James Egan
Scientific Computing
Final Report
5/4/07

ODE's and Apogees: A Harrowing Tale of Survival

Author's Note: Rather than including a lot of .avi files in the email, I have included the .m files alone. These can be run as explained throughout the reading. Think of the .m files not merely as the results of incompetence in Matlab's avi functions, but rather as intentional supplements to the text.

Introduction:

The situation is dire indeed. Scientists posit that in less than a few decades, the Earth may very well be struck by a gigantic meteor. More specifically, there is about a 1/45000 chance that in the year 2036, we could collide with a meteor whose size is roughly that of North Carolina.

Not wanting the Earth to be unprepared, I decided to focus my efforts on solving this problem once and for all. I decided that, using my body of knowledge of Scientific Computing, I would be able to model both the problem and the solution in Matlab. There are those that say that the situation is under control. Scientists plan to launch a small probe in the near future that will act as a gravity pump. The probe will have rocket boosters so that it can stay alongside the meteor, gently nudging it off-track until there is 0 chance of collision. I decided to test whether such an idea could possibly work.

1D Modeling:

The main task for modeling this problem was to create in Matlab a set of .m files that could effectively model the gravitational interactions of multiple bodies. First, I wrote functions to solve for a system of bodies that can travel only along 1 dimension.

There are some things to note before running the function. First of all, note in the ODE function that at each step, we are simply adding up for each body the total forces imparted to it by the other bodies. That is, the equation:

$$F = (GMm) * r^{-2}$$

is calculated for each pair of bodies that involve the particular planet whose current force we are solving for.

Then at each step of the ODE, a planet's position changes by its current velocity, and its velocity changes by the calculated F.

Also note that for now, we are ignoring all possible collisions. We define *tol*, which is a relatively 'large enough' distance for this particular 1D example. While 2 planets are <= *tol* distance apart, they impart no forces on each other. This is just a placeholder procedure, we'll be able to detect collisions later on, when they really matter.

Run the function Xmotion_newest() and enjoy the show.

Notice the nifty plotting design used. For the 1D case, the size of each body roughly correlates with its mass. A preset list of different colors can accommodate up to 7 planets at a time!

Notice the smoothness of the animation? This was attained through use of the *deval* function. Because the ode45 function uses timesteps of variable lengths, a simple naïve plotting of the output will result in an unnatural animation, one in which time is rarely constant. By using *deval* on the structure output by ode45, we are able to calculate specific positions of the bodies at regular time intervals, which results in an overall smoother video.

Higher-Dimensional Modeling:

After creating a 1D model, the next step was to create a system of functions that could model the gravitational interactions between bodies that can travel in multiple dimensions.

Run XYmotion_new() now.

The output is much what one would expect for a 2-dimensional model. The XYmotionODE_new.m file is based on the process of simply splitting the total F up into components for each body.

For a simple implementation of XYZmotion_new(), the process is very similar - we just split the calculated F into X, Y and Z components. The actual implementation of XYZmotion_new() is more complicated, so we'll discuss it later.

Modeling the Earth-Moon System:

While in the 2D case, I decided to try and model the Earth-Moon orbit. I wanted to try and model the orbit in 2D first, and then go on to recreate the 3D orbit at a later time.

Using Wikipedia, I looked up all the pertinent details of the numerical relation between the Earth and the Moon.

My theory was that, even though I was modeling a solar system in which the Earth was stationary and no other heavenly bodies existed, recreating a 2D orbit would be simply a matter of choosing the correct initial X and Y velocities for the Moon.

In attempting this, I used Wikipedia's measurements for the Moon's Apogee (farthest point from the Earth) and Perigee (closest point to the Earth).

I gave the Moon an initial position of (Apogee, 0) and used *fsolve* to try and discover a pair of initial xVel and yVel that correctly got the Moon to the point (-Perigee, 0). If you take a look at the bottom of XYmotion_new.m (where the plotting is happening) I calculate the return value (X + Perigee) at the time when the Moon's y-position becomes 0 (after the initial timestep, of course). If at that point the Moon is located at (-Perigee, 0), the function would return 0.

(Currently, XYmotion_new() has initial values defined at the top. These are the results of my *fsolve* trials.)

