Numerical simulation of orbital resonance under gravitational field and stability analysis

Zhen Li

Department of Physics, Southeast University

April 14, 2019

Abstract: Some types of orbital resonance under gravitational field has been simulated in numerical way with the Eular method improved by predictor-corrector method in this paper. Here presents the introduction of orbital resonance, the algorithm of numerical calculation, the simulation results and comparison with real orbital resonance, such as that between Jupiter and asteroid belt.

Key Words: orbital resonance, numerical simulation, Eular method.

I. Introduction to orbital resonance

Orbital resonance occurs when orbiting bodies regularly influence each other with gravity, which usually because their orbit periods are related by a ratio of small integers (Wikipedia, 2019). In most cases, it results in an unstable interaction, two orbiting bodies changing orbits until the resonance no longer exists, for example, the gaps in the rings of Saturn. However, an orbital resonance system can be stable in some circumstances, such as the 1:2:4 resonance of Jupiter's moons Ganymede, Europa and Io.

II. Algorithm of numerical calculation

Orbital resonance can be simulated by a many-body system. For celestial body *i*, the acceleration

$$\boldsymbol{a_i} = \ddot{\boldsymbol{r}}_i = G\sum_{j\neq i}\frac{m_j(\boldsymbol{r_j}-\boldsymbol{r_i})}{\|\boldsymbol{r_j}-\boldsymbol{r_i}\|^3}$$

in which m_j and $\boldsymbol{r_j}$ separately refer to the mass and position vector of celestial body *j*.

Then take a small step τ from time *k* to $k+1$ and apply to the Eular method (Pang, 2011).

$$\boldsymbol{v_i^{(k+1)}} = \boldsymbol{v_i^{(k)}} + \tau\boldsymbol{a_i^{(k)}} + O(\tau^2)$$

$$\boldsymbol{r_i^{(k+1)}} = \boldsymbol{r_i^{(k)}} + \tau \boldsymbol{v_i^{(k)}} + O(\tau^2)$$

To be more accurate, predictor-corrector method is applied after calculating all of vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ (Pang, 2011).

$$\boldsymbol{v_i^{(k+1)}} = \boldsymbol{v_i^{(k)}} + \frac{\tau}{2}(\boldsymbol{a_i^{(k)}} + \boldsymbol{a_i^{(k+1)}}) + O(\tau^3)$$

$$\boldsymbol{r_i^{(k+1)}} = \boldsymbol{r_i^{(k)}} + \frac{\tau}{2}(\boldsymbol{v_i^{(k)}} + \boldsymbol{v_i^{(k+1)}}) + O(\tau^3)$$

All of the vectors are resoluble in Cartesian system *Oxyz*. As for the program, there is no much difficulty describing vectors and their operation.

It is necessary to simulate a two-body system with the aim of simply verifying the algorithm. The parameters are as follows.

i	m_i	$\boldsymbol{r_i^{(0)}}$	$\boldsymbol{v_i^{(0)}}$	k
1	10000	(0,0,0)	(0,0,0)	1
2	1	(1,0,0)	(0,100,0)	

Theoretically celestial body 1 should be almost stay static while celestial body 2 need to have a nearly circular orbit in surface *xOy*.

Set $\tau = 0.001$ and total simulation time $T = 10$. The motion of two celestial body fits in with the expectation (Fig.1). The deviation can be explained by the influence of gravity that celestial body 2 has.

