Lab 2

In this lab we will solve a real problem which means we will encounter a number of side issues to distract us from our main task. There are many annoying subtleties when working with computers. In this lab we will encounter difficulties in working with inverse trigonometric functions. Typically when doing calculations we want to avoid writing loops if we can help it since this is slow.. There are, however, some cases when loops cannot be avoided. We will encounter some in this lab.[1]

We wish to find the velocity of the Earth as a function of time as it orbits the Sun. This does not sound too hard. To make things a little shorter we will only find the angular speed of the Earth, $d\theta/dt \equiv \dot{\theta}(t)$. From Kepler's second law we know that the angular speed in an elliptical orbit is not constant, the radius of the orbit sweeps out equal area in equal time. When we study central forces in classical mechanics we learn that this can be understood in terms of conservation of angular momentum and further that we can easily write down equations of motion and solve, in terms of integrals, for how both the radial and angular position of an object behaves in such an orbit. We are interested in the angular position which for an elliptical orbit is given by

$$t(\theta) = \frac{\tau}{2\pi}(1-e^2)^{3/2}\int_0^\theta \frac{d\theta'}{(1+e\cos\theta')^2} = \frac{\tau}{2\pi}\left[2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\tan(\theta/2)\right) - \frac{e\sqrt{1-e^2}\sin\theta}{1+e\cos\theta}\right]$$

Here τ is the period of the orbit, $0 \le e < 1$ is the eccentricity of the orbit, and I have chosen $t = 0$ at periapsis (the distance of closest approach). The difficulty we encounter is not in doing the integral, that we **can** do, it is in inverting this function. We have found $t(\theta)$ but really want $\theta(t)$. One approach to inverting this would be to do what we did in the last lab, use a root finder. That is time consuming and not easily amenable to taking the derivative we need. In this lab we will instead use a spline. This will also make it easy to calculate the derivative to get the angular velocity.

The equation given above comes from a direct solution of the equations of motion. Historically this result has been manipulated. In fact, even before Newton's laws this problem was studied observationally. An alternative way to write the problem is to use **Kepler's equation:** $\omega t = \psi - e\sin\psi$. Here ωt is called the *mean anomaly* and ψ is called the *eccentric anomaly*.[2] We are using the denoting the average speed by $\omega = 2\pi/\tau$. Note that $0 \le \omega t \le 2\pi$ for $0 \le \psi \le 2\pi$. We can relate ψ to the angular position in many ways, we will use $\cos\theta = \frac{\cos\psi - e}{1 - e\cos\psi}$. Note that in the

[1] In many cases array slicing can be used to avoid writing loops. That is true for one of the cases we encounter below. Array slicing will be explored in a future lab.

[2] Astronomers seem to like anomalies.

pre-computer age over one hundred methods were developed to solve this problem. It drove the development of many numerical techniques.

Conceptually the procedure we want to follow is straightforward:

1. Calculate $t(\theta)$ for a grid of θ values using one of the two methods discussed above.
2. Construct a spline that inverts this to give $\theta(t)$.
3. Calculate the angular velocity, $\dot{\theta}(t)$, by calculating the first derivative of our spline. Note that we can do this at **any** time, t, so we will do this on a finely spaced time grid.
4. …
5. Profit?

Naturally it will not be quite so easy.

We will consider the motion of the Earth but the approach works for any object in any elliptical orbit around any other object. To this end it makes sense to define functions for the equations we use in terms of e and τ instead of plugging in numbers. Ultimately we do need numbers. For the Earth: $e = 0.01671123$ and $\tau = 365.25636$ days.

As a final note, we **are** solving a real problem but if we were going to do this for a real research problem we would need to be more careful than we are here. We would want to perform more checks, consider more methods, …. In particular we are not going to do a good job of exploring the accuracy of our methods.

The main result of this lab will be two figures. To make the steps you follow clear and to aid in grading I have created regions after each major part to include the details you feel are important. You should think of it as follows; "in a month and a half when we are working on the midterm exam with limited time and have forgotten all about this lab, what information will help us quickly understand what we did here."

1. Let us begin with the first equation for $t(\theta)$. Further let us only evaluate it at 360 values of θ, that is in steps of 1 degree. As noted above it is best to define this $t(\theta)$ as a function. Evaluate it at the array of θ values as suggested. You should find a couple of problems!
    a. The first issue is in what units do the trigonometric functions use to evaluate angles? All libraries I know of (that is, sane ones) use radians. It is easiest for us to think in terms of degrees but the computer wants radians, thus we need to convert our array, θ, into radians.
    b. The bigger issue is you should find negative values for $t(\theta)$. Clearly this cannot be correct! What happened? Look up the documentation for arctan(). The first

