PB1010 - Fysikk 1
Oblig 1

Sølve Kjelseth

February 16, 2025

0. Introduction

This is for Oblig 1 in Physics where the mechanics of a free falling ball is dissected.
Note: The LaTeX and python code is open source, github link.

0.0. Table of Contents

0.1. List of Figures

1. Task 1

I do not understand why the assignment wants to have ground level at $h = 0m$ and positive direction downwards at the same time(as stated in part a), that means the sphere has a starting position at $h = -50m$. The math works either way and even if it seems counterintuitive I choose to adhere to the requirements of the task and assumes this direction going forward unless otherwise stated.

1.0. Part a

To solve part a I choose to use the ISEE method.

1.0.0. Identify

I find some important variables, such as ground level, $h_{ground} = 0m$ and distance between the ground and the sphere $x_{sphere-ground} = 50m$. I also see that positive direction is downwards, that means $h > 0$ will be meters under ground. That makes starting position $h_0 = -50m$. Air resistance is omitted, this way only gravity affects acceleration. This is constant acceleration, downwards, thereby positive $a = g = 9.825ms^{-2}$ (This is the official number for g in Oslo). Therefore the equations for motion in constant acceleration can be used. I also note that the start time of the free fall is at $t_0 = 0s$ and it has no initial velocity $v_0 = 0ms^{-1}$. Lastly I see that everything in this part is single dimension motion.

1.0.1. Set up

For the first question, find $h(t)$, we know the goal variable is position, we "know" the time variable as all possible values will be put in, and finally the last known is acceleration. this means the velocity less equation can be used.

$$x = x_0 + v_0 t + \frac{1}{2}at^2 \tag{1}$$

For the next question, find $v(t)$, we know the goal variable is velocity, and here time is also "known" as it is the function input and since we know acceleration as well the position less equation can be used.

$$v = v_0 + at \tag{2}$$

For the final question, find t_{ground}, we can rearrange Equation 1 to find the time when we calculate further. A sketch of the situation is provided on the next page.

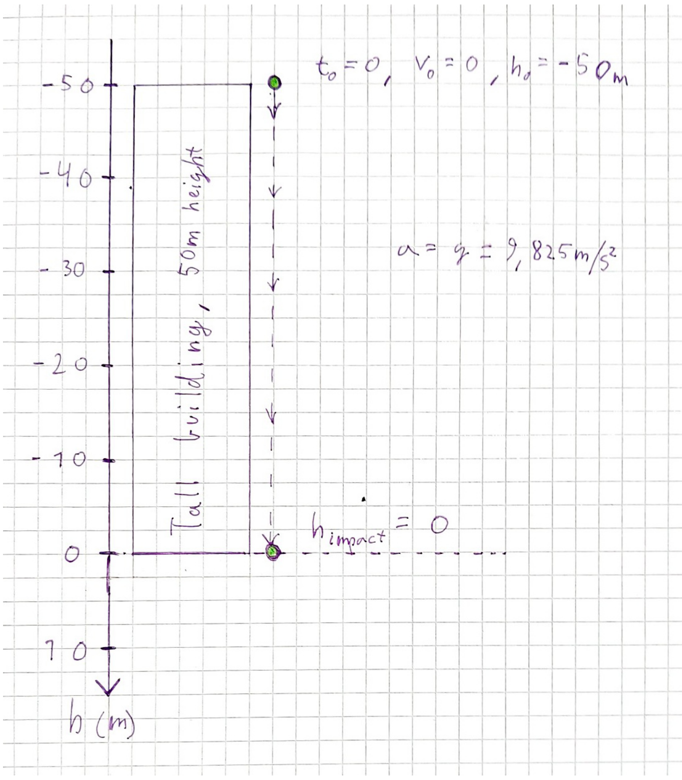

Figure 1.0: Sketch of sphere falling from building

1.0.2. Execute

Filling the used variables in Equation 1, and making it into a function:

$$h(t) = h_0 + v_0 t + \frac{1}{2} g t^2$$

Filling in values, here $v_0 = 0ms^{-1}$ simplifies the equation by removing a term.

$$h(t) = \frac{1}{2} 9.825ms^{-2} t^2 - 50m \tag{3}$$

Moving on to the next, filling in the used variables in Equation 2 and making it a function.

$$v(t) = v_0 + gt$$

Filling in values, $v_0 = 0ms^{-1}$ simplifies this equation too.

$$v(t) = 9.825ms^{-2} t \tag{4}$$

The time of impact will be at $h(t) = 0m$, filling this in Equation 3 and solving for t.

$$0m = \frac{1}{2}9.825ms^{-2}t^2 - 50m$$

$$50m = \frac{1}{2}9.825ms^{-2}t^2$$

$$100m = 9.825ms^{-2}t^2$$

$$\frac{100m}{9.825ms^{-2}} = t^2$$

$$t = \pm\sqrt{\frac{100m}{9.825ms^{-2}}} \approx 3.19s \tag{5}$$

Now as time starts at 0, the negative solution for t does not make sense in this task and is therefore omitted from the results.