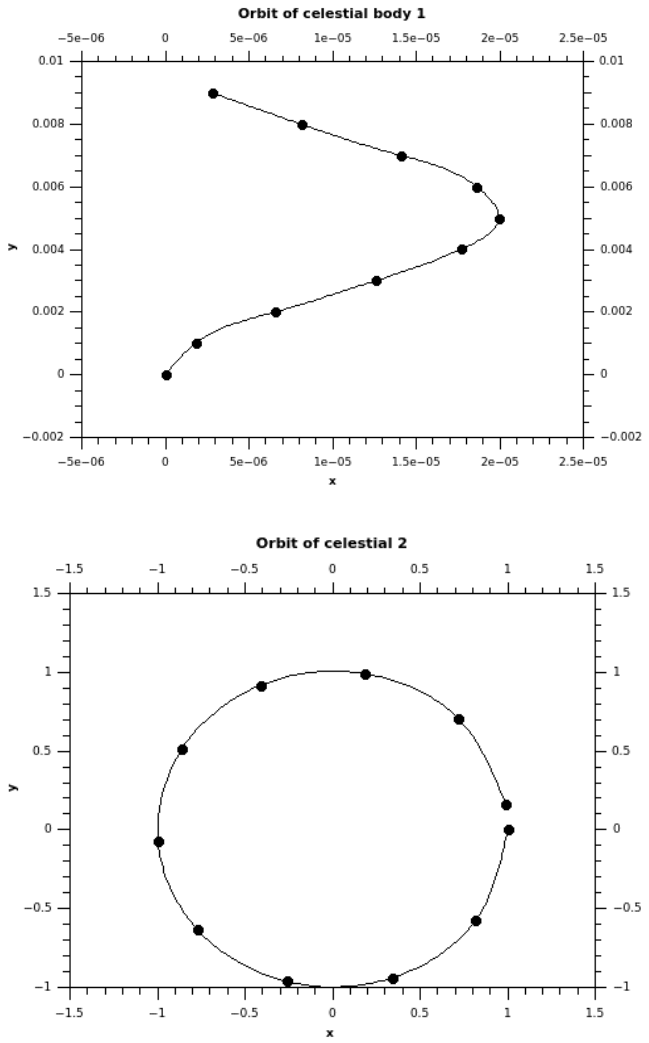

Fig.1 Orbit of celestial body 1(upper) and 2(below).

III. Simulation results and comparison with real orbital resonance

Firstly consider a much smaller asteroid added to a star-planet system (like the system simulated in II) while the ratio of their initial orbit period is a simple fraction. Set $\tau = 0.001$, $T = 1000$, and the simulation results show that orbital resonance 1/2 (Fig.2), 3/7 (Fig.3), 2/5 (Fig.4) and 1/3 (Fig.5) are not stable for long period of time.

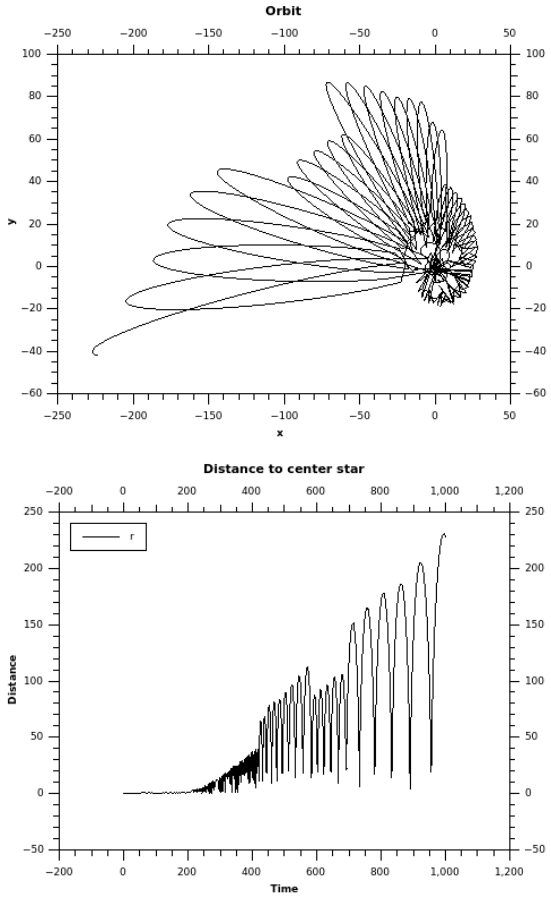

Fig.2 Resonance 1/2, orbit (upper) and distance to center star (below)

