

Orbital Mechanics and Computational Solutions

Conor Sayres

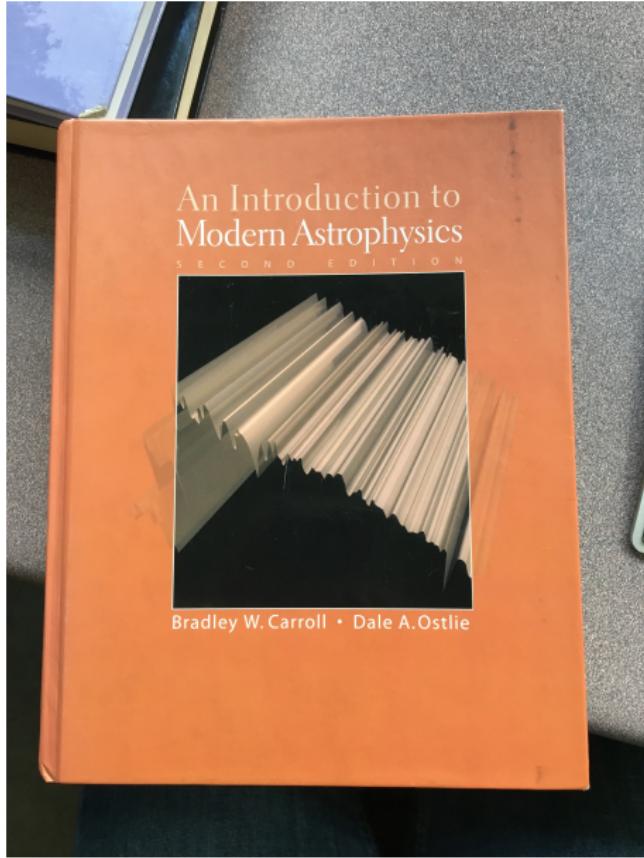
University of Washington Dept. of Astronomy

csayres at uw

April 2, 2018

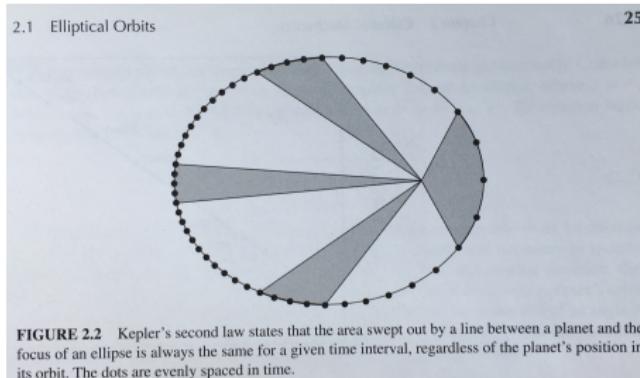
Overview

- 1 Mathematical Background
- 2 The N-Body Problem
- 3 Computing Leapfrog Solutions with Python
- 4 References



Kepler's Laws

- ① **Kepler's First Law** A planet orbits the Sun in an ellipse, with the Sun at one focus of the ellipse
- ② **Kepler's Second Law** A line connecting a planet to the Sun sweeps out equal areas in equal time intervals.
- ③ **Kepler's Third Law** $P^2 = a^3$. Where P is orbital period of planet in years, and a is average distance of planet from the Sun in astronomical units (AU, distance from Earth to Sun)



Newton's Laws

- ① **Newton's First Law** An object at rest will remain at rest and an object in motion will remain in motion in a straight line at a constant speed unless acted upon by an external force.
- ② **Newton's Second Law** The *net* force (the sum of all forces) acting on an object is proportional to the object's mass and its resultant acceleration. $\mathbf{F}_{net} = \sum_{i=1}^n \mathbf{F}_i = m\mathbf{a} = \frac{d\mathbf{p}}{dt}$
- ③ **Newton's Third Law** For every action there is an equal and opposite reaction.

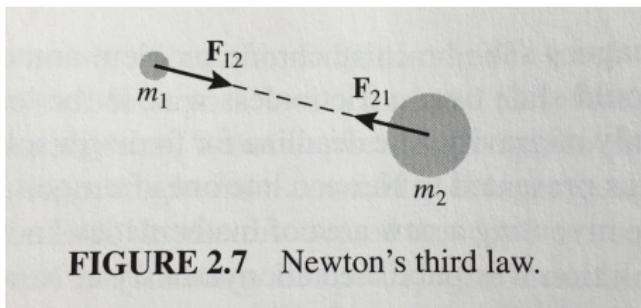


FIGURE 2.7 Newton's third law.