1.0.3. Evaluate

I feel like the magnitude of my answers is in the correct order and $3.19s$ seems quite reasonable time for an object to fall 50 meters, it is maybe a bit fast, but this is also with no air resistance. I will also want to check units, starting with Equation 3.

$$m = m \cdot s^{-2} \cdot s^2 - m$$

$$m = m - m$$

$$m = m$$

This checks out correct, onto Equation 4.

$$m \cdot s^{-1} = m \cdot s^{-2} \cdot s$$

$$m \cdot s^{-1} = m \cdot s^{-1}$$

And the final one, Equation 5.

$$s = \sqrt{\frac{m}{m \cdot s^{-2}}}$$

$$s = \sqrt{\frac{1}{s^{-2}}}$$

$$s = \sqrt{s^2}$$

$$s = s$$

This also ended with correct units, This makes me conclude that the answers are correct with a very high probability.

1.0.4. Solution

This is the function for $h(t)$:

$$h(t) = \frac{1}{2} 9.825ms^{-2}t^2 - 50m \tag{3}$$

This is the function for $v(t)$:

$$v(t) = 9.825ms^{-2}t \tag{4}$$

The sphere hits the ground approximately $3.19s$ after letting it go.

$$t = \pm\sqrt{\frac{100m}{9.825ms^{-2}}} \approx 3.19s \tag{5}$$

1.1. Part b

To solve part b I choose to use the ISEE method and to build upon the finds in part a.

1.1.0. Identify

I find the same initial variables as in part a, $t_0 = 0s$, $h_0 = -50m$, $v_{0h} = 0ms^{-1}$, this time I have specified the direction of v and a as we are now dealing with two dimensional motion. I choose to keep the vectors decomposed and calculate separately for each axis. The new information I have is that $v_{0x} = 7.2ms^{-1}$ and I choose to set the initial position for the sphere at $x_0 = 0m$ and positive direction along the horizontal movement. As there is no air resistance we can see that only gravity affects the acceleration, $a_h = g = 9.825ms^{-2}$ and $a_x = 0ms^{-2}$. This is constant and therefore I can still use equations for motion with constant acceleration.

1.1.1. Set up

The first question, find the time until impact, is more of a trick question, as the horizontal movement have no impact on the vertical movement, this means the time will be the same as calculated in part a. The second question, find the horizontal movement at time of impact, the goal variable is known, x and we know time of impact from part a, we also know that since there is no acceleration the speed is always the same, I can therefore use any of the equations and I decided to use the velocity less as it involves initial velocity as it is known and it simplifies nicely when acceleration is zero.

$$x = x_0 + v_0 t + \frac{1}{2}at^2 \tag{1}$$

A sketch of the situation is provided on the nexrt page.