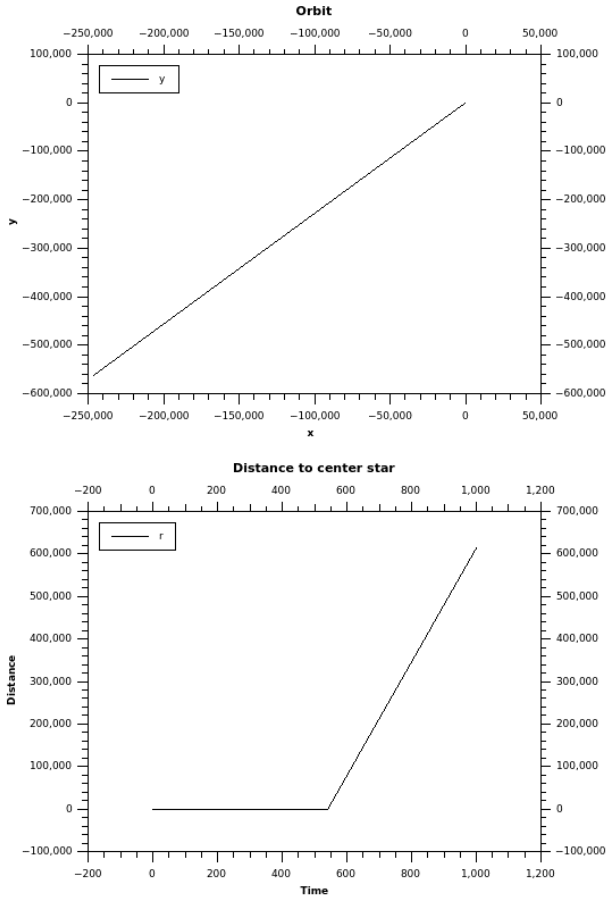

Fig.4 Resonance 2/5, orbit (upper) and distance to center star (below)

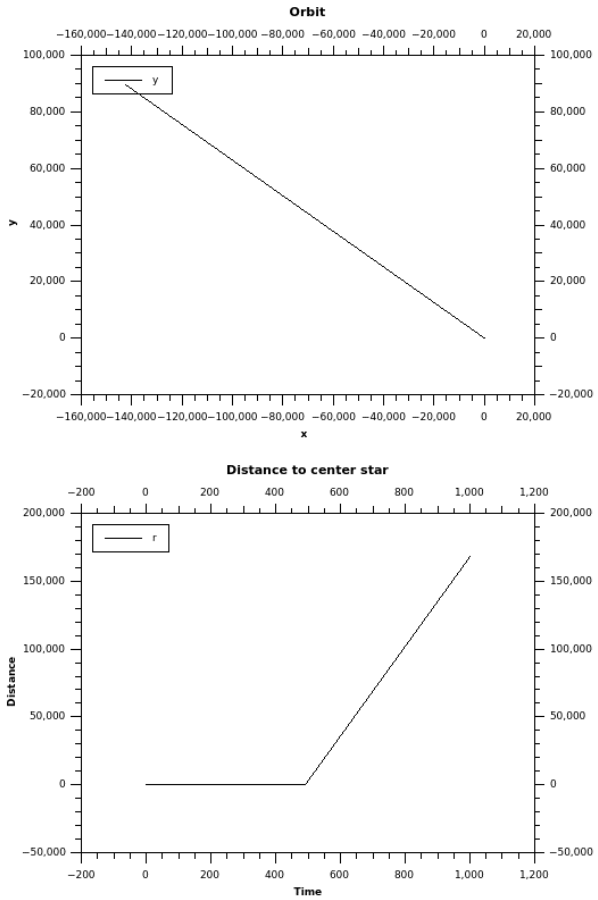

Fig.3 Resonance 3/7, orbit (upper) and distance to center star (below)

