PHYS580 Lab06 Report

Yicheng Feng
PUID: 0030193826

September 29, 2019

Workflow: I use the Linux system, and code C++ in terminals. For visualization, I use the C++ package – ROOT (made by CERN) to make plots. At the end, the reports are written in LaTeX.

The codes for this lab are written as the following files:

- `runge_kutta.h` and `runge_kutta.cxx` for the class `RungeKutta` to solve general ordinary differential equation sets.
- `planet_spin_orbit.h` and `planet_spin_orbit.cxx` for the class `PlanetSpinOrbit` to calculate ODE set of spinning and orbiting motion of Hyperion.
- `lab6.cxx` for the main function to make plots.

To each problem of this lab report, I will attach the relevant parts of the code. If you want to check the validation of my code, you need to download the whole code from the link `https://github.com/YichengFeng/phys580/tree/master/lab6`.

(1) Use the provided starter program `hyperion.m` (or your own equivalent program) to study the motion of Hyperion, one of Saturns moons, in the dumbbell model discussed in class. First, observe and display the orbital motion and the motion of the dumbbell axis. In particular, try different initial conditions for the Hyperion center of mass location and velocity, and the dumbbell axis orientation and angular velocity. Study both the case of a hypothetical circular orbit as well as a few (2-3) elliptic orbits with different eccentricities. Make sure to include the real orbit of Hyperion with perihelion $a(1-e) \approx 1.3 \times 10^6$ km $\equiv 1$ HU (Hyperion unit) and eccentricity $e = 0.123$. How would you characterize the nature of the spinning motion of Hyperion, based on your simulations? Does the kind of spinning motion depend on the type of orbit, and if yes, how?

Physics explanation:

We fix the major radius of the orbit $a = r_{\min}/(1-e) = 1/(1-0.123)$ HU $= 1.1403$ HU, and fix the initial condition $x(0) = r_{\min}$, $y(0) = 0$, $v_x(0) = 0$, and $v_y(0) = v_{\max}$. Then, we change the initial position and velocity of the mass center by inputing different eccentricities e.

$$r_{\min} = a(1-e), \qquad v_{\max} = 2\pi\sqrt{\frac{1+e}{a(1-e)}} = 2\pi\sqrt{1+e}. \tag{1}$$

We will use the values $e = 0, 0.123, 0.3, 0.6$.

For the initial conditions of the dumbbell axis orientation (θ) and auglar velocity (ω). We go through two sets $\theta(0) = 0$, $\omega(0) = 0$ and $\theta(0) = 0.2$ rad, $\omega(0) = 0.2$ rad/Hyr.

To reduce the error in calculation, we use the Runge-Kutta 4^{th} order approximation, with time step $\Delta t = 0.001$ Hyr.

This problem is similar as the pendulum with periodic driving force.

Except for the circular orbit ($e = 0$, Fig. 1 and Fig. 5), all elliptical orbits (Figs. 2, 3, 4, 6, 7, 8) have chaotic spin motions, no matter what the initial conditions are.

Yes, the kind of spinning motion depend on the type of orbit.

For the circular orbit, the angular velocity ω has clear periodic pattern depending on time. The orientation θ seems not periodic, because both the revolution and spin motion can affect the orientation. If we focus on the moments when θ doesn't change over time ($\omega = 0$), we can also find it periodic motion.

For the elliptical orbits, there are some time, in which the angular velocity ω oscillate around a high value quickly with small amplitude and the orientation θ therefore go through many periods quickly. With increasing e, the oscillation center of ω gets higher, and the relative amplitude gets smaller. There are some time when $\omega < 0$. Those dips will increase along e. The initial condition also seems to have effect on those dips.

Plots:

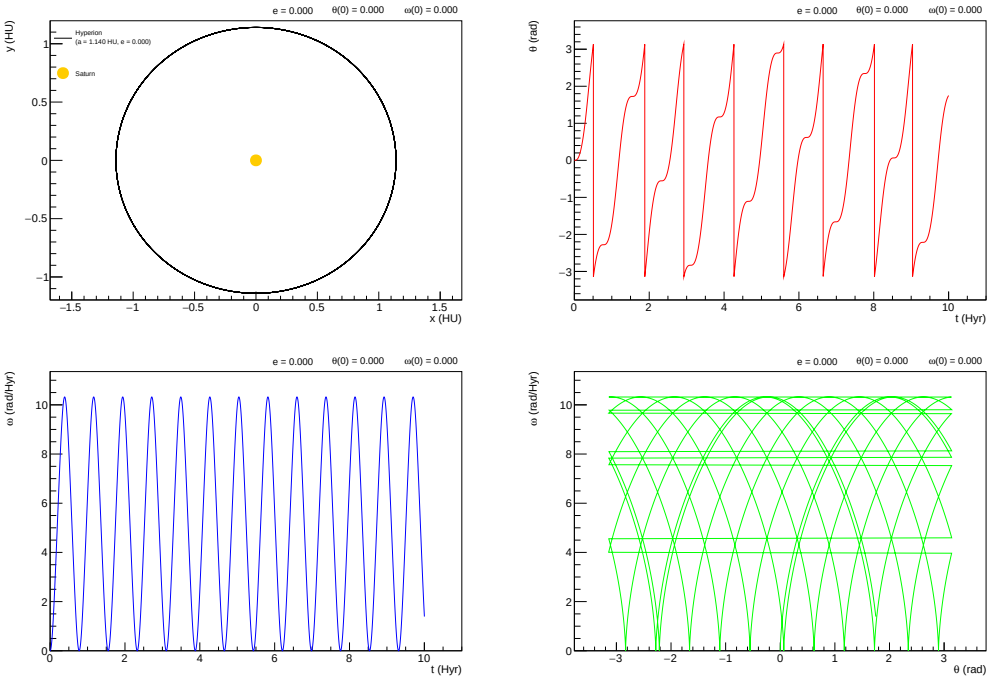

Figure 1: Hyperion circular orbit $e = 0$, with $\theta(0) = 0$, $\omega(0) = 0$

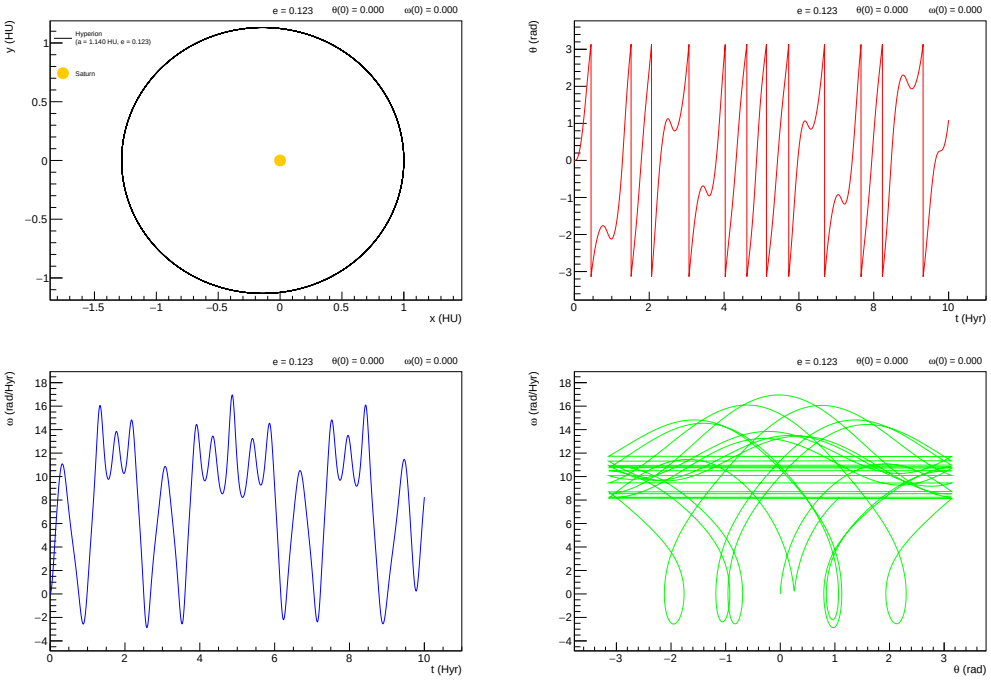

Figure 2: Hyperion real elliptical orbit $e = 0.123$, with $\theta(0) = 0$, $\omega(0) = 0$

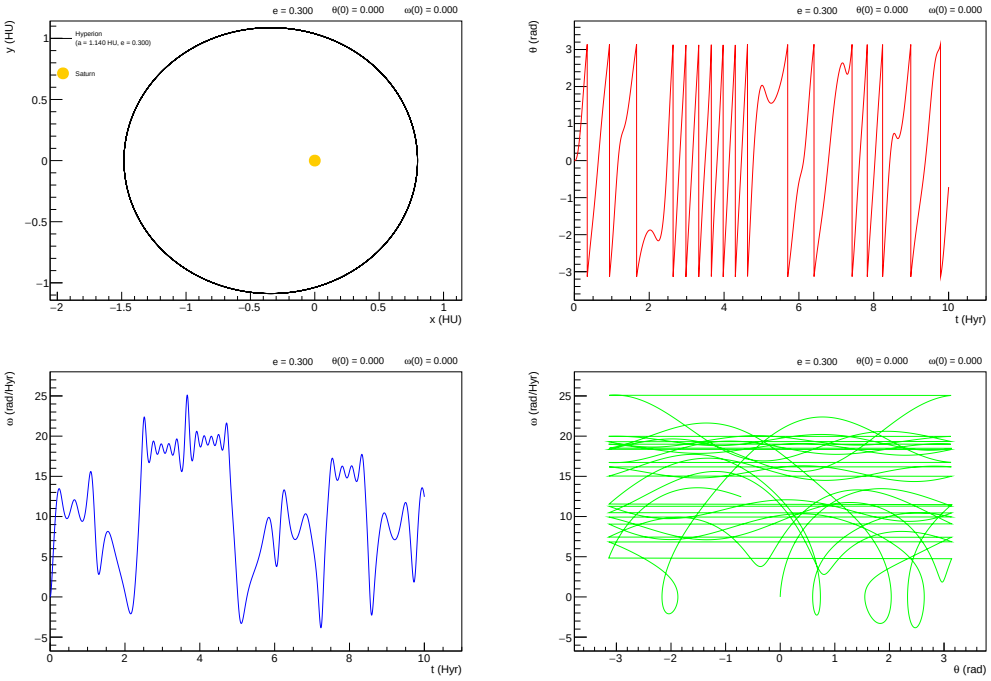

Figure 3: Hyperion elliptical orbit $e = 0.3$, with $\theta(0) = 0$, $\omega(0) = 0$

