PHYS580 Lab05 Report

Yicheng Feng
PUID: 0030193826

September 23, 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_orbit.h` and `planet_orbit.cxx` for the class `PlanetOrbit` to calculate ODE set of the system composed of one planet and one stationary star.
- `two_planet_orbit.h` and `two_planet_orbit.cxx` for the class `TwoPlanetOrbit` to calculate ODE set of the system composed of two planets and one stationary star.
- `three_body_2d.h` and `three_body_2d.cxx` for the class `ThreeBody2D` to calculate ODE set of the system composed of three objects in moving in the same plane.
- `lab5.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/lab5`.

(1) Observe and display motion along circular, parabolic, and hyperbolic orbits in our Solar System, using the provided starter program (planet_EulerCromer.m) or your own equivalent. First set initial conditions that would match the orbit of one of the solar system planets, and then calculate the (real) orbit of that planet. Next, change the initial velocity to take the planet to a hypothetical new orbit of a different type (if you start from a planet with a significantly elliptical orbit, such as Pluto or Mercury, this could be more challenging). Make sure to first theoretically estimate the necessary initial velocities that would be needed for circular and parabolic orbits.

Physics explanation:

As instructed, we use the Euler-Cromer approximation to calculate the orbit of Mercury with time step $\Delta t = 0.001$ AU. From the table, some of the parameters of Mercury are eccentricity $e = 0.206$, major radius $a = (0.307 + 0.467)/2 = 0.387$ AU, and mass $M_P = 3.30 \times 10^{23} = 1.66 \times 10^{-7} M_S$. Then, we can use the given equations for the minimal and maximal velocity

$$v_{\min} = \sqrt{GM_S}\sqrt{\frac{1-e}{a(1+e)}\left(1+\frac{M_P}{M_S}\right)}, \qquad v_{\max} = \sqrt{GM_S}\sqrt{\frac{1+e}{a(1-e)}\left(1+\frac{M_P}{M_S}\right)}. \tag{1}$$

We set the planet at the perihelion $r_{\min}$ and with maximal velocity $v_{\max}$ as the initial state, and we put this on the positive x-axis with velocity along the positive y-axis. Then, the initial conditions can be written as

$$\begin{aligned} x(0) =& r_{\min} = 0.307 \text{ AU}, \\ y(0) =& 0, \\ v_x(0) =& 0, \\ v_y(0) =& v_{\max} = \sqrt{4\pi^2}\sqrt{\frac{1+0.206}{0.387(1-0.206)}(1+1.66\times 10^{-7})} = 12.45 \text{ AU/yr.} \end{aligned} \tag{2}$$

We can use the initial above to calculate the **real** orbit of Mercury. (Fig. 1) We also calculate the Mercury's period from our numerical calculation $T = 0.2407$ yr, which is consistent with the given value in the table.

To get the **circular** orbit of Mercury from the same initial position, we need to change the initial velocity. The gravity fully provides the centripetal force

$$\frac{GM_SM_P}{r^2} = M_P\frac{v^2}{r} \quad \Rightarrow \quad v_y(0) = v(0) = \sqrt{\frac{2GM_S}{r_{\min}}} = \sqrt{\frac{\times 4\pi^2}{0.307}} \text{ AU/yr} = 11.34 \text{ AU/yr.} \tag{3}$$

We use put the initial conditions $x(0) = 0.307$ AU, $y(0) = 0$, $v_x(0) = 0$, and $v_y(0) = 11.34$ AU/yr into the Euler-Cromer approximation, and get the circular trajectory as shown in the left pad of Fig. 2.

To get the **parabola** trajectory of Mercury from the same initial position, we need to increase its initial velocity. At this critical point, the total energy of the system is 0.

$$0 = E = \frac{1}{2}M_Pv^2 - \frac{GM_SM_P}{r} \quad \Rightarrow \quad v_y(0) = v(0) = \sqrt{\frac{2GM_S}{r_{\min}}} = \sqrt{\frac{2\times 4\pi^2}{0.307}} \text{ AU/yr} = 16.04 \text{ AU/yr} \tag{4}$$

The parabola trajectory of Mercury is shown in the middle pad of Fig. 2.

To get the **hyperbola** trajectory of Mercury, we continue increasing its initial velocity to $v_y(0) = 17.64$ AU/yr, greater than the escape velocity calculated above. The hyperbola trajectory is shown in the right pad of Fig. 2.

