电 子 科 技 大 学 实 验 报 告

课程名称：数学实验

实验地点：科 A227

指导教师：陈小杰

评　　分：

完成实验学生信息：

选课序号	姓名	学号	贡献百分比/%	备注（主要工作）
	蔡与望	2020010801024	33	编写程序
	丁岩	2020010801007	33	编写程序
	胡义磊	2020010801005	33	编写程序

1. 学生人数按照任课教师要求限定；
2. 对于“评价、改进、总结和体会”都要认真填写，和其他内容是评价实验成绩的重要参考。

综合实验项目：椭球面上两点之间的最短距离

目录

椭球面上两点之间的最短距离

1. 问题分析

1.1 问题重述

已知椭球面方程: $\frac{x^2}{a^2}+\frac{y^2}{a^2}+\frac{z^2}{b^2}=1$。设 $a=6000, b=5000$。又已知该椭球面上两点 $P_1(2200,3600,z_1), P_2(2900,3300,z_2)$。请设计算法估算 P_1 和 P_2 两点在椭球面上的最短距离。这里 z_1, z_2 均大于 0。

1.2 问题分析

求椭球上任意两点间的最短弧长，用数学来推算解析的话十分复杂，因此考虑使用计算机来求得近似解。

其基本思想是，将一个观测点从 P_1 开始，逐步挪向 P_2；并且该点每一次前进的方向，都是在它所有可选的方向中，让自己和 P_2 的距离缩短最多的方向。显然这是一个“贪心”策略，即每一步的选择与之前的选择无关，只要每一次都做出最优选择，最后得出的选择也是最优的。

有了基本思想，我们就很容易发现解决问题的关键：选择出当前状态下的最优方向。下

面，我们就将围绕这一核心，建立模型并得出解答。

2. 模型假设

（1） 当每一步的步长足够小，观测点可到达的椭球面区域就可视作平面；
（2） 当观测点与终点的直线距离小于某一足够小的固定值，就可视作其到达终点；
（3） 观测点每次依直线移动，但这一直线距离近似等于在椭球表面移动的曲线弧长。

3. 变量与符号说明

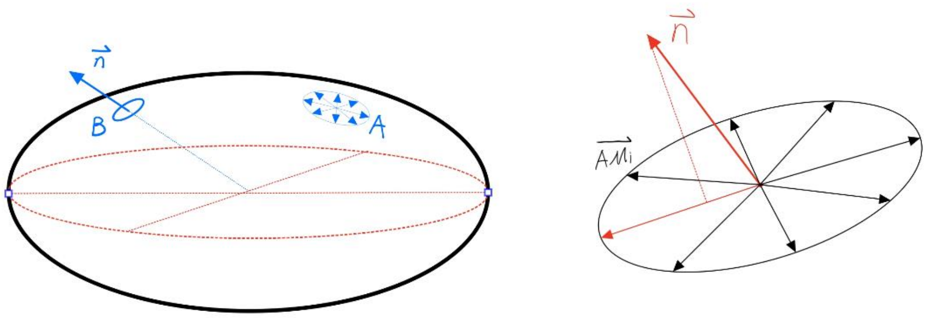
变量名	说明
A	观测点
B_i	观测点下一步能够到达的、椭球面上的点集
B_M	使 $\vec{n}\cdot\overrightarrow{AB_i}$ 值最大的点
d_i	观测点该次移动的直线距离
D	观测点总共移动的距离
ϵ	模型假设（2）中的固定值
$\vec{n}$	P_2 处的椭球面外法向量

4. 模型建立与求解

4.1 模型建立

如图为一椭球面，A、B 为椭球面上两点，以圆心为原点建立直角坐标系和极坐标系。则若 A 的直角坐标为 (x,y,z)，极坐标为 (ρ,θ,φ)，有映射关系：

$$x = a\cos\theta\cos\alpha$$
$$y = a\cos\theta\sin\alpha$$
$$z = b\sin\theta$$



4.2 算法设计

（1）首先建立椭球面，对给定的 P_1,P_2 两点，计算出两点的直角坐标，并且计算出 P_2 的外法向量 $\vec{n}$。

（2）设观测点为 A（第一步时观测点为 P_1），B_i 是在椭球面上以 A 为圆心半径极小的圆上一点，计算出向量 $\overrightarrow{AB_i}$。