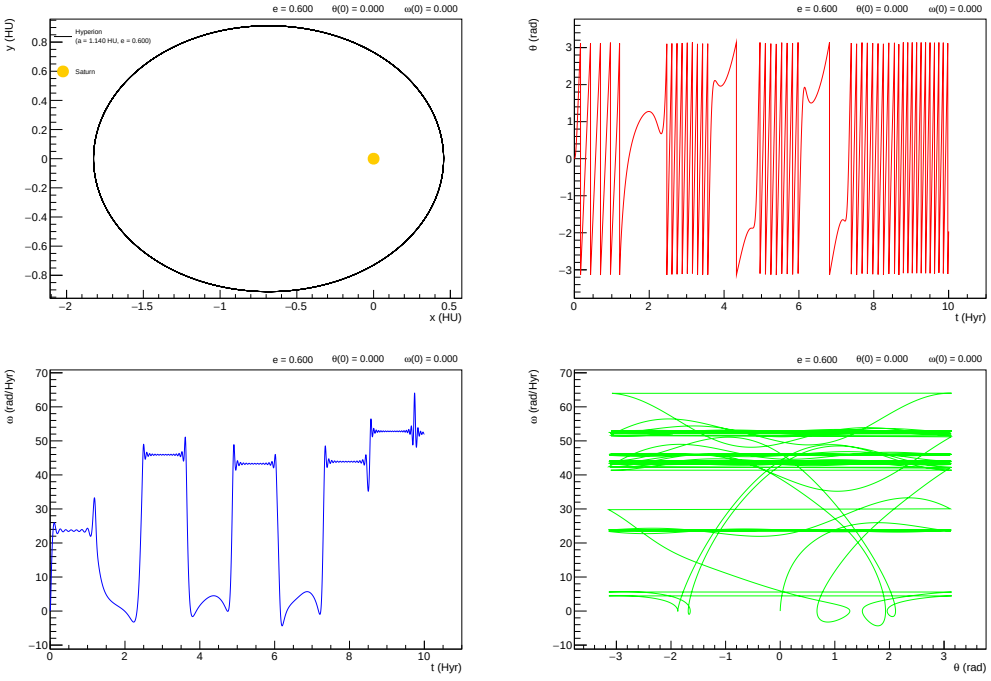

Figure 4: Hyperion elliptical orbit $e = 0.6$, with $\theta(0) = 0$, $\omega(0) = 0$

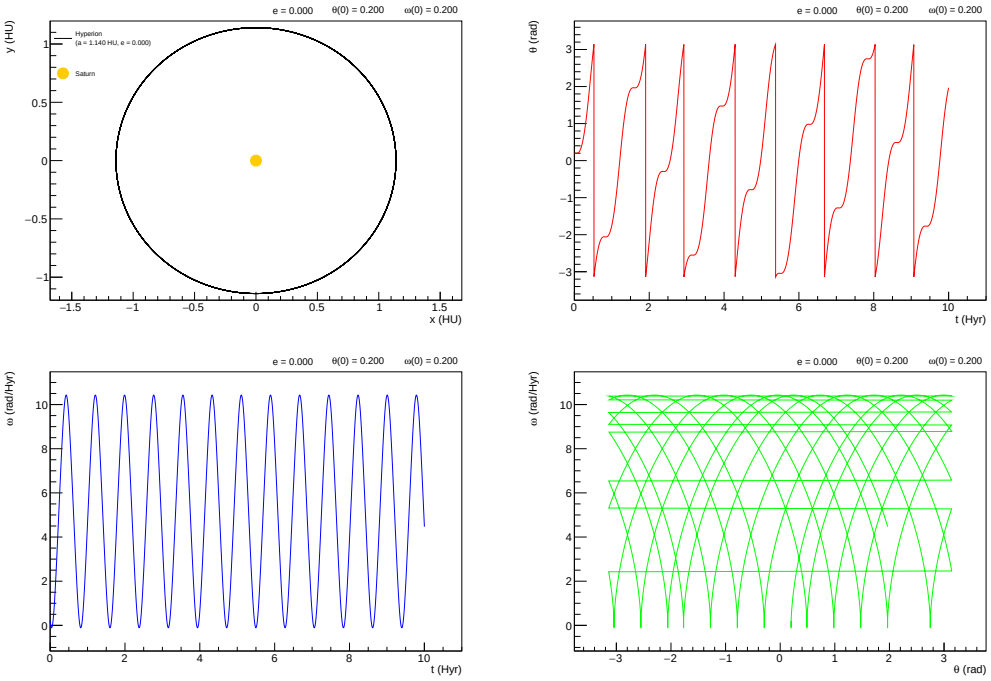

Figure 5: Hyperion circular orbit $e = 0$, with $\theta(0) = 0.2$, $\omega(0) = 0.2$