Plots:

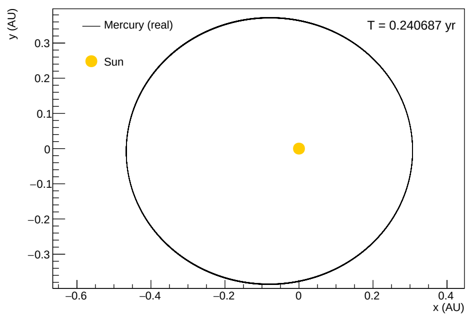

Figure 1: The Euler-Cromer approximation for the real orbit of Mercury.

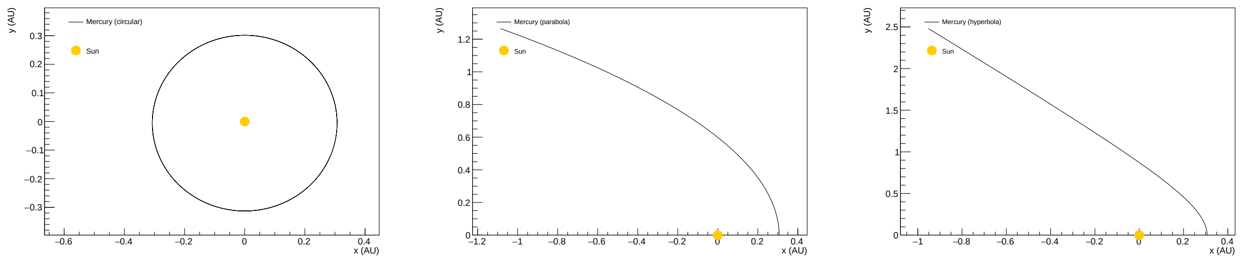

Figure 2: The Euler-Cromer approximation for the parabola (left pad) and hyperbola (right pad) orbits of Mercury.

Relevant code:

For the Euler-Cromer approximation

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

void RungeKutta::cal_Euler_Cromer() {
        if(!check() || !clear()) {
                cout << "WARNING: check() not passed, no calculation!" << endl;
                return;
        }
        //cout << "check() passed" << endl;

        _n_stps = 0;

        double t = _t_start;
        vector<double> x = _x_start;

        vector<double> F1(_n_eqns);

        int nfv = _fv.size(); // nfv = _n_eqns
        while(!_stop(t, x)) {
                // Fill into output
                _t.push_back(t);
                for(int ifv=0; ifv<nfv; ifv++) {
                        _x[ifv].push_back(x[ifv]);
                }
                _n_stps ++;

                // F1
                // update x right after each calculation
                //for(int ifv=0; ifv<nfv; ifv++) {
                for(int ifv=nfv-1; ifv>=0; ifv--) {
                        F1[ifv] = _fv[ifv](t, x);
                        x[ifv] += F1[ifv]*_dt;
                }
                t += _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 planetary motion

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

double PlanetOrbit::f_x(double t, const vector<double> &x) {
        return x[2];
}

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

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

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

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

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

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

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

For the initial condiions of the planetary motion

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

PlanetOrbit::PlanetOrbit(double a, double e, double mp) {
        _a = a;
        _e = e;
        _mp = mp;
        _GM = 4*M_PI*M_PI;
        _t_end = 5;

        _t_start = 0;
        _x_start = _a*(1-_e);
        _y_start = 0;
        _vx_start = 0;
        _vy_start = sqrt(_GM*(1+_e)/(1-_e)/_a*(1+_mp));
        _c_start.push_back(_x_start);
        _c_start.push_back(_y_start);
        _c_start.push_back(_vx_start);
        _c_start.push_back(_vy_start);

        _energy = 0.5*_mp*(_vy_start*_vy_start + _vx_start*_vx_start);
        _energy += -_GM*_mp/sqrt(_x_start*_x_start + _y_start*_y_start);

        _alg = 0;
        _dt = 0.001;
}

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

For the change of the initial velocity

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

void PlanetOrbit::increase_v_start(double r) {
        _t_end = 1;
        _vy_start = sqrt(2*_GM/_x_start)*r;
        _c_start[3] = _vy_start*r;
}

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

(2) Repeat the orbit calculation but with values appropriate for a comet - specifically, the Shoemaker-Levy 2 that has a perihelion of 1.933 AU and eccentricity of 0.572. Obtain its period by keeping track of time in your simulation, and crosscheck against the theoretical prediction and also, if you can, against actual measurements. What about Halley's comet with perihelion of 0.589 AU and eccentricity of 0.967? Do these two comets follow Keplers third law, T^2/a^3 = const., according to your simulations?

