Assignment 1

Ayden S. McCann

March 2021

1 Introduction / Discussion

The physical problem explored in this assignment is the simple case of a ball dropped from a height, falling under gravity and bouncing on a surface defined at x = 0. This is represented physically by the following equations:

$$\frac{dv}{dt} = \frac{F}{m} = -g \tag{1}$$

$$\frac{dx}{dt} = v \tag{2}$$

In order to convert this analytic equation into a form that can be approached numerically it must be discretised in the following way:

$$\frac{dv}{dt} = \frac{v(t + \Delta t) - v(t)}{\Delta t} = -g \tag{3}$$

$$\frac{dx}{dt} = \frac{x(t + \Delta t) - v(t)}{\Delta t} = v \tag{4}$$

This is then incorporated into the following algorithm:

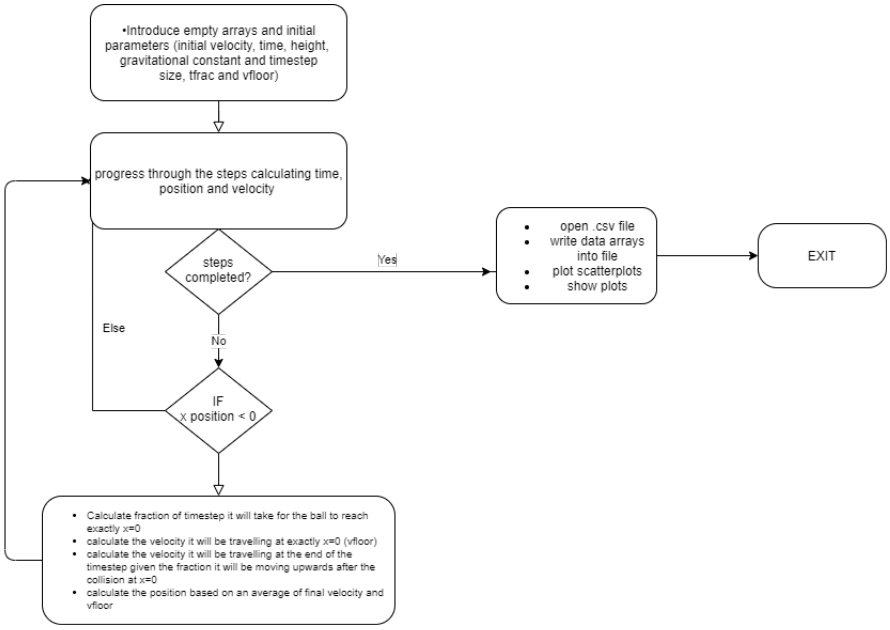

Figure 1: The flow chart illustrating how my final algorithm works.

This is the final version i arrived to after a number of trial-and-error attempts to rectify a severe flaw in my initial code.

Initially, i had decided to represent the floor by searching for x positions less than zero, setting the x position to zero and reversing the velocity. This however does not work well, especially for large timesteps (discussed later).

2 Exercise 1

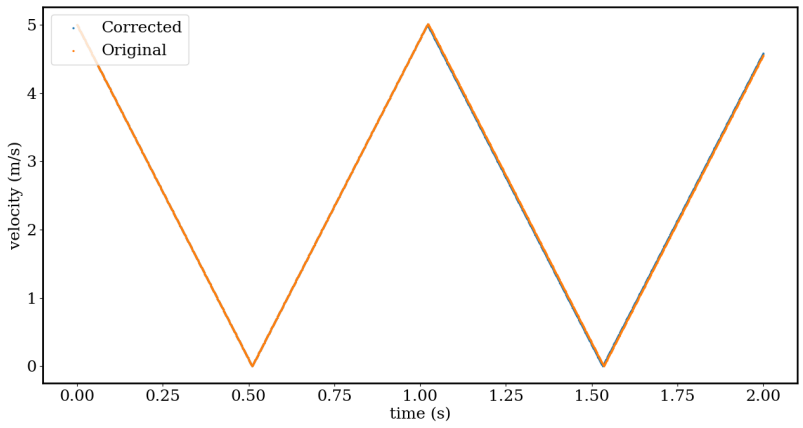

Figure 2: A plot of velocity over time for the original and corrected systems.

