

0.1 Homework 8

Due: Monday, October 26, 2007 at 10:00 p.m.

This week, you will extend your orbit code to animate the motion of minor planets in the outer solar system. Minor planets mean Pluto and the other dwarf planets, as well as all of the other satellites of the Sun. In particular, the outer solar system has the Kuper Belt objects (also known as Trans-Neptunian Objects). There are about 350 of these objects known with good orbital parameters.

You will ignore the gravity between these minor planets (we don't usually know their masses anyways) and assume the only gravity is from the Sun, which we will put at the origin of the coordinate system. However, since we're now dealing with many planets, we need to integrate the orbits in 3 dimensions. We could get away with 2 dimensions in the single object case because we could always rotate to the orbital plane.

Given that you know how to integrate the orbit of a single object, the challenges of this homework are two-fold: 1) Dealing with inputting and transforming the orbital parameters into 3-d Cartesian initial conditions, and 2) Handling the book-keeping of 350 planets.

For the latter challenge, we will use a structure to hold all of the information about a minor planet and then use an array of those structures to hold the full set of planets. Your ODE integrator, plotting, and printing routines should accept this array of structures, then use a loop to process each planet in turn. Note that the loop may contain variables that hold temporary computations about the planet under consideration. That is, if a quantity is not going to be used again, you don't need to put it into the data structure. But all quantities that you'll need from one time step to the next should go into the structure.

Also important to keeping this code in a manageable form is to split your work into functions. For example, you may want to have functions that operate on a single planet to compute its energy or angular momentum, or that rotate a coordinate vector by a given angle. Divide the problem into pieces that you can reuse. Keep `main()` pretty simple!

We will approach our solution in a series of steps. Part of the lesson of this homework is to see how you can build up a complicated program one step at a time!

The orbital parameters for the minor planets are in the following file:
http://www.physics.arizona.edu/~doug/minor_planets_orbits.txt.
(There is also a link to this file on the course web site.)

Each line of this file describes a separate minor planet. As you'll see, there are a fair number of parameters to understand. We will include them into the code in stages, so that you can build up to the full complexity, but we'll describe the physics here before we get to the code aspects.

As you know, all bound orbits in the inverse square force are ellipses. The motion

of an object is fully determined by specifying the position and velocity at a single time. Since we're working in 3 dimensions, there must be 6 pieces of information to specify the motion. However, physically we care about the properties of the ellipse, so we will quote quantities about the geometry of the orbit. Clearly, we must specify 6 such quantities to fully specify the orbit. The first two give the shape and size of the ellipse: the semi-major axis of the ellipse (recall that this fixes the period of the orbit) and the eccentricity of the ellipse. Then there are 3 angles to specify the orbital plane and the direction of perihelion within that plane. Finally, there is a measure of where the object along the orbit at a given reference time; this is specified as the angle advanced relative to the perihelion point.

Hence, the columns in the file are:

- 1) the absolute magnitude, which is related to the size,
- 2) the true anomaly, which is the angle around the orbit,
- 3) the angle of perihelion,
- 4) the angle of the ascending node,
- 5) the angle of inclination, all three of which set the orbital plane
- 6) the orbital eccentricity,
- 7) the semi-major axis,
- 8) and the name of the object. Some of the names you'll recognize; others are just numbers awaiting names.

More on how to use these quantities below.

1) Because the full problem is relatively big, we will build the code in pieces. To begin, we will write a code that simply sets up an array of structures, reads in the data file, writes out the first 10 rows, and plots the eccentricity vs. semi-major axis of all the minor planets. No time evolution yet!

We are supplying the following code to get you started. In particular, this will set up the structure definition, define an array of 1000 elements (of which you will only use `nplanets`, depending on the length of the input file), and read the file.