Physics explanation:

For the Shoemaker-Levy 2, we have $r_{\text{min}} = 1.933$ AU and $e = 0.572$.

$$a = r_{\text{min}}/(1-e) = 1.933/(1-0.572) \text{ AU} = 4.516 \text{ AU}, \tag{5}$$

$$T = 2\pi\sqrt{\frac{a^3}{GM_S}} = 2\pi\sqrt{\frac{4.516^3}{4\pi^2}} = 9.59803 \text{ yr}. \tag{6}$$

The numerical result is show in the left pad of Fig. 3, and the calculated period is $T_n = 9.598111$ yr. The relative difference between the theoratical and numerical calculation is $|T_n - T|/T = 8.4 \times 10^{-6}$. The period from observation is $T_e = 9.60$ yr by Wikipedia, so the relative difference between the observation and numerical calculation is $|T_n - T_e|/T_e = 2.0 \times 10^{-4}$.

For the Halley's comet, we have $r_{\text{min}} = 0.589$ AU and $e = 0.967$.

$$a = r_{\text{min}}/(1-e) = 0.589/(1-0.967) \text{ AU} = 17.85 \text{ AU}, \tag{7}$$

$$T = 2\pi\sqrt{\frac{a^3}{GM_S}} = 2\pi\sqrt{\frac{17.85^3}{4\pi^2}} = 75.40533 \text{ yr}. \tag{8}$$

The numerical result is show in the right pad of Fig. 3, and the calculated period is $T_n = 75.733986$ yr. The relative difference between the theoratical and numerical calculation is $|T_n - T|/T = 4.4 \times 10^{-3}$. The period from observation is $T_e = 75.32$ yr by Wikipedia, so the relative difference between the observation and numerical calculation is $|T_n - T_e|/T_e = 5.5 \times 10^{-3}$.

From the small difference between periods from theoratical and numerical calculations, we can see the Kepler's third law is satisfied. We can also calculate the ratio T^2/a^3. For the Shoemaker Levy 2 comet, the simulation gives $T_n^2/a^3 = 9.598111^2/4.516^3 = 1.0002527$. For the Halley's comet, the simulation gives $T_n^2/a^3 = 75.733986^2/17.85^3 = 1.0084793$. As expected, the two values are close and consistent to 1.

Plots:

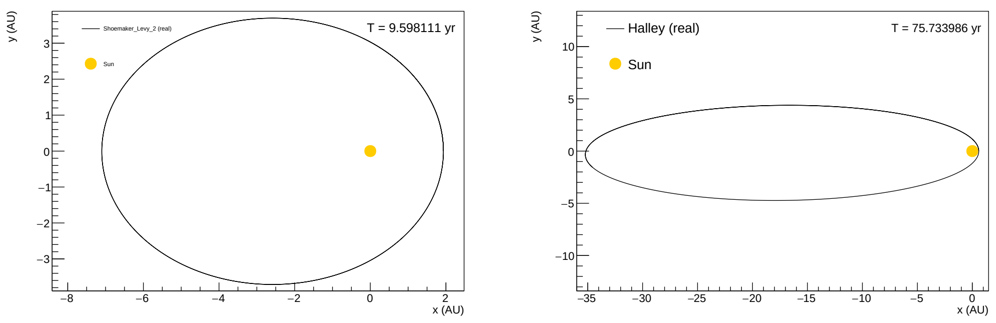

Figure 3: The Euler-Cromer approximation for the orbit of the Shoemaker Levy 2 comet (left pad) and the Halley's comet (right pad).

Relevant code:

For the Euler-Cromer approximation

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

void RungeKutta::cal_Euler_Cromer() {
        if(!check() || !clear()) {
                cout << "WARNING: check() not passed, no calculation!" << endl;
                return;
        }
        //cout << "check() passed" << endl;

        _n_stps = 0;

        double t = _t_start;
        vector<double> x = _x_start;

        vector<double> F1(_n_eqns);

        int nfv = _fv.size(); // nfv = _n_eqns
        while(!_stop(t, x)) {
                // Fill into output
                _t.push_back(t);
                for(int ifv=0; ifv<nfv; ifv++) {
                        _x[ifv].push_back(x[ifv]);
                }
                _n_stps ++;