As mentioned above, the initial code i had written to simulate the bouncing ball was inaccurate for the following reasons:

The x position was updating via multiplying the current velocity by the time since the last timestep. When in reality, the distance travelled during this time would be the **average** of the current velocity and the previous timestep's velocity multiplied by the time.

Secondly and most critically the reflections from the floor.

The code initially 'bounced' the ball from the surface by finding timesteps where $x < 0$, setting $x = 0$ and reversing the velocity to send the ball back upwards. However since the ball has passed below zero it has travelled more than the initial drop-height. As a result, it has gained additional velocity and will (in this idealised system) pass above the initial drop height, violating conservation of energy.

My first solution to correcting for this incorporated a simple equation to calculate the distance below zero the ball had travelled, and deduct off this additional velocity:

$$v^2 = u^2 + 2gs \tag{5}$$

Where v is final velocity, u is initial velocity, g is the gravitational constant and s is the distance travelled.

After making this correction and observing better, but still non-ideal results, i deduced that the following was to blame:

Since there is a nonzero time for the ball to reach zero from the last positive x timestep, once the ball reaches zero and begins to accumulate negative distances it has only been a fraction of a timestep.

Merely deducting off this value and treating it as an entire timestep is another approximation that was leading to inaccuracy.

The solution (as detailed in Figure 1) was to approximate the fraction of the timestep the ball was above zero. This is shown in equation 6 is where x(i) is the x position of the final timestep above x=0 and x(i+1) is the x position below 0. Then calculate the final velocity of the ball as it 'impacts' the surface at x=0 v_{floor}, where v(i) is the velocity at the last positive timestep and x(i) is the height above zero (7).

$$t_{frac}(i) = \frac{x(i)}{x(i) + |x(i+1)|} \tag{6}$$

$$v_{floor}(i) = \sqrt{2 * g * x(i) + v(i)^2} \tag{7}$$

Then, approach the remainder of the timestep by reflecting this, more accurate velocity, proceed to deduct off gravitational acceleration for the remainder of the timestep (v_{final}) (8) and then calculate the distance travelled from the floor by averaging the final speed and v_{floor}, and multiplying it by the remainder of the timestep where it was travelling upwards (x_{corr}) (9)

$$v_{final}(i) = v_{floor}(i) - g * dt * (1 - t_{frac}(i)) \tag{8}$$

$$x_{corr}(i) = \frac{v_{floor} + v_{final}}{2} * dt * (1 - t_{frac}(i)) \tag{9}$$

These changes substantially increase the accuracy of the results, especially for higher timestep values. In Figure 2, the algorithm was iterated with 200 steps of 0.1 second. As can be clearly seen the original code progressively gains additional velocity from falling below zero, This subsequently causes the ball to gain additional nonphysical velocity. This error can be characterised in the following way:

The worst possible scenario for attaining error in this scenario is where the ball is below zero for a large fraction of a timestep. In the limit of maximum error this approaches dt. Therefore the positional error maximum error can be regarded as for any given bounce:

$$\Delta x = v_{floor}\, dt + \frac{g\, dt^2}{2} \tag{10}$$

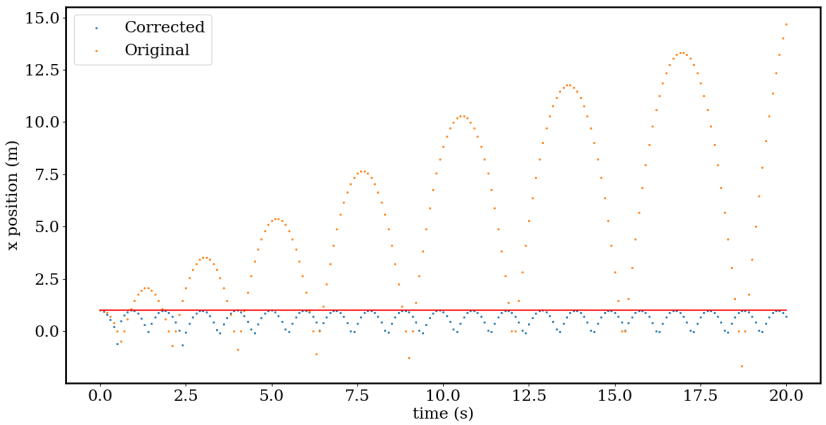

Figure 3: An example of how large timesteps can lead to inaccuracy in the original equation.