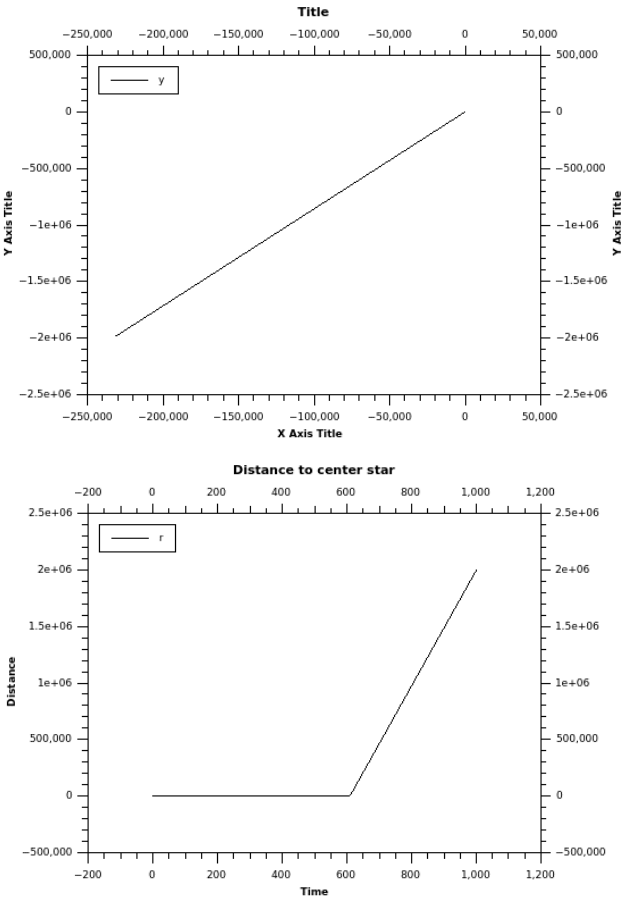

Fig.5 Resonance 1/3, orbit (upper) and distance to center star (below)

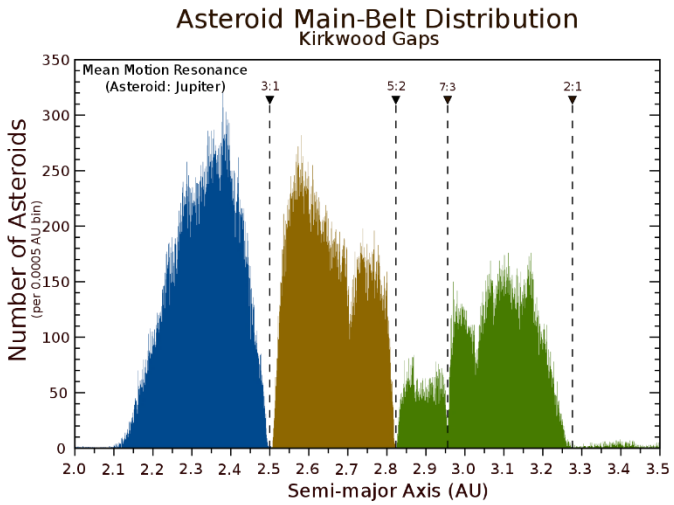

Fig.6 Asteroid main-belt distribution (Wikipedia, 2019)

Interestingly, in asteroid belt, there are few asteroids which orbit periods are 1/2, 3/7, 2/5, or 1/3 time of Jupiter (Fig.6). The gaps there are known as Kirkwood gaps.