                // F1
                // update x right after each calculation
                //for(int ifv=0; ifv<nfv; ifv++) {
                for(int ifv=nfv-1; ifv>=0; ifv--) {
                        F1[ifv] = _fv[ifv](t, x);
                        x[ifv] += F1[ifv]*_dt;
                }
                t += _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 planetary motion

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

double PlanetOrbit::f_x(double t, const vector<double> &x) {
        return x[2];
}

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

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

```

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

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

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

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

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

For the initial condiions of the planetary motion

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

PlanetOrbit::PlanetOrbit(double a, double e, double mp) {
        _a = a;
        _e = e;
        _mp = mp;
        _GM = 4*M_PI*M_PI;
        _t_end = 5;

        _t_start = 0;
        _x_start = _a*(1-_e);
        _y_start = 0;
        _vx_start = 0;
        _vy_start = sqrt(_GM*(1+_e)/(1-_e)/_a*(1+_mp));
        _c_start.push_back(_x_start);
        _c_start.push_back(_y_start);
        _c_start.push_back(_vx_start);
        _c_start.push_back(_vy_start);

        _energy = 0.5*_mp*(_vy_start*_vy_start + _vx_start*_vx_start);
        _energy += -_GM*_mp/sqrt(_x_start*_x_start + _y_start*_y_start);

        _alg = 0;
        _dt = 0.001;
}

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

For the change of the initial velocity

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

void PlanetOrbit::increase_v_start(double r) {
        _t_end = 1;
        _vy_start = sqrt(2*_GM/_x_start)*r;
        _c_start[3] = _vy_start*r;
}

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

(3) Modify the starter program (`two_planets.m`, or create your own equivalent) to observe and display how planet 2 affects (perturbs) the orbital motion of planet 1. The starter program assumes that the Sun is stationary, which is fine when the planets are much less massive than the Sun (true for the Solar System). Find out how much more massive Jupiter would have to be for its influence to make Earth's orbit visibly precess and eventually make it non-periodic, or even eject(!) Earth from the Solar System. Jupiters orbit may be approximated as circular with radius 5.20 AU (the actual eccentricity is only about 0.048). Then, verify how the result changes, if at all, if you properly treat all 3 bodies (Sun, Earth, Jupiter) as mobile.

Physics explanation:

We still use the **elliptic orbit** of the Earth and Jupiter to calculate the initial conditions of the two planets. We use the Euler-Cromer approximation with time step $\Delta t = 0.001$ yr, and evolve the system to 20 years.

