FinalProject-NBodyDynamics

[redacted]

Due before class (1:00 pm) on the last day of class: Friday August 5.

Your task is to write a gravitational N-body code: a program that evolves a system of N particles over time, accounting for their mutual gravitational forces. Starting from initial arrays of particle masses, positions, and velocities, your code will need to step through many time points and calculate the instaneous accelerations, velocities, and positions of the particles at these times. At each time, your code must calculate the vector forces among all N particles, use those forces to determine the instantaneous accelerations, update both the velocities and positions of the particles, store those values, and then repeat this process over and over again for every time point.

This may sound daunting, but don't worry! You will be given a module to do the physics for you and you can break the rest down into smaller steps.

This project is designed to synthesize many of the topics we have covered in class. For the project, you will use variables, functions, loops, arrays, file input/output, plotting, numerical integration, and more. You will need to tie multiple concepts together to get through it: a good strategy throughout is to make sure you are *very* clear on what each aspect of the code should do, and test each component to make sure they work in isolation, before linking them all together.

Some parts of the project build off of earlier ones, but some can be completed in isolation. If you get stuck on one part, keep going; you may be able to finish later parts of the project. As with other assignments, you should feel free to discuss with other students, but you must turn in your own work.

To access the provided python modules, you will either need to (Anaconda folks) download them and place them in your working directory (or in your `$PYTHONPATH`) or (JupyterHub folks) add the appropriate directory to your path as done in previous tutorials and homeworks.

1 N-body Dynamics: Background and Definitions

Previously, you wrote code to calculate the net gravitational force on particle i due to all other particles in a system of N total particles. This net force was a sum of many individual force vectors

$$\vec{F}_i = \sum_{i \neq j} \vec{F}_{ij}$$

where $\vec{F}_{ij}$ represents the individual force of particle j acting on particle i. As outlined below, in this project you will combine your force calculation capabilities with equations of motion, to calculate the trajectories of multiple particles all acting under each others' gravity.

Let's express the position of particle i at time t as a position vector $\vec{r}_i(t) = (x_i, y_i, z_i)$, the velocity vector of this particle as $\vec{v}_i(t)$, and the acceleration vector of this particle as $\vec{a}_i(t)$.

Mathematically, we can determine these time-dependent vectors for the particle by solving the following equations

$$\begin{aligned}
\vec{r}_i(t) &= \vec{r}_i(t_0) + \int_{t_0}^{t} \vec{v}_i(t')dt' \quad (1)\\
\vec{v}_i(t) &= \vec{v}_i(t_0) + \int_{t_0}^{t} \vec{a}_i(t')dt' \quad (2)\\
\vec{a}_i(t) &= \frac{\vec{F}_i(t)}{m_i} \quad (3)
\end{aligned}$$

where $\vec{F}_i(t)$ is the force vector acting on the particle at time t, which is calculated from the positions and masses of all of the other particles. In some cases we could solve this system of equations analytically, but that won't be the case in general.

Numerically, we can approximate these integrals by breaking them up into discrete time steps. If we make these steps small enough, then we should get a fairly accurate approximation to how positions and velocities change with time. We can use a common numerical integration method called the 'leapfrog' integrator to update positions and velocities from time t to a time Δt later, provided Δt is small enough:

$$\begin{aligned}
\vec{r}_i(t+\Delta t) &= \vec{r}_i(t) + \vec{v}_i(t)\Delta t + \frac{1}{2}\vec{a}_i(t)\Delta t^2 \quad (4)\\
\vec{v}_i(t+\Delta t) &= \vec{v}_i(t) + \frac{1}{2}\left[\vec{a}_i(t) + \vec{a}_i(t+\Delta t)\right]\Delta t \quad (5)\\
\vec{a}_i(t+\Delta t) &= \frac{\vec{F}_i(t+\Delta t)}{m_i} \quad (6)
\end{aligned}$$

This leapfrog integrator is famous for being a good balance between accuracy and complexity. In this project, I will provide you with a Python function called `updateParticles` that encodes a single step of this leapfrog integration. Your job will be to link multiple steps together and to calculate, store, and visualize the xyz trajectories of multiple particles over a long time span.