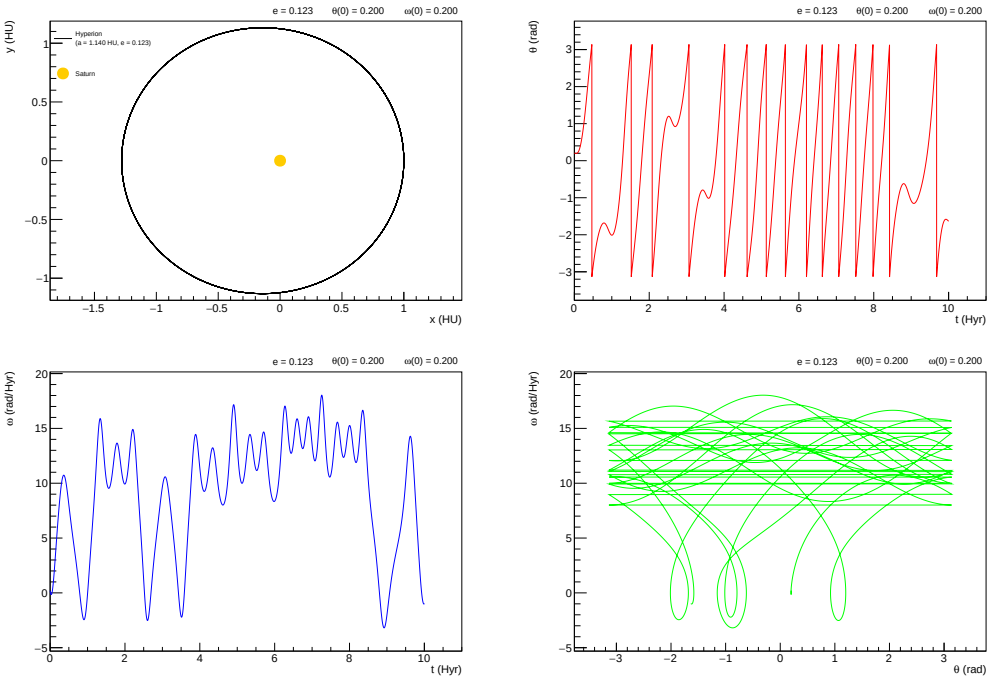

Figure 6: Hyperion real elliptical orbit $e = 0.123$, with $\theta(0) = 0.2$, $\omega(0) = 0.2$

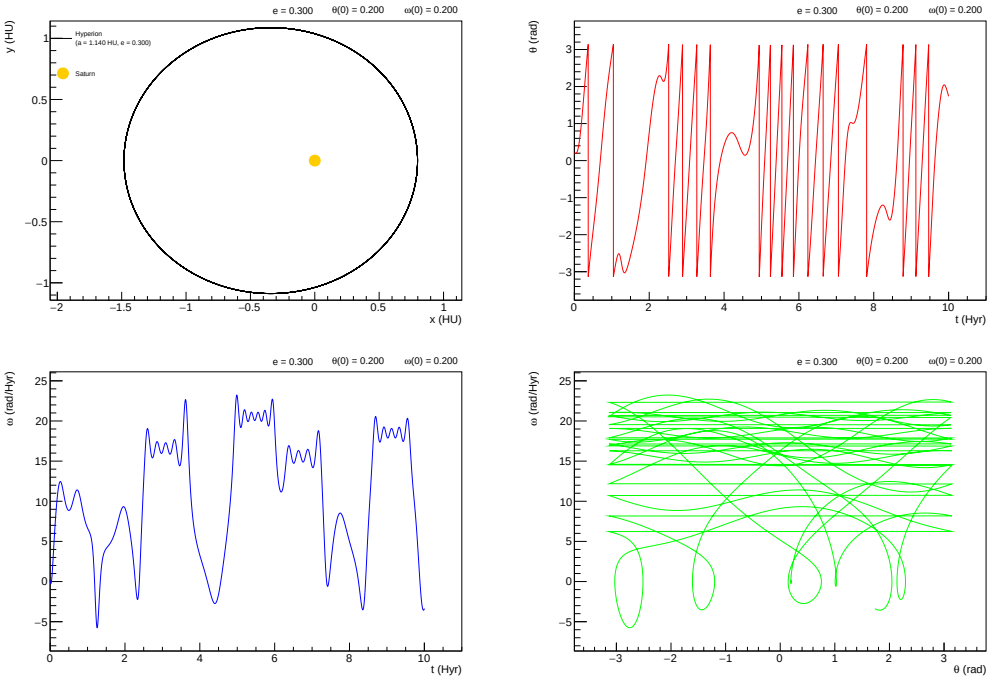

Figure 7: Hyperion elliptical orbit $e = 0.3$, with $\theta(0) = 0.2$, $\omega(0) = 0.2$

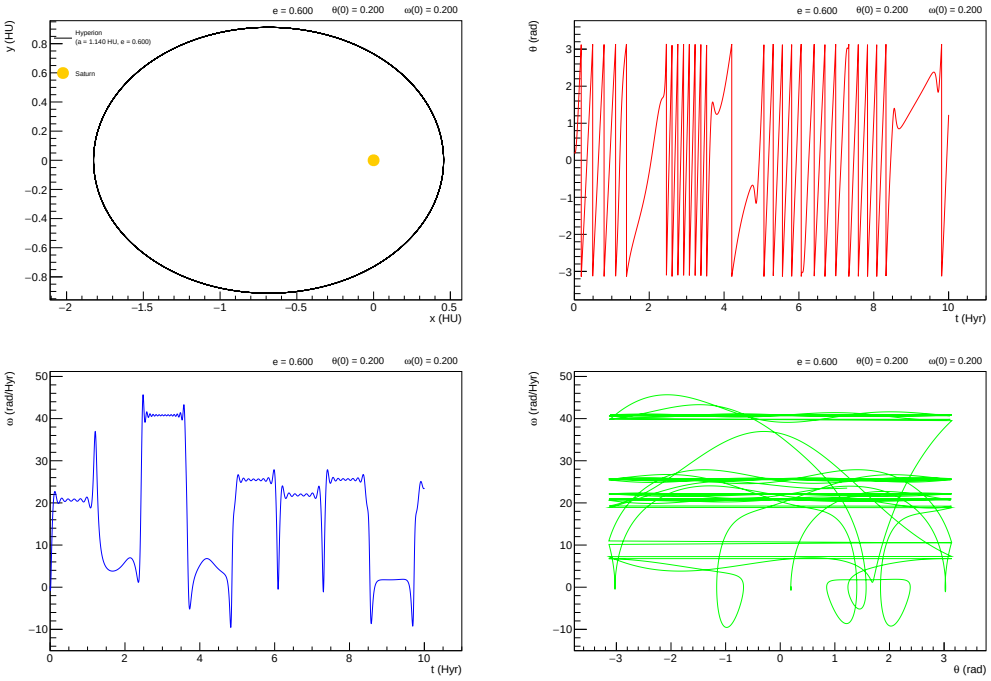

Figure 8: Hyperion elliptical orbit $e = 0.6$, with $\theta(0) = 0.2$, $\omega(0) = 0.2$

