Name: _____ , _____

PHYSICS 401 : SPRING SEMESTER 2020

Project #4: The 3-body problem and on-screen animations

Please note: Do this assignment early, and you will know the excitement of discovery. Be the first to see the secret three-body orbit, but don't show this to your classmates, if they haven't done it yet. Everyone deserves to see the orbit completed for the first time, by him/herself.

Reference Reading: Sections 3.1-3.3; 5.1-5.3, 5.6

Downloads:

1) Download `MovingKObject.java` and `PlanetKApp.java` from my website to your Java working directory, say `javawork`, which contains the open source physics folder `org`.

2) Compile `MovingKObject.java`, then `PlanetKApp.java`, and run `PlanetKApp`, as shown in class.

1. `PlanetKApp.java` is the main program which create the **input panel** and the **animation window**. The subprogram `MovingKObject.java` supplies the drawable items in `draw` and the animation updating `doStep`. The speed of the animation is controlled by the loop number of the for-loop in the main program. Instead of changing this number in the main program by hand, let's import it directly on-screen just like the initial positions and velocities. Let this integer be `nspeed`, declare it in `MovingKObject.java` as `int nspeed` and import it to `PlanetKApp.java` on-screen by following the example of importing the initial position `x`. Set its default value as 10. Now replace the number `10` in the main program by `planet.nspeed`. Recompile and run. When the program is working, check that when you changed `nspeed` to 100 or 1000, the animation is greatly speeded up.

2. Now replace the two function calls in the `doStep` method of `MovingKObject.java` by your symplectic integrator `sym2bstep(double cof)` for solving the Kepler problem. (This is the case of positionstep at dt/2, velocitystep at dt and positionstep at dt/2.)

a) Recompile and run at `dt=0.2`, stop after 10 periods, hand in an image of this run. (Click "File" on the drawing panel, then "Save image", then "eps". Eps is the best, withmuch greater resolution than the other two formats.)

b) In `PlanetKApp.java`, uncomment the line `planet.trail.clear()` in the `doStep` method and run at `dt=0.1`. Observe the trail as you set `nspeed` =10, 100, 1000. Describe what is happening when you set `nspeed` =759. What is the signifiance of this number? Hand in an image of this case when the elliplical orbit is at 6 o'clock. (The orbit starts at 3 o'clock.) Now set `dt=0.2` and `nspeed` =30,000. Describe what you see.

3. Save `MovingKObject.java` as `MovingTObject.java` ("T"=Three-Body). Inside `MovingTObject.java`, be sure to change `class MovingKObject` to `class MovingTObject` and carefully replace the Keplerian acceleration with the restricted three-body, time-dependent acceleration

$$\mathbf{a}(t) = -\frac{1}{2}\Big(\frac{\mathbf{d}_1(t)}{d_1^3(t)} + \frac{\mathbf{d}_2(t)}{d_2^3(t)}\Big), \qquad \text{with} \qquad \mathbf{d}_1(t) = \mathbf{r} - \mathbf{r}_1(t), \qquad \mathbf{d}_2(t) = \mathbf{r} - \mathbf{r}_2(t).$$

where $\mathbf{r} = (x, y)$ is the position of our third body, and where the two massive bodies revolve around each other in circular motion with

$$\mathbf{r}_1(t) = (\,0.5\cos(t), 0.5\sin(t)\,), \qquad \mathbf{r}_2(t) = -\mathbf{r}_1(t).$$

At $t = 0$, $\mathbf{r}_1(t) = (0.5, 0)$ and $\mathbf{r}_2(t) = (-0.5, 0)$. You now declare `public void accel(double t){.....}` and call it via `accel(t);`. You must also modify your `sym2bstep(double cof)` method to keep track of time:

```
postionstep(0.5*cof);
t=t+0.5*cof*dt;
velocitystep(1.0*cof);
positionstep(0.5*cof);
t=t+0.5*dt*cof;
```

The rule is to update t after every positionstep with the positionstep's time-step coefficient: `t=t+coeff*dt`. In `doStep` remove the line `t=t+dt`, since the time is already updated in `sym2bstep(double cof)`. You must declare all the new variables you use in `accel(double t)`.

In the `draw` part, replace the "sun" at the origin by the position of $\mathbf{r}_1(t)$. Create another "sun" with color GREEN at the position of $\mathbf{r}_2(t)$.
In the main program `PlanetKApp.java`, replace `MovingKObject` by `MovingTObject` everywhere, make the animation window scale as

```
frame.setPreferredMinMax(-1.2, 1.2, -1.2, 1.2);
```

and comment out the line `planet.trail.clear()` in the `doStep`. Set the default starting values as
$\mathbf{r}_0 = (0.0, 0.058)$, $\mathbf{v}_0 = (0.49, 0)$, $dt = 0.001$ and $nspeed = 200$.

Compile `MovingTObject`, `PlanetKApp.java` and run `PlanetKApp`. Run the program long enough so that and you can see the completed, closed, secret orbit. Hand in an image of this closed orbit. (Be sure that the aspect ratio is correct, that the overall shape of the orbit is circular.) What is the period of this orbit? (This is a tricky question. A completed period is where all three bodies have returned to their starting positions. Hint: what is the period of each massive body?)

4. Instead of plotting the orbit in the space-frame, we can also animate the orbit in the co-rotating frame in which the two massive bodies are at rest. To do this, we counter-rotate the trajectory and let `trail` and the `RED` body have coordinates `xrot` and `yrot` defined below:

```
rcos=Math.cos(t)
rsin=Math.sin(t)
xrot= rcos*x+rsin*y
yrot=-rsin*x+rcos*y.
```

We also fix the two massive bodies in their starting positions. This orbit has a ballet-like quality. Hand in an image of a completed orbit in the rotationg frame.

5. Hand in another other *closed* orbit in the rotating frame for 10 bonus points. (But don't waste time on this, it is very difficult to find one.) Hand-in the one chaotic orbit. (This is easy, almost any starting point will do. Try to find one that is *most* chaotic looking in the rotating frame.)