Princeton University
Department of Operations Research
and Financial Engineering

ORF 201
Computer Methods in Problem Solving

Lab 5: Earth–Moon Simulation

Due Sunday, Mar 5, 11:59 pm

---

1. INTRODUCTION

This assignment is about simulating the motion of heavenly bodies as they move under the influence of gravity. We will first describe the physics that underlies the simulation. If you are rusty on physics (or haven't taken the relevant course yet), have no fear since all the important formulas will be carefully laid out and you should be able to program them even if you don't fully understand them.

2. NEWTON'S LAW OF GRAVITATION

Newton's *law of gravitation* says that two masses, m_1 and m_2, experience an attractive force whose magnitude is given by

$$F = G\frac{m_1 m_2}{r^2}.$$

Here G is a universal constant (called the *Gravitation constant*) and r denotes the distance between the two masses. Of course, force is really a vector—the above formula only specifies its length. The assertion that the force is *attractive* determines its direction. Indeed, the magnitude given above must be multiplied times a unit vector pointing in the correct direction. Let $\mathbf{p}_1 = (x_1, y_1)$ be the position vector[1] for body 1 and let $\mathbf{p}_2 = (x_2, y_2)$ denote the position vector for body 2 (these position vectors are relative to some arbitrary coordinate system). Then, the distance r between the two bodies is given by

$$r = \|\mathbf{p}_1 - \mathbf{p}_2\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Consider the force, $\mathbf{F}_1$, on body 1 caused by the gravitational attraction of body 2. A unit vector pointing toward body 2 is given by

$$\frac{\mathbf{p}_2 - \mathbf{p}_1}{r}$$

[1] We use boldface to denote vectors. In this lab all vectors are assumed to be two-dimensional.

and so the force vector is given by

$$\begin{aligned}\mathbf{F}_1 &= G\frac{m_1 m_2}{r^2}\,\frac{\mathbf{p}_2-\mathbf{p}_1}{r}\\ &= G\frac{m_1 m_2}{r^3}(\mathbf{p}_2-\mathbf{p}_1)\end{aligned}$$

3. Newton's Second Law of Motion

In addition to his law of gravitation, Newton also formulated laws of motion. *Newton's second law of motion* says that the acceleration $\mathbf{a}=(a^x,a^y)$ on a body is equal to the force $\mathbf{F}=(F^x,F^y)$ divided by the mass:

$$\mathbf{a}=\mathbf{F}/m.$$

Therefore, the acceleration of body 1 due to the gravitational pull of body 2 is given by

$$\mathbf{a}_1=\mathbf{F}_1/m_1=G\frac{m_2}{r^3}(\mathbf{p}_2-\mathbf{p}_1). \tag{1}$$

Similarly, the acceleration of body 2 due to the gravitational pull of body 1 is given by

$$\mathbf{a}_2=G\frac{m_1}{r^3}(\mathbf{p}_1-\mathbf{p}_2). \tag{2}$$

4. Computer Simulation

To make a computer simulation, we discretize time and look at successive discrete points in time. We can think of this as a sequence of *snapshots* that are displayed in rapid succession to make a movie. We assume that the time interval between these snapshots is a small number Δt. We also assume that positions and accelerations are computed at the endpoints of the time intervals but that velocities are computed at the midpoints of the time intervals. Finally, we assume that we are given initial position and velocity vectors for both bodies and that the initial velocity vector corresponds to the midpoint of the first time interval.

To understand how the calculations go, suppose that the simulation has been running for awhile and let's see how one updates the position, velocity, and acceleration over one time interval. Let t denote the current time, which is at the beginning of one of these discrete time intervals. We have the position at time t and the velocity at time $t+\Delta t/2$. We assume that the velocity at this midpoint time is a good approximation to the average velocity over the entire time interval $[t,t+\Delta t]$ and so update the position vectors by incrementing them by the velocity times the length of the time interval:

$$\begin{aligned}\mathbf{p}_1(t+\Delta t) &= \mathbf{p}_1(t)+\mathbf{v}_1(t+\Delta t/2)\Delta t\\ \mathbf{p}_2(t+\Delta t) &= \mathbf{p}_2(t)+\mathbf{v}_2(t+\Delta t/2)\Delta t.\end{aligned}$$

Now that new positions are known (i.e., at the beginning of the next time interval), we can use equations (1) and (2) to compute acceleration vectors at the new time (that is, $t+\Delta t$). Given the acceleration vector $a_1(t+\Delta t)$, we assume that it is a good approximation to the average