This code is also available at

<http://www.physics.arizona.edu/~doug/program8.start.c>.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

#define NDIM 3
#define MAXNAME 80

struct planet {
    double absolute_magnitude;
    double true_anomaly;
```

```

    double angle_of_perihelion;
    double angle_of_ascending_node;
    double inclination;
    double eccentricity;
    double semimajor_axis;
    char name[MAXNAME];
    // Add any other variables you want!
};

#define MAXPLANETS 1000

int read_file(char *filename, struct planet P[]) {
    // Read the given file and put the information into the array of planets.
    // Return the number of planets in the list.
    FILE *fp;
    int nplanets;
    char line[200];

    fp = fopen(filename,"r");
    if (fp==NULL) {
        fprintf(stderr,"File %s not found.\n", filename);
        abort();
    }

    nplanets = 0;
    while (fgets(line,200,fp)!=NULL) {
        if (line[0]=='#') continue;
        sscanf(line, "%lf %lf %lf %lf %lf %lf %lf",
            &(P[nplanets].absolute_magnitude),
            &(P[nplanets].true_anomaly),
            &(P[nplanets].angle_of_perihelion),
            &(P[nplanets].angle_of_ascending_node),
            &(P[nplanets].inclination),
            &(P[nplanets].eccentricity),
            &(P[nplanets].semimajor_axis));
        strncpy(P[nplanets].name, line+74, MAXNAME);
        P[nplanets].name[MAXNAME-1] = '\0';
        nplanets++;
        if (nplanets>=MAXPLANETS) break;
    }
    fclose(fp);
    printf("Read %d planets from file %s.\n", nplanets, filename);
    return nplanets;
}

```

```

int main() {
    struct planet Planets[MAXPLANETS];
    int nplanets;
    nplanets = read_file("MPC_listing_of_TNO.v2.txt", Planets);
    // Elements Planets[0] to Planets[nplanets-1] are now filled
    // with planet information.
    return 0;
}

```

You should take this code and add two functions, one to print the information about the first 10 planets and the other to use Philsplot to plot the eccentricity vs semi-major axis. Check by hand that the printed information matches that in the input file!

You should make your graph with Philsplot, as one of the things we're trying to do is to validate that we have read the file correctly. Although the range of semi-major axes goes out to several hundred AU, we recommend you stop at 150 AU so you can see some of the interesting behavior. Note that you can put all of the Philsplot commands in this function, including `close_plot()`, which will then prompt you to close the window. That way, when it's time to add the animation of the solar system, we can just call another `open_plot()` and do both plots in the code.

Looking at the above code, you will probably see a number of unfamiliar aspects about reading from files, manipulating strings, etc. Part of your assignment is to read this code carefully and consult the documentation (like your C book, the course notes, or the on-line 'man' pages) to make sure you understand what every line is doing. Your report should include a detailed assessment of each line in `read_file()` and `main()`, e.g., a comment for every line to say what is happening.

Note that we pass the array into functions. Because arrays are passed by memory location, any elements of the array that get manipulated in the function will affect the original array, not a copy of it. This makes it easy to pass information between functions.

In the plot itself, you will see two interesting features. First, there are a bunch of minor planets all with a semi-major axis of 39 AU. Pluto is one of these, which is why these are called plutinos. The reason that 39 AU is special is that this corresponds to an orbital period that is exactly 1.5 times that of Neptune. The gravitational interaction with Neptune causes this "resonant" behavior to be more stable. In the early history of the solar system, Neptune's radius moved outwards and as the 3:2 resonance swept outwards, it trapped minor planets into it. Similar attractive and repulsive resonances govern a number of other interesting features in the solar system!

Second, you'll see that at high semi-major axis, there are only objects at high eccentricity. Effectively this means that nearly all of the objects have a perihelion distance below 50 AU. There are two reasons for this. First, minor planets get very faint when they are far away; indeed, the flux drops as $1/r^4$, one factor of $1/r^2$ for the light received from the Sun and one factor of $1/r^2$ for the fraction of reflected light

that gets back to the Earth! So we are much more likely to find objects when they are close, such as when they are at perihelion. Second, minor planets from around 40 AU sometimes get scattered by interactions with each other. When they get scattered up in energy, they get a large semimajor axis but their perihelion remains small (as in the midterm assignment!). Perhaps most of the objects beyond 50 AU are on scattered orbits from smaller radius.

2) Now that you have an array of structures and have loaded the file of orbital parameters, it is time to set up the Cartesian initial conditions and integrate the orbits. You should not do this in `main()`. Instead, write functions to be called from `main()`, e.g.,

