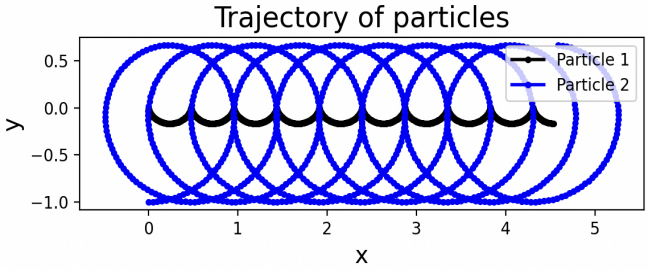
HW2 Report

We want to model how force between two objects (in this case gravitational force) affects their change in position w.r.t. time. Using $\mathbf{F = ma}$, $\mathbf{p=mv}$ and $\mathbf{F} = \mathbf{Gmm}\Delta x/\mathbf{r}^3$, where F,p, and v are vectors, and the kinematic equation $\mathbf{x_f = x_i + v_i + \frac{1}{2}at^2}$, we can solve for the necessary acceleration, final position and final velocity of the system. t in this case is actually Δt, a small increment within our interval which will help us approximate change in behavior of 2 objects, and our experiment works by integrating changes in Δt over a certain interval. Vectors were programmed to have x and y components, an index to label respective objects (1 or 2), and our initial timestep. My main debugging strategy was to mimic the sample code we were given as much as possible to understand syntax/what was going on, as most of my errors were because of an incorrect number or symbol.

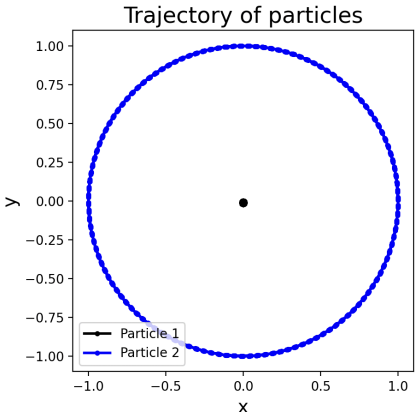
1) real (fp) can be used in orbits.f90 and timestep.f90 because it is declared in the utility file, which is called at the start of orbits under the program block and timestep under the module block. (sort of like calling python files)

2) Orbits.f90 sets up our graph, runs the actual integration and sets/updates our initial condition (starting position, velocity, time, mass, etc). timestep.f90 is used in orbits in each iteration. It runs our calculation for gravitational force, change in acceleration and velocity with set values, changed in iterations of orbits.f90.

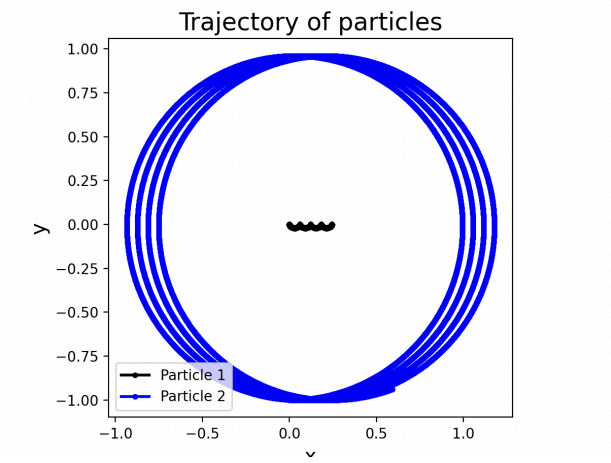
3) To increase the experiment to 3 objects, we would need to add a third object with its velocity, acceleration and momentum values, and 2 more force equations to account for the 2 forces any one of three objects would be experiencing. For N objects, I would probably use 1 object as the source of the force (probably the one with the largest mass) and have every other object share a force with that source. Sort of like the sun in a solar system.
4) To increase the dynamics in space, momentum, velocity, acceleration, and position would need 3 components, and distance would have a z component included as well (used in the gravitational force equation). I think this might be easier to implement than n bodies, as it's still similar to kinematics in 2D but with an extra component. Conceptually it's probably similar to topics in 3D calculus but would be tedious to actually do.
5)



Particle 2 mass = 0.1



Particle 1 initial velocity = (-1,0)



nsteps = 10,000 and tFinal = 25

6) **-wall** generates warnings about common errors in code,
   -wextra gives warnings about subroutine arguments that never get used (I experienced this)
   -g generates debugging info
   -ffree-line-length-none removes character limit on lines
   -fPIC creates shared libraries (I think also makes .ex files run on ram?)
   -fcheck=all runtime checks, undeclared variables, incorrect arguments,
   -fbacktrace creates snapshot of program at time of error to help with debugging
   -c compiles to an object file instead of a new program, useful for when using multiple files (yay)
   -o compiles to object file ending in .f90

7) In essence we modeled two small objects, gave them x and y coordinates and set up values for position, momentum, velocity, and acceleration in 2D using a kinematic equation. We related them through the gravitational force equation and set up an initial condition (1 particle's mass was significantly larger than the other) to model behavior that resembles an orbit in a 2D plane. Changing timesteps can make our integration smoother or more discrete, and changing values like mass or position would change the trajectory of our particle's direction/orbit. We centered things for the sake of clarity.