

Newton's Laws and Kepler Orbits

Dynamics of Solar System Objects

The orbits around the Sun of solar system bodies[1] are determined by Newton's Law of Universal Gravitation, and Newton's equations of motion. These are systems of ordinary differential equations with solutions specified by initial conditions at some time. The orbital elements of objects involved provide precisely such initial conditions.

Motion of planets around the Sun

Johannes Kepler discovered three laws relating to the motion of planets around the Sun.

1. Planets move in elliptical orbits, with the Sun at one focus.
2. The line joining the planet to the Sun sweeps out equal areas in equal times.
3. The square of the planet's orbital period is proportional to the cube of the semi-major axis of the orbit.

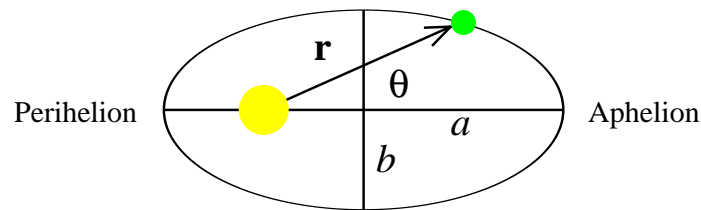


Figure 1: Kepler orbit.

In Fig. 1, a is the semimajor axis, b is the semiminor axis. The Perihelion and Aphelion points lie on the major axis, and are the closet and farthest distances, respectively.

Newton's law of gravity

Isaac Newton proved that Kepler's laws are obeyed by an isolated bound system of any two objects interacting through an inverse square central force. Gravitation is such a force:

$$\mathbf{F}_{12} = -\frac{Gm_1m_2}{r_{12}^3}\mathbf{r}_{12}, \quad \mathbf{F}_{21} = -\frac{Gm_1m_2}{r_{21}^3}\mathbf{r}_{21}. \quad (1)$$

These forces obey Newton's Third Law:

$$\mathbf{F}_{12} = -\mathbf{F}_{21}, \quad (2)$$

which is illustrated in Fig. 2.

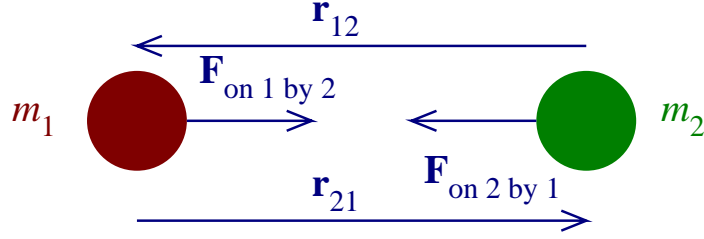


Figure 2: Newton's third law.

Exact solution of the Kepler problem

Since the Sun is by far the most massive object in the Solar System, most discussions use a Copernican coordinate system with the Sun fixed at the origin. However, this system is only approximately inertial. The approximation is an excellent one for comets, and even for a massive planet like Jupiter. If the two bodies have comparable masses, e.g., in a binary star system, then the center of mass system is inertial. The exact solution in this case can be expressed in terms of the reduced mass and relative coordinate

$$\mu = \frac{m_1 m_2}{m_1 + m_2}, \quad \mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2, \quad m_1 \mathbf{r}_1 + m_2 \mathbf{r}_2 = 0. \quad (3)$$

If $m_2 = M \gg m_1 = m$ then $\mu \simeq m$, $\mathbf{r} \simeq \mathbf{r}_1$, and $\mathbf{r}_2 \simeq 0$, which is the Copernican coordinate system.

The Kepler orbit can be written in parametric form

$$r(\theta) = \frac{a(1 - \epsilon^2)}{1 - \epsilon \cos \theta}, \quad b = a\sqrt{1 - \epsilon^2}, \quad (4)$$

where a is the semimajor axis, ϵ is the *eccentricity*, and b the semiminor axis. The orbital speed is given by

$$v = \sqrt{G(m_1 + m_2) \left(\frac{2}{r} - \frac{1}{a} \right)}. \quad (5)$$

The conserved energy of the system is given by

$$E = \frac{1}{2} \mu v^2 - \frac{Gm_1 m_2}{r} = -\frac{Gm_1 m_2}{2a}. \quad (6)$$

The period of the orbit is given by Kepler's third law

$$T^2 = \frac{4\pi^2}{G(m_1 + m_2)} a^3. \quad (7)$$

The orbital angular momentum can be computed using $\mathbf{L} = \mathbf{r} \times (\mu \mathbf{v})$.

Of course, even the c.m. frame is not strictly inertial if one takes into account the motion of the solar system through the galaxy.

Solving Newton's Equation of Motion Numerically

We need to solve Newton's equation of motion

$$\frac{d^2 \mathbf{r}}{dt^2} = -\frac{GM}{r^3} \mathbf{r} \quad (8)$$

given the initial position and velocity of the planet. This is a second order ordinary differential equation. For computational purposes, it is convenient to rewrite higher order differential equations as systems of coupled first order equations.