```
set_initial_conditions(Planets, nplanets);
RK_integrate(Planets, nplanets, tstart, tfinal, tstep, tplot);
```

We'll begin by keeping the initial conditions simple: use only the semi-major axis and eccentricity as in homework 7, and place all the planets in the x - y plane with perihelion on the positive x axis. Start all planets at perihelion.

You will at minimum need to add the Cartesian position and velocity to the planet structure. Note that by simply adding variables to the structure definition, these elements are automatically created for each planet in the array and automatically passed between your functions. Very convenient!

You should integrate for 1000 years with a time step of 0.01 year, plotting every `tplot` years. I found that plotting every year works fairly well.

Include `PhilsPlot` calls so that the x - y position of all planets are plotted and animated. After you've called `putpoint_plot()` for each planet, you can call `flush_plot()`, and then call an equal number of `delpoint_plot()`. This will cause the planets to animate on the screen. If you want to put the Sun at the center of the plot, yellow is color 7 and you'll want to use `memory=0` so that the Sun point is not deleted. Remember that you can use `delay_plot()` to tune the speed of the animation.

Limiting the plot to 100 AU in each direction will allow you to see more detail in the Kuiper Belt. Also, we found it useful to make the dots for the largest minor planets bigger and in a different color. Choosing absolute magnitude less than 3 will select Pluto, Sedna, Eris, Makemake, Quaoar, and Orcus, the biggest six KBOs. The definition of absolute magnitude is described at the end of this document, but it doesn't matter here save to know that smaller numbers mean bigger planets.

Of course, the plot will look strange because we have short-cut the initial conditions. Explain the behavior in your report.

3) Add code to compute the energy and norm of the angular momentum of a planet. Save the initial values for each planet (by adding elements to the structure) and print out the change in these quantities after 1000 years for the first 20 planets. Report on the quality of the integrations based on this.

Note that the angular momentum is a vector in 3-d. You should compute all three components and use the vector norm.

Notice that you have to compute the 3-d vector norm fairly often. Perhaps you should write a function to do that!

4) Now we'll make the initial conditions more realistic in two steps. In the first step, we will start the planets not at perihelion. However, the orbital plane will remain the x - y plane and the perihelion position will still be along the positive x axis.

Recall that in polar coordinates, the equation of an ellipse with one focus at the origin is given by

$$r = \frac{a(1 - e^2)}{1 + e \cos \theta} \quad (1)$$

where the angle of perihelion is at $\theta = 0$ (you should check that this formula produces our perihelion distance of $r_p = a(1 - e)$). The value of θ at the reference time is the true anomaly (column 2 in the file, in degrees!). We can then use $x = r \cos(\theta)$, $y = r \sin(\theta)$, $z = 0$ to set the position.

The velocity is a little trickier. An easy way to do this is to compute energy and angular momentum at perihelion, where we know $v_p = \sqrt{GM(1 + e)/r_p}$ and insist that the velocity at $\theta = 0$ conserve those quantities. The tangential velocity v_θ is set by conservation of angular momentum, so $v_\theta = r_p v_p / r$. The energy is $E = (1/2)(v_\theta^2 + v_r^2) - GM/r$. Since we know r , v_θ , and E (from perihelion), we can solve for v_r^2 .

Pushing through the algebra, we find

$$v_\theta = (1 + e \cos \theta) \sqrt{\frac{GM}{a(1 - e^2)}} \quad (2)$$

$$v_r = \pm \sqrt{\frac{GM}{a} \frac{e^2}{1 - e^2} (1 + \cos^2 \theta)} \quad (3)$$

You choose the correct sign of v_r by noting that it is positive for $0 < \theta < \pi$ and negative for $\pi < \theta < 2 * \pi$. Once you have v_θ and v_r , you can project to the x and y velocity. $v_z = 0$ of course.

Remember that all angles in the file are given in degrees, but C will want radians for its trigonometry functions.

It is a good idea to check your code by seeing that the energy and angular momentum of the result matches the values at perihelion.

5) Now we're ready to add the angles of the orbital plane. This admittedly is confusing, because it's a 3-d geometry problem. Please look at the figure at http://en.wikipedia.org/wiki/Argument_of_periapsis to help with the following (figure also attached in hardcopy).

First, you need to know that the solar system reference system is defined by the ecliptic plane, which is the orbital plane of the Earth around the sun. We'll take this to be the x - y plane. A reference direction in this plane is defined by the position of the Earth at the vernal equinox (i.e., where the Earth's equatorial plane intersects the ecliptic plane). We'll take this to be the positive x axis. [As you may know, the earth's axis and even its orbital plane around the Sun precess slowly, which means these defined directions would change relative an inertial frame set by extragalactic objects. Technically these directions are defined at a particular time, known as the 'epoch'.]

