Trajectory Simulation

Jan Johannsen

November 22, 2023

1 Introduction

Simulation is a powerful tool, which can be used to help understand or evaluate processes working within complex systems. One example for such a process is the drop of a ping pong ball onto a surface. One might want to simulate this process, to find both the speed with which the ball reaches the surface as well as the height of the subsequent bounces. This can be accomplished by using mathematical models to calculate the trajectory of the falling ball based on its previous position and the forces impacting it. Depending on the needed accuracy of the simulation the Problem can be simplified in order to make computation both less complex and thus less time intensive. In the following I will describe a simplified version of the problem and the mathematical models used to simulate it.

2 Description of the Problem

The problem discussed in the following is dropping a ping pong ball onto a flat surface. To initially lower the complexity of the simulation not every possible parameter is used. The following assumptions are made before the simulation. First, the ball is perfectly round and will fall onto a perfectly flat surface. Second, the ball falls in a perfectly straight line towards the ground. Third, the ball is dropped on earth, however the air resistance will not be a factor. This means the simulation will take place in a single dimension, which will be representing the height during the fall and bounces of the ball. This also means that sound won't be part of the simulation.

3 Possible Mathematical Model

When analyzing motion along a straight line a few base concepts need to be introduced. When looking at the motion of objects in space the central metric is their velocity. The velocity of an Object describes its direction in combination with its speed. Velocities are usually represented through vectors. In the simplified version of the problem the ball is dropped in a single dimension thus there are only two directions for the ball to travel in. An objects velocity can then be used to calculate its acceleration. Acceleration is the change in speed and/or direction of an objects velocity. An objects velocity can be calculated with the equation:

$$a = \frac{v - u}{t}$$

With a being the acceleration of the object, u being the initial velocity, v being the final velocity and t being the time between u and v. Since air-resistance is not measured into the calculation in the simplified version of the problem, the change in speed of the ball is coming from its free fall acceleration. The free fall acceleration is independent of an objects properties and lies at $9.8m/s^2$ at sea level on earth. This means when dropped the ping pong ball will accelerate at a constant speed of $9.8m/s^2$. With constant acceleration the current velocity of the ball can be calculated using the following equation:

$$v = v_0 + at$$

With v_0 being the starting position. To calculate the distance traveled from its initial starting point the following equation can be used:

$$x - x_0 = v_0 t + \frac{1}{2} a t^2$$

With x_0 being the starting point and x being the current position at t. Through solving this quadratic equation for t the time the ball takes to reach the ground can be determined. With the equations mentioned above one is now able to find the velocity with which the balls is impacting the ground. When reaching the ground the ball is met with normal force $\vec{F_N}$ from the surface pressing against the ball. The normal force can be calculated through Newtons second law $\vec{F_n et} = m\vec{a}$, which states that the net force on a body is equal to the product of the body's mass and its acceleration. This means that if the ball is perfectly elastic it will return upward with the same velocity it had just before reaching the ground. However due to different materials of the ball as well as the surface not all of the kinetic energy exerted on impact stays as kinetic energy. Thus the upwards velocity on the rebound of the ball is lower than on impact. When calculating the upwards movement one has to remember that the gravitation acceleration now works opposite to the velocity of the ball.

4 Euler-Integration

4.1 Introduction

When looking at real problems one always wants to calculate the exact result. Actually doing so requires an equally exact mathematical model, usually containing various forces with differing effects on the problem at hand. However once this model is finished it gets apparent just how many calculations are necessary to compute results. More often than not this means a model has to exclude or approximate certain factors during calculation to fit the needed efficiency of its use case. The best approach is then to start stripping or approximating less impactful factors first in order to keep the results of the model as exact as possible. Still one problem remains, key to many physical problems, including bouncing ping pong balls, is the differential equation or ordinary differential equation (ODE in short). Working with modern computers trying to solve problems like these analytically is either highly inefficient or downright impossible. This results in the need for a method of approximating ODE's through numerical means. While in many regards out shined by newer methods the Euler Integration introduced this core idea.