paragraph under Notes describes the problem. Multiple arguments, θ, have the same value of tan() so how do we know how to invert it? To more clearly see the problem evaluate arctan(tan(3*pi/4)). We do **not** get the value we put in! In this case the basic problem is what should the arctangent of -1 be? There is no correct answer, it depends on the situation. We need more information to get the correct answer for our particular case. For the arctangent we can do something about this. Back in kindergarten when we learned trigonometry we saw that $\tan\theta = y/x$. If we know the signs of x and y then we know what quadrant we are in so we know the correct value of θ. This is handled in math libraries by defining a function that takes two arguments, it is typically called atan2 or arctan2. In numpy it is called arctan2(). What matters when using this function are the signs of x and y. For our case we are evaluating something like $\tan^{-1}[\tan(\theta/2)]$. When we transition from $\theta < \pi$ to $\theta > \pi$ does x or y change sign? With the answer to this question we can now use the arctan2() function to correctly calculate the inverse and get a positive, monotonically increasing function $t(\theta)$. Note, to determine the sign of a number you can use the sign() function.

c. With the correction from the previous part we can now use scipy.optimize.InterpolatedUnivariateSpline() to construct the inverted function $\theta(t)$. To get the angular speed we calculate its first derivative. We have used a spline since it is a smooth function so we know we can calculate its derivative at any point. The spline has a method called derivatives(). It does the sensible thing of calculating all the derivatives at a given point (since we are using a cubic spline there are four derivatives, the zeroth, first, second, and third). Unfortunately what we want is the first derivative evaluated at an array of times so we have to some work.

d. First construct an array of times from 0 to the period. Let us use 5,000 values in this array. Normally when we do calculations we want to avoid writing loops, they are slow. Sometimes we cannot avoid it, this is such a case. Even so, it is advantageous to construct an array of the size we need to store the values we want to calculate. Given our array of times the zeros_like() function can be used to construct an array of the same size and type for storing the angular velocities we will calculate. If v is the array of velocities we can make a loop using len() to get the length of the array and xrange() to produce a list of integers as follows:

```
for j in xrange(len(v)) :
    v[j] = …. (fill in based on our problem)
```

e. We finally have calculated the angular velocity! Keep your time and angular velocity arrays handy as we will use them below.

2. Let us now repeat this procedure using Kepler's equation. In this case we really are calculating $t(\psi)$ and $\theta(\psi)$. Construct a ψ grid as you did for θ above. In fact, you can just reuse the old grid if you like.
    a. Calculate $t(\psi)$ from Kepler's equation. This is straightforward, there is no subtlety in using Kepler's equation.
    b. Next we want to calculate $\theta(\psi)$. Unfortunately this involves the arccosine and this suffers from the same problem as the arctangent, it is not uniquely defined. This case is even worse as we do not have the equivalent of arctan2() that we can use. We will instead rely on mathematics and use symmetry. For $\psi \le \pi$ there is no problem, we can invert using the arccos() function. When you look up its documentation you see it returns a value in the range 0 to π which is what we want. For $\psi > \pi$ we do have a problem as the arccos() function will still only return a value in the range 0 to π. There are a number of transformations we can use to get around this. I will point out one; notice that $\cos(2\pi - \theta) = \cos(\theta)$. Thus from the fact that $\cos(2\pi - \theta) = \frac{\cos(2\pi - \psi) - e}{1 - e\cos(2\pi - \psi)}$ we can calculate $\theta = 2\pi - \cos^{-1}\left[\frac{\cos(2\pi - \psi) - e}{1 - e\cos(2\pi - \psi)}\right]$ which is valid for $\psi \ge \pi$. We can now proceed as we did for calculating the angular speed in part 1d;[3] use zeros_like() to construct an array for θ of the same size as ψ. Use a loop to fill in the first half of the array directly then use another loop to fill in the second half of the array using the transformed relationship. Recall that len() gives us the length of an array so we can easily determine the index of the middle of the array. Further make sure you check the documentation of xrange() to see how to specify the starting and ending values; what should the ending value be for this second loop?
    c. Now that we have constructed the arrays we can calculate the angular velocity as in part 1d.

[3] This is a case where using array slicing is faster and easier. By being forced to write loops like we are doing here we will better appreciate the beauty of array slicing when we encounter it.

3. With our two arrays of speeds we can finally produce two plots to show the results. Include both figures below.
    a. **Main Figure:** For the main figure plot both versions of $\dot{\theta}(t)$. Though the computer likes to work in radians we prefer degrees so convert your angular speeds. Apply all the good practices you learned in the previous lab to this plot. Also draw the horizontal line for the average speed of the Earth, ω, in your figure. Note that we are plotting quantities with units and this should be reflected in your axis labels. Also note that matplotlib tries to guess good ranges for the axes. In this case we are plotting the time up to the period of the orbit. It does not make sense to go beyond this. To change the x-axis limits we can use xlim().
    b. **Extra Figure:** To help gauge how well the two methods we used agree calculate the absolute magnitude of the difference between the two speeds you calculated and normalize it by the average speed. It makes sense to plot this as a semilog plot using semilogy(). Again apply the good practices we have learned for making plots.

Enter the names of all people who worked on this lab as a group. By including a name here you are verifying that said person was actively involved in the work and understands the material:

Names or ids	