Relevant code:

For the Runge-Kutta 4$^{\text{th}}$ order approximation

```
1 //------------------------------------------------------------------------//
2
```

```
3  void RungeKutta::cal_rk4() {
4          if(!check() || !clear()) {
5                  cout << "WARNING: check() not passed, no calculation!" << endl;
6                  return;
7          }
8          //cout << "check() passed" << endl;
9
10         _n_stps = 0;
11
12         double t = _t_start;
13         vector<double> x = _x_start;
14
15         vector<double> F1(_n_eqns);
16         vector<double> F2(_n_eqns);
17         vector<double> F3(_n_eqns);
18         vector<double> F4(_n_eqns);
19
20         vector<double> x1(_n_eqns);
21         vector<double> x2(_n_eqns);
22         vector<double> x3(_n_eqns);
23         vector<double> x4(_n_eqns);
24
25         double t1, t2, t3, t4;
26
27         int nfv = _fv.size(); // nfv = _n_eqns
28         while(!_stop(t, x)) {
29                 // Fill into output
30                 _t.push_back(t);
31                 for(int ifv=0; ifv<nfv; ifv++) {
32                         _x[ifv].push_back(x[ifv]);
33                 }
34                 _n_stps ++;
35
36                 // F1
37                 t1 = t;
38                 for(int ifv=0; ifv<nfv; ifv++) {
39                         x1[ifv] = x[ifv];
40                 }
41                 for(int ifv=0; ifv<nfv; ifv++) {
42                         F1[ifv] = _fv[ifv](t1, x1);
43                 }
44                 // F2
45                 t2 = t + 0.5*_dt;
46                 for(int ifv=0; ifv<nfv; ifv++) {
47                         x2[ifv] = x[ifv] + F1[ifv]*0.5*_dt;
48                 }
49                 for(int ifv=0; ifv<nfv; ifv++) {
50                         F2[ifv] = _fv[ifv](t2, x2);
51                 }
52                 // F3
53                 t3 = t + 0.5*_dt;
54                 for(int ifv=0; ifv<nfv; ifv++) {
55                         x3[ifv] = x[ifv] + F2[ifv]*0.5*_dt;
56                 }
57                 for(int ifv=0; ifv<nfv; ifv++) {
58                         F3[ifv] = _fv[ifv](t3, x3);
59                 }
60                 // F4
61                 t4 = t + _dt;
62                 for(int ifv=0; ifv<nfv; ifv++) {
```

```
                        x4[ifv] = x[ifv] + F3[ifv]*_dt;
                }
                for(int ifv=0; ifv<nfv; ifv++) {
                        F4[ifv] = _fv[ifv](t4, x4);
                }

                t += _dt;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x[ifv] += (F1[ifv]/6 + F2[ifv]/3 + F3[ifv]/3 + F4[ifv]/6)*_dt;
                }
        }

        _t.push_back(t);
        for(int ifv=0; ifv<nfv; ifv++) {
                _x[ifv].push_back(x[ifv]);
        }
        _n_stps ++;
}

//------------------------------------------------------------------------//
```

For the ODE set of the orbiting and spinning motions

```
//------------------------------------------------------------------------//

double PlanetSpinOrbit::f_x(double t, const vector<double> &x) {
        return x[3];
}

//------------------------------------------------------------------------//

double PlanetSpinOrbit::f_y(double t, const vector<double> &x) {
        return x[4];
}

//------------------------------------------------------------------------//

double PlanetSpinOrbit::f_theta(double t, const vector<double> &x) {
        return x[5];
}

//------------------------------------------------------------------------//

double PlanetSpinOrbit::f_vx(double t, const vector<double> &x) {
        return -_GM*x[0]/pow(x[0]*x[0]+x[1]*x[1], 1.5);
}

//------------------------------------------------------------------------//

double PlanetSpinOrbit::f_vy(double t, const vector<double> &x) {
        return -_GM*x[1]/pow(x[0]*x[0]+x[1]*x[1], 1.5);
}

//------------------------------------------------------------------------//

double PlanetSpinOrbit::f_omega(double t, const vector<double> &x) {
        double domegadt = -3*_GM/pow(x[0]*x[0]+x[1]*x[1], 2.5);
        domegadt *= (x[0]*cos(x[2])+x[1]*sin(x[2]))*(x[0]*sin(x[2])-x[1]*cos(x[2]));
        return domegadt;
}

```

```
39 //------------------------------------------------------------------------//
```

For the initial conditions

```
1  //------------------------------------------------------------------------//
2
3  PlanetSpinOrbit::PlanetSpinOrbit() {
4          _a = 1/(1-0.123);
5          _e = 0;
6          _mp = 5.97e24/1.99e30;
7          _GM = 4*M_PI*M_PI;
8          _t_end = 5;
9
10         _t_start = 0;
11         _x_start = _a*(1-_e);
12         _y_start = 0;
13         _theta_start = 0;
14         _vx_start = 0;
15         //_vy_start = sqrt(_GM*(1+_e)/(1-_e)/_a*(1+_mp));
16         _vy_start = sqrt(_GM*(1+_e)/(1-_e)/_a);
17         _omega_start = 0;
18         _c_start.push_back(_x_start);
19         _c_start.push_back(_y_start);
20         _c_start.push_back(_theta_start);
21         _c_start.push_back(_vx_start);
22         _c_start.push_back(_vy_start);
23         _c_start.push_back(_omega_start);
24
25         _alg = 0;
26         _dt = 0.001;
27 }
28
29 //------------------------------------------------------------------------//
```

(2) Now modify `hyperion.m` (or create your own program) to study the "butterfly effect", i.e., follow the evolution from two slightly different initial conditions for the dumbbell axis. Track both the angular orientation difference and angular velocity difference for the same circular and elliptic orbits as in (1), including the real Hyperion orbit. Use the results to further substantiate the conclusions you reached in (1), and calculate the Lyapunov exponents for each case.