3 Exercise 2

As described above, increasing dt increases the possible error, as it increases the possible time the ball could be falling below zero and gaining nonphysical velocity.

By changing the timestep used in Figure 2 of 0.1 to 0.01 (Figure 3), the potential positional error (10) decreases from ~ 0.497 m to ~ 0.045 m on the first bounce from 1m. As a result, after 20 seconds of iteration the maximum height reached by the uncorrected system decreases from $\sim$ 15m to $\sim$ 2.5m.

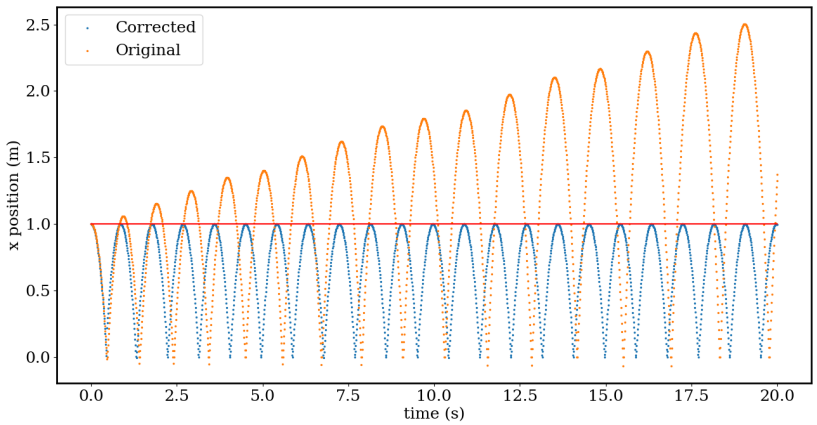

Figure 4: The corrected and uncorrected systems iterated for $dt = 0.1$.

4 Exercise 3

By changing the initial velocity to 50m/s and the initial height to 0m, some other interesting differences can be observed between the two simulations.

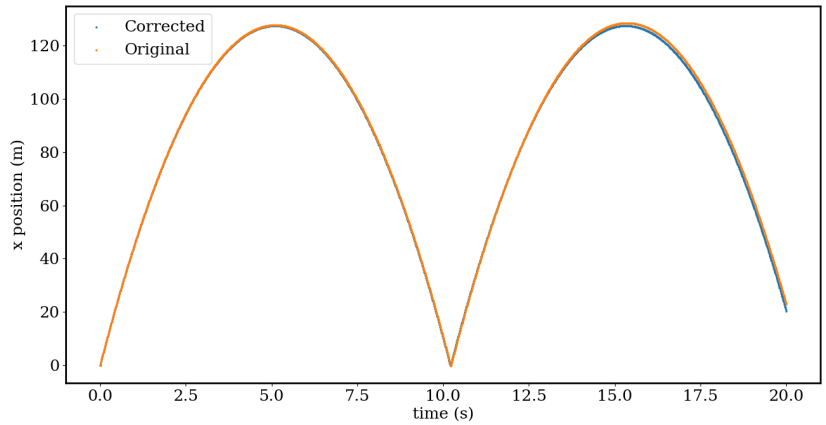

Figure 5: The two systems iterated for an initial velocity of 50m s^{-1} and an initial height of 0m .

On closer inspection there is a difference between the two systems even before the bounce.

