Growth Memo Project 2
N Body Dynamics with Monte Carlo Sampling

Nodoka Masamune

September 2025

1

The scientific goal of this project was to determine which out of the Euler method, RK2, RK4, and leapfrog integration method produced the most accurate long term simulations, and then simulate the dynamics of various N-body systems, ultimately simulating a stellar cluster.

The technical challenge was writing code that could calculate the steps of the integration methods, as well as writing code that could generate a stellar cluster that reflected observed stellar clusters.

Through this project I gained a far deeper understanding of how numpy arrays worked. Previously I worked with two dimensional arrays, but only in a very shallow capacity. I had to develop an understanding of three dimensional arrays- not only creating them, but also manipulating them and calling specific columns, rows, and indices, and storing these values. This was definitely a slow process at the beginning! Every time I began array splicing I would have to have a test run with a very small array to remind myself how it worked exactly.

I also learned how to write doc-strings for my .py files! This was the first time where anything like that had been required in my code, as this is also the first class where I'm turning in complex code and packages to be graded.

I think the key challenge I faced was in modifying my initial integrators from 2D to 3D. This increased the complexity of the arrays I was working with significantly, and I hadn't anticipated struggling as much as I did. Again, to remind myself how array slicing worked I would generate a very small, simple array and have examples of what different slices would produce. I also drew a lot of the array shapes I wanted out in my notebook, and would compare the shapes of the arrays my code would generate with what I really wanted. Having a sketch of what I want was incredibly helpful, and I think I'll continue to rely on pen and paper as I code.

Having to type out and encode every single calculation done in a single step of integration for all the methods really meant that I had to understand how each integrator worked. This was a struggle at the beginning, but I could probably explain them in my sleep now. I feel similarly about the Kroupa IMF and Plummer sphere models. While the math was initially very overwhelming, going through and writing down each mathematical step and then later coding it made it far more understandable (shocking, I know!). This is doubly true for me since this is my first time hearing of either the Kroupa IMF or Plummer models (since my background is in physics a lot of this is new to me!).

Through coding an ODE class I definitely feel like I've gained a better understanding of how classes work. Last time I still felt a little confused at the end of the project, but now I feel a lot more confident in my ability to write a class without having to refer back to older code as much as I did this time around.

I tried not to use AI as much for this project as well. Since it still feels like I'm learning fundamentals, overly relying on AI at this point in my learning feels like it would do more harm than good. The one part where I ended up using it a lot was during my attempts at vectorization, where I would give it an example of my inputs and what I want to output (such as 'I want to multiply two arrays, [a , b] and [d, e] and I want to get a 2 x 2 array [[ad, ae][bd, be]]. What function should I use?' This was very helpful since there are so many different array multiplication and addition functions, and finding the exact function I wanted proved to be very difficult. Ultimately, I think I will continue to use AI this way- not as a coding tool or generator per se, but a more efficient and succinct Google without having to wade through outdated StackExchange interactions.

I think a really cool part of this project is being able to plot the projected trajectories of Earth and Jupiter for the 2- and 3- body simulations. It's a really cool way to visually confirm that my integration code was working as expected. I think a cool extension would've been to create an animation showing the locations of Earth and Jupiter as time went on, even though it's much smaller than the ultimate scale of the project. This was especially true for Jupiter since I had to modify my code a lot so that it worked for more than paired interactions.

I definitely want to try vectorizing my code properly. Before realizing I had to vectorize it, I had written a lot of my integrator and it was a mishmash of arrays and loops. This made it pretty difficult to vectorize fully in a meaningful way, and it felt like I would have to rework through my understanding of how to calculate everything and rewrite large sections of code. Unfortunately I wasn't able to fully vectorize it before the deadline, but I would love to be able to simulate larger populations for longer times, so I'm definitely interested in getting it to work so that I can do that without running code for hours.

Another thing I would like to improve are my plotting functions. Since I ended up hardcoding a lot of things like titles and file names I wasn't able to define a single function that worked for every plot generated; however, I know there's definitely a way to be able to set titles and file names in a way that means it would work in a loop. This would decrease the lines of code I would have to write and would also increase the readability of my scripts significantly.

I think I've made a lot of progress in my understanding of how to structure large scale projects that involve multiple .py scripts and outputs. I definitely feel far more confident in my submission this time compared to my last project.

I think I would tell myself to approach my initial ODE integrators script with the mindset to get it to work for more bodies and more dimensions than just the Earth Sun 2D test. I think I would've saved myself a lot of frustration if I didn't hardcode things into my integrator that I later had to drastically change, which also meant going over my code with a fine tooth comb to make sure I had replaced every instance of hardcoding with the proper variable.