first task

Problem Formulation

task description

develop a simulation which throws a ball at the objects of different size and shape in the given picture.

- Input: Image of randomly scattered polygons, circles, ellipses each with and without holes.
- Task: Throw a ball and hit targets on the image one after another
- Output: Animation corresponding to the task description

Numerical Methods

1. Shooting Method

Inanumerical analysis, theaa**shooting method**ais a method for solving aaboundary value problemaby reducing it to anainitial value problem. It involves finding solutions to the initial value problem for different initial conditions until one finds the solution that also satisfies the boundary conditions of the boundary value problem.

- Code uses Newton's method for finding correct initial velocity and angle
- Adjusts parameters to hit target's coordinates
- Convergence criterion is *error* < 0.1

in our case,

- starting point is (0,0)
- target point is (x_{target}, y_{target})
- we need to find velocities (v_x, v_y) that will allow us to hit the target

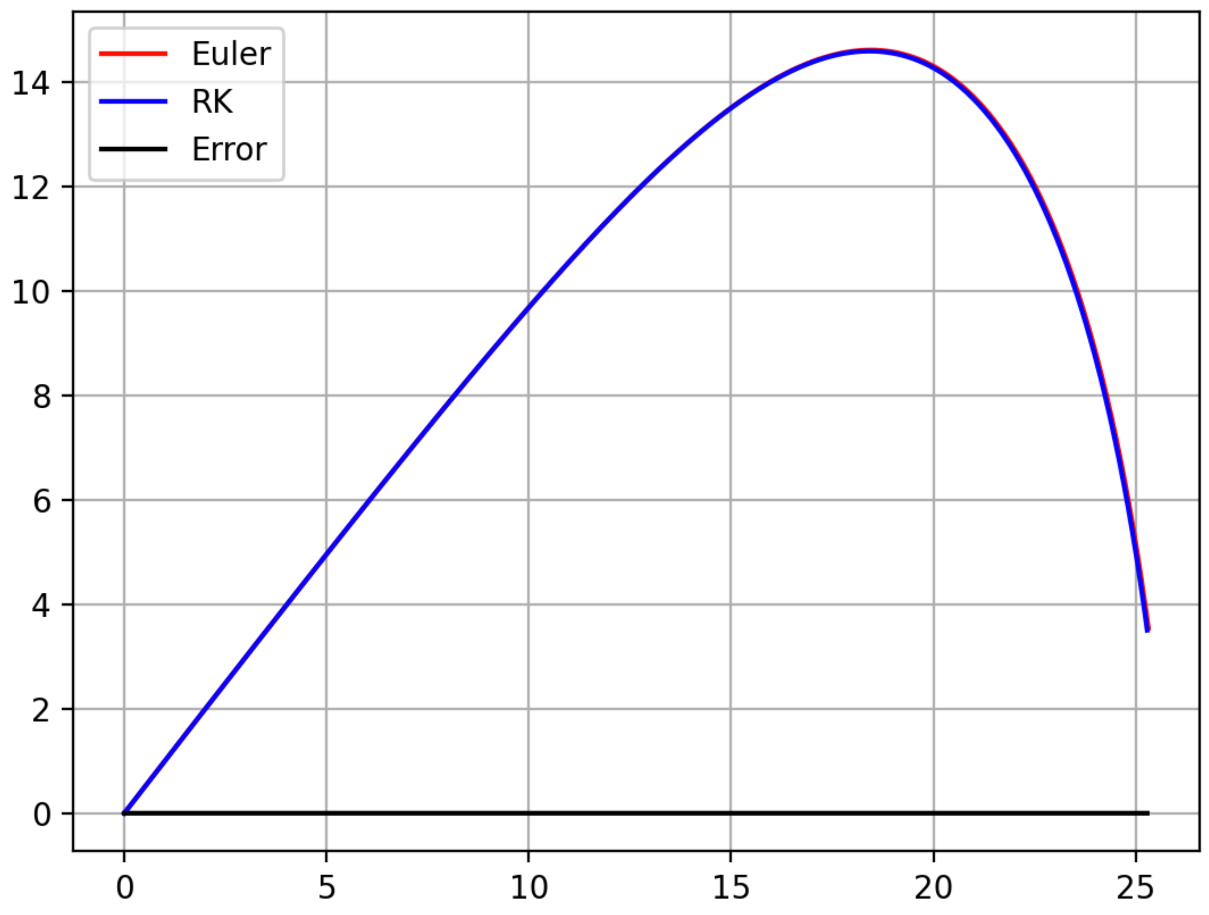
Algorithm

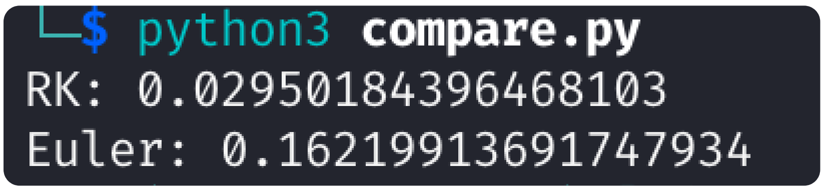
1. make initial guess for launch velocities (v_x, v_y)
   some initial velocities for x and y axis are given. I took (10,10).
2. calculate trajectory using RK4/Euler
   use the chosen numerical method (in my case RK4 and Euler) to calculate the ball's trajectory based on the initial guess of the velocities (solve the equation of motion).