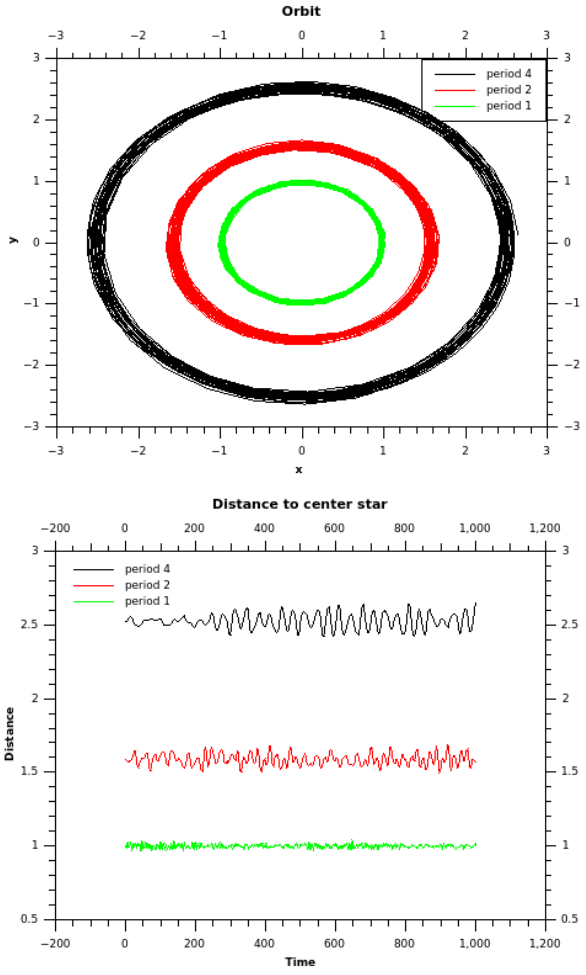

Fig.7 Orbital resonance 1:2:4, orbits (upper) and distance to center star (below).

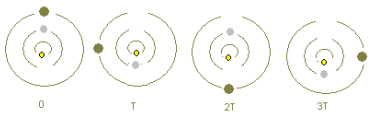

Fig.8 Orbital resonance of Jupiter's moons (Wikipedia, 2019).

In addition, there are also stable orbital resonances, for example, resonance 1:2:4 (Fig.7), which is the one of Jupiter's moons Ganymede, Europa and Io (Fig.8). Below are one set of parameters of this type of resonance.

i	m_i	$\boldsymbol{r}_i^{(0)}$	$\boldsymbol{v}_i^{(0)}$	k
1	10000	(0,0,0)	(0,0,0)	1
2	1	(1,0,0)	(0,100,0)	
3	1	(1.5874,0,0)	(0,79.37008,0)	
4	1	(2.51984,0,0)	(0,62.99608,0)	

Another stable resonance exists at 3/2 (Fig.9), which is the ratio of orbit period between Neptune and Pluto. Below are one possible set of parameters.

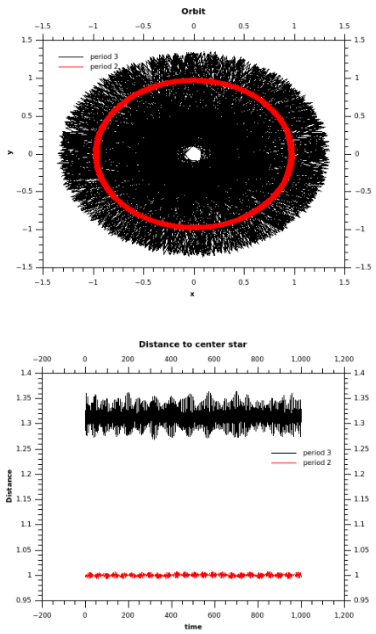

Fig.9 Resonance 3/2, orbits (upper) and distance to center star (below).

i	m_i	$\boldsymbol{r}_i^{(0)}$	$\boldsymbol{v}_i^{(0)}$	k
1	10000	(0,0,0)	(0,0,0)	1
2	1	(1,0,0)	(0,100,0)	
3	0.1	(1.31037,0,0)	(0, 87.358046,0)	

What's more, mass of orbital-resonance objects has little influence on the stability (Fig.10, Fig.11).

