CSS 490: Orbital Mechanics and Computational Solutions

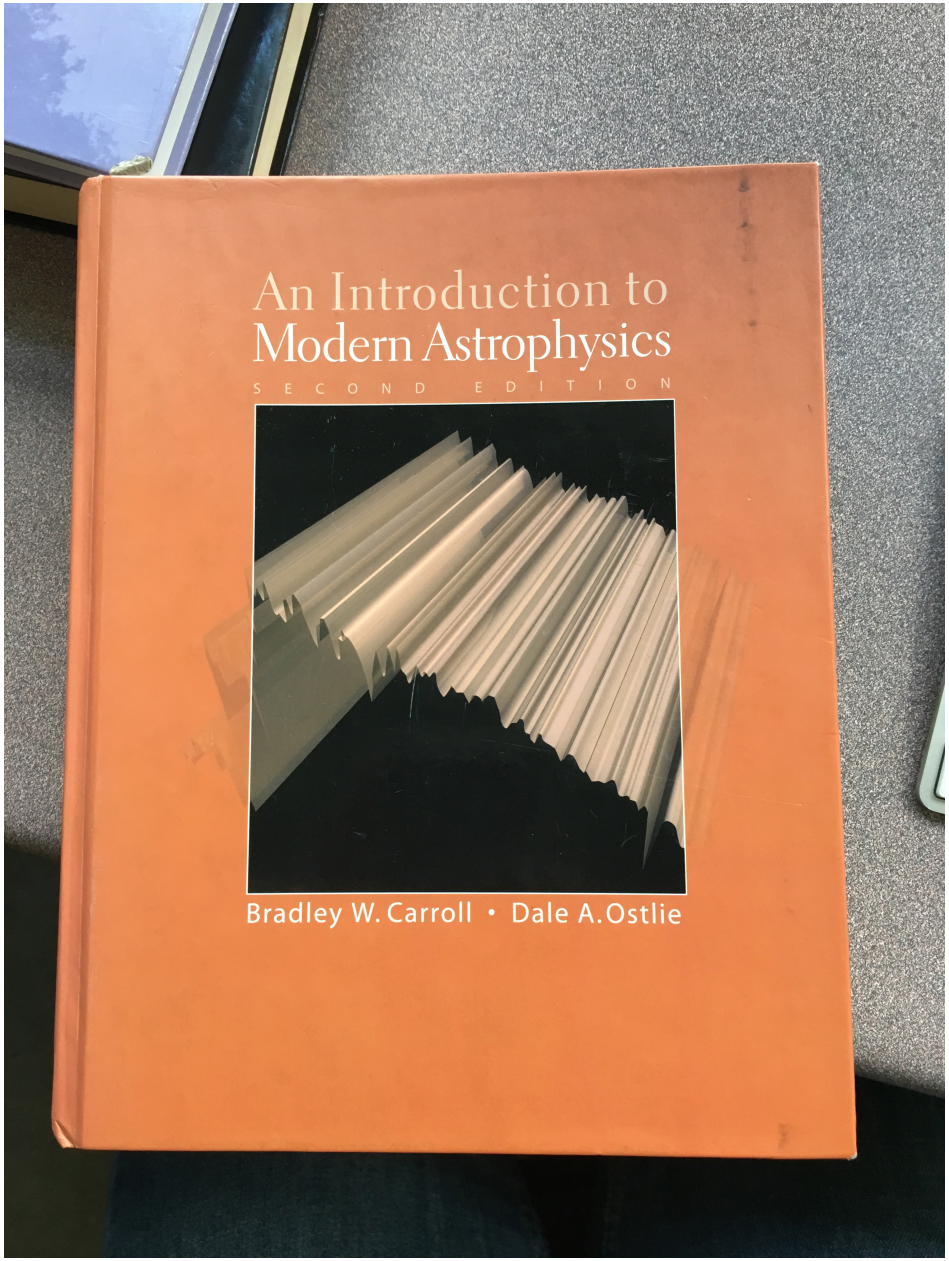
Conor Sayres

University of Washington Dept. of Astronomy

csayres at uw

April 3, 2018

Overview


An Introduction to
Modern Astrophysics
SECOND EDITION
Bradley W. Carroll • Dale A. Ostlie