Physics explanation:

To reduce the error in calculation, we use the Runge-Kutta 4^{th} order approximation, with time step $\Delta t = 0.001$ Hyr. We focus on the initial conditions of the orientation and angular velocity $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr. To make the variations, we also calculate the cases with $\theta_1(0) = 0.2001$ rad, $\omega_1(0) = 0.2$ rad/Hyr, and $\theta_2(0) = 0.2$ rad, $\omega_2(0) = 0.2001$ rad/Hyr. Then, the differences $|\theta_1 - \theta_0|$, $|\omega_1 - \omega_0|$, $|\theta_2 - \theta_0|$, and $|\omega_2 - \omega_0|$ will be plotted along time in logarithm scale.

To get the Lyapunov exponents, we follow the instruction and get the local maximums of the $\Delta\theta$–t ($\Delta\omega$–t) curves, and use linear fitting (blue dashed lines) in the logarithm scale. The values of the Lyapunov exponents (relative to e) are listed in the following table.

Lyapunov exponent of	$\Delta\theta_{01}$	$\Delta\omega_{01}$	$\Delta\theta_{02}$	$\Delta\theta_{02}$
$e = 0$	0.1006	0.0997	0.0906	0.1223
$e = 0.123$	0.9621	0.9285	0.9591	0.9566
$e = 0.3$	1.5799	1.3366	1.5295	1.5071
$e = 0.6$	2.1350	1.6013	2.4219	2.4990

I also checked the results with smaller time step $\Delta t = 0.0005$ Hyr, the Lyapunov exponents don't change much, which means the results above are reasonable with respect to the calculation errors.

Plots:

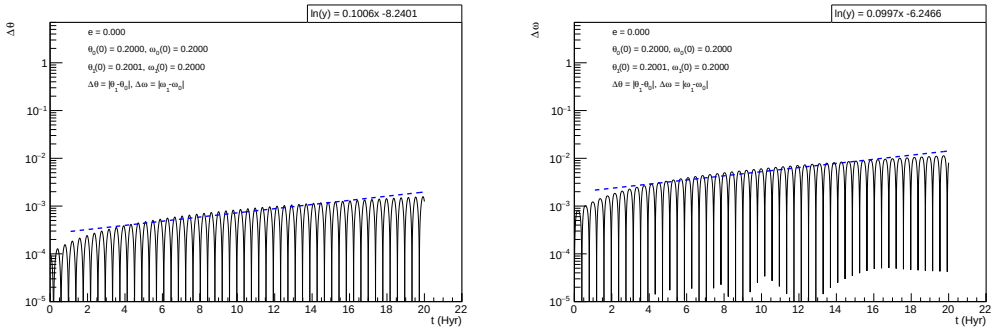

Figure 9: Hyperion circular orbit $e = 0$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_1(0) = 0.2001$ rad, $\omega_1(0) = 0.2$ rad/Hyr

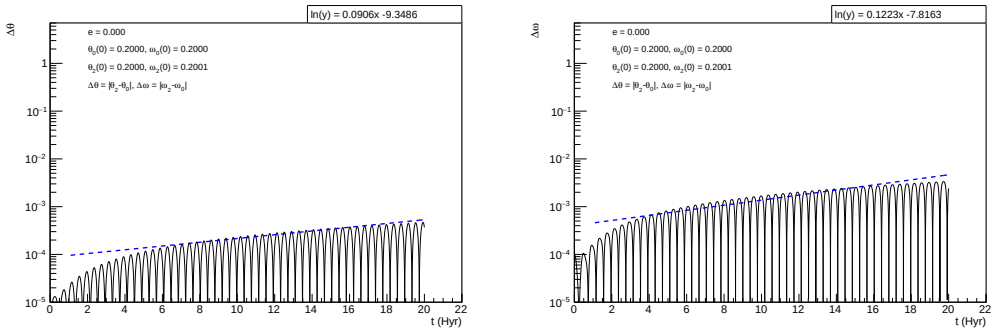

Figure 10: Hyperion circular orbit $e = 0$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_2(0) = 0.2$ rad, $\omega_2(0) = 0.2001$ rad/Hyr

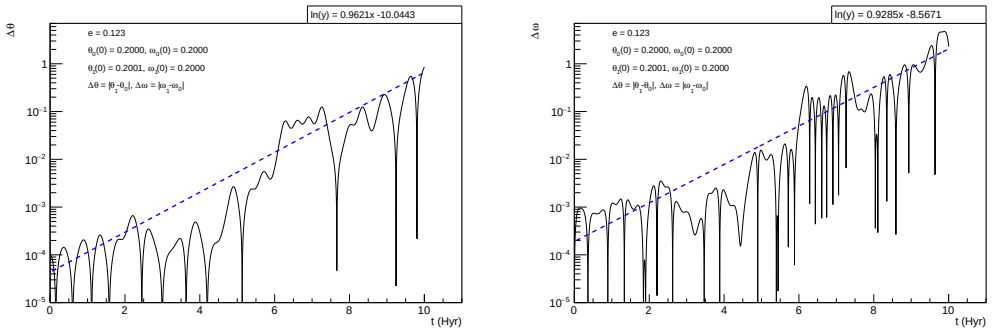

Figure 11: Hyperion real elliptical orbit $e = 0.123$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_1(0) = 0.2001$ rad, $\omega_1(0) = 0.2$ rad/Hyr