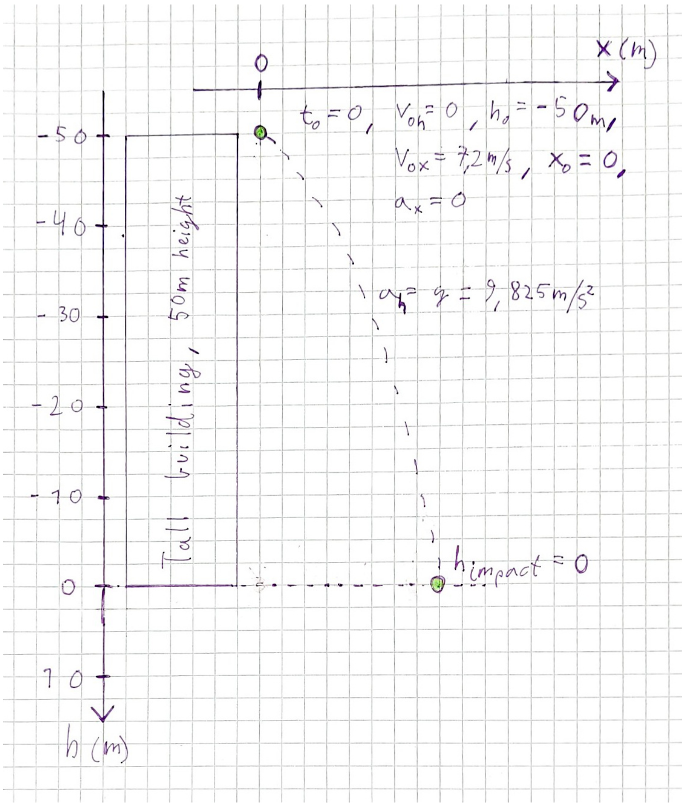

Figure 1.1: Sketch of sphere falling with initial horizontal velocity

1.1.2. Execute

I start by simplifying this equation as both $x_0 = 0$ and $a_x = 0$ this removes two terms. Also filling in the used variables.

$$x_{impact} = v_0 t_{impact}$$

Now with values is filled in.

$$x_{impact} \approx 7.2ms^{-1} \cdot 3.19s \approx 22.97m \tag{6}$$

1.1.3. Evaluate

I feel like the magnitude of my answer is in the correct order and $22.97m$ seems quite reasonable for an object to move that distance when the velocity is $v_0 = 7.2ms^{-1}$, and travels for $t_{impact} = 3.19s$. I will also want to check units to Equation 6.

$$m = m \cdot s^{-1} \cdot s$$

$$m = m$$

Here the units check out. This makes me conclude that the answer is correct with a very high probability.

1.1.4. Solution

The sphere hits the ground after approximately $3.19s$, the same as in part a.

$$t = \pm\sqrt{\frac{100m}{9.825ms^{-2}}} \approx 3.19s \tag{5}$$

The sphere moved approximately $22.97m$ horizontaly.

$$x_{impact} \approx 7.2ms^{-1} \cdot 3.19s \approx 22.97m \tag{6}$$

1.2. Part c

To solve part c I choose to use the ISEE method.

1.2.0. Identify

This part does not need any numerical values as the task is to make a drawing, and set up an equation, therefore no important known values are needed. It is specified that correct sign should be used such that positive direction is upwards.

1.2.1. Set up

I start with the free body diagram, as it provides some context.

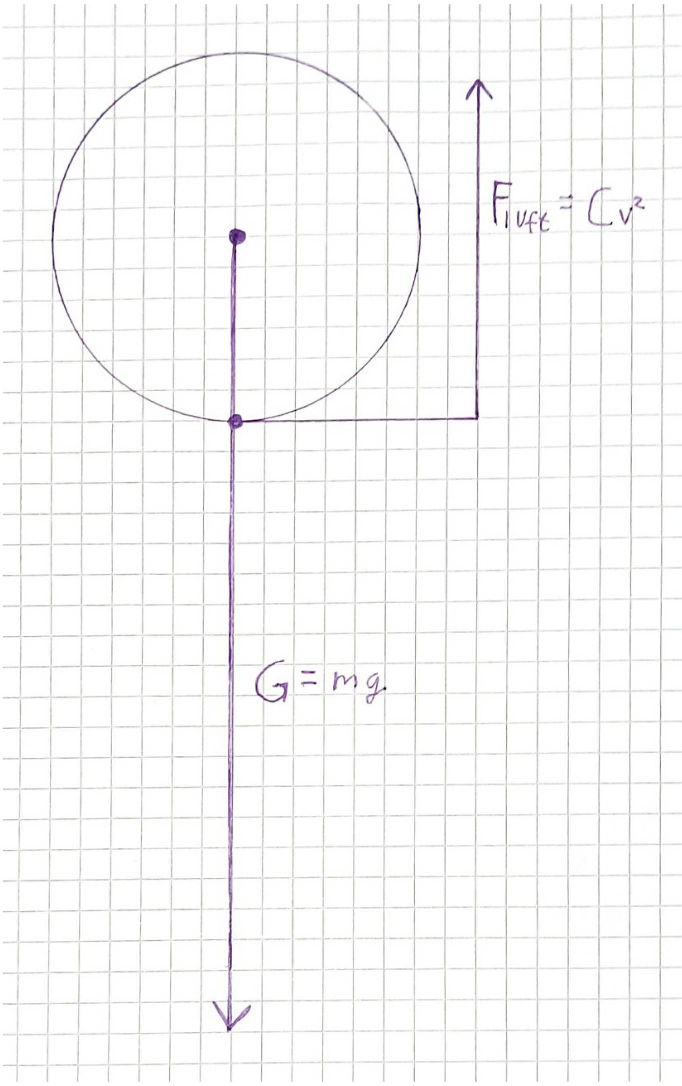

Figure 1.2: Free body diagram of sphere with air resistance

We only need to use one equation, Newtons 2nd law.