3. compare final position with target's position
   after simulating the trajectory, compare the final and target position. the error is how far we are from the correct guess.
4. adjust initial velocities using Newton's method
   Apply Newton's method to refine the initial velocity estimates in order to minimize the error. This involves approximating the Jacobian matrix, which describes how the final position varies with respect to the initial velocities. The Jacobian is computed numerically by using a small perturbation. The Jacobian matrix is approximated numerically using the backwards differentiation formula.
5. repeat until convergence
   Keep adjusting the initial velocity estimates until the error falls below a specified tolerance. Once the error is sufficiently small, the method is considered to have converged, and the current estimates represent the optimal initial velocities.

- non-convergence:
  If the method fails to converge within the maximum number of allowed iterations, it will report a failure. This could suggest that the initial guess was too inaccurate or that the tolerance level was set too low.

Numerical Experiments

I compared implicit Euler's method and RK. the results are almost the same(dt=0.001), but time it takes to calculate differs.





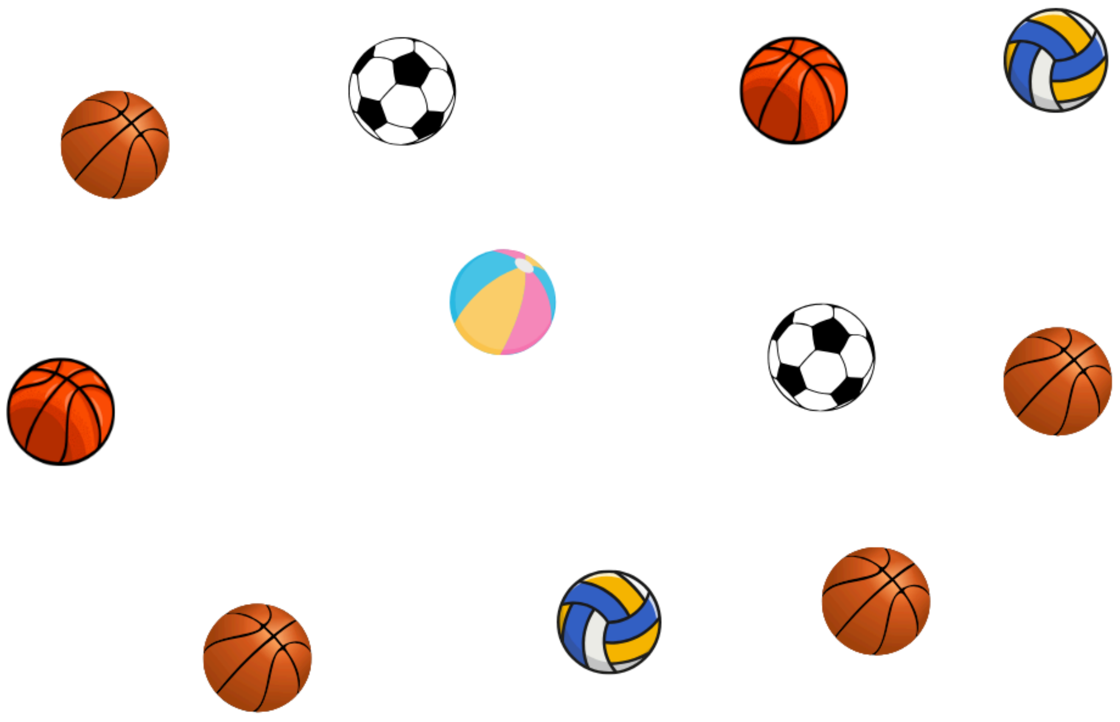
```
└─$ python3 compare.py
RK: 0.029501843964681O3
Euler: 0.16219913691747934
```