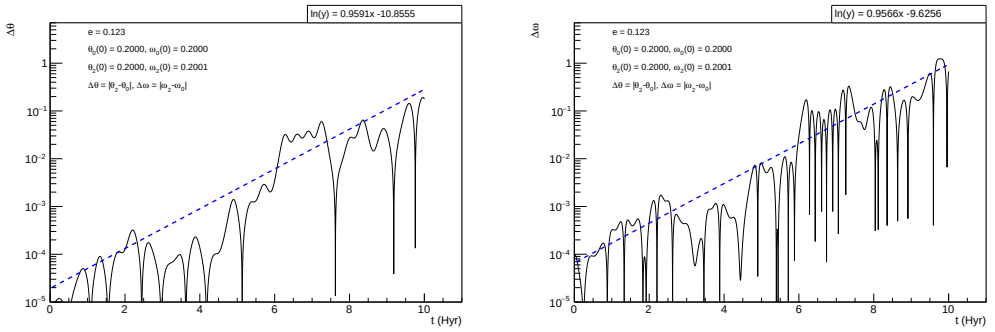

Figure 12: Hyperion real elliptical orbit $e = 0.123$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_2(0) = 0.2$ rad, $\omega_2(0) = 0.2001$ rad/Hyr

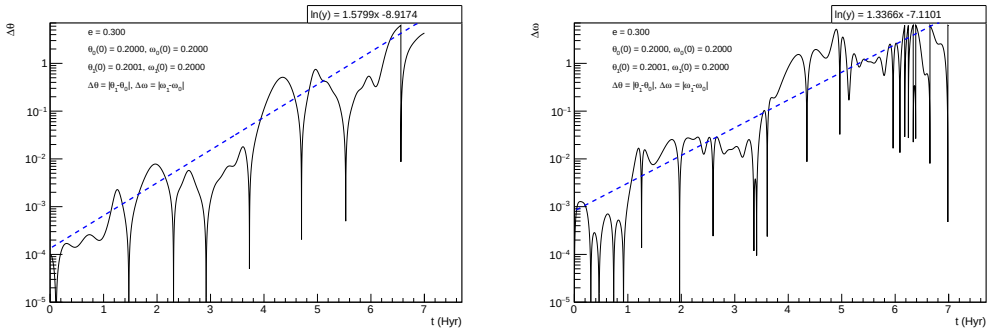

Figure 13: Hyperion elliptical orbit $e = 0.3$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_1(0) = 0.2001$ rad, $\omega_1(0) = 0.2$ rad/Hyr

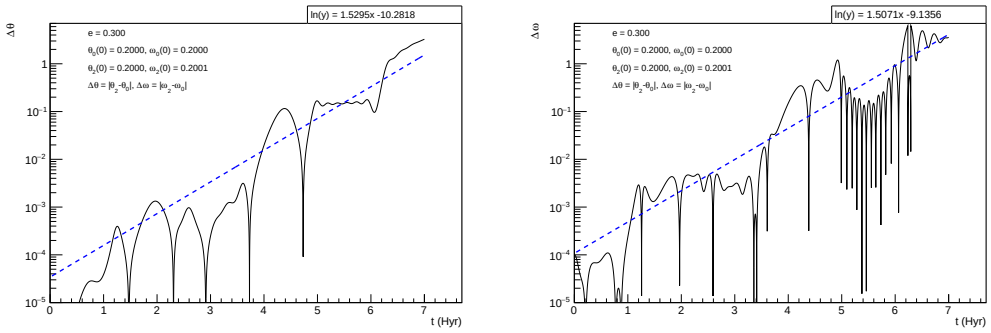

Figure 14: Hyperion elliptical orbit $e = 0.3$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_2(0) = 0.2$ rad, $\omega_2(0) = 0.2001$ rad/Hyr

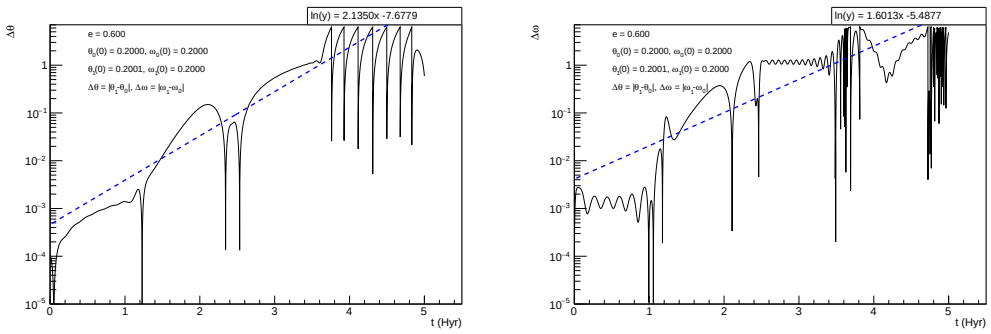

Figure 15: Hyperion elliptical orbit $e = 0.6$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_1(0) = 0.2001$ rad, $\omega_1(0) = 0.2$ rad/Hyr