To do this, use the velocity \mathbf{v} to obtain

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \quad (9)$$

$$\frac{d\mathbf{v}}{dt} = -\frac{GM}{r^3}\mathbf{r}. \quad (10)$$

Algorithms for solving such systems are discussed in Chapter 16 of Numerical Recipes[2].

Euler's algorithm

A very simple algorithm was introduced by Euler to solve sets of coupled first order differential equations.

Consider the simple equation

$$\frac{dx}{dt} = f(x, t). \quad (11)$$

Given an initial value $x(0)$, the differential equation determines an unique solution $x(t)$.

Suppose that this solution is a smooth function with well behaved derivatives. Given $x(t)$ at any time t , the value of $x(x + dt)$ at a later time can be approximated by a Taylor expansion

$$x(t + dt) = x(t) + \left. \frac{dx}{dt} \right|_t dt + \mathcal{O}(dt^2) = x(t) + f(x, t)dt + \mathcal{O}(dt^2). \quad (12)$$

Given the initial value $x(0)$ and a step size dt , the solution is determined as a sequence of points

$$x(0), x(dt) = x(0) + f(x(0), 0)dt, x(2dt) = x(dt) + f(x(dt), dt)dt, \dots \quad (13)$$

The advantage of writing the equations in first order form is now apparent. A general system of ordinary differential equations can be written in first order form by replacing x and f with vectors

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t). \quad (14)$$

and Euler's algorithm can be expressed in vector form

$$\mathbf{x}(0), \mathbf{x}(dt) = \mathbf{x}(0) + \mathbf{f}(\mathbf{x}(0), 0)dt, \mathbf{x}(2dt) = \mathbf{x}(dt) + \mathbf{f}(\mathbf{x}(dt), dt)dt, \dots \quad (15)$$

C++ Program for Kepler Orbit of Halley's Comet

The following program implements Euler's algorithm to generate one orbit of Halley's comet using the Euler algorithm.

_____ Program 1: <http://www.physics.buffalo.edu/phy410-505/topic3/halley.cpp> _____

```
#include <cmath>
#include <cstdlib>
```

```

#include <fstream>
#include <iostream>

using namespace std;

const double G = 6.67e-11;      // Newton's constant in SI units
const double M = 1.99e30;      // Mass of the Sun in kg

double rAphelion = 5.28e12;    // aphelion distance in m
double vAphelion = 9.12e2;     // aphelion speed in m/s

void EulerStep(                // take one time step using Euler algorithm
    const double dt,          // step size - input
    double& x,                // x position - input/output
    double& y,                // y position - input/output
    double& vx,                // v_x - input/output
    double& vy                // v_y - input/output
) {
    double r = sqrt(x*x + y*y);
    double ax = - G*M*x / (r*r*r);
    double ay = - G*M*y / (r*r*r);
    x += vx * dt;
    y += vy * dt;
    vx += ax * dt;
    vy += ay * dt;
}

int main() {

    double x = rAphelion;
    double y = 0;
    double vx = 0;
    double vy = vAphelion;

    double dt;                // Euler algorithm time step
    cout << "Enter value of dt in seconds: ";
    cin >> dt;
    int skip;
    cout << "Enter number of data points to skip: ";
    cin >> skip;
    ofstream dataFile("comet.data");
    double t = 0;
    dataFile << t << '\t' << x << '\t' << y << '\n';
    int step = 0;
}

```

We will use a `bool` (logical variable which takes values `true` or `false`) to keep track of whether the comet has completed a full orbit around the Sun. The initial $y = 0$, and since $v_y > 0$ the comet moves counterclockwise around the Sun. So for the first half of the orbit, y will remain non-negative.

Program 1: <http://www.physics.buffalo.edu/phy410-505/topic3/halley.cpp>

```
bool yIsNegative = false;           // initial y = 0
```

Here we use the `do { ... } while (condition);` construct to iterate.

Program 1: <http://www.physics.buffalo.edu/phy410-505/topic3/halley.cpp>

```
do {
    EulerStep(dt, x, y, vx, vy);
    t += dt;
    ++step;
}
```

Note the use of the logical operators `!` (logical NOT) and `&&` (logical AND) to test when the comet moves past perihelion:

Program 1: <http://www.physics.buffalo.edu/phy410-505/topic3/halley.cpp>

```
if ( !yIsNegative && y < 0 ) // comet crosses x axis
    yIsNegative = true;

if (step % skip == 0)
    dataFile << t << '\t' << x << '\t' << y << '\n';

} while ( !(yIsNegative && y > 0) );

dataFile.close();
}
```

Improving the Kepler Orbit Program

Astronomical units

It is a good idea to avoid using very large or very small numerical values in computer programs, in order to avoid potential problems with numerical overflow or underflow. This can usually be done by using a system of units that is natural to the problem being considered.