$$\sum F = ma \tag{7}$$

1.2.2. Execute

I start with modifying Equation 7, replacing the $\sum F$ with the forces drawn in Figure 1.2.

$$F_{luft} - G = ma \tag{8}$$

Now filling in for the forces and doing general arithmetic.

$$Cv^2 - mg = ma$$

$$\frac{Cv^2}{m} - g = a$$

$$a = \frac{dv}{dt} = -g + \frac{Cv^2}{m} \tag{9}$$

1.2.3. Evaluate

In Figure 1.2 I made G larger than F_{luft} as without high starting speed or other forces, F_{luft} can never be larger than G. Checking units for Equaton 9.

$$m \cdot s^{-2} = m \cdot s^{-2} + \frac{kg \cdot m^{-1} \cdot (m \cdot s^{-1})^2}{kg}$$

$$m \cdot s^{-2} = m \cdot s^{-2} + m^{-1} \cdot m^2 \cdot s^{-2}$$

$$m \cdot s^{-2} = m \cdot s^{-2} + m \cdot s^{-2}$$

$$m \cdot s^{-2} = m \cdot s^{-2}$$

Here the units checks out. This makes me conclude that the answer is correct with a very high probability.

1.2.4. Solution

Free body diagram is drawn in Figure 1.2.

Newtons 2nd law for this situation:

$$F_{luft} - G = ma \tag{8}$$

Can be rewritten to this:

$$a = \frac{dv}{dt} = -g + \frac{Cv^2}{m} \tag{9}$$

1.3. Part d

To solve part d I choose to use the ISEE method and use results from part c.

1.3.0. Identify

I find v_T to be the terminal velocity. I also find a value for $C = 5.00 \cdot 10^{-2} kgm^{-1}$ and $m = 100g$. I need to use Equation 9 from part c.

1.3.1. Set up

The first thing to do is to correct $m = 100g$ into $m = 0.1kg$ such as to keep all units as SI-units. Terminal velocity will cause no acceleration as there is no change in speed, this means that $\frac{dv}{dt} = 0$ at terminal velocity. It can be set up like this.

$$0 = -g + \frac{Cv^2}{m}$$

1.3.2. Execute

solving this for v using arithmetic.

$$g = \frac{Cv^2}{m}$$

$$mg = Cv^2$$

$$\frac{mg}{C} = v^2$$

$$v = \pm\sqrt{\frac{mg}{C}} \tag{10}$$

Only the negative answer makes sense with positive direction upwards. Filling in values for m, g and C.

$$v = -\sqrt{\frac{0.1kg \cdot 9.825ms^{-2}}{5.00 \cdot 10^{-2} kgm^{-1}}} \approx -4.43ms^{-1} \tag{11}$$

1.3.3. Evaluate

I feel like the magnitude of my answer is in the correct order and $-4.43ms^{-1}$ seems quite reasonable terminal speed for a object with mass $m = 100g$, it is maybe a bit slower than intuitivly expected for me. I will also want to check units for Equation **??**.

$$m \cdot s^{-1} = \sqrt{\frac{kg \cdot m \cdot s^{-2}}{kg \cdot m^{-1}}}$$

$$m \cdot s^{-1} = \sqrt{\frac{m \cdot s^{-2}}{m^{-1}}}$$

$$m \cdot s^{-1} = \sqrt{m^2 \cdot s^{-2}}$$

$$m \cdot s^{-1} = \sqrt{\frac{m^2}{s^2}}$$

$$m \cdot s^{-1} = \frac{m}{s}$$

$$m \cdot s^{-1} = m \cdot s^{-1}$$

1.3.4. Solution

Using Equation 9 with $\frac{dv}{dt} = 0$ you can get equation for terminal velovity.

$$v = \pm\sqrt{\frac{mg}{C}} \tag{10}$$

Filling in values for c we find that the teminal velocity is approximately $-4.43ms^{-1}$.

$$v = -\sqrt{\frac{0.1kg \cdot 9.825ms^{-2}}{5.00 \cdot 10^{-2}kgm^{-1}}} \approx -4.43ms^{-1} \tag{11}$$

2. Task 2

Now I may have overdone this section as I taught this was a good exercise to extend my python skills, in other words, the code does exactly as described, and more. But it does not reassemble the example code a whole lot. If clarifications on how the code works is needed, feel free to reach out. In general the code as shown in Subsection `2.4.` runs two simulations one with positive direction upwards named Sphere 1 and one with positive direction downwards, named Sphere 2 as the program handles both.

2.0. Part a