4.2 Deriving Euler's Method

Euler-Integration or also known as Euler's method is based on the definition of the derivative

$$f(y(t)) = \frac{dy}{dt} \tag{1}$$

Euler makes use of a simple numerical approximation of the above function

$$\frac{y(t+\Delta t) - y(t)}{\Delta t} \cong f(y(t)) \tag{2}$$

Which says $f(x(t))$ is approximately y at time t plus a chosen interval Δt minus y at time t divided by Δt. This can then be further rewritten to the following

$$y(t+\Delta t) = y(t) + \Delta t f(y(t)) \tag{3}$$

Which gives us the approximate y value at Δt time into the future from the current time t. To verify the above being an approximate version of $F(y(t))$ one can look at y one Δt into the future

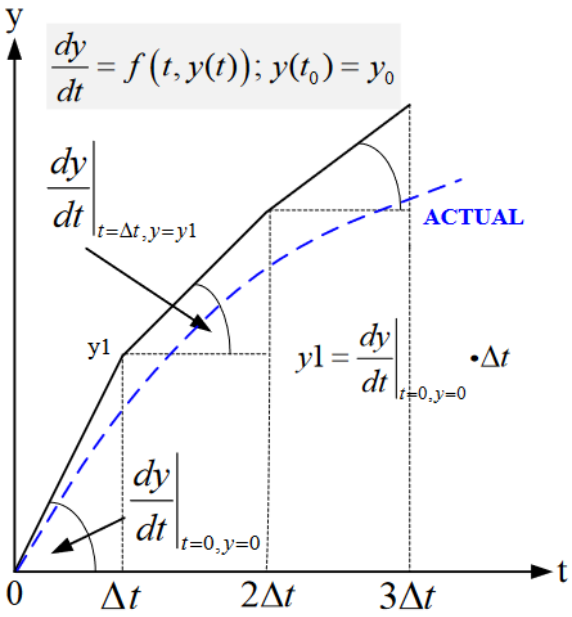
$$y(t_k + 1) = F(y(t_k)) \tag{4}$$

With k being an arbitrary number of Δt time intervals the above can be rewritten to

$$y((k+1)\Delta t) = y(k\Delta t + \Delta t) = y(t_k + \Delta t) \tag{5}$$

4.3 Which inaccuracies result out of the Euler-Integration?

While this approximation gives a solution to the inefficiencies of computing ODE's analytically Euler's method naturally doesn't come without flaws. It is important to mention that the accuracy of this approximation strongly depends on the size of delta t. The closer the value gets to 0, the more accurate the approximation gets. Furthermore since Euler-Integration uses previous results when calculating multiple steps into the future errors from earlier steps increase following ones. This can be seen in the following graphic.



4.4 What happens at y(t) = 0?

When reaching $y(t_k) = 0$ at time t_k during the process of integrating with Δt time steps the next results at time $t_k + 1$ is calculated only through the derivative $f(y(t_k))$ times Δt. If the derivative is still unequal to zero the next value $y(t_k + 1)$ will also be unequal to zero. If however the derivative is zero every subsequent value following $y(t)$ will be zero as well.

5 Implementation

5.1 Software Structure

After finding a physical model to describe the given problem and introducing ways of approximation to simplify the problem for numerical calculation one can now implement a simulation. This simulation will calculate the position of the ping pong ball at every time step Δt for a given time frame or maximum number of bounces. To further illustrate the effect of using Euler-Integration the simulation will also include calculations with the analytical approach for comparison purposes. However since these aren't the main focus of the simulation the analytical approach will only be calculated until reaching the ground for the first time. Finally the results of the simulation will be plotted onto a graph for visualisation purposes. To implement the above, the program can be split into the following steps

