расстояние на сфере радиуса R между точками A, B
double distOnSphere(double R, const Point& A, const Point& B);
```

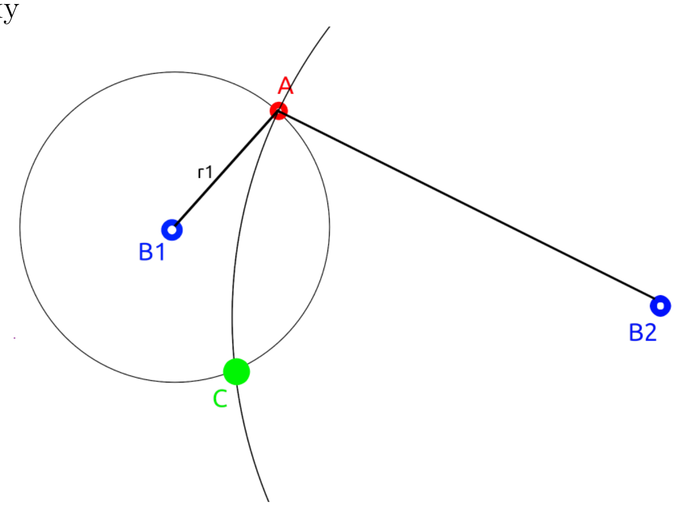
Эти функции нам понадобятся, чтобы реализовать основное решение. Конечно, есть более специализированные функции, которые не показаны здесь, их можно посмотреть в коде. Их роль имеет место только как вспомогательные элементы основных функций.

3 Решение

Для решения задачи полезно первым делом решить ее в плоскости. Полученные принципы мы перенесем на случай сфер. Отметим красной точкой A корабль, маяки же будут синими $B_1, \ldots, B_n$. Когда мы знаем, что точка A на расстоянии r_i от B_i, то это тоже самое, что точка A на окружности радиуса r_i и центром B_i.



Если мы посмотрим на еще какую-то точку B_j с радиусом r_j, то получим следующую картину



то заметим, что окружности пересекаются в двух точках. Конечно, возможно, что это будет касание, тогда мы сразу можем понять результат. В противном случае нам требуется третья окружность, которая уточнит точку, потому что A и C неотличимы относительно первых двух окружностей. Однако, если B_3 будет на прямой B_1B_2, то окружность пересечет сразу две точки A, C. Таким образом, для восстановления исходной точки нам требуются три точки, лежащие не на одной прямой.

Если мы попробуем перенести это решение на сферический случай, то столкнемся всего с несколькими изменениями, а именно

1. вместо пересечения окружностей на плоскости у нас пересечение сфер в пространстве
2. если центры сфер (включая центр Земли) в одной плоскости, то ответ неоднозначен

Первое очевидно, поэтому поясним только второе. Это аналог плоского случая, когда все точки на одной прямой. В случае, когда все точки на одной плоскости, то мы не можем понять, какая именно точка будет в ответе, та, которая по одну сторону этой плоскости, или та, что по другую.

Основываясь на этом, мы можем получить следующий алгоритм решения

1. выберем два маяка, которые не лежат на одной прямой с центром Земли
2. пересечем три сферы в одной или двух точках
3. если точка одна, то она и является ответом
4. иначе переберем третью окружность и уточним ответ

4 Формулы

Сформулируем основные математические переходы, которые лежат в основе реализации полученного решения.

4.1 Расстояние между точками на сфере

Пусть $dist(A, B)$ - это длина кратчайшей кривой между точками A, B, лежащими на сфере с радиусом R. Алгоритмически это расстояние выражается следующим образом

- строим плоскость ABO, где O - центр сферы
- пересекаем сферу и плоскость, получаем окружность в пространстве
- выбираем меньшую дугу, соединяющую AB на полученной сфере

Получаем $dist(A, B) = R \cdot \angle(\bar{A}, \bar{B})$, где $\bar{A}, \bar{B}$ - вектора из O в A, B соответственно. Угол получается как $\arccos \frac{\bar{A} \cdot \bar{B}}{|\bar{A}||\bar{B}|}$.

