**上机报告：太阳系作业**

物理学院

1400012141

邵智轩

**一 题目描述**

编制程序， 描述太阳系行星随时间变化的运动轨迹， 以及在各种意外情况下太阳系的稳定性分析。 利用编制的程序来分析一些物理问题， 例如如果太阳质量下降一半会如何？以及碰撞问题等。

**二 题目分析**

利用牛顿第二定律计算每一微元时间dt的位移和速度变化即可。只要将dt取得足够小，就可以保证结果的精确性。由于物理计算涉及到很多矢量，如位置，速度，引力，不妨定义矢量结构。

整个太阳系分为恒星、行星、卫星三层，构造树状结构。则整个系统的运行就类似于树的周游，访问每个结点并计算其受力和运动。

碰撞问题按照球体的完全弹性碰撞处理，代入公式即可。

**三 数据结构**

**1 矢量类型 struct Vector3D**

定义三维矢量Vector3D类型，成员为double x,y,z三个浮点数。有以下成员函数：

1. double magnitude()//求模
2. double distance(double a=0, double b=0, double c=0)//求与点(a,b,c)的距离
3. 重载与标量的“+、-、×、/”四则运算
4. double dot(Vector3D a=Vector3D())//点乘运算
5. Vector3D cross(Vector3D a = Vector3D())//叉乘运算
6. 重载输出算符<<，使其按照“os << c.x << '\t' << c.y << '\t' << c.z;”的形式输出，方便把数据输出到xlsx文件。

有了以上这些功能，计算时就只需把相应矢量代入公式即可。

**2.1 质点类型 struct object**

定义质点类型（没有形状），成员为double mass、Vector3D location、Vector3D speed。有以下成员函数：（有一些在太阳系中没有用到）

1. Vector3D momentum()//动量
2. double kinetic\_energy()//动能
3. Vector3D angular\_momentum(Vector3D reference=Vector3D())//相对参考点的角动量
4. void unit\_move(Vector3D a, double dt)//dt时间内的微元运动，算法附后

**2.2 球体类型 struct object: public object**

定义球体类型（有半径信息），新增成员double radius。新增以下成员函数：

1. bool WillNotCollide(sphere\* other)//判断两球体是否已经达到碰撞条件（球心距离小于半径之和）
2. void collision(sphere\* other)//三维球体完全弹性碰撞。具体算法会在算法部分介绍。
3. void print2D\_xy(int n=36)//打印xy球面的圆 n为打印的点的个数

**2.3 天体类型 struct celestial\_body : public sphere**

姑且认为所有天体都近似为球体。新增成员string name。

除了行星和恒星定义为其子类外，其他天体如卫星都为celestial\_body类型。

新增成员函数：

1 double Theta(Vector3D core=Vector3D())//计算其在球坐标下的θ角

2 double Phi(Vector3D core=Vector3D())//计算其在球坐标下的φ角

3 重载运算符“<<”：friend ostream & operator << (ostream & os, const celestial\_body &c)//输出天体信息，如名字，位置，质量等

**2.4 行星类型 struct planet: public celestial\_body**

定义行星类型，新增成员int Satellites\_num、int max、celestial\_body\*\* Satellites。分别为行星的卫星实际数量，数量上限和存放卫星的**顺序表**。表中存放指向卫星的指针。

新增以下成员函数：

1. void Satellites\_insertSort()//按照卫星与行星的距离对卫星的顺序进行排序（直接插入排序）。
2. void insert\_Satellite(celestial\_body\* x)//插入卫星

**2.5 恒星类型 struct star public celestial\_body**

定义恒星类型，新增成员int Planet\_num、int max、planet\*\* Planets。分别为恒星的行星实际数量，数量上限和存放行星的**顺序表**。表中存放指向行星的指针。

新增以下成员函数：

1. void Planets\_insertSort()//按照恒星与行星的距离对行星的顺序进行排序（直接插入排序）。
2. void insert\_Planet(planet\* x)//插入行星
3. void unit\_planet\_move(double dt)//整个恒星系在微元时间dt内的运动。算法后面会详细介绍。
4. void planet\_move(double dt, double time)//整个恒星系在时间time内的运动，以dt为微元时间。
5. celestial\_body \* search(string x)//搜素恒星系中名字为x的天体，并输出其信息