2 Write Calculation Tools:

|1| (2%) Make sure you have access to a function that calculates net force vectors from a list of masses and a list of positions. You wrote such a function in HomeworkD; feel free to use either you own version or `forces.py` on Canvas and `/home/hama2717/astr2600/nbody/code/`. Test this function in a situation where you know the answer. (Explain what you expect the output to be and why.) The net force vectors can either be a list of 3-element `numpy` arrays (one force vector for each mass) or a 2D numpy array.

|**2**| (3%) Various code snippets relevant to this project are stored in `/home/hama2717/astr2600/nbody/code/` on `scorpius` (they are also available on Canvas). Import the `updateParticles` function from the `leapfrog.py` into your `jupyter` notebook. Display the docstring for `updateParticles` so you know how it is called. When called, this function executes one time step of the leap-frog integrator. `updateParticles` imports and calls a function called `calculateForceVectors` to determine the net forces on the particles; you may need to modify this to link it with your force calculation function (and all of the other functions it may depend on) if you're not using the previously referenced `forces.py`. Test this function in a situation where you know the answer. (Explain what you expect the output to be and why.)

|**3**| (20%) Write a function called `calculateTrajectories` whose purpose is to evolve particle initial positions and initial velocities in time. It should start from initial positions and velocities, take a step forward in time, calculate new positions and velocities, store these new values in arrays, and then repeat that process over and over again. Inside this loop, your function should call the `updateParticles` function whenever you want to advance the positions and velocities by a time step. Be *sure* you are very clear on how to call `updateParticles` in from the previous step! Your function *must* contain a detailed docstring.

- The outputs (return values) of the function *must* be:
  - times (1D array, with elements identifying each time step)
  - positions at all times (multi-dimensional array or arrays)
  - velocities at all times (multi-dimensional array or arrays)

Your returned times, positions, and velocities arrays should all contain your initial time (t=0), positions, and velocities as well.

The following describes one possible way that your `calculateTrajectories` function could work. Here, `nParticles` is the number of particles, `nDimensions=3` is the number of spatial dimensions (x, y, z), and `nTimes` is the number of different time points where you calculate positions and velocities.

- The inputs to this function could be:
  - masses (1D array, with `nParticles` elements)
  - initial positions (2D array, `nParticles` × `nDimensions` elements)
  - initial velocities (2D array, `nParticles` × `nDimensions` elements)
  - the total time to evolve the system (a float, in seconds)
  - the size of each time step, (a float, in seconds)
- The outputs (return values) of the function could be:
  - times (1D array, with `nTimes` elements)
  - positions at all times (3D array, `nTimes` × `nParticles` × `nDimensions`)
  - velocities at all times (3D array, `nTimes` × `nParticles` × `nDimensions`)

This structure is provided to try to help you keep things tidy, but feel free to organize your code differently. For example, if you prefer, you could store x, y, z, v_x, v_y, v_z as separate 2D arrays (with shape of `nTimes` × `nParticles`) instead of using the 3D arrays above. Additionally the axes of your arrays may have a slightly different ordering depending on how you create them (e.g. `nTimes` × `nParticles` × `nDimensions` vs. `nParticles` × `nDimensions` × `nTimes`).

If you get confused about how to access portions of your arrays, have a look at their shapes. Be sure you are clear on which axes represent which attributes (TimeStep vs Particle# vs Physical

Dimension).

3 Earth in (circular) orbit around the Sun:

|**4**| (10%) Test your code with the following 2-body problem, which approximates the Earth in a circular orbit around the Sun.

particle #	mass (kg)	x (AU)	y (AU)	z (AU)	v_x (m/s)	v_y (m/s)	v_z (m/s)
0	1.989e30	-3e-6	0.0	0.0	0.0	-8.94e-2	0.0
1	5.972e24	0.999997	0.0	0.0	0.0	2.98e4	0.0

Evolve this system for a total time of 1000 days, using time-steps of $\Delta t = 0.1$ days. Be careful with units! Your `calculateTrajectories` code uses SI units.

|**5**| (10%) Make three plots summarizing the results:

- One plot should have time on the horizontal axis, and show the x positions of both particles on the vertical axis. Estimate the period of orbit by eye; does this make sense?
- One plot should show the trajectories that the particles trace out in the $x - y$ plane. *The Earth's orbit should be pretty close to a perfect circle. Scale the axes so it appears this way.*
- One plot should show the x velocity of the Sun as a function of time. Based on this plot, to what precision would we need to measure the radial velocity of a Sun-like star to see an Earth-like planet orbiting it? *(Estimate this by eye.)*

For all positions and times, convert back to AU and days for the above plots.

|**6**| (5%) Repeat the Earth-Sun calculation and plots, starting from the same initial conditions but *(with python)* decreasing the initial velocities by a factor of 2. How does the orbit change?

4 The Circumbinary Exoplanet Kepler-16ABb:

|**7**| (5%) Read in the initial conditions describing the Kepler-16ABb circumbinary planet system (Two stars & one planet), stored in the file `/home/hama2717/astr2600/nbody/initialconditions/kepler16.txt` (and on Canvas). The file contains 7 columns: one for masses, three for positions, and three for velocities. Read in the data file and organize these initial conditions into arrays that could be fed as inputs to your `calculateTrajectories` function.

The data stored in this file, for three particles, should look like this:

mass (kg)	x (m)	y (m)	z(m)	vx (m/s)	vy (m/s)	vz (m/s)
1.372e+30	7.635e+09	-9.659e+05	1.845e+09	-1.05e+03	-6.822	1.303e+04
4.029e+29	-2.6e+10	3.289e+06	-6.281e+09	3.576e+03	23.23	-4.437e+04

mass (kg)	x (m)	y (m)	z(m)	vx (m/s)	vy (m/s)	vz (m/s)
6.325e+26	3.776e+10	-4.092e+07	8.193e+10	-3.63e+04	-10.81	1.702e+04

Note you have 3 particles and 3 dimensions. Don't mix up how they are each stored in your arrays!

|**8**| (5%) Plot the initial positions of these particles in the $x - z$ plane (in AU units). Scale the symbol area to be proportional to the particle masses. To each point, add a vector signifying the initial velocity of that point. Make your code flexible enough that it doesn't depend on hard coding the fact that there are 3 objects.

|**9**| (5%) Use your N-body code to calculate the planet and stars' orbits. Evolve the system for a total of 500 days, using a time step of $\Delta t = 0.5$ days.

|**10**| (10%) Make an animation, showing the motion of the three particles over that 500 days. From frame to frame in this movie, the positions of the particles should move, and the title of the plot should update to reflect the passage of time. Units for the plot should be AU and days.

5 Choose Your Own Adventure

|**11**| (5%) The scorpius directory `/home/hama2717/astr2600/nbody/initialconditions/` (also posted to Canvas) contains additional sets of initial conditions in txt file form. Alternatively, there is a Python function (`systems.py`) that can be used to generate your own initial conditions according to some customizable parameters. Choose a set of initial conditions from among these options and read in the appropriate data file (or make your own) and calculate their trajectories. You might need to experiment with the step size and length of time you evolve over to see how the system behaves. After you start working on the Final Project, I will post a few details about some of the datasets in a Canvas announcement. Be sure you check in to see if any specific hints pertain to the dataset you chose.

|**12**| (10%) Visualize your four-dimensional results in some manner that captures *all four dimensions*. For example, this could be a 3D movie, separate panels plotting xyz positions as a function of time, different projections of the trajectories with arrows or color gradients expressing time, a movie in two or three panels, a 3D movie for stereoscopic red-blue glasses, or other representations you can think of.

6 Code Check & Revision

|**13**| (10%) The revision process is an extremely important part of coding. To encourage you to revise and improve your code, 10% of the grade for this assignment will be awarded for submitting

a draft version of your project by August 1 at 5pm. To earn the full 10%, this draft *does not* need to be complete or polished; it merely needs to demonstrate serious effort has been made. If you don't have code written for some section, you *must* briefly write out (in human words) your plan for how to approach it, some pseudocode, some question you would like answered about it, or where you are stuck. I will provide feedback/assistance within a couple of days, if appropriate.

It is not possible to get >90% on the final project if you do not turn in a draft by August 1st at 5pm.