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 Thrid 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)

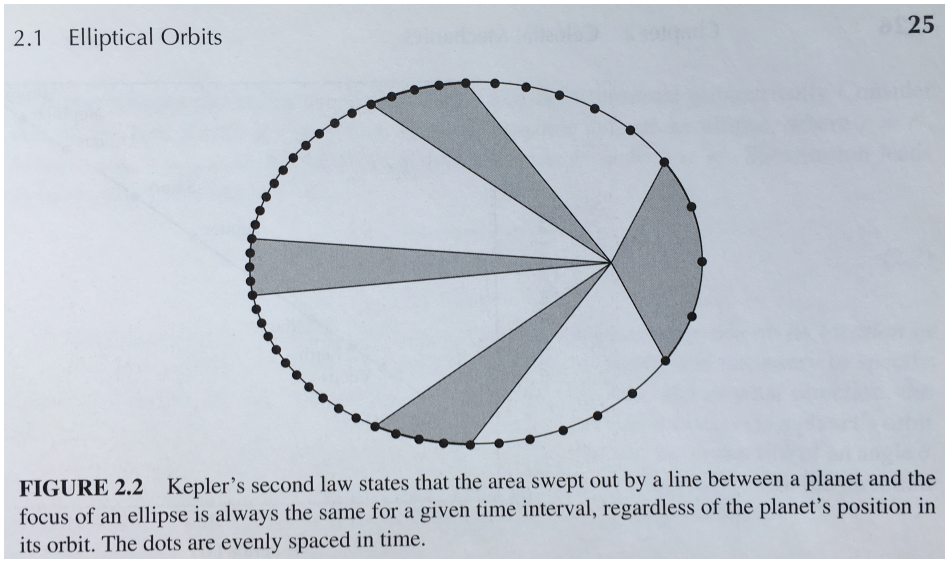


FIGURE 2.2 Kepler's second law states that the area swept out by a line between a planet and the focus of an ellipse is always the same for a given time interval, regardless of the planet's position in its orbit. The dots are evenly spaced in time.

Newton's Laws

1. **Newtons'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 Thrid Law** For every action there is an equal and opposite reaction.

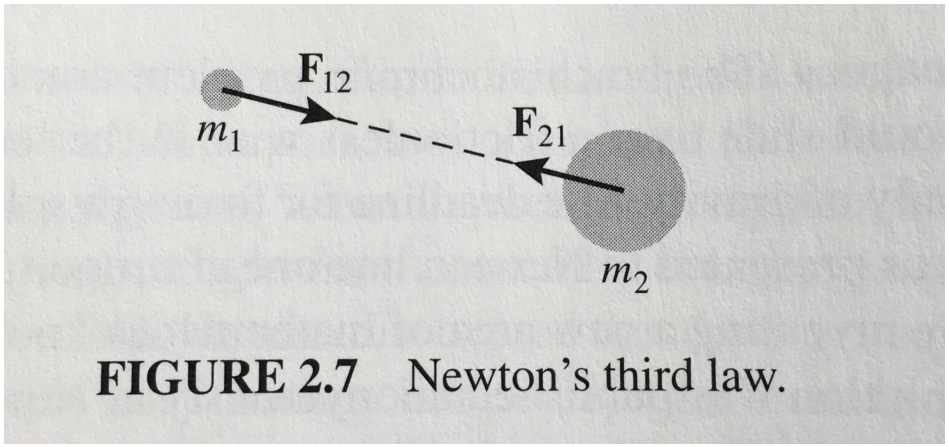


FIGURE 2.7 Newton's third law.

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.