Perhaps I needed to go about it more intelligently, or perhaps I was simple being too naïve in thinking that I could disregard variables such as spin, z-directional orbital components, pull of the Sun, and motion of the Earth. But iteratively running *fsolve* and even generously upping the FunEval number didn't yield acceptable results.

The actual information that eventually shows up in XYZmotion_new() is a set of initial data that I think looks pretty good after a while, at least visually. The correct masses and radii are entered, but initial velocities are just the best that I could get from multiple trials of *fsolve* and visual evaluation of the orbit.

Modeling the Earth-Moon-Meteor System:

Once I had successfully created a 3-dimensional gravity model of interacting bodies, and created the best-looking 2-dimensional representation of the orbit of the Moon around the Earth, it was time to add in the meteor. In XYZmotion_new(), the 2-dimensional orbit I calculated runs in the XY plane, and the meteor is coming in from the upper-left corner (-X, 0, Z). I entered in numbers for the meteor (again based on information from Wikipedia).

The plan is to send a body of our own out from the Earth to the upper-left corner (with a velocity of -X, 0, Z) towards the oncoming meteor. Additionally we give the body some constant vector force F, so that it can nudge the meteor away from the Earth.

The key difference in the way XYZmotion_new() works is that here we are actually doing something about collisions. Each planet now has a radius, and as we are calculating

the force between 2 planets, we check their radii to ensure they haven't collided. If a collision has occurred, we return *NaN* and are done. The function knows not to plot any *NaN* timesteps.

Run XYZmotion_new() to see how things look. I replaced the hollow circles with more appropriately 3D-looking filled-in circles.

My overall strategy was to use Matlab's *fminsearch* function to calculate the constant F that should exist on the expelled body.

My first idea was to work backwards from an ideal arrangement. That is, given the model with a fixed Earth and 1 Moon, the best arrangement would be one in which we could get the meteor and expelled body to pass right along the outside of the Moon, on the side opposite the Earth. That way they could both go shooting off into the nothingness of space. My first strategy was to set up that ideal arrangement (based on how I knew the Earth and Moon would act) and work backwards from there.

The problem with this strategy is that you can't really use ode45 timeslices as initial conditions. That is, because of the way that ode45 wants to choose the timesteps, taking a position/velocity from 1 ode and making it the initial position/velocity of another will not result in the same behavior.

After discovering that, I tried to find something that I *could* calculate and then use to get the ideal behavior, some aspect of the meteor's path I could set. So first I tried to simply have the function return the minimum distance ever reached between the meteor and the expelled body. Doing *fminsearch* on this, I was able to find a constant F that met those requirements.

But getting these bodies close together doesn't imply anything about resulting behavior. I tried next to make sure that the bodies got close *and* that they had the same vector velocities when they were close. That way, the constant F on the expelled body could greatly influence the meteor's path away from the Earth.

But I wasn't able to really influence the meteor enough to avoid a collision. I realized that nothing could be done with the meteor starting out as close as it was (after all, the scientists are planning to act while the meteor is still years and years in coming). So I punted the meteor out to the extreme upper-right corner, and tried to repeat the process.

Now is where my faulty Earth-Moon orbit calculations came back to haunt me. After a not-too-long amount of time, the Moon eventually crashes into the Earth. Since the actions of the meteor and the expelled body occur far away from the Earth and the Moon, I decided to solve this problem by combining the Earth and Moon into a single large body whose mass and radius were the sum of those of both.

And that is where I currently am. XYZmotion_findF([xF yF zF]) contains my current attempts at finding both the spot at which the two bodies are close and when they have

similar velocities. I ran fminsearch(@XYZmotion_findF(F) to try and get these minimal initial conditions, but the skews on my results are not yet fine enough. And of course XYZmotion_disasterAverted() is all set to take in those values for F once I find them.

Future Work.....

If I were to spend more time on this project, I would try to start back at square 1 in the modeling, at least until I could get an accurate depiction of the Earth-Moon orbit. Also, I would come up with more partitioned strategies for using *fminsearch* and *fsolve* (and look harder into what other functions I could use in their place). That is, I tried to do a lot of things in single function calls.

Even though I ran into a lot of snags, though, I'm pleased with the work I did. It's a good foundation of methods for how to tackle this problem. And besides, I've got some 20-odd years before I *really* have to do anything about this meteor (so fear not, citizens).