First, we set the Sun stationary, and increase the mass of Jupiter from its real mass $\times 1$ to 10, 100, 500, 1000 times. The results are shown in Fig. 4. From $\times 1$ to $\times 500$ Jupiter mass, the Earth has precession with increasing deviations (still periodic), because of the interaction with Jupiter. When $\times 1000$ Jupiter mass, the motion of the Earth seems chaotic (non-periodic), and then "ejected" out of the system. However, this may not be right in physics, because here the Jupiter mass is too huge (nearly equal to the Sun mass), the Sun must be considered mobile.

Second, we set the Sun also movable, and increase the mass of Jupiter from its real mass $\times 1$ to 10, 100, 500, 1000 times. The results are shown in Fig. 5. The system is put into the frame of the mass center, and the total momentum is 0. The origin point $(0, 0)$ is the mass center. From $\times 1$ to $\times 100$ Jupiter mass, the Earth has precession with increasing deviations (still periodic), because of the interaction with Jupiter. When $\times 500$ and $\times 1000$ Jupiter mass, the motion of the Earth seems chaotic (non-periodic), and then "ejected" out of the system, which seems a bit more reasonable.

Plots:

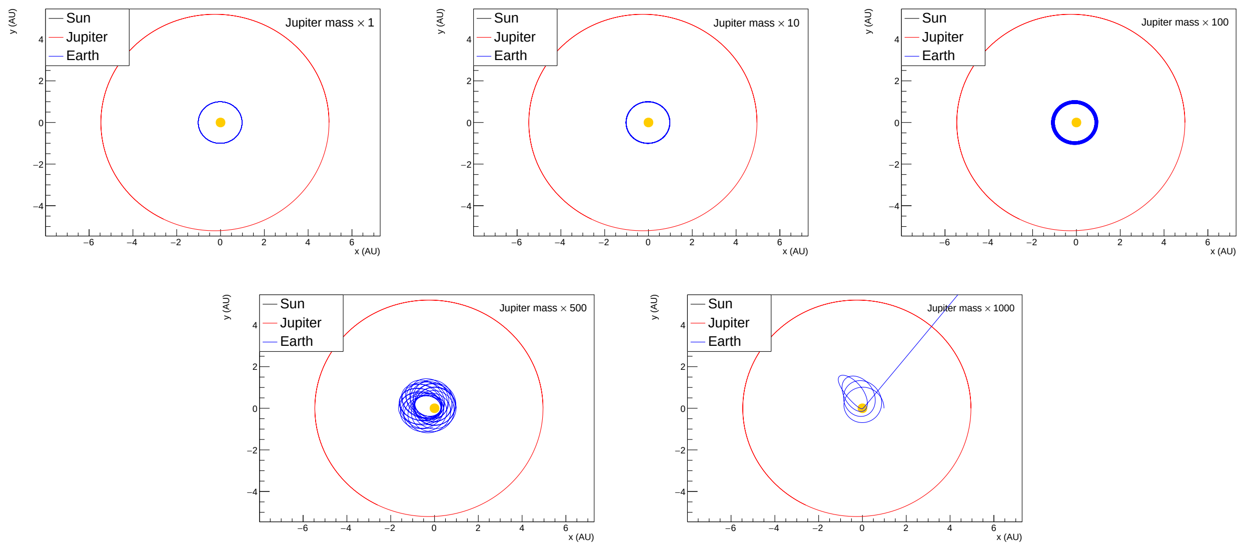

Figure 4: The Sun is set stationary. The plots show the evolution of the Jupiter and Earth with different Jupiter mass (×1, 10, 100, 500, 1000).

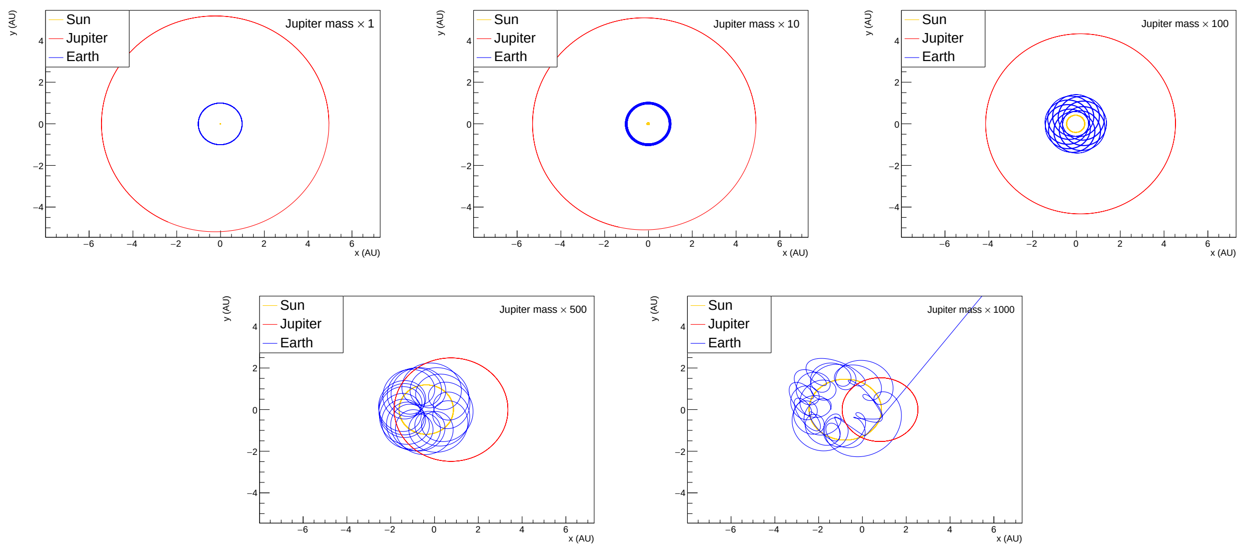

Figure 5: The plots show the three body system (2D case), the evolution of the Sun, Jupiter, and Earth with different Jupiter mass (×1, 10, 100, 500, 1000).

Relevant code:

For the Euler-Cromer approximation

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

void RungeKutta::cal_Euler_Cromer() {
        if(!check() || !clear()) {
                cout << "WARNING: check() not passed, no calculation!" << endl;
                return;
        }
        //cout << "check() passed" << endl;

        _n_stps = 0;

        double t = _t_start;
        vector<double> x = _x_start;

        vector<double> F1(_n_eqns);

        int nfv = _fv.size(); // nfv = _n_eqns
        while(!_stop(t, x)) {
                // Fill into output
                _t.push_back(t);
                for(int ifv=0; ifv<nfv; ifv++) {
                        _x[ifv].push_back(x[ifv]);
                }
                _n_stps ++;

                // F1
                // update x right after each calculation
                //for(int ifv=0; ifv<nfv; ifv++) {
                for(int ifv=nfv-1; ifv>=0; ifv--) {
                        F1[ifv] = _fv[ifv](t, x);
                        x[ifv] += F1[ifv]*_dt;
                }
                t += _dt;
        }

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

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

For the stationary Sun, the ODE set of Earth-Jupiter system

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

double TwoPlanetOrbit::f_x1(double t, const vector<double> &x) {
        return x[4];
}

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

double TwoPlanetOrbit::f_y1(double t, const vector<double> &x) {
        return x[5];
}

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

double TwoPlanetOrbit::f_x2(double t, const vector<double> &x) {
        return x[6];
}

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

double TwoPlanetOrbit::f_y2(double t, const vector<double> &x) {
        return x[7];