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

$G = 6.673x10^{-11}\mathrm{Nm^2kg^{-2}}$ is the **Universal Gravitational Constant**

The 2-Body Problem

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

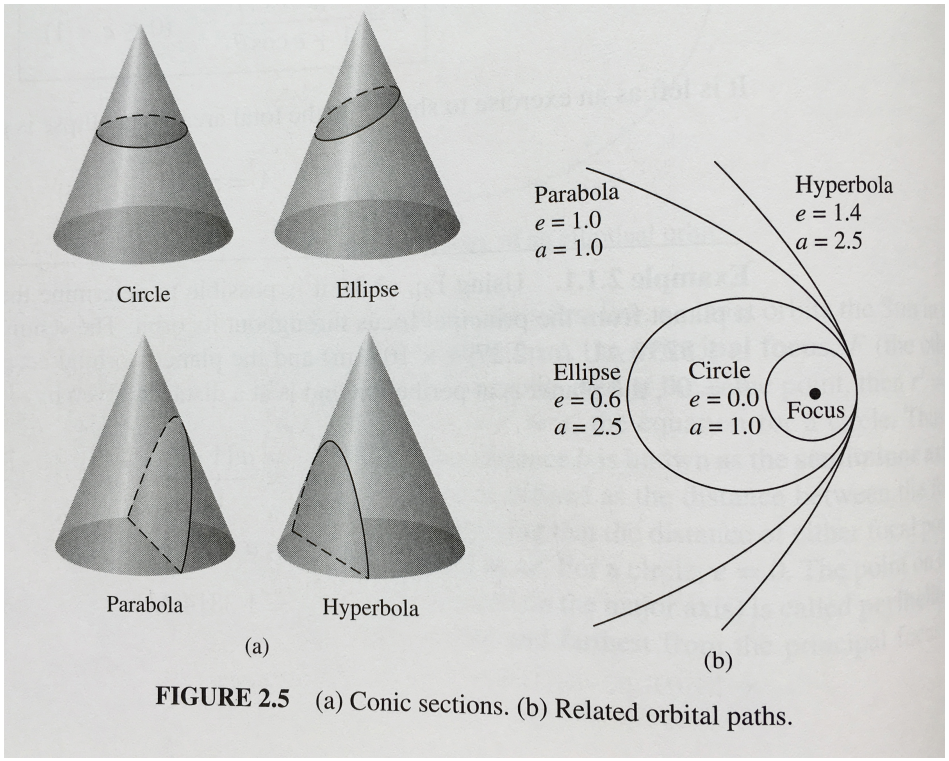
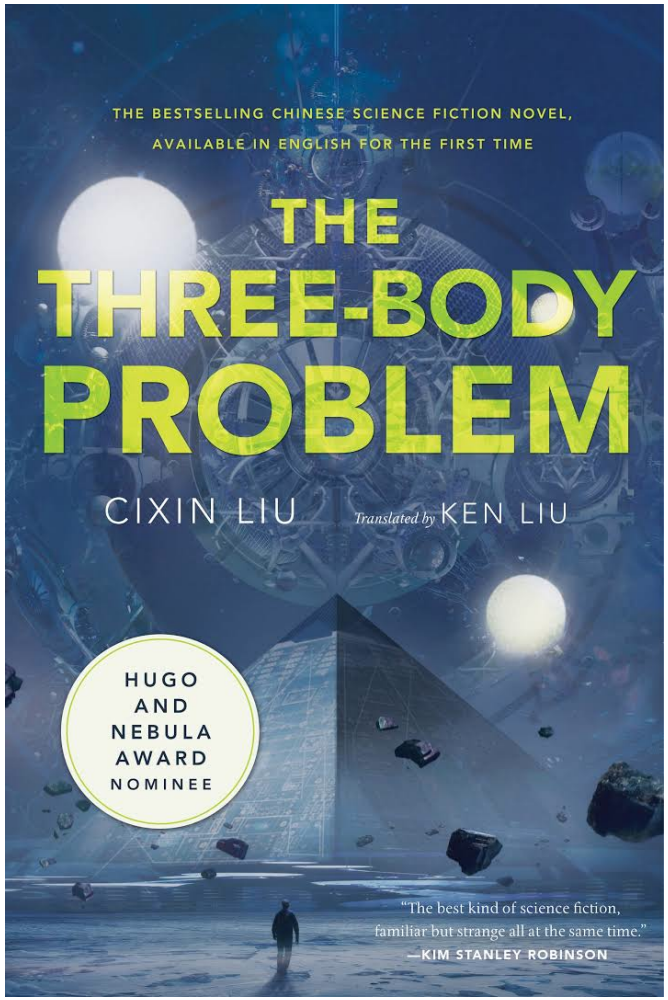


