Ariel Gans
Ms Woodford
4 June, 2018

# N-Body Simulation

## Comp Sci 12 ISU

## Pros and Cons of Using Python

A pro of python as always is the ease of mapping out ideas. One more specific thing that I found useful was the versatility of the *list* file type. *List* allows multiple variable types in one array-flavoured variable, which allowed me to store all of my planet data in one place. *Lists* also have versatile "slicing" capabilities, allowing me to have my functions give two outputs and to slot those outputs right into multiple indices in the *list* at once. A huge con is definitely python's sluggish runtimes. While the code may have been harder to write in a language like C++, it would definitely run faster, which is a huge plus when you're trying to simulate 100 particles with any reasonable level of accuracy.

## Difficulties

Most of the time I spent on this project was spent trying to figure out why my code would sometimes send planets in the complete wrong direction. Eventually I figured out that the problem stemmed from how I was calculating the angle between planets. I was using a function in the *math* library called *atan*, which takes a float ratio and returns your angle. The problem with this function is that turning the two floats into a ratio means the function doesn't know which quadrant the angle is in, and guesses. This means it gets it wrong half of the time. Luckily, there's a function called *atan2* which takes the two floats as separate arguments so that their signs are preserved. Using *atan2* in place of at*an* fixed all my problems.

## History of the N-Body Simulation

Issac Newton has been doing N-Body Simulations since the 17th century when he cracked the basic physics behind the movements of the planets. An important discovery of Newton's is that not only was the sun affecting the positions of the planets as they move around it, but the planets were affecting each other in the same way they were being affected by the sun. This is the core of the N-Body problem. You can't just take an initial position and velocity and extrapolate from there. You need to know all the gravitational forces. Newton actually concluded that the N-Body problem is unsolvable since you need to calculate each step with an infinitely small timescale to account for the fact that the positions of every body are constantly changing. What he didn't know is that the universe doesn't work with continuous motion. My simulation doesn't really solve the N-Body problem and it links back to why Newton concluded that the N-Body problem is unsolvable. My simulation is limited by the fact that it has to recalculate every timestep, which means it's a little bit wrong each time. A perfect N-Body simulation calculates continuously.

## Example

I couldn't find *the best* N-Body simulation but here's one off github: https://github.com/harrism/mini-nbody

## The Code

My script is split into four functions.

The *main* function is a pretty ordinary *main* function. It initiates the *argparse* so that arguments can be parsed, reads the input and plugs it into a list. Then it plugs the list and the timestep into the *iterate* function $\Delta t$ / *timestep* times. Now the list has been updated with the final positions of the planets. Then the *main* function generates a saves the plot of the planets' postions.

The *iterate* function uses the physics functions *velocity* and *newton* to update the positions of the planets after one timestep.

The *newton* function takes one planet and applies the Newton's gravity equation to it and each other planet in the system and adds all the forces up in both dimensions in separate variables.

The *velocity* function takes one planet and uses the *newton* function to determine the total force on the planet, then it uses those two forces to calculate the velocity in the x and y dimensions that planet is experiencing at this timestep.

## Issues

The two main issues with the script are its runtime, which has been discussed before, and its inaccuracy. The inaccuracy is due to the fact that in order for the script to run in any reasonable time you need to have a  large timestep, which means the planets tend not to want to orbit each other because the timesteps are so large they just fly away before they can be pulled back.