（3）计算 $\vec{n}\cdot\overrightarrow{AB_i}$ 的值，并找出使该值最大的向量 $\overrightarrow{AB_m}$，该向量的方向就是当前状态下的最优方向。

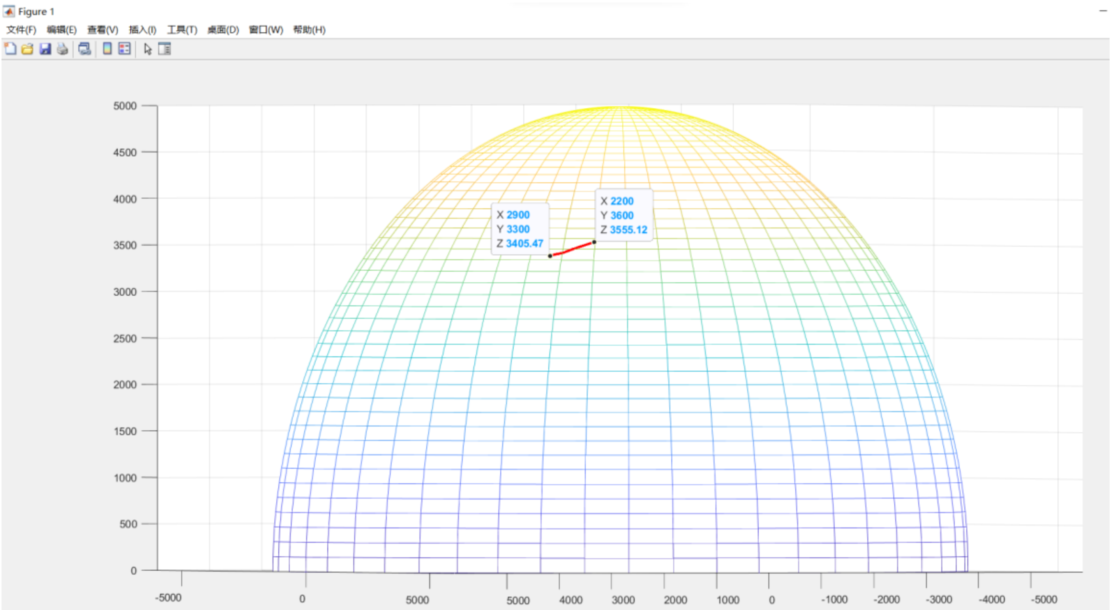
（4）计算当前 A 与 B_m 的距离，该距离就为观测点该次移动的直线距离 d_i。