parameters

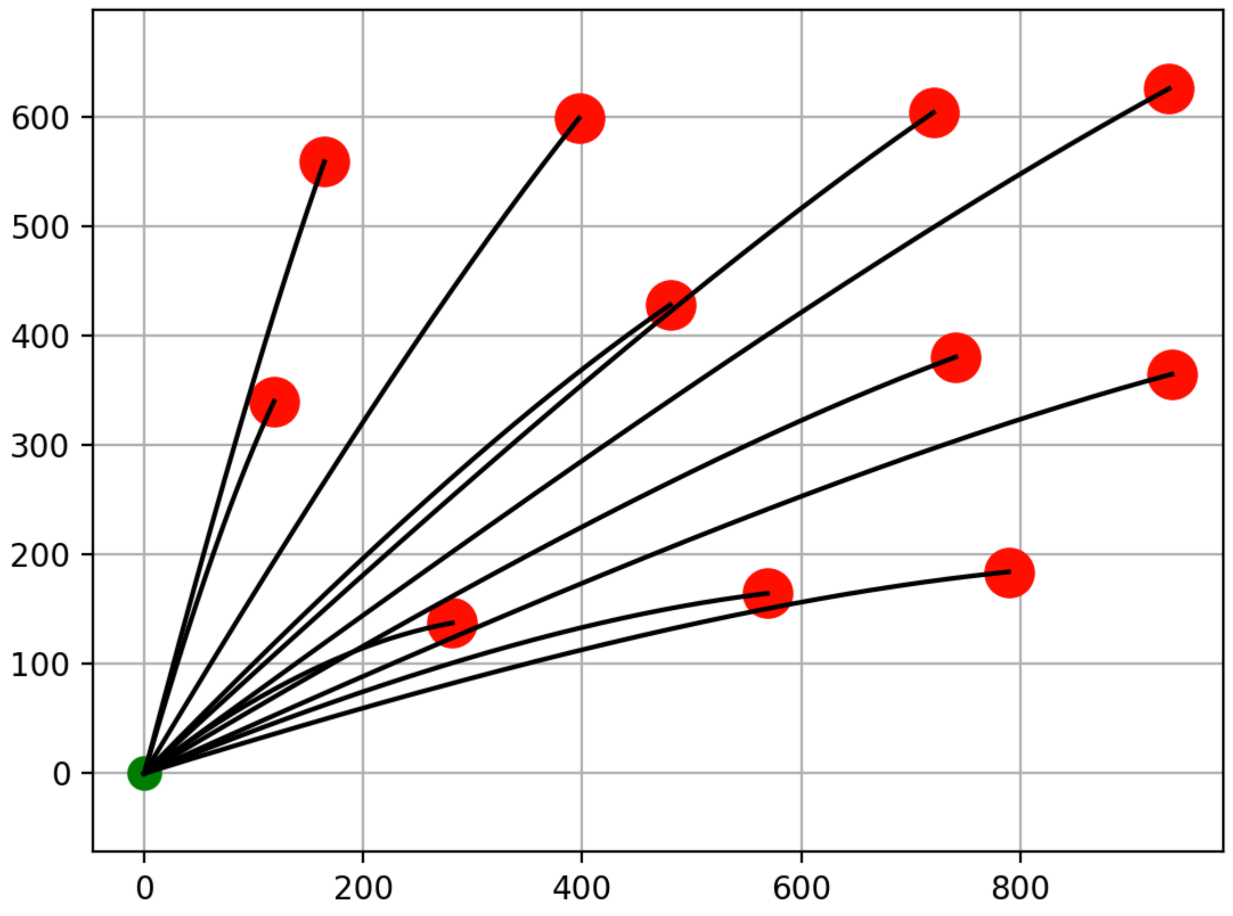
- Air Drag $k = 0.001$
- Air Drag $k = 0.001$
- Time step: $dt = 0.001s$
- Maximum simulation time: $4s$
- Shooting method tolerance: $0.1m$
- Maximum iterations: 100

Test

I took the following image to test the implementation:



the output:



```
36.141153123825966 124.96759846796077
56.50760123135213 214.60219041252662
83.13072125084747 61.60187380024827
146.89159404752323 244.81284748533355
170.0867822140586 174.4032579616144
194.96004905912767 78.69823123446251
301.7310654321086 277.68588475510825
291.0780579774923 173.4126423267227З
305.18124870068493 94.89741913877761
436.236192748079 318.06454249917675
406.8918178320583 183.4543788226947
Finished Calculating Speeds and Angles
```

Sources:

- https://en.wikipedia.org/wiki/Shooting_method
- https://en.wikipedia.org/wiki/RungeЦKutta_methods
- https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/writing-mathematical-expressions
- https://en.wikipedia.org/wiki/Newton%27s_method#Multidimensional_formulations
- https://en.wikipedia.org/wiki/Backward_Euler_method