Programming Assignment

The Kessel Run

Goals: You will learn how to implement explicit Runge-Kutta methods to solve a 2D second-order partial differential problem.

Assignment: Implement Euler's method to move a ship through a series of gravity wells, plotting the longest and shortest paths through the system using ImGui and ImPlot.

Git repository: https://github.com/STIM-Lab/ece3340-kessel
Server: `tuxedo.egr.e.uh.edu`

Implementation Guidelines

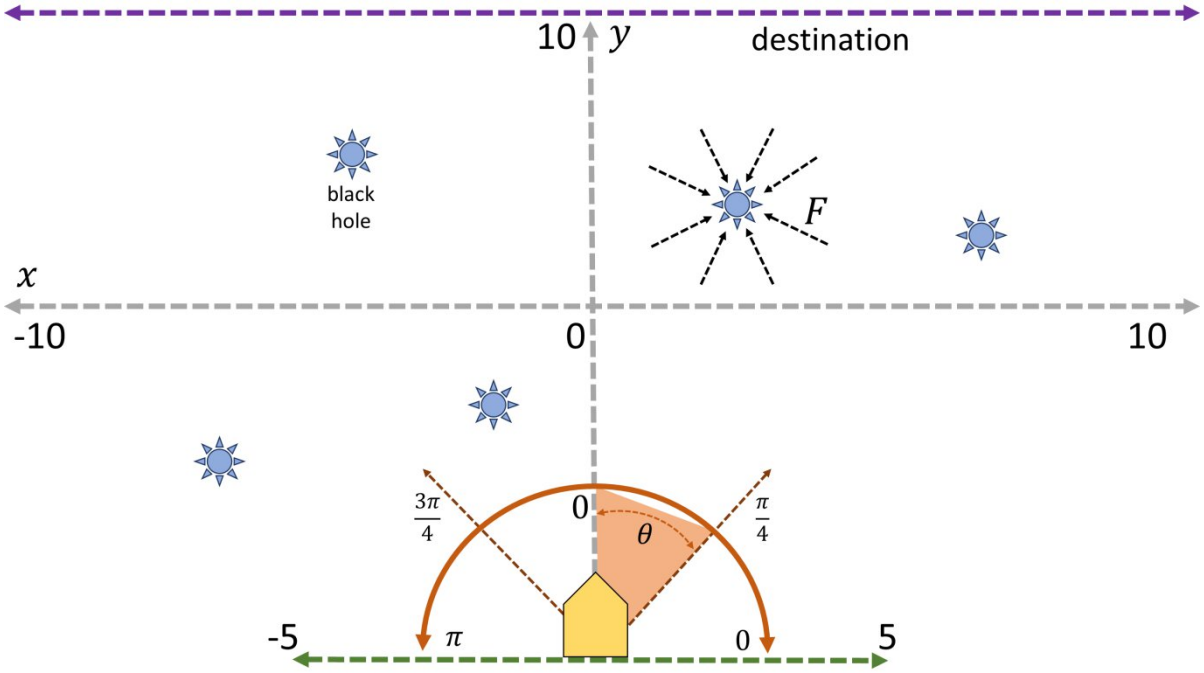
1. Pull the repository and make sure that you have all of the necessary libraries installed via VCPKG. The same libraries are used from PA1. You will also require a new library: **`implot`**

2. Load a gravity file. Several scenes containing different patterns of gravity wells are provided in the **`/scenes`** directory as **`.txt`** files. These files can be loaded using command-line arguments:

   ```
   ./kessel gravity_10.txt
   ```

   This will load the gravity file into the corresponding **`gx`** (x coordinate), **`gy`** (y coordinate), and **`m`** (mass) vectors. These are declared in the **`kessel.h`** file and can be accessed in any of the other source files.

3. Randomly select the starting parameters for your ship. All distance units are in parsecs ($5.38552341 \times 10^{14}$ meters) and angles are in radians. The initial position $p_0 = [p_x\ p_y]$ of your ship is at $p_y = -10$ and $p_x \in [-5, 5]$ selected randomly using a uniform distribution. The initial speed $s_0 \in [2, 5]$ parsecs/sec drawn from a uniform distribution and the initial direction is $\theta_0 \in \left[{}^{\pi}/_{4},\ {}^{3\pi}/_{4}\right]$ drawn from a normal distribution:



4. Use Euler's method to integrate the path of your ship through the field. Terminate under the following conditions:
   - Your ship leaves the playing field along the x-axis: $x < -10$ or $x > 10$
   - Your ship exceeds a maximum acceleration of $\|a(t)\|_2 = 4$ (4 parsecs per second per second)

- Your ship arrives at $y > 10$, making it a successful run

5. If the run was successful, measure the length of the path. If the recent run is shorter than the shortest or longer than the longest, save the path.

6. Execute steps 3 - 5 until you are comfortable finding the shortest and longest paths (this should require around 200 runs). Render the results using **`implot`**.