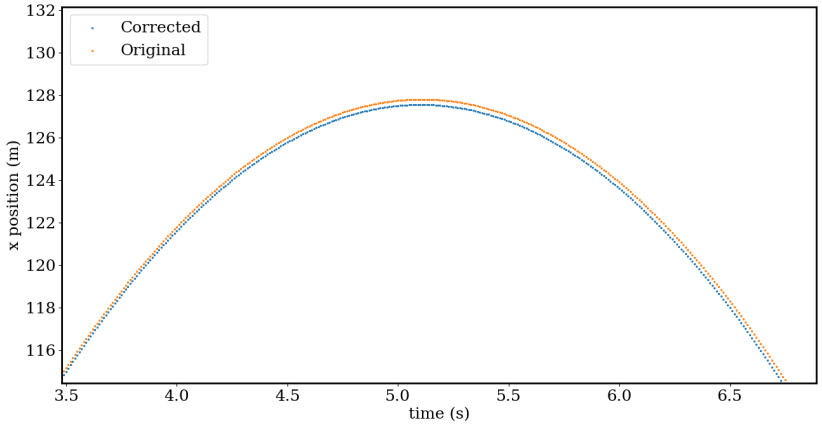

Figure 6: Figure 4 zoomed to illustrate the differences in the two systems. Before the first bounce.

Given that the only difference between the two systems is that the original took the velocity at the end of the previous timestep to calculate distance, while the corrected code takes the average velocity over the previous timestep.

By printing the results to .csv and comparing with analytical results the velocities of the first 200 steps (a time range prior to the first bounce) were compared in the following way:

$$\Delta x_{total} = \sum_{i=0}^{199} x(0) * t(i) - \frac{g * t(i)^2}{2} - x(i) \tag{11}$$

Where $x(i)$ is the positional value of the given system calculated at the ith timestep. for each equation and $t(i)$ is the accumulated time at the ith timestep.

The results were the following and demonstrate the superior accuracy of the corrected system.

System	Total Accumulated Positional Error
Original	19.31 m
Corrected	$4.15 * 10^{-12}$ m

5 Exercise 4

In order to consider inelastic collisions with the table, a modification was made to the code wherein at each reflection triggered by the ball reaching

$x \leq 0$ a multiplier of 0.9 is applied to the velocity, simulating a loss of energy akin to a real world system.

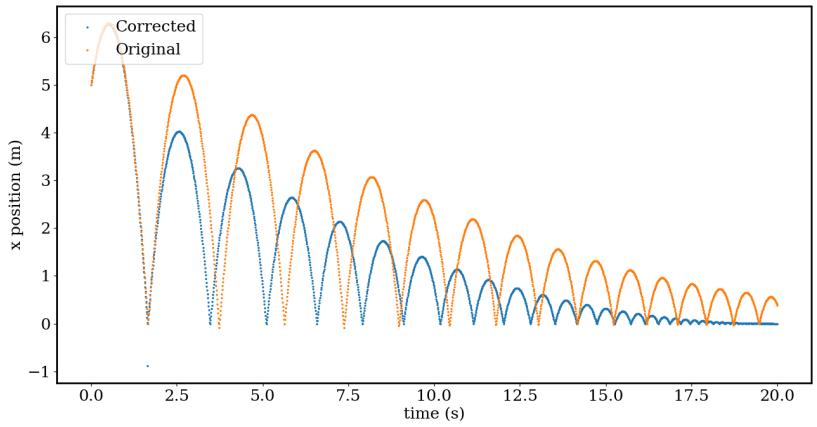

Figure 7: The two systems with $dt = 0.01$, initial heights of 5m, initial velocities of 5ms^{-1} and a 10% velocity reduction at each bounce.

Given the error that the original system is prone to, with the input parameters described in Figure 6, the ball never decays to a velocity of 0. Rather, it tends towards a stable trajectory where its sub-zero positional error acts to give it exactly the 10% velocity it loses per bounce. This is better illustrated in Figure 7.