**四 算法及其关键函数**

**1 运动问题**

**1.1 万有引力计算：Vector3D Gravitation(double mass, Vector3D r)**

万有引力加速度g满足：。代码如下：

Vector3D Gravitation(double mass, Vector3D r)//计算质量为mass的物体对相对位置为r的物体的万有引力加速度

{ return r\*(-1)\*constG\*mass / pow(r.magnitude(), 3); }

**1.2 质点的微元运动：void object:: unit\_move(Vector3D a, double dt)**

采用了最简单的算法：

void unit\_move(Vector3D a, double dt)

{

location = location + speed\*dt;

speed = speed + a\*dt;

}

**1.3 恒星系微元运动：void star:: unit\_planet\_move(double dt)**

整个太阳系相当于一个高度为3的树（自上而下3层分别为恒星、行星、卫星）的结构，现在的问题就是树的周游。计算每个周游的结点的微元运动。

为了合理的简化计算，这里假设恒星的位置是固定的，即不考虑恒星的微元运动。

采用深度优先方式。对于每个行星，先考虑其卫星的微元运动（如果它有的话），卫星的引力为：①所有其他卫星的引力②恒星系内所有行星对它的引力③恒星对它的引力。

再考虑行星自身的运动，它所受的引力为：①其卫星对它的引力②恒星系内所有其他行星对它的引力③恒星对它的引力。

函数代码较长，就不贴在这了。当然，也可以对行星只计算恒星的引力，对卫星只计算恒星和其所属行星的引力，就能很大程度地加快运行速度。而且结果证明这样的系统很稳定，甚至可以说更稳定。但是为了模拟真实的过程，我还是选择了这种。

**1.4 输入星体的数据以及单位的处理**

我选用的是kg，m，s单位制。然而直接使用必然造成输入数据时的不便，故先定义如下单位：

#define AU (1.4959787e+11)//距离的单位都是米

#define pi 3.14159265358979

#define minute 60

#define hour 3600

#define day (23\*hour+56\*minute+4.0990)

#define year 365.256364\*day//时间单位都是秒

在网上查到了行星与月球的数据后直接输入。有前面的数据结构可知，须求出初位置和初速度。为了简便，初位置我一律取在行星的近日点，且都设在y=0平面上。初位置取在，算法如下：

Vector3D get\_initial\_location(double average\_distance, double eccentricity, double inclination)//给出行星的初位置（近日点)

{

return Vector3D((1 - eccentricity)\*average\_distance\*cos(inclination), 0, (1 - eccentricity)\*average\_distance\*sin(inclination));

}

函数中的average\_distance，行星的平均距离即为行星的半长轴长，eccentricity为离心率，inclination为轨道倾角（弧度制）。

初速度曾考虑过两种算法，其一是由公式得到。

其二是由得出的。

通过误差分析，即比较两种算法得到的地球周期。第一种算法为370天左右，第二种算法为366天左右。故最终采取第二种方法得到初速度。

**2 行星、卫星的直接插入排序**

使用了课上学习的直接插入排序算法，按照某一时刻行星（卫星）与恒星（行星）的距离对行星（卫星）在顺序表中的位置进行直接插入排序。

**3 球体碰撞问题：void sphere:: collision(sphere\* other)**

首先是判断：两球球心距小于等于两球半径之和则构成碰撞。在周游太阳系中所有结点时，不仅计算每个行星的微元运动，同时也会判断是否与其他星体发生碰撞。

其次是某一瞬间的碰撞问题，碰撞前后只有速度改变。为**三维球体完全弹性碰撞**。

转换到质心系中解决该问题，步骤如下：

①计算质心的速度

②计算两球相对于质心的速度（转换到质心系）

③连心线方向的单位向量

④与连心线平行的速度分量反向，垂直的速度分量不变

⑤加上质心速度，转换回原参考系

算法如下：

void sphere:: collision(sphere\* other)//三维球体完全弹性碰撞

{

const Vector3D vc = (momentum() + other->momentum()) / (mass + other->mass);//质心速度守恒