FIGURE 2.5 (a) Conic sections. (b) Related orbital paths.

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. We choose this method because 1) it is a simple, and 2) energy is conserved in the case of our orbit problem (a feature that isn't shared by all ODE solvers). The general procedure will be: update positions at each integer time step n, and update velocities between each position update $n/2$ where n is odd.

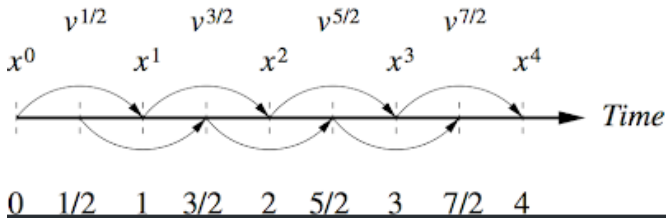


Figure: From Mark Krumholtz's Web Page

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

First, discretize time

$$t_{n+1} = t_n + dt \tag{9}$$

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:

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

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

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:

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

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

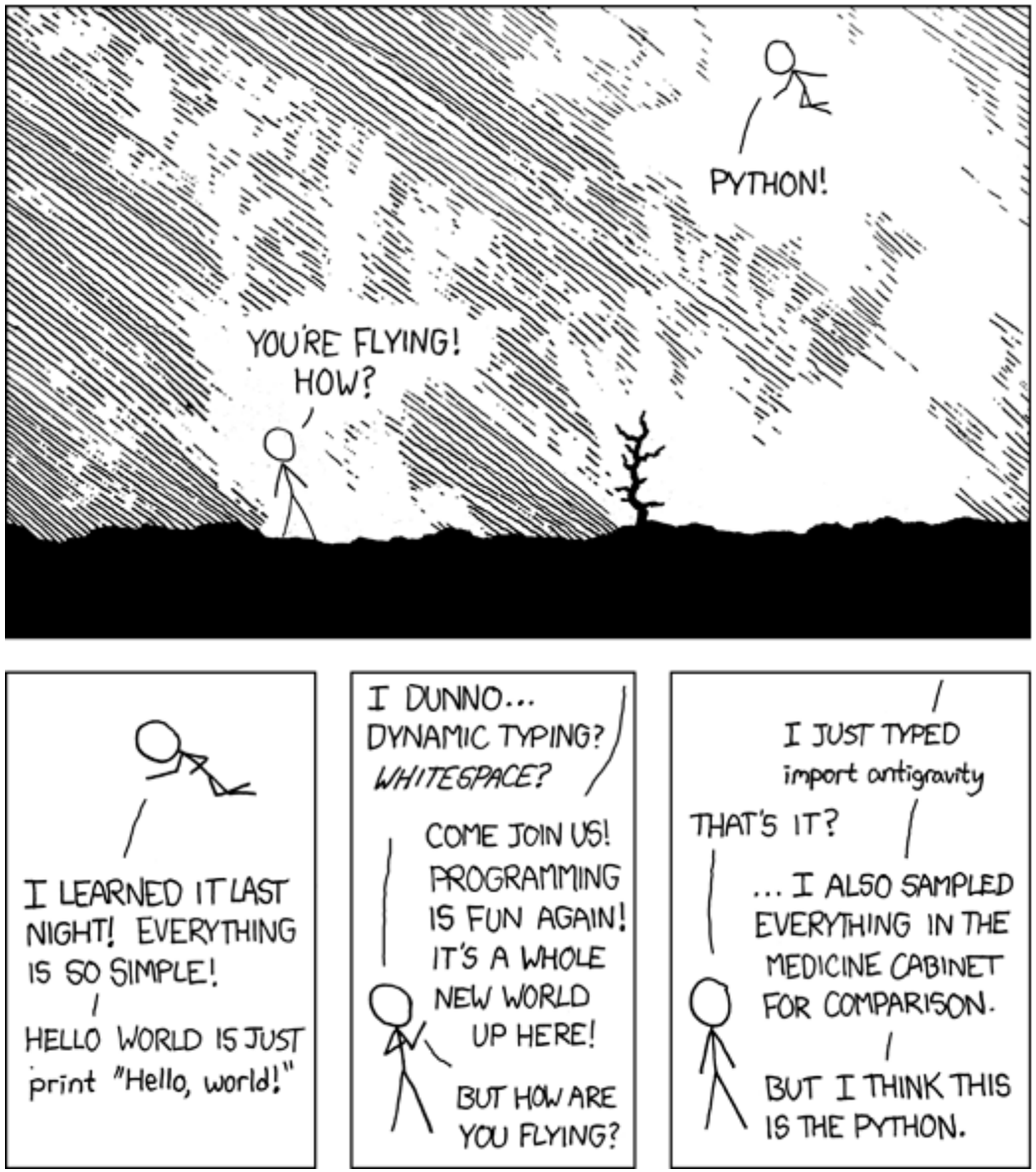
where $v_{x,1}^{(n=1/2)}$ is supplied as the initial x-velocity component of the 1st body. With these two steps (1st position, 2nd velocity) we can step forward in time to solve for trajectories in an N-body simulation.