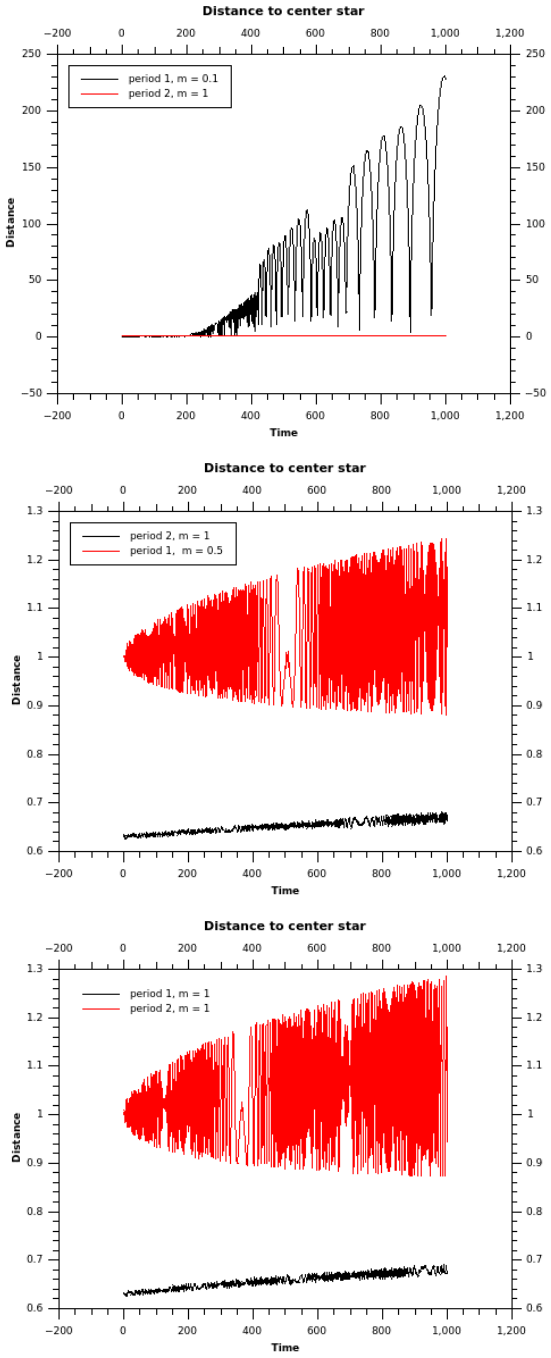

Fig.10 Resonance 1/2

Upper: mass 0.1, 1

Middle: mass 0.5, 1

Below: mass 1, 1

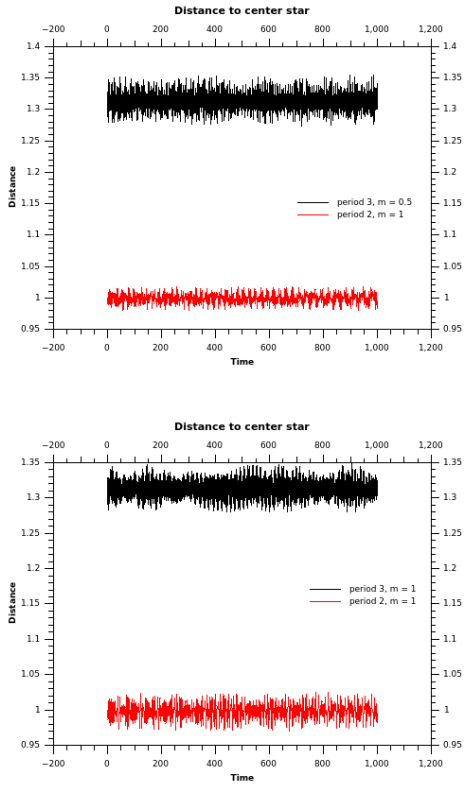

Fig.11 Resonance 3/2

Upper: mass 0.5, 1

Below: mass 1, 1

IV. Conclusion

With the help of numerical simulation, a set of stable/unstable orbital resonances are found, which can explain some astronomical phenomena. The Algorithm here is also suitable for many-body system. However, perturbation theory in celestial mechanics is needed to have a deeper understanding of orbital resonance (李广宇, 2015).

References:

Pang, T. (2011). Predictor-corrector methods.

Wikipedia. (2019). Orbital resonance. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Orbital_resonance