Vector3D r12 = (other->location - location) / (other->location - location).magnitude();//连心线单位向量

Vector3D v1c0 = speed - vc; Vector3D v2c0 = other->speed - vc;//相对质心的速度

Vector3D v1cn = r12 \* (v1c0.dot(r12));//平行分量

Vector3D v2cn = r12 \* (v2c0.dot(r12));

speed = v1c0 - v1cn \* 2 + vc;

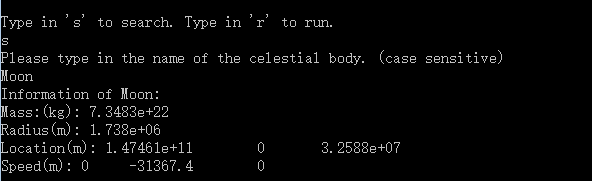
other->speed = v2c0 - v2cn \* 2 + vc;

}

**五 结果分析**

**1 搜索天体**

以天体的名字为关键字，在恒星系内深度优先搜索该天体，搜索成功时输出该天体的信息。



**2 星体运动**

行星我以地球和最外面的海王星为例，卫星以月球为例进行分析。总模拟时间为一个海王星绕日周期（约167.8年），微元运动时间dt为0.1秒。程序运行时间为13小时。8颗行星轨迹图如下：

有趣的是靠内的四颗行星的轨道半径与外面的四颗行星相比非常小，几乎“挤在一起”。

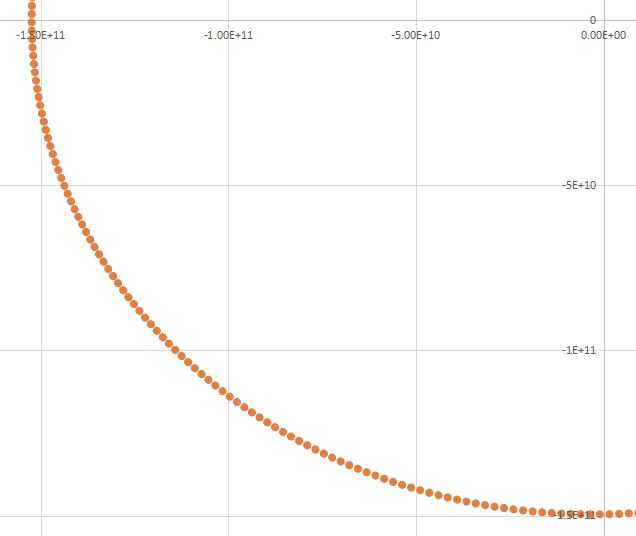
**2.1 地球**

由下图看出，轨道曲线还是可以很好的闭合的。与实际出入明显的一点是周期比真实值大。

由放大图看得更加清楚。最后一个点是第366天。可见模拟出的地球周期略大于366天，比真实值365.36天偏大。

再来分析地球轨道的稳定性。下图中蓝色的是地球第0到366天，橙色为（167年后）第59823到第60189天，可以看到轨道几乎没有变化。

放大之后可以看到，橙色的点和蓝色的点几乎是点点重合的。运行相当稳定。



**2.2 海王星**

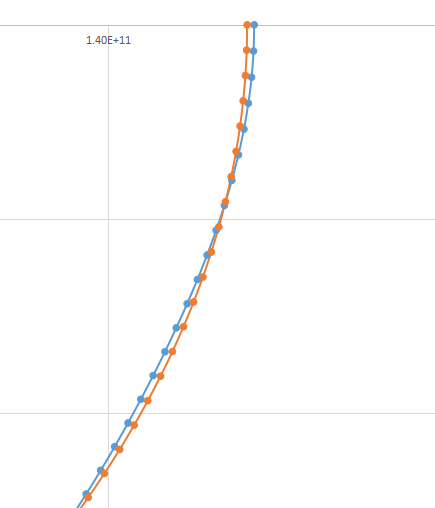
再以最外层的海王星为例。我按照搜索得到的周期运行。最后发现海王星没能绕完1周。根据最后一次的坐标可以算出离绕满一周还差0°1′11.01″。