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


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

```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 velocities of object 1 at each timestep
    v2: N x [vx,vy], array of velocities 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
```

Python - astropy

Astrology made easy: Accessing solar system ephemeris astropy

```
from astropy.time import Time
from astropy.coordinates import get_body_barycentric

# my birthday
myBirthday = Time("1986-10-23 00:00")
marsPos = get_body_barycentric("mars", myBirthday).xyz.value
# print XYZ position of mars on my birthday with respect to the
# solar system's center of mass (basically the sun)
print(marsPos)
```

DEMOS

- `https://github.com/csayres/leapfrog/`
- leapfrog2D.py
- solarsytem.py

Areas to investigate...in no particular order

- numpy.sum(numpy.abs(thrustArr)) in solarSystem.py is a proxy for the amount of onboard fuel you've used, can you minimize it to reach your goal?
- Research (Google!) Hohmann transfer orbits, and gravitational slingshots/assists to see if you can figure out how to utilize these to minimize fuel consumption. Check out Galileo's trajectory on youtube, it used two Earth passes and one Venus pass to get to Jupiter.
- Implement realistic thruster acceleration constraints (a max value for thrustArr).
- As thrusts are used, fuel is consumed, and the probe's mass decreases, can you model this realistically, and does it change your solutions?

Areas to investigate...in no particular order

- Vary dt and compare the resulting trajectories. As $dt \rightarrow 0$ you should approach the *exact* solution, however number of iterations quickly begin to kill you: Python loops are very slow. If you really want to be a hero and investigate very small timesteps, code up this solver in C/C++ or (gasp) FORTRAN.
- Try using a different ODE solver (scipy.integrate.ode), and compare solutions with the Leapfrog approach.
- Rather than using precomputed positions for solar system bodies, use initial positions/velocities and let the solver update their trajectories in time. What happens? Does the solar system blow up? Again how sensitive is the solar system N-body problem to choice of dt?
- try: 'import antigravity' in a Python shell

References

- An Introduction to Modern Astrophysics. Carroll and Ostlie.
- https://sites.google.com/a/ucsc.edu/krumholz/teaching-and-courses/ast119_w15/class-11
- Google
- If you think N-body sims are cool (UW Astro): http://depts.washington.edu/astron/research/n-body-shop/