李广宇. (2015). 普遍摄动理论. In 李广宇, 天体测量和天体力学基础 (pp. 160-166).

```
/*
 * Useage: time number_of_planet planet_file correct_time [central_object]
 *
 *      time:                   The total simulation time.
 *
 *      number_of_planet:       The total number of planet to be simulated.
 *
 *      planet_file:            The text file with initial values of all the planets.
 *                              Each line contains the information of one planet.
 *                              The format needs to be:
 *                              mass position_x position_y position_z velocity_x velocity_y velocity_z
 *                                  The program will not check the validity of data given.
 *
 *      correct_time:           The time that predictor-corrector method is used
 *
 *      [central_object]:       An optional argument.
 *                              Set one of the planets as central body,
 *                              and the relative position "xr" "yr" "zr",
 *                              relative velocity "vxr" "vyr" "vzr",
 *                              and distance to central body "dist"
 *                              of other planets will be shown in the output file.
 */

#include <iostream>
#include <fstream>
#include <cmath>
#include <string.h>

using namespace std;

const double dt = 0.001;

class vector
{
    double x;
    double y;
    double z;
public:
    vector(double xx = 0, double yy = 0, double zz = 0)
    {
        x = xx;
        y = yy;
        z = zz;
    }
    friend vector operator +(vector &a, vector &b);
    friend vector operator -(vector &a, vector &b);
    friend vector operator *(double n, vector &v);
    friend ostream &operator <<(ostream &out, vector &v);
    double length()
    {
        return sqrt(x * x + y * y + z * z);
    }
    void setvector(double xx = 0, double yy = 0, double zz = 0)
    {
        x = xx;
        y = yy;
        z = zz;
    }
};

ostream &operator <<(ostream &out, vector &v)
{
```

```
    out<<v.x<<'\t'<<v.y<<'\t'<<v.z;
    return out;
}

vector operator*(double n, vector &v)
{
    vector r(n * v.x, n * v.y, n * v.z);
    return r;
}

vector operator-(vector &a, vector &b)
{
    vector r(a.x - b.x, a.y - b.y, a.z - b.z);
    return r;
}

vector operator+(vector &a, vector &b)
{
    vector r(a.x + b.x, a.y + b.y, a.z + b.z);
    return r;
}

class planet
{
public:
    vector *r = 0;
    vector *v = 0;
    vector a;
    double m;
    planet()
    {
        ;
    }
    ~planet()
    {
        if(r)
            delete []r;
        if(v)
            delete []v;
    }
    void initplanet(double m, double x, double y, double z, double vx, double vy, double vz, int n)
    {
        (*this).m = m;
        r = new vector[n + 1];
        v = new vector[n + 1];
        r[0].setvector(x, y, z);
        v[0].setvector(vx, vy, vz);
    }
};

void predictorcorrector(planet p[], const int i, const double dt, const int iteration, const int nplanet)
{
    for(int k = 0; k < iteration; k++)
    {
        for(int n = 0; n < nplanet; n++)
        {
            vector rp = p[n].v[i] + p[n].v[i + 1];
            rp = (dt / 2) * rp;
            rp = rp + p[n].r[i];
            vector a1(0, 0, 0);
            vector a2(0, 0, 0);
            for(int j = 0; j < nplanet; j++)
```

```
            {
                if(n != j)
                {
                    vector r1 = p[n].r[i] - p[j].r[i];
                    r1 = (p[j].m / pow(r1.length(), 3)) * r1;
                    a1 = a1 - r1;
                    vector r2 = p[n].r[i + 1] - p[j].r[i + 1];
                    r2 = (p[j].m / pow(r2.length(), 3)) * r2;
                    a2 = a2 - r2;
                }
            }
            vector vp = a1 + a2;
            vp = (dt / 2) * vp;
            vp = vp + p[n].v[i];
            p[n].r[i + 1] = rp;
            p[n].v[i + 1] = vp;
        }
    }
}

int main(int argc, char *argv[])
{
    if(argc == 1)
    {