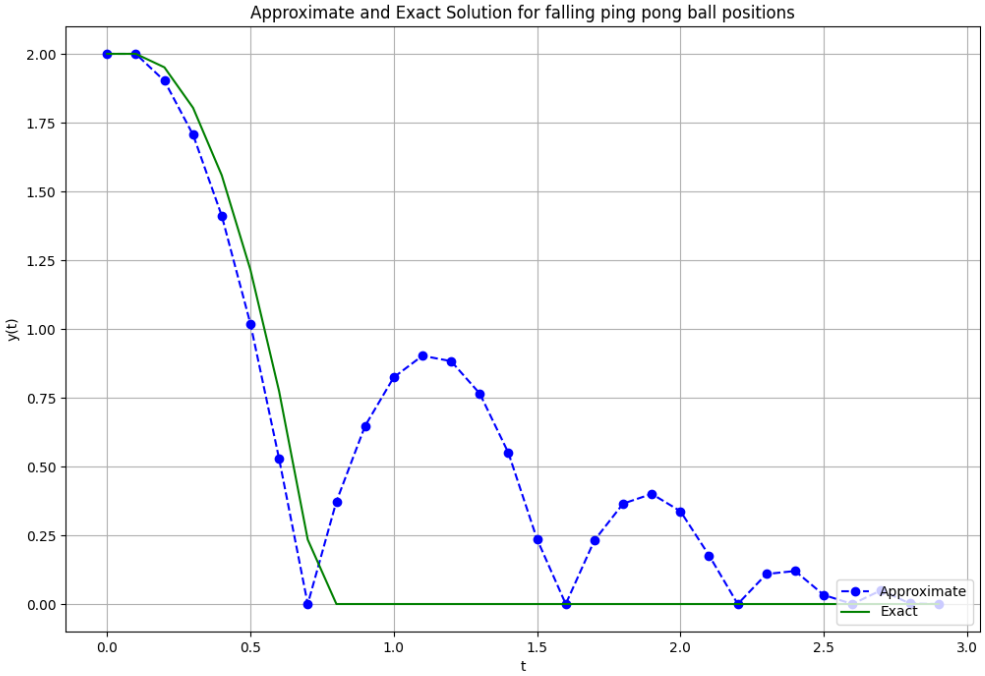
1. Initializing the variables
2. Defining the analytical and velocity function
3. Initialize the data structure to save the results
4. Calculate y values with Δt steps through Euler-Integration until one end condition is met
5. Calculate the exact values until reaching 0
6. Plot the results

5.2 Experiment Results

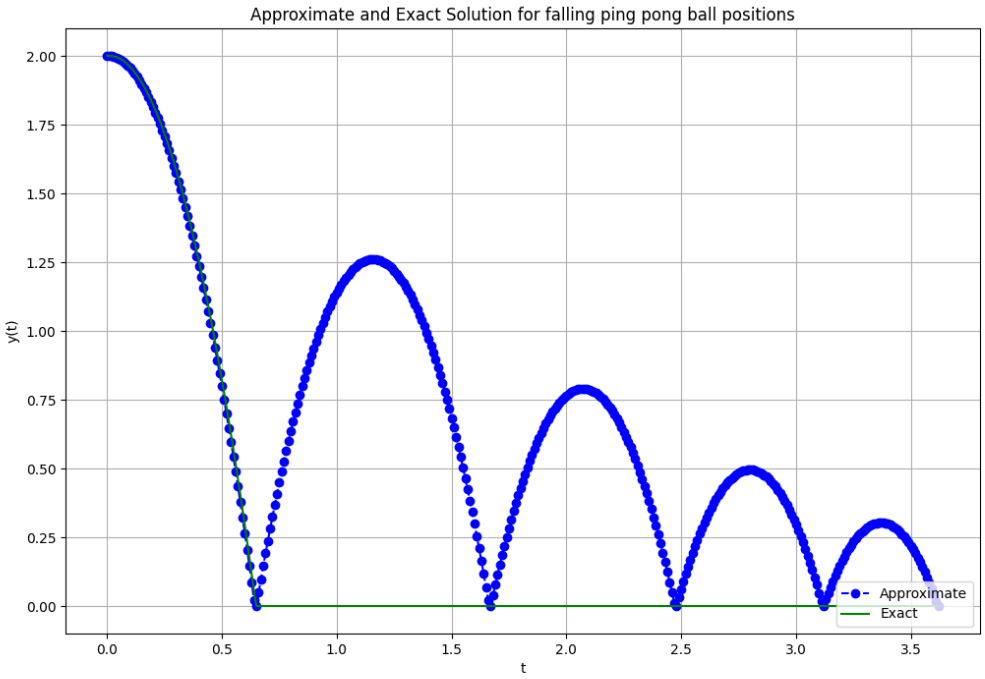
The following Experiments are based on a simple python script utilizing matplotlib to visualize results. Both experiments had the following parameters initialized with the same values:

1. start height $y_0 = 2\text{m}$
2. start velocity $v_0 = 0m/s$
3. gravitational pull $g = 9.81m/s$
4. energy preservation constant $e = 0.8$
5. ball radius $r = 0mm$
6. table height $y_Table = 0m$
7. max time $endTime = 10s$
8. max bounces $maxBounces = 5$

Experiment number 1 had Δt set to 0.1s, resulting in the following plot



Experiment number 2 had Δt set to 0.01s, resulting in the following plot



5.3 Interpretation

When comparing the experiment results one will find that the larger time steps from experiment one produced significantly worse results than experiment two when compared to the exact calculations

during the initial free fall. The error resulting from the usage of Euler-Integration in this example comes from the approximation not factoring in the change in velocity during the time step. The exact error for one step can be calculated with the following equation

$$Err = -g * \Delta t^2 * \frac{1}{2} \tag{6}$$

If Δt decreases the effect on the error is exponential. This means, while the steps were increased by a factor of 10, the error decreased by a factor of 100.

6 Adding Air Resistance

6.1 Program Structure Refactor

Before adding drag to the simulation the structure of the program will be reworked into three functions. This is done to split functionality into reusable and problem specific parts. The first function eulerIntegration(function, timeSincePrev, previousValue, timeSinceStart) is generic and will a numerical function for the derivative of a problem. Given the numerical function its input timeSinceStart the previous value and a step Distance euler Integration is then used to approximate the following value. The next functions are problem specific and will contain logic for the simulation of the ball. The main function eulerSimulateFallTrajectoryODE() has a simialr structure to the previously described program, with the addition of a submethod approximateTimeOfImpact(target, function, previousValue, stepDistance, currentTime, maxDepth), which is used to better determine the exact height of the ball after bouncing instead of setting any value below zero as zero. This is done through splitting the stepsize in half a set number of times and calculating Euler Integration for steps in the middle of both halves. Depending on which one is closer to zero the process is repeated until either the max number of repeats is reached or the exact time of reaching zero is found.

6.2 Air Resistance

Improving the simulation to account for air resistance can be achieved through changing the input function when calling eulerIntegration(). With the current program structure the problem specific function is defined inside eulerSimulateFallTrajectoryODE() with the initial velocity as a baseline. This definition is replaced on every bounce taking the velocity after the bounce as the new baseline for the velocity calculations. To account for drag the previous function $v(t) = v_0 + -g * t$ now has to extended by $-c * (v_0 + -g * t)$. The drag coefficient c is always inverted so the result is opposite of the velocity. This is done since the drag is affecting the ball from two different angles depending on whether the ball is ascending or descending.

6.3 Experiments and Results

To Test the results of the added changes the same baseline for experiments as the previous ones were chosen, with the addition of the drag coefficient c being set at 0,17. The previous experiments were conducted with values being initialized as follows:

1. start height $y_0 = 2\text{m}$
2. start velocity $v_0 = 0m/s$
3. gravitational pull $g = 9.81m/s$
4. energy preservation constant $e = 0.8$
5. ball radius $r = 0mm$
6. table height $y_Table = 0m$
7. max time $endTime = 10s$
8. max bounces $maxBounces = 5$

To best compare the updated version of the program with the previous one the two experiments will again be done with Δt being 0.1s and 0.01s respectively. When comparing both experiment 1 and experiment 2 to the ones conducted prior one can see that the height of subsequent bounces shrank significantly. Furthermore the total time until reaching the max number of observed bounces also decreased due to the ball being in the air for shorter periods of time between bounces.