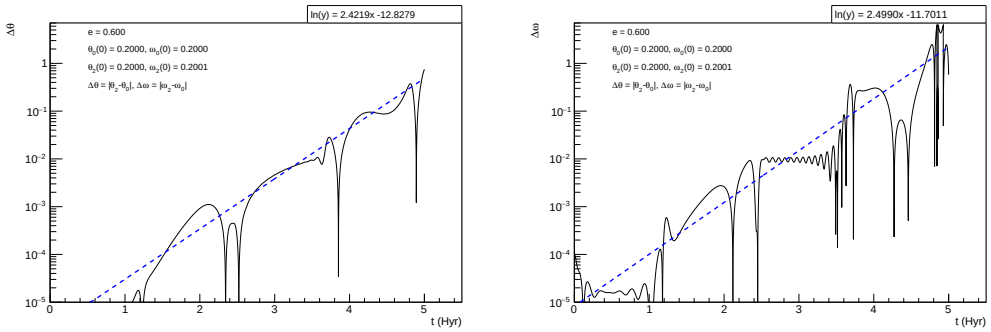

Figure 16: Hyperion elliptical orbit $e = 0.6$, difference between $\theta_0(0) = 0.2$ rad, $\omega_0(0) = 0.2$ rad/Hyr and $\theta_2(0) = 0.2$ rad, $\omega_2(0) = 0.2001$ rad/Hyr

Relevant code:

For the Runge-Kutta 4$^{\text{th}}$ approximation, spinning and orbiting motion ODE set, and initial condition setting, the relevant codes have already shown in problem (1).

For the different cases

```
//-------------------------------------------------------------------------------//

void butterfly_effect_plot(double e, double t_end, double fit_t_start, double fit_t_end) {

        TString str_e = Form("e%.3d", int(e*100));

        double a = 1/(1-0.123);
        double r_min = a*(1-e);
        double r_max = a*(1+e);
        double f = a*e;
        double b = sqrt(a*a - f*f);

        double theta_start0 = 0.2;
        double omega_start0 = 0.2;
        PlanetSpinOrbit pso0(a, e, 0);
        pso0.set_alg(G_ALG);
        pso0.set_t_end(t_end);
        pso0.set_theta_start(theta_start0);
        pso0.set_omega_start(omega_start0);
        pso0.cal();
        vector<double> t0 = pso0.get_t();
        vector< vector<double> > x0 = pso0.get_x();
        int n_stps0 = pso0.get_n_stps();
        //theta_to_pipi(x0[2]);

        double theta_start1 = 0.2001;
        double omega_start1 = 0.2;
        PlanetSpinOrbit pso1(a, e, 0);
        pso1.set_alg(G_ALG);
        pso1.set_t_end(t_end);
        pso1.set_theta_start(theta_start1);
        pso1.set_omega_start(omega_start1);
        pso1.cal();
        vector<double> t1 = pso1.get_t();
        vector< vector<double> > x1 = pso1.get_x();
        int n_stps1 = pso1.get_n_stps();
        //theta_to_pipi(x1[2]);

```

```
        double theta_start2 = 0.2;
        double omega_start2 = 0.2001;
        PlanetSpinOrbit pso2(a, e, 0);
        pso2.set_alg(G_ALG);
        pso2.set_t_end(t_end);
        pso2.set_theta_start(theta_start2);
        pso2.set_omega_start(omega_start2);
        pso2.cal();
        vector<double> t2 = pso2.get_t();
        vector< vector<double> > x2 = pso2.get_x();
        int n_stps2 = pso2.get_n_stps();
        //theta_to_pipi(x2[2]);

        // ... other code ...

        vector<double> dtheta01;
        vector<double> domega01;
        vector<double> dtheta02;
        vector<double> domega02;
        for(int i=0; i<n_stps0; i++) {
                dtheta01.push_back(fabs(x1[2][i]-x0[2][i]));
                domega01.push_back(fabs(x1[5][i]-x0[5][i]));
                dtheta02.push_back(fabs(x2[2][i]-x0[2][i]));
                domega02.push_back(fabs(x2[5][i]-x0[5][i]));
        }
        theta_to_twopi(dtheta01);
        theta_to_twopi(domega01);
        theta_to_twopi(dtheta02);
        theta_to_twopi(domega02);

        // ... plot code ...
}

//------------------------------------------------------------------------//
```

For the fitting to get Lyapunov exponent

```
//------------------------------------------------------------------------//

TF1* fit_Lyapunov_exp(const vector<double> &t, const vector<double> &x, double t_start, double t_end)

        vector<double> tlm;
        vector<double> xlm;
        int n = 0;
        for(int i=1; i<t.size()-1; i++) {
                if(t[i]<t_start || t[i]>t_end) continue;
                if(x[i]<x[i-1] || x[i]<x[i+1]) continue;
                if(x[i]<1e-4) continue;
                //cout << n << "   " << t[i] << "   " << x[i] << endl;
                tlm.push_back(t[i]);
                xlm.push_back(log(x[i]));
                n ++;
        }

        TGraph *g = new TGraph(n, &tlm[0], &xlm[0]);
        //g->SetMarkerColor(kRed);
        //g->Draw("same P");

        gStyle->SetOptFit();
        TF1 *line = new TF1("line", "[0] + [1]*x", t_start, t_end);
        g->Fit(line);
```

```
25
26          TF1 *expo = new TF1("expo", "exp([0] + [1]*x)", t_start, t_end);
27          expo->SetLineColor(kBlue);
28          expo->SetLineStyle(7);
29          expo->SetParameters(line->GetParameters());
30
31          TLegend *l_fit_para = new TLegend(0.6,0.9,0.9,0.95);
32          l_fit_para->SetHeader(Form("ln(y) = %.4fx %+.4f", line->GetParameter(1),
33          line->GetParameter(0)), "");
34          l_fit_para->Draw("same");
35
36          return expo;
37  }
38
39  //--------------------------------------------------------------------------//
```