Newton's Law of Universal Gravitation

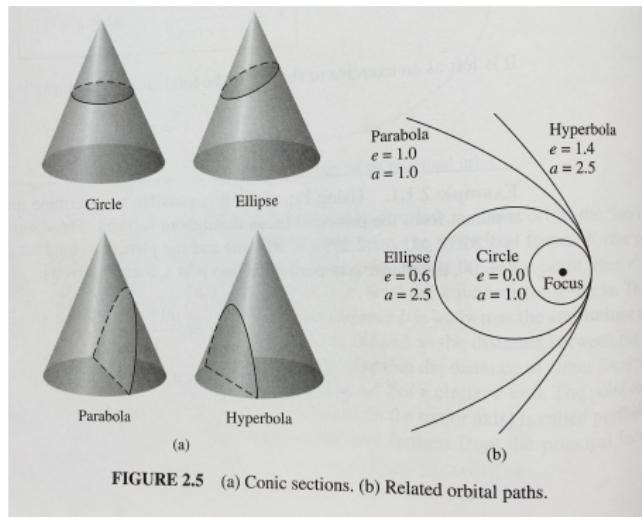
Combining Kepler's and Newton's laws, Newton was able to derive (see Carroll and Ostlie text) the following expression describing the force that holds planets in their orbits.

$$F = G \frac{Mm}{r^2}$$

$G = 6.673 \times 10^{-11} \text{ Nm}^2\text{kg}^{-2}$ is the **Universal Gravitational Constant**

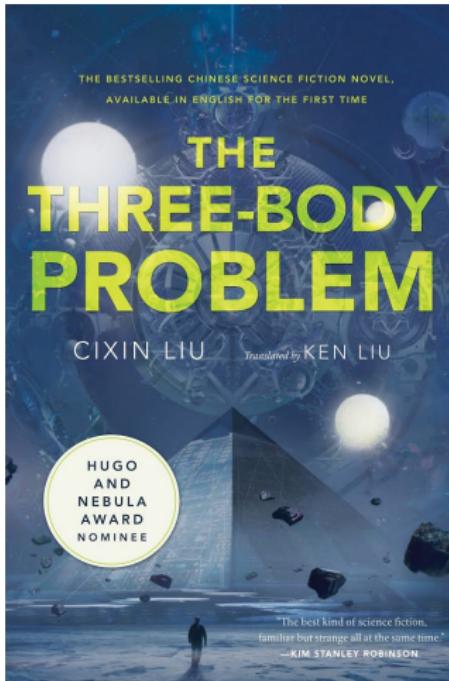
The 2-Body Problem

...can be solved exactly (with effort)! The solutions are elliptic, parabolic or hyperbolic trajectories.



The 3-Body Problem

...cannot be solved exactly.



Solving the N-Body Problem

The strategy for solving the N-Body problem will be:

- ① Formulate the problem as a system of ordinary differential equations (ODEs) that govern the motion of all objects in time.
- ② Discretize time, and use an iterative (time stepping) ODE solver to determine the trajectories of all bodies as a function of time.
- ③ For each body in the system, plug in the initial conditions at time=0 (mass, position, velocity) and watch what happens.

ODE system for N=2, in the xy plane

$$\frac{dx_1}{dt} = v_{x,1} \quad (1)$$

$$\frac{dy_1}{dt} = v_{y,1} \quad (2)$$

$$\frac{dx_2}{dt} = v_{x,2} \quad (3)$$

$$\frac{dy_2}{dt} = v_{y,2} \quad (4)$$

$$\frac{dv_{x,1}}{dt} = \frac{Gm_2}{r_{12}^2} \left(\frac{x_2 - x_1}{r_{12}} \right) \quad (5)$$

$$\frac{dv_{y,1}}{dt} = \frac{Gm_2}{r_{12}^2} \left(\frac{y_2 - y_1}{r_{12}} \right) \quad (6)$$

$$\frac{dv_{x,2}}{dt} = \frac{Gm_1}{r_{12}^2} \left(\frac{x_1 - x_2}{r_{12}} \right) \quad (7)$$

$$\frac{dv_{y,2}}{dt} = \frac{Gm_1}{r_{12}^2} \left(\frac{y_1 - y_2}{r_{12}} \right) \quad (8)$$

Leapfrog Method

We will use the Leapfrog method to solve our system of differential equations. The Leapfrog is a simple solver that has a nice feature (that isn't shared by all ODE solvers) that energy is conserved. The idea is that positions are updated at each time step n , and velocities are updated half way between each time step.

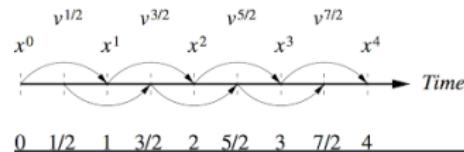


Figure: From Mark Krumholtz's Web Page

Leapfrog Method - 2 Body (x component for 1st body)

First, discretize time

$$t_{n+1} = t_n + dt \quad (9)$$

Here dt is a time step chosen by us. With a discrete time step we can begin massaging the equations shown previously. For now we'll just focus on the x components for the first body in our 2 body Leapfrog example:

$$\frac{dx_1}{dt} = v_{x,1} \rightarrow \frac{x_1^{(n+1)} - x_1^{(n)}}{dt} = v_{x,1}^{(n+1/2)} \quad (10)$$

$$x_1^{(n+1)} = x_1^{(n)} + dt \left(v_{x,1}^{(n+1/2)} \right) \quad (11)$$

where $x_1^{(n=0)}$ is supplied as the initial x position of the 1st body.