Next, we consider that the orbital plane of the object is tilted relative to the ecliptic plane. The intersection between these two planes is a line. The direction along this line that points toward where the orbit goes from south to north is called the ascending node. The angle in the ecliptic plane between the reference direction x and the ascending node is called the angle of the ascending node. The opening angle between orbital plane and the ecliptic plane is called the angle on inclination. With these two angles, we have specified the orientation of the orbital plane.

We then need to specify the orientation of the ellipse within the orbital plane. This is done by specifying the angle within the orbital plane between the ascending node and the direction to perihelion. This is known as the angle of perihelion (or periapsis).

These three angles combined with the semi-major axis, eccentricity, and true anomaly comprise the 6 quantities we need to fully determine the initial position and velocity of the planet.

Even if you understand the figure and angle definitions, it can be a brain-twister to work out the rotations needed to actually set up the Cartesian coordinates. Here's the plan:

First, set up the position and velocity as in part 4. That is, set up the position and velocity for a given semi-major axis, eccentricity, and true anomaly assuming that the orbit is counter-clockwise in the x - y plane with perihelion at the positive x axis.

Next, rotate the 3-d position and velocity around the z axis by the angle of perihelion.

Next, rotate the 3-d position and velocity around the x axis by the angle of inclination.

Finally, rotate the 3-d position and velocity around the z axis by the angle of the ascending node.

To do this, you should write a function to rotate a 3-d vector around the z axis by a given angle, and a similar function to rotate around the x axis. [Note: rotating around an arbitrary direction is notably more complicated, so you probably don't want to overdesign this part.] Be sure to rotate the position and velocity in the same manner. Also, remember that rotations in 3-d don't commute, so the order above does matter.

Complete your program by adding these rotations to your initial condition routine. Now you can watch the outer solar system in action! Try the x - y projection to start, but also look at the x - z projection. Note that you will occasionally see planets do things that don't look like ellipsoidal motion, e.g., speeding up at points that aren't the closest approach to the Sun. This is because you're projecting 3-d motion into 2 dimensions.

On the x - y plot, you'll see an interesting behavior that the swath of planets seems to expand in radius at the beginning of the animation. Is the Kuiper Belt pulsating? No, this is just a selection effect: nearly all of these objects were discovered in the last 15 years, whereas the orbits are hundreds of years. Objects are much more likely to be discovered near perihelion when they appear brightest. So today most of the Kuiper Belt objects are still near their closest approach; when we run the animation forward for a hundred years, they generally move away from the Sun! The corollary to this is that there are lots more Kuiper Belt objects waiting to be discovered further out, and perhaps we'll only find them when they move in closer to the Sun.

If you study the initial position of the planets, you'll also see that they aren't evenly distributed in angle around the Sun. This also is a selection effect: more of the searches have been from telescopes in the northern hemisphere, so the ecliptic plane has not been uniformly searched.

Absolute magnitude: It doesn't matter for the homework, but in case you care about the definition of the absolute magnitudes in the file, here's the definition. The definition of absolute magnitudes in the solar system are different than for astronomical objects. Magnitudes are a logarithmic measurement of flux, on the scale $m = -2.5 \log_{10} \text{flux}$. Every 5 magnitudes larger means that the flux is 100 times fainter.

Absolute magnitudes attempt to correct for the distance to the object so that we're talking about intrinsic luminosity rather than flux. For galactic and extragalactic work, the standard distance is 10 pc. For solar system work, the choice is that the object will be 1 AU from the Sun and observed from the Sun's position.

The point here is that intrinsically more luminous objects are bigger. Remember that essentially all of the optical luminosity of a planet is due to the reflected light from the Sun, so the luminosity goes as the square of the diameter times the reflectivity, known as the albedo. Holding the albedo constant, every 5 magnitudes larger will correspond to a factor of 10 smaller diameter.

The known objects in the Kuiper Belt cover a spread of 15 magnitudes, which is a factor of a million in luminosity and a factor of 1000 in diameter. The brightest KBOs can be observed with modest telescopes; the faintest ones were discovered with the Hubble Space Telescope!