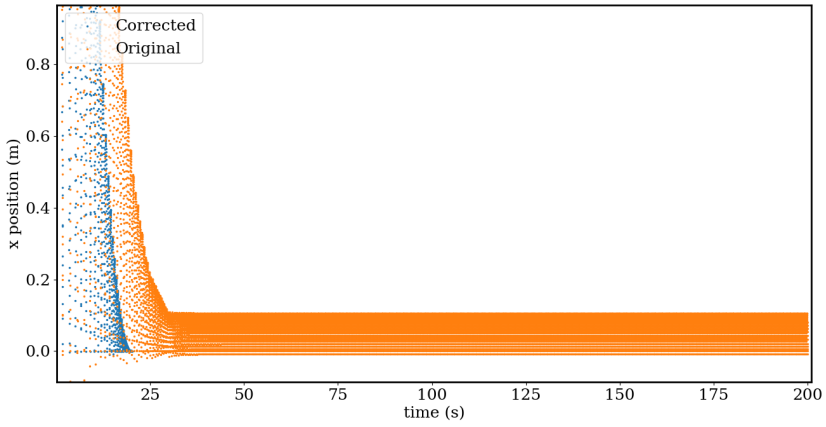

Figure 8: Figure 6 iterated for 200 seconds to demonstrate the behaviour of the original system.

It is clear however that the corrected system is behaving exactly as expected and its results are accurate.

6 Code

```
import numpy as np  # for fast numerical computing with python
import matplotlib.pyplot as plt  # visualisation of the results
import matplotlib as mpl

steps = 2000 # time steps
tval = np.empty(steps + 1)  # time values
xval = np.empty(steps + 1)  # position (original) value at each time
vval = np.empty(steps + 1)  # velocity (original) value at each time
xvalcor = np.empty(steps + 1)  # position (corrected) value at each time
vvalcor = np.empty(steps + 1)  # velocity (corrected) value at each time

tfrac = np.empty(steps +1)
vfloor = np.empty(steps +1)

xval[0] = xvalcor[0] = 0.0  # initial height of the ball
vval[0] = vvalcor[0] = 5.0  # initial velocity of the ball

tval[0] = 0.0  # initial time (t=0 initially)
g = 9.8  # gravitational acceleration
dt = 0.01  # size of the time step

for i in range(steps):  # loop for 300 timesteps
    tval[i + 1] = tval[i] + dt
    xval[i + 1] = xval[i] + vval[i] * dt
    vval[i + 1] = vval[i] - g * dt
    vvalcor[i + 1] = vvalcor[i] - g * dt
    xvalcor[i + 1] = xvalcor[i] + ((vvalcor[i + 1] + vvalcor[i]) * dt) * 0.5
    #changed to use average velocity between current and last timestep

    if xval[i] < 0:  # Reflect the motion of the ball when it strikes the surface
        vval[i + 1] = -vval[i]*0.9
        xval[i + 1] = 0

    if xvalcor[i+1] < 0:  # corrected reflection off surface
        tfrac[i+1] = xval[i]/(xval[i]+xvalcor[i+1])
        # calculate which fraction of the timestep it will take to hit
        # x=0
        vfloor[i] = np.sqrt(xvalcor[i]*2*g+(vvalcor[i])**2)
```

```
        # calculate the velocity it would be travelling at
        # x=0
        vvalcor[i+1] = 0.9*(vfloor[i] - g*dt*(1-tfrac[i+1]))
        # calculate the velocity at the end of the timestep
        xvalcor[i+1] = ((vvalcor[i+1]+vfloor[i])/2)*(1-tfrac[i+1])*dt
        # calculate the position at the end of the
        # timestep
    else:
        tfrac[i] = 0
        vfloor[i] = 0

# save the results to file
f = open("bouncing_ball_results.csv", "w")  # create text file
f.write("xval, vval, vvalcor, xvalcor, tval\n")  # write column names
for i in range(steps + 1):  # write the results to file as text
    f.write("{0}, {1}, {2}, {3}, {4} \n".format(xval[i],
    vval[i], vvalcor[i], xvalcor[i], tval[i]))

f.close()  # close the file

x_datacor = xvalcor
x_data = xval
t_data = tval

mpl.rcParams['font.family'] = 'Serif'
plt.rcParams['font.size'] = 18
plt.rcParams['axes.linewidth'] = 2
plt.scatter(t_data, x_datacor, s =2, label = "Corrected")
# plot the results in a scatter plot
plt.scatter(t_data, x_data, s =2, label = "Original")
#plt.plot((0,steps*dt), (1,1), linestyle = '-', color='red')
plt.xlabel('time (s)')  # axis labels
plt.ylabel('x position (m)')

plt.legend(loc="upper left")



plt.show()  # show plot
```