4.2 Пересечение трех сфер

У нас частный случай, когда одна сфера в начале координат и центры двух других находятся на ней. Формально мы хотим найти множество точек t (или векторов из O), которые удовлетворяют следующим уравнениям

- $|t| = 1$ (напомним, что Земля имеет радиус 1 по соглашению)
- $|t - O_1| = r_1$
- $|t - O_2| = r_2$

Чтобы найти решения этой системы, мы воспользуемся свойством $|v| = \sqrt{v \cdot v}$, то есть длина - это корень квадрата скалярного произведения. Перепишем с учетом этого

- $t^2 = 1$
- $(t - O_1)(t - O_1) = r_1^2$
- $(t - O_2)(t - O_2) = r_2^2$

после упрощения получим

- $t^2 = 1$
- $t^2 - 2tO_1 + O_1^2 = r_1^2$
- $t^2 - 2tO_2 + O_2^2 = r_2^2$

где окончательно получим

- $t^2 = 1$
- $tO_1 = \frac{O_1^2 - r_1^2 + 1}{2} = g_1$
- $tO_2 = \frac{O_2^2 - r_2^2 + 1}{2} = g_2$

последние два уравнения представляют из себя два уравнения плоскости. Раскрыв подобное скалярное произведения, мы бы получили

$$t_x a + t_y b + t_z c = d$$

Система из двух плоскостей, которые точно пересекаются, имеют своим решением прямую. Для ее нахождения мы найдем любым способом две точки, после чего получим исходную прямую. В нашем решении мы разделяем случай вертикальной прямой и всех остальных. В подобные тонкости реализации мы углубляться не будем.

4.3 Пересечение сферы и прямой

Когда мы получили прямую, то нам надо пересечь ее с исходной сферой. Прямая имеет векторное уравнение

$$line: \ \bar{s} + t\bar{v}$$

где s, v начальная точка и вектор направления соответственно, а t пробегает значения $(-\infty, +\infty)$. Мы просто хотим найти такой параметр t, что расстояние от точки прямой до O будет равняться 1. Это можно выразить через скалярное произведение походим образом

$$(s + tv)(s + tv) = 1$$
$$s^2 + 2t(sv) + t^2v^2 = 1$$

это квадратное уравнение относительно переменной t. Решая его, мы и найдем две точки (возможно одну, если прямая касается сферы).

4.4 Конкретизация одной из двух точек

Мы проходим по всем остальным точкам (маякам) и смотрим на расстояние до наших двух кандидатов, если они разные, то мы однозначно восстановили ответ, а иначе это точка лежит в одной плоскости, и мы должны ее пропустить. Для этого достаточно использовать уже имеющийся $dist(A, B)$.

Из-за особенностей чисел с плавающей точкой данный раздел заменем на более численно стойкий алгоритм, который математически эквивалентен приведенному выше. В нем мы находим такую точку, у которой разница расстояний максимальна, чтобы мы могли более точно определить верного кандидата.

5 Тестирование

Подобные алгоритмы, требующие большого количества формул, подлежат большому тестированию на наличие описок в коде и других неприятностей.

5.1 Генератор

Первым делом разумно написать генератор тестов, который предоставляет следующий формат теста

- n - количество маяков
- α_1, ϕ_1, d_1 - координаты первого маяка и расстояние до корабля
- $\ldots$
- α_i, ϕ_i, d_i
- $\ldots$
- α_n, ϕ_n, d_n

Для этого мы используем равномерное распределение по углам, генерируем данным образом $n+1$ точку, где первые n будут являться маяками, а относительно $n+1$ будет посчитаны расстояния через функцию $dist(A, B)$. Здесь можно генерировать точки разными способами, мы дополнительно добавили генерацию, где много точек лежат в одной плоскости.

5.2 Bash

Чтобы автоматизировать тестирование, мы написали *bash* скрипт, который перенаправляет вывод генератора в основное решение, после чего мы сравниваем $n+1$ точку с полученным ответом, и если они близки, то считаем решение верным. Данный процесс мы повторяем тысячи раз.

5.3 Результаты тестирования

Тестирование порядка 10000 тестов из 10 точек показало отсутствие ошибок. Данное решение можно считать достаточно правильным. Аспекты численной устойчивости не рассматривались в данной работе. Хорошей практикой будет использование чисел с более мощной точностью для промежуточных расчетов.