（5）以 B_m 为新的观测点 A，重复（2）~（4）过程，直到 d_i 小于某一足够小的固定值 ϵ。

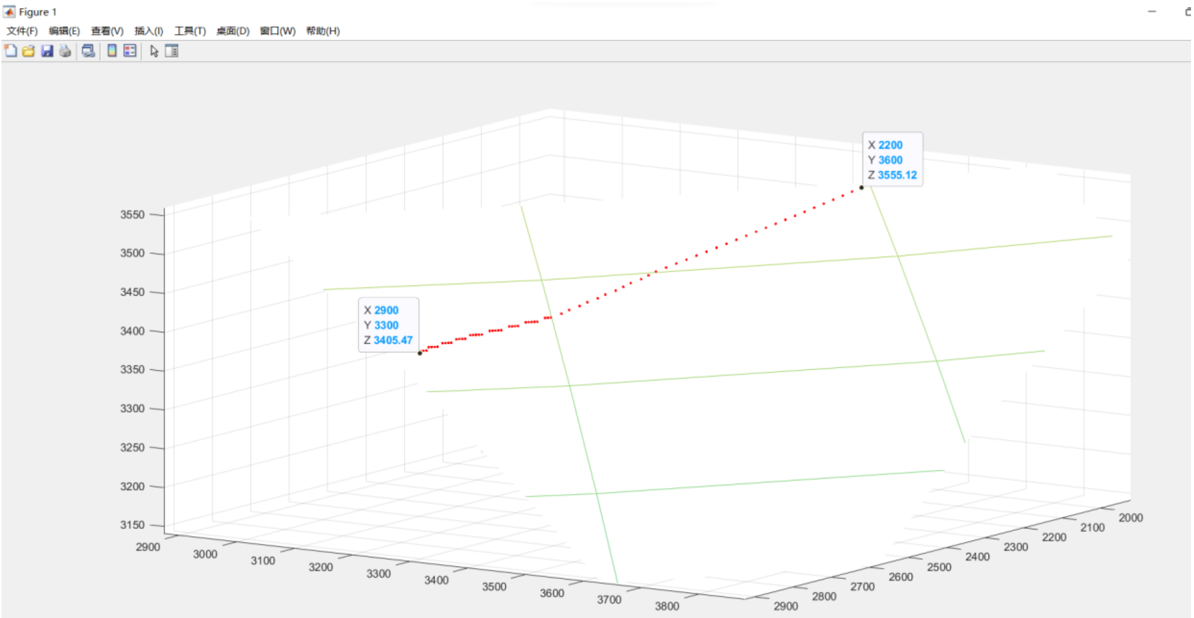
（6）计算观测点总共移动的距离 D。

4.3 模型求解

绘制出椭球面以及面上两点间最短路径，如图所示：



放大观察其点轨迹：



同时输出总距离与迭代步数：

```
>> main
total_distance=783.457350
total_steps=77
P1=2200.000000   3600.000000   3555.121501
P2=2900.000000   3300.000000   3405.469457
```

5. 实验结果分析

本实验测得椭球面上指定两点的最短路径长度结果为 783.4574，经过了 77 次迭代。

由于 P_1, P_2 两点间隔不远，所以我们可以通过两点间的直线距离大致判断该结果的合理性。通过简单的计算，直线距离 P_1P_2 长 776.1416；而结果的弧长略大于该值，所以我们可以认为该结果比较合理。

由此我们可以说，我们的算法较好地实现了对椭球面上两点最短距离的求解。

6. 优缺点及改进方向

本程序的优点在于算法思想简单，每次计算只需要关注当前状态，而不需要考虑过去状态。

但本程序缺点也十分明显。每一步的计算量较大，而且更为精确的结果又依赖于大量的迭代，导致程序的总计算量大。

另外，由于每次计算只关注当前状态，计算出的当前最优解可能在某些情况下并不是整体的最优解。

7. 心得体会与总结

通过本次实验，我们小组的 MATLAB 绘图技巧有了进一步的提升，掌握了通过创建极坐标来绘制直角坐标系图的方法；同时对贪心算法的核心思想有了较深的理解，并应用其解决了实际的数学问题。

附件

附件 1. 本实验的 MATLAB 程序

```
a = 6000;
b = 5000;
ellipsoid_axes = [a a b];
P1 = [2200 3600];
P2 = [2900 3300];
goal = 5;
total_distance = 0;

P1 = [P1 getEllipsoidZ(ellipsoid_axes, P1)];
P2 = [P2 getEllipsoidZ(ellipsoid_axes, P2)];
P2_normal = getEllipsoidNormal(ellipsoid_axes, P2);

cur_pos = P1;
nexts = [];

while (true)
    next_pos = getNextPos(ellipsoid_axes, cur_pos, P2_normal);
    nexts = [nexts; next_pos];
    walked_distance = getDistance(cur_pos, next_pos);
```

```
        total_distance = total_distance + walked_distance;
        dst_distance = getDistance(next_pos, P2);

        if dst_distance < goal
                break
        else
                cur_pos = next_pos;
        end

end

fprintf('total_distance=%f\n',total_distance);
fprintf('total_steps=%d\n',total_steps);
fprintf('P1=');
fprintf('%f    ',P1);
fprintf('\nP2=');
fprintf('%f    ',P2);

[theta, alpha] = meshgrid(linspace(0, pi / 2, 50), linspace(0, 2 * pi, 50));
z = b * sin(theta);
x = a * cos(theta) .* cos(alpha);
y = a * cos(theta) .* sin(alpha);
mesh(x, y, z)
hold on;

plot3(P1(1), P1(2), P1(3), 'r.', 'markersize', 6)
plot3(P2(1), P2(2), P2(3), 'r.', 'markersize', 6)
plot3(nexts(:, 1), nexts(:, 2), nexts(:, 3), 'r.', 'markersize', 6)
```