Leapfrog Method - 2 Body (x velocity component for 1st body)

And for velocity we get:

$$\frac{dv_{x,1}}{dt} = \frac{Gm_2}{r_{12}^2} \left(\frac{x_2 - x_1}{r_{12}} \right) \rightarrow \frac{v_{x,1}^{(n+3/2)} - v_{x,1}^{(n+1/2)}}{dt} = \left[\frac{Gm_2}{r_{12}^2} \left(\frac{x_2 - x_1}{r_{12}} \right) \right]^{(n+1)} \quad (12)$$

$$v_{x,1}^{(n+3/2)} = v_{x,1}^{(n+1/2)} + dt \left[\frac{Gm_2}{r_{12}^2} \left(\frac{x_2 - x_1}{r_{12}} \right) \right]^{(n+1)} \quad (13)$$

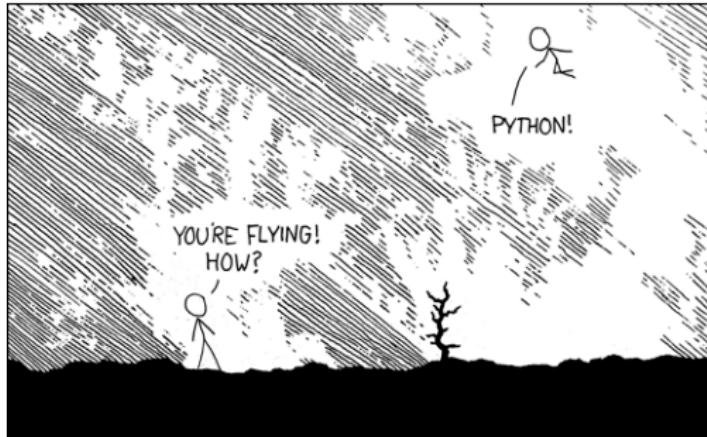
where $v_{x,1}^{(n=1/2)}$ is supplied as the initial x-velocity component of the 1st body. With these two steps (1st position, 2nd velocity) we can step forward in time to solve for trajectories in an N-body simulation.

Python

It's my favorite. I recommend using the Anaconda distribution of Python, you can download it here: <https://www.anaconda.com/download>. It comes with many useful packages including:

- Numpy - fast array based routines (similar to Matlab, if you've used that)
- Matplotlib - plotting and visualization tools (again, similar to Matlab)
- Astropy - a package that makes handling astronomical coordinates and time easy

Python



```

import numpy

G = 6.674e-11 # grav constant

def twoBody(t, dt, pos1_o, pos2_o, v1_o, v2_o, m1, m2):
    """Leapfrog solver for 2 body problem

    t: total duration of time in seconds
    dt: time step in seconds
    pos1_o: [x,y] initial position of object 1 in meters
    pos2_o: [x,y] initial position of object 2 in meters
    v1_o: [vx,vy] initial velocity of object 1 in meters/sec
    v2_o: [vx,vy] initial velocity of object 2 in meters/sec
    m1: mass of object 1 in kg
    m2: mass of object 2 in kg

    returns:
    tVec: vector of timesteps in seconds (of length N, t/dt)
    pos1: N x [x,y], array of positions of object 1 at each timestep
    pos2: N x [x,y], array of positions of object 2 at each timestep
    v1: N x [vx,vy], array of positions of object 1 at each timestep
    v2: N x [vx,vy], array of positions of object 2 at each timestep
    """
    # initialize outputs to zero
    # they will be populated in the
    # leapfrog loop.
    tVec = numpy.arange(0,t,dt)
    n = len(tVec)
    pos1 = numpy.zeros((n,2))
    pos2 = numpy.zeros((n,2))
    v1 = numpy.zeros((n,2))
    v2 = numpy.zeros((n,2))

    # initial conditions
    pos1[0,:] = pos1_o
    pos2[0,:] = pos2_o
    v1[0,:] = v1_o
    v2[0,:] = v2_o

    for i in range(n-1):
        pos1[i+1,:] = pos1[i,:] + dt*v1[i,:]
        pos2[i+1,:] = pos2[i,:] + dt*v2[i,:]
        rVec = pos2[i+1,:]-pos1[i+1,:]
        r = numpy.linalg.norm(rVec)
        v1[i+1,:] = v1[i,:] + dt*(G*m2/r**2)*(rVec/r)
        v2[i+1,:] = v2[i,:] + dt*(G*m1/r**2)*(-1*rVec/r)

    return tVec, pos1, pos2, v1, v2

```

DEMOS

Areas to investigate...

- An Introduction to Modern Astrophysics. Carroll and Ostlie.
- [https://sites.google.com/a/ucsc.edu/krumholz/
teaching-and-courses/ast119_w15/class-11](https://sites.google.com/a/ucsc.edu/krumholz/teaching-and-courses/ast119_w15/class-11)