- Earth's semimajor axis $1 \text{ AU} = 1.496 \times 10^{11} \text{ m}$
- Kepler's third law

$$G(M_{\text{Sun}} + m_{\text{Earth}}) = \frac{4\pi^2 a^3}{T^2} \quad (16)$$

- Measure distance in AU and time in years

$$G(M_{\text{Sun}} + m_{\text{Earth}}) = 4\pi^2 \text{ AU}^3/\text{yr}^2 \quad (17)$$

Setting Initial Values

The orbital elements of solar system bodies provide initial data that in principle determine their orbits. These include the semimajor axis length a and the eccentricity ϵ . From these two data the initial position and velocity, at the aphelion point for example, can be fixed so as to reproduce the desired Kepler orbit.

- Choose semimajor axis in the x direction and specify the aphelion distance $x(t = 0)$

$$x(0) = a(1 + \epsilon) . \quad (18)$$

- Choose the velocity at aphelion in the positive y direction and set $v_y(t = 0)$ equal to its magnitude.
- The speed is determined from the orbit equations:

$$v = \sqrt{G(m_1 + m_2) \left(\frac{2}{r} - \frac{1}{a} \right)} . \quad (19)$$

- Solve this equation at $t = 0$ for the eccentricity and semimajor axis

$$\epsilon = x(0)v_y(0)^2 / (G(M + m)) - 1 , \quad (20)$$

$$a = x(0) / (1 + \epsilon) . \quad (21)$$

Lagrange Interpolation

The period of the orbit can be determined more accurately by interpolating the numerical solution to obtain the time at aphelion.

- Consider a set of data points (x_i, y_i) in 2 dimensional space.
- Any two points $(x_0, y_0), (x_1, y_1)$ can be connected by a straight line

$$y(x) = \frac{(x - x_1)}{(x_0 - x_1)}y_0 + \frac{(x - x_0)}{(x_1 - x_0)}y_1 . \quad (22)$$

- A parabola can be drawn to pass through any three points:

$$y(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}y_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}y_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}y_2 . \quad (23)$$

- Simple to code in C++

```
#include <vector>

double interpolate(          // value of parabola at
    const double x0,        // this value of x, given 3 points
    const vector<double> x, // with these x values and
    const vector<double> y  // these y values
) {

    return (x0 - x[1]) * (x0 - x[2]) / (x[0] - x[1]) / (x[0] - x[2]) * y[0]
        + (x0 - x[2]) * (x0 - x[0]) / (x[1] - x[2]) / (x[1] - x[0]) * y[1]
        + (x0 - x[0]) * (x0 - x[1]) / (x[2] - x[0]) / (x[2] - x[1]) * y[2];
}
```

Algorithms for interpolation and extrapolation are discussed in Chapter 3 of Numerical Recipes[3].

The Euler-Cromer Algorithm

An alternative to the simple Euler algorithm was introduced by Cromer[4], who found that it tended to conserve energy better in problems with periodic motion.

The modification consisted in using an updated value of the velocity of the particle to compute the position:

$$\begin{aligned}\mathbf{v}(t + dt) &= \mathbf{v}(t) + \mathbf{a}(\mathbf{r}(t))dt , \\ \mathbf{r}(t + dt) &= \mathbf{r}(t) + \mathbf{v}(t + dt)dt .\end{aligned}$$

Notice the change from the simple Euler algorithm in the second equation!

Homework Problem

Modify the program halley.cpp to measure the period of the orbit. Compare this with the prediction from Kepler's third law to determine the accuracy of the simulation, and hence determine the number of steps required to achieve a given accuracy with the Euler and Euler-Cromer algorithms. Apply the more accurate algorithm to simulate the orbits of some of the comets in NASA's comet fact sheet.

References

- [1] International Astronomical Union – Minor Planet Center:
<http://www.cfa.harvard.edu/iau/mpc.html>, NASA Jet Propulsion Laboratory – Solar System Bodies website: <http://ssd.jpl.nasa.gov/?bodies>. NASA Comet Fact Sheet:
<http://nssdc.gsfc.nasa.gov/planetary/factsheet/cometfact.html>
- [2] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, “Numerical Recipes in C” (Cambridge University Press 1992), §16.0 Integration of Ordinary Differential Equations,
<http://www.nrbook.com/a/bookcpdf/c16-1.pdf>.
- [3] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, “Numerical Recipes in C” (Cambridge University Press 1992), §3.1 Polynomial Interpolation and Extrapolation,
<http://www.nrbook.com/a/bookcpdf/c16-1.pdf>.
- [4] A. Cromer, “Stable Solutions using the Euler Approximation”, Am. J. Phys. **49**, 455 (1981),
<http://dx.doi.org/10.1119/1.12478>.