        cout<<"Useage:\ttime number_of_planet planet_file correct_time [center_object]\n"<<endl;
        cout<<"\ttime:\t\t\tThe total simulation time.\n"<<endl;
        cout<<"\tnumber_of_planet:\tThe total number of planet to be simulated.\n"<<endl;
        cout<<"\tplanet_file:\t\tThe text file with initial values of all the planets."<<endl;
        cout<<"\t            \t\tEach line contains the information of one planet."<<endl;
        cout<<"\t            \t\tThe format needs to be:"<<endl;
        cout<<"\t            \t\tmass position_x position_y position_z velocity_x velocity_y
velocity_z\n"<<endl;
        cout<<"\t            \t\tThe program will not check the validity of data given."<<endl;
        cout<<"\tcorrect_time:\t\tThe time that predictor-corrector method is used\n"<<endl;
        cout<<"\t[central_object]:\tAn optional argument."<<endl;
        cout<<"\t                \tSet one of the planets as central body,"<<endl;
        cout<<"\t                \tand the relative position \"xr\" \"yr\" \"zr\","<<endl;
        cout<<"\t                \trelative velocity \"vxr\" \"vyr\" \"vzr\", "<<endl;
        cout<<"\t                \tand distance to central body \"dist\""<<endl;
        cout<<"\t                \tof other planets will be shown in the output file.\n"<<endl;
        return 0;
    }

    if(argc != 5 && argc != 6)
    {
        cerr<<"Wrong arguments"<<endl;
        return -1;
    }

    ifstream in(argv[3]);
    if(!in)
    {
        cerr<<"File not found"<<endl;
        return -1;
    }
    ofstream fout("planet_data.dat");
    const int ntime = atoi(argv[1]) / dt;
    const int nplanet = atoi(argv[2]);
    const int corrector = atoi(argv[4]);
    int center = 0;

    if(argc == 6)
```

```
    {
        center = atoi(argv[5]);
        if((center > nplanet) || (center < 0))
        {
            cerr<<"Wrong arguments"<<endl;
            return -1;
        }
    }

    planet *p = new planet[nplanet];

    for(int i = 0; i < nplanet; i++)
    {
        double m, x, y, z, vx, vy, vz;
        in>>m;
        in>>x;
        in>>y;
        in>>z;
        in>>vx;
        in>>vy;
        in>>vz;
        p[i].initplanet(m, x, y, z, vx, vy, vz, ntime);
    }

    for(int t = 0; t < ntime; t++)
    {
        for(int i = 0; i < nplanet; i++)
        {
            p[i].a.setvector(0, 0, 0);
            for(int j = 0; j < nplanet; j++)
            {
                if(i != j)
                {
                    vector r = p[i].r[t] - p[j].r[t];
                    r = (p[j].m / pow(r.length(), 3)) * r;
                    p[i].a = p[i].a - r;
                }
            }
        }
        for(int i = 0; i < nplanet; i++)
        {
            vector dr = dt * p[i].v[t];
            vector dv = dt * p[i].a;
            p[i].r[t + 1] = p[i].r[t] + dr;
            p[i].v[t + 1] = p[i].v[t] + dv;
        }
        predictorcorrector(p, t, dt, corrector, nplanet);
    }

    fout<<"Time\t";
    for(int i = 0; i < nplanet; i++)
    {
        fout<<"Planet"<<(i + 1)<<"x\tPlanet"<<(i + 1)<<"y\tPlanet"<<(i + 1)<<"z\tPlanet"<<(i +
1)<<"vx\tPlanet"<<(i + 1)<<"vy\tPlanet"<<(i + 1)<<"vz\t";
        if((center > 0) && (i != (center - 1)))
        {
            fout<<"Planet"<<(i + 1)<<"xr\tPlanet"<<(i + 1)<<"yr\tPlanet"<<(i + 1)<<"zr\tPlanet"<<(i +
1)<<"vxr\tPlanet"<<(i + 1)<<"vyr\tPlanet"<<(i + 1)<<"vzr\tPlanet"<<(i + 1)<<"dist\t";
        }
    }
    fout<<endl;
    for(int t = 0; t <= ntime; t+=1000)
```

```
    {
        fout<<t * dt<<"\t";
        for(int i = 0; i < nplanet; i++)
        {
            fout<<p[i].r[t]<<"\t"<<p[i].v[t]<<"\t";
            if((center > 0) && (i != (center - 1)))
            {
                vector rr = p[i].r[t] - p[center - 1].r[t];
                vector vr = p[i].v[t]-p[center - 1].v[t];
                fout<<rr<<"\t"<<vr<<"\t"<<rr.length()<<"\t";
            }
        }
        fout<<endl;
    }
    cout<<"Output file:\tplanet_data.dat"<<endl;
    delete []p;
    return 0;
}
```