Time	Position $\mathbf{p} = (x, y)$				Velocity $\mathbf{v} = (v^x, v^y)$				Acceleration $\mathbf{a} = (a^x, a^y)$			
	Earth		Moon		Earth		Moon		Earth		Moon	
0.0	0.0	0.0	1.0	0.0								
					0.0	-1.0	0.0	1.0				
0.1	0.0	-0.1	1.0	0.1					1.886	0.377	-1.886	-0.377
					0.189	-0.962	-0.189	0.962				
0.2	0.019	-0.196	0.981	0.196					1.715	0.699	-1.715	-0.699
					0.36	-0.892	-0.36	0.892				
0.3	0.055	-0.285	0.945	0.285					1.505	0.965	-1.505	-0.965
					0.511	-0.796	-0.511	0.796				
0.4	0.106	-0.365	0.894	0.365					1.271	1.178	-1.271	-1.178
					0.638	-0.678	-0.638	0.678				
0.5	0.17	-0.433	0.83	0.433					1.023	1.341	-1.023	-1.341
					0.74	-0.544	-0.74	0.54				
0.6	0.244	-0.487	0.756	0.487					0.768	1.46	-0.768	-1.46
					0.817	-0.398	-0.817	0.398				
0.7	0.325	-0.527	0.675	0.527					0.51	1.54	-0.51	-1.54
					0.868	-0.244	-0.868	0.244				

TABLE 1. $G = 1$, $m_1 = m_2 = 2$, $\Delta t = 0.1$

acceleration over the midpoint-to-midpoint time interval $[t + \Delta t/2, t + 3\Delta t/2]$ and update the velocity as follows:

$$\mathbf{v}_1(t + 3\Delta t/2) = \mathbf{v}_1(t + \Delta t/2) + \mathbf{a}_1(t + \Delta t)\Delta t.$$

The velocity of body 2 is updated in a similar fashion:

$$\mathbf{v}_2(t + 3\Delta t/2) = \mathbf{v}_2(t + \Delta t/2) + \mathbf{a}_2(t + \Delta t)\Delta t.$$

At this point, we iterate the process just described to carry out the simulation. A sample calculation using $G = 1$, $m_1 = m_2 = 2$, $\Delta t = 0.1$ is shown in Figure 1. The first two lines represent initial data. Each subsequent line is computed in succession. Note that each computation requires knowledge of only the previous two lines of the table.

5. GETTING STARTED

First, you need to create some directories and copy some files. Begin like this:

```
cd public_html/JAVA/ORF201
mkdir gravity
cd gravity
```

Then copy the following files:

```
cp /u/orf201/public_html/JAVA/ORF201/gravity/index.html .
cp /u/orf201/public_html/JAVA/ORF201/gravity/gravity.html .
```

```
cp /u/orf201/public_html/JAVA/ORF201/gravity/Gravity.java .
```

6. COMPILING

First, compile the java code that you just copied over:

```
javac Gravity.java
```

Then use appletviewer to see what the code currently does:

```
appletviewer gravity.html
```

An applet window should pop up with some editable text fields, two buttons (start and stop), and an empty drawing canvas below. At the moment, if you push the start button nothing happens. That is because the main part of the code has been stripped out. It is your job to fill it in according to the instructions given below.

7. PROGRAMMING NOTES

Just as in last week's assignment, the Java code you are given as a starting point contains the "user interface" part but has the "core" part of the code deleted. It is your job to recreate the core part. The code that has been deleted is marked in several places with comments indicating the type of stuff that needs to go there. In addition to filling in these obvious points of deletion, you might find it convenient to write some *methods* that will facilitate your programming. Feel free to do this.

As you can see from the default values in the textfields, our system of units is clearly not the MKS system. In fact, since the mass of the moon is taken to be one, we refer to our unit of mass as *moon units*. Distance and time units are also nonstandard. We refer to the entire system as the *Zappa* system. In the Zappa system, the universal constant of gravitation is exactly 1.

On most computers, using a time step in the range of 0.01 to 0.1 gives a fairly reasonable simulation.

The only graphics functions you'll need this week are `GL.circf()` `GL.color()`. Be sure to choose the radius of the circle carefully (note the coordinate system defined by the call to `GL.ortho2()`). Also, the radius should be an appropriate function of the mass.

8. MINIMUM REQUIREMENTS

At a minimum:

(1) Your simulation must correctly implement the model of gravitation described in this assignment.
(2) It must show the Earth and Moon revolving around each other.
(3) It must work for a variety of input masses, positions, and velocities.
(4) The apparent size of the Earth/Moon must reflect the mass.