附件 1.2 子函数 1

```
function vec = getUnitVector(pointA, pointB)
      % Get the unit vector pointing from A to B.
      dx = pointB(1) - pointA(1);
      dy = pointB(2) - pointA(2);
      dz = pointB(3) - pointA(3);

      vec = [dx, dy, dz];
      vec = vec / norm(vec);
end

function next_pos = getNextPos(ellipsoid_axes, cur, normal_vector)
      % Get the rectangular coordinate the next_pos step should land on.
      max_product = 0;
      next_pos = [0 0 0];
```

```
    cur_polar = rectToPolar(ellipsoid_axes, cur);
    next_polars = getNextPolars(cur_polar);

    for next_polar = next_polars
        next_rect = polarToRect(ellipsoid_axes, next_polar);
        next_unit = getUnitVector(cur, next_rect);
        product = next_unit * normal_vector';

        if product > max_product
            max_product = product;
            next_pos = next_rect;
        end

    end

end

function rect_coord = polarToRect(ellipsoid_axes, polar_coord)
    % Convert polar coordinates to rectangular coordinates.
    a = ellipsoid_axes(1);
    b = ellipsoid_axes(2);
    c = ellipsoid_axes(3);
    theta = polar_coord(1);
    alpha = polar_coord(2);

    x = a * cos(theta) * cos(alpha);
    y = b * cos(theta) * sin(alpha);
    z = c * sin(theta);

    rect_coord = [x y z];
end

function polar_coord = rectToPolar(ellipsoid_axes, rect_coord)
    % Convert rectangular coordinates to polar cordinates.
    a = ellipsoid_axes(1);
    b = ellipsoid_axes(2);
    c = ellipsoid_axes(3);
    x = rect_coord(1);
    y = rect_coord(2);
    z = rect_coord(3);

    tan_alpha = (y / b) / (x / a);
    alpha = atan(tan_alpha);
```

```
    tan_theta = sqrt((z^2 / c^2) / (x^2 / a^2 + y^2 / b^2));
    theta = atan(tan_theta);

    polar_coord = [theta alpha];
end

function next_polars = getNextPolars(cur_polar)
    % Get all ends of vectors radiating from `cur_polar`, among which the next_pos step will select
one.
    radius = 0.01;
    precision = 0.001;
    num = 2 * radius / precision + 1;

    theta = linspace(cur_polar(1) - radius, cur_polar(1) + radius, num);
    alpha = linspace(cur_polar(2) - radius, cur_polar(2) + radius, num);
    [theta, alpha] = meshgrid(theta, alpha);
    theta = reshape(theta, [1 num^2]);
    alpha = reshape(alpha, [1 num^2]);

    next_polars = [theta; alpha];
end

function z = getEllipsoidZ(ellipsoid_coord, point)
    % Get Z coordinate of a point on an ellipsoid.
    a = ellipsoid_coord(1);
    b = ellipsoid_coord(2);
    c = ellipsoid_coord(3);
    x = point(1);
    y = point(2);

    z = c * sqrt(1 - x^2 / a^2 - y^2 / b^2);
end

function vec = getEllipsoidNormal(ellipsoid_coord, point)
    % Get the normal vector of a point on an ellipsoid.
    a = ellipsoid_coord(1);
    b = ellipsoid_coord(2);
    c = ellipsoid_coord(3);
    x = point(1);
    y = point(2);
    z = point(3);

    vec = [2 * x / a^2, 2 * y / b^2, 2 * z / c^2];
    vec = getUnitVector([0 0 0], vec);
```

```
end

function distance = getDistance(pointA, pointB)
    % Calculate the distance between A and B.
    distance = sqrt((pointB(1) - pointA(1))^2 + (pointB(2) - pointA(2))^2 + (pointB(3) - pointA(3))^2);
end
```

附件 2. main 程序的输出结果

```
>> main
total_distance=783.457350
total_steps=77
P1=2200.000000   3600.000000   3555.121501
P2=2900.000000   3300.000000   3405.469457
```