海王星的z坐标分量为余弦函数，如下图。

**2.3 月球**

由下图看出，模拟得到的月球绕地周期约为（略短于）27天，（第27天时已超过1周）小于真实值。

下图可以看出，月球绕地球旋转，同时和地球一起绕太阳公转。



再来分析月球轨道的稳定性。下图中蓝色的是月球第0到27天，橙色为（167年后）第60162到第60189天，可以看到轨道发生了很大程度的平移，而且平均地月距离也略有增大。

月球相比其他行星的轨道不那么稳定。可能是因为月地距离比较近，尺度相对较小，而其引力加速度又较大，故积累的误差比较多也比较明显。

**3 星体碰撞**

为了比较方便的人为制造碰撞。我添加了一颗金星的镜像行星（关于yz平面的镜像），观察它的碰撞。为了方便观察我在碰撞时用void print2D\_xy(int n=36)函数绘制出球体的xy截面。

为了确保在有非对称的，其他行星引力作用的情况下依然能撞上，在第一次实验时，我将金星与镜像金星的半径先调大了100倍。

碰撞结果如下图：

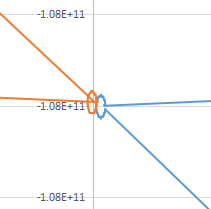
可以看到，碰撞结果具有高度的对称性。如果多运行一会儿，甚至能发生3次碰撞，见下图。

于是我又尝试按原半径碰撞，结果如下图：

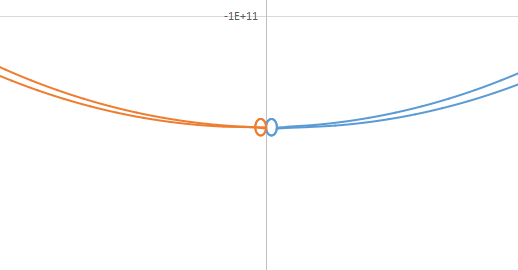
可以看到，碰撞后两颗行星的运动轨迹是很不对称的。

为什么半径不同，碰撞效果会如此不同呢？把碰撞瞬间的图放大即可理解了。

这是原半径的碰撞瞬间图（圆是用print2D\_xy()绘制的），可以看出由于非对称的受到其他行星的引力（尽管很微小），造成了金星与其镜像的轨道略有错位，不是对心碰撞，所以产生了偏离。



而这是半径放大一百倍后的碰撞瞬间图。由于球体半径变大，所以“更接近于”对心碰撞，效果也就更对称。



**4 太阳质量减小一半**

太阳质量如果减小为原来的一半会发生什么情况呢？

为了作图的方便，图上只展示了四颗行星的运动轨迹。

可以看到四颗行星的轨迹都不再是闭合的椭圆。而是抛物线或双曲线。如水星就明显是双曲线，而金星则很接近于抛物线。试分析如下：

在近日点位置，行星初速度不变而太阳质量减半后，行星轨道的能量变为，e为离心率。水星e为0.2左右，比较大，故接近于双曲线；而金星e为0.0068，较小，故接近与抛物线。可以想象，若在行星运行至远日点时改变太阳质量，则行星轨道的能量变为仍为椭圆，只不过是“非常偏”的椭圆。

**六 经验心得**

体会到了逻辑清晰的数据结构在编程中的重要性。可以大大简化代码，并且写起来也比较轻松，不容易出bug。运用了不少课程中学到的知识或思想，如顺序表、树、排序等。并熟悉了C++的类的使用，如构造、析构、成员函数、继承派生等。运算符重载的功能，大大简化了物理公式的编写。运用“#define”处理单位转换问题，使得输入数据时可以直接用各种单位，如AU，year，mass\_of\_Earth，而最终代入公式运算的数据为熟悉的千克米秒制。在处理碰撞问题时，体会到了在质心系中处理问题的便捷性，实践了“力学”课上学到的知识。学会了直接将数据输出到excel文件，并在excel上处理大规模数据，并绘制图表。

总之，独立完成了一个比较复杂的项目，并首次体会用程序解决实际的物理问题，我感到受益匪浅。

搜索

以天体的名字为关键字，在恒星系内深度优先搜索，搜索成功就输出