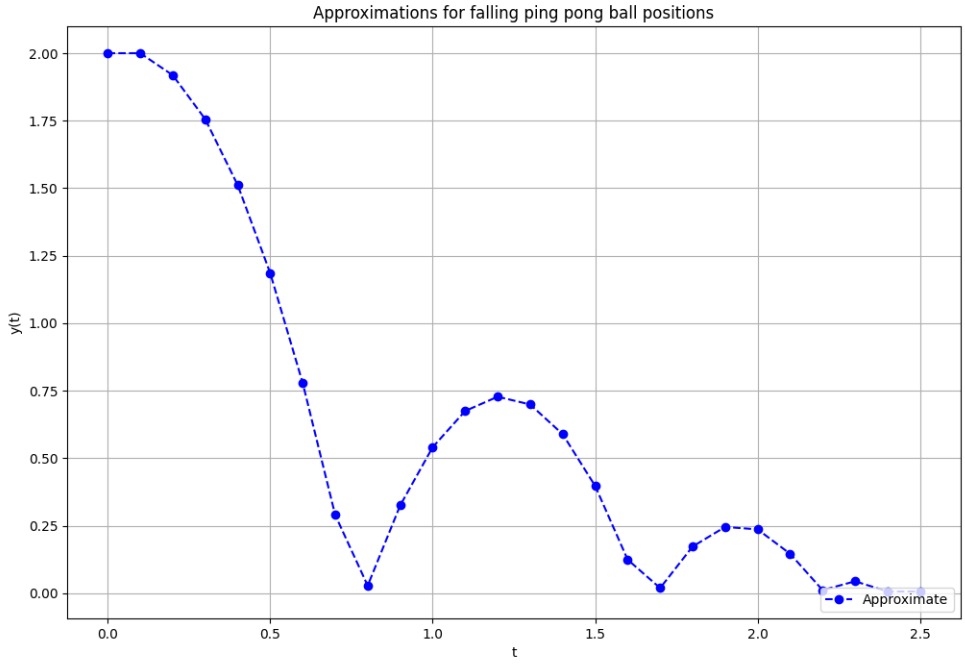


Figure 1: Approximations with drag deltaT 0.1

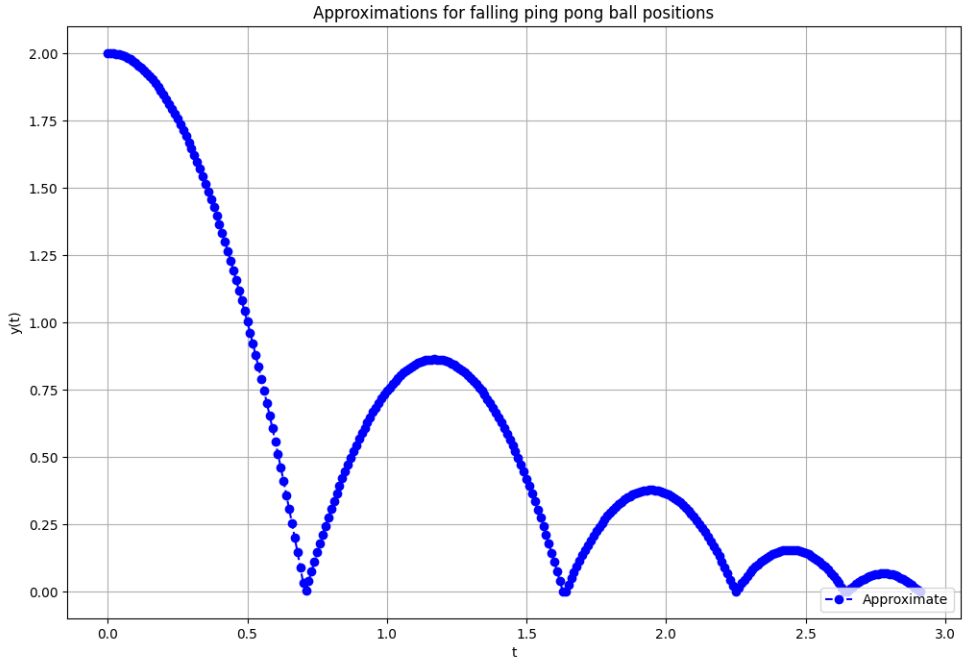


Figure 2: Approximations with drag deltaT 0.01

References

[1] Halliday, David and Resnick, Robert and Walker, Jearl (2013) *Fundamentals of physics*, John Wiley & Sons

[2] Fathoni, M.F. and Wuryandari, A.I., 2015, December. Comparison between Euler, Heun, Runge-Kutta and Adams-Bashforth-Moulton integration methods in the particle dynamic simulation. In 2015 4th International Conference on Interactive Digital Media (ICIDM) (pp. 1-7). IEEE.

[3] Deriving Forward Euler and Backward/Implicit Euler Integration Schemes for Differential Equations, 2023, Steve Brunton

[4] Qingkai Kong, Timmy Siauw, Alexandre Bayen, Python Programming and Numerical Methods: A Guide for Engineers and Scientists, Academic Press, 2020