}

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

double TwoPlanetOrbit::f_vx1(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        return -_GM*x[0]/pow(x[0]*x[0]+x[1]*x[1], 1.5) - _GM*_m2*(x[0]-x[2])/rEJ/rEJ/rEJ;
}

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

double TwoPlanetOrbit::f_vy1(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        return -_GM*x[1]/pow(x[0]*x[0]+x[1]*x[1], 1.5) - _GM*_m2*(x[1]-x[3])/rEJ/rEJ/rEJ;
}

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

double TwoPlanetOrbit::f_vx2(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        return -_GM*x[2]/pow(x[2]*x[2]+x[3]*x[3], 1.5) - _GM*_m1*(x[2]-x[0])/rEJ/rEJ/rEJ;
}

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

double TwoPlanetOrbit::f_vy2(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        return -_GM*x[3]/pow(x[2]*x[2]+x[3]*x[3], 1.5) - _GM*_m1*(x[3]-x[1])/rEJ/rEJ/rEJ;
}

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

For the mobile Sun, the ODE set of the Earth-Jupiter-Sun 2D three-body system

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

double ThreeBody2D::f_x1(double t, const vector<double> &x) {
        return x[6];
}

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

double ThreeBody2D::f_y1(double t, const vector<double> &x) {
        return x[7];
}

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

double ThreeBody2D::f_x2(double t, const vector<double> &x) {
        return x[8];
}

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

double ThreeBody2D::f_y2(double t, const vector<double> &x) {
        return x[9];
}

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

double ThreeBody2D::f_x3(double t, const vector<double> &x) {
        return x[10];
}

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

double ThreeBody2D::f_y3(double t, const vector<double> &x) {
        return x[11];
}

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

double ThreeBody2D::f_vx1(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        double rES = sqrt((x[0]-x[4])*(x[0]-x[4])+(x[1]-x[5])*(x[1]-x[5]));
        double rJS = sqrt((x[2]-x[4])*(x[2]-x[4])+(x[3]-x[5])*(x[3]-x[5]));
        return -_GM*_m2*(x[0]-x[2])/rEJ/rEJ/rEJ -_GM*_m3*(x[0]-x[4])/rES/rES/rES;
}

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

double ThreeBody2D::f_vy1(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        double rES = sqrt((x[0]-x[4])*(x[0]-x[4])+(x[1]-x[5])*(x[1]-x[5]));
        double rJS = sqrt((x[2]-x[4])*(x[2]-x[4])+(x[3]-x[5])*(x[3]-x[5]));
        return -_GM*_m2*(x[1]-x[3])/rEJ/rEJ/rEJ -_GM*_m3*(x[1]-x[5])/rES/rES/rES;
}
```

```

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

double ThreeBody2D::f_vx2(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        double rES = sqrt((x[0]-x[4])*(x[0]-x[4])+(x[1]-x[5])*(x[1]-x[5]));
        double rJS = sqrt((x[2]-x[4])*(x[2]-x[4])+(x[3]-x[5])*(x[3]-x[5]));
        return -_GM*_m3*(x[2]-x[4])/rJS/rJS/rJS -_GM*_m1*(x[2]-x[0])/rEJ/rEJ/rEJ;
}

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

double ThreeBody2D::f_vy2(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        double rES = sqrt((x[0]-x[4])*(x[0]-x[4])+(x[1]-x[5])*(x[1]-x[5]));
        double rJS = sqrt((x[2]-x[4])*(x[2]-x[4])+(x[3]-x[5])*(x[3]-x[5]));
        return -_GM*_m3*(x[3]-x[5])/rJS/rJS/rJS -_GM*_m1*(x[3]-x[1])/rEJ/rEJ/rEJ;
}

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

double ThreeBody2D::f_vx3(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        double rES = sqrt((x[0]-x[4])*(x[0]-x[4])+(x[1]-x[5])*(x[1]-x[5]));
        double rJS = sqrt((x[2]-x[4])*(x[2]-x[4])+(x[3]-x[5])*(x[3]-x[5]));
        return -_GM*_m1*(x[4]-x[0])/rES/rES/rES -_GM*_m2*(x[4]-x[2])/rJS/rJS/rJS;
}

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

double ThreeBody2D::f_vy3(double t, const vector<double> &x) {
        double rEJ = sqrt((x[0]-x[2])*(x[0]-x[2])+(x[1]-x[3])*(x[1]-x[3]));
        double rES = sqrt((x[0]-x[4])*(x[0]-x[4])+(x[1]-x[5])*(x[1]-x[5]));
        double rJS = sqrt((x[2]-x[4])*(x[2]-x[4])+(x[3]-x[5])*(x[3]-x[5]));
        return -_GM*_m1*(x[5]-x[1])/rES/rES/rES -_GM*_m2*(x[5]-x[3])/rJS/rJS/rJS;
}

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

For the transformation to the frame of mass center of the three-body system

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

ThreeBody2D::ThreeBody2D(double a1, double e1, double m1,
double a2, double e2, double m2, double a3, double e3, double m3) {
        // body 1: Earth
        // body 2: Jupiter
        // body 3: Sun

        _a1 = a1;
        _e1 = e1;
        _m1 = m1;
        _a2 = a2;
        _e2 = e2;
        _m2 = m2;
        _a3 = a3;
        _e3 = e3;
        _m3 = m3;

        _GM = 4*M_PI*M_PI;
        _t_end = 5;

        _t_start = 0;
        _x1_start = _a1*(1-_e1);
        _y1_start = 0;
        _x2_start = _a2*(1-_e2);
        _y2_start = 0;
        _x3_start = 0;
        _y3_start = 0;
        _vx1_start = 0;
        _vy1_start = sqrt(_GM*(1+_e1)/(1-_e1)/_a1);
        _vx2_start = 0;
        _vy2_start = sqrt(_GM*(1+_e2)/(1-_e2)/_a2);
        _vx3_start = 0;
        _vy3_start = 0;

        // automatically go to the frame of mass center
        double cx = (_m1*_x1_start + _m2*_x2_start + _m3*_x3_start)/(_m1 + _m2 + _m3);
        double cy = (_m1*_y1_start + _m2*_y2_start + _m3*_y3_start)/(_m1 + _m2 + _m3);
        double cvx = (_m1*_vx1_start + _m2*_vx2_start + _m3*_vx3_start)/(_m1 + _m2 + _m3);
        double cvy = (_m1*_vy1_start + _m2*_vy2_start + _m3*_vy3_start)/(_m1 + _m2 + _m3);

        _x1_start -= cx;
        _y1_start -= cy;
        _x2_start -= cx;
        _y2_start -= cy;
        _x3_start -= cx;
        _y3_start -= cy;
        _vx1_start -= cvx;
        _vy1_start -= cvy;
        _vx2_start -= cvx;
        _vy2_start -= cvy;
        _vx3_start -= cvx;
        _vy3_start -= cvy;

        _c_start.push_back(_x1_start);
        _c_start.push_back(_y1_start);
        _c_start.push_back(_x2_start);
        _c_start.push_back(_y2_start);
        _c_start.push_back(_x3_start);
        _c_start.push_back(_y3_start);
        _c_start.push_back(_vx1_start);
        _c_start.push_back(_vy1_start);
        _c_start.push_back(_vx2_start);
        _c_start.push_back(_vy2_start);
        _c_start.push_back(_vx3_start);
        _c_start.push_back(_vy3_start);

        _alg = 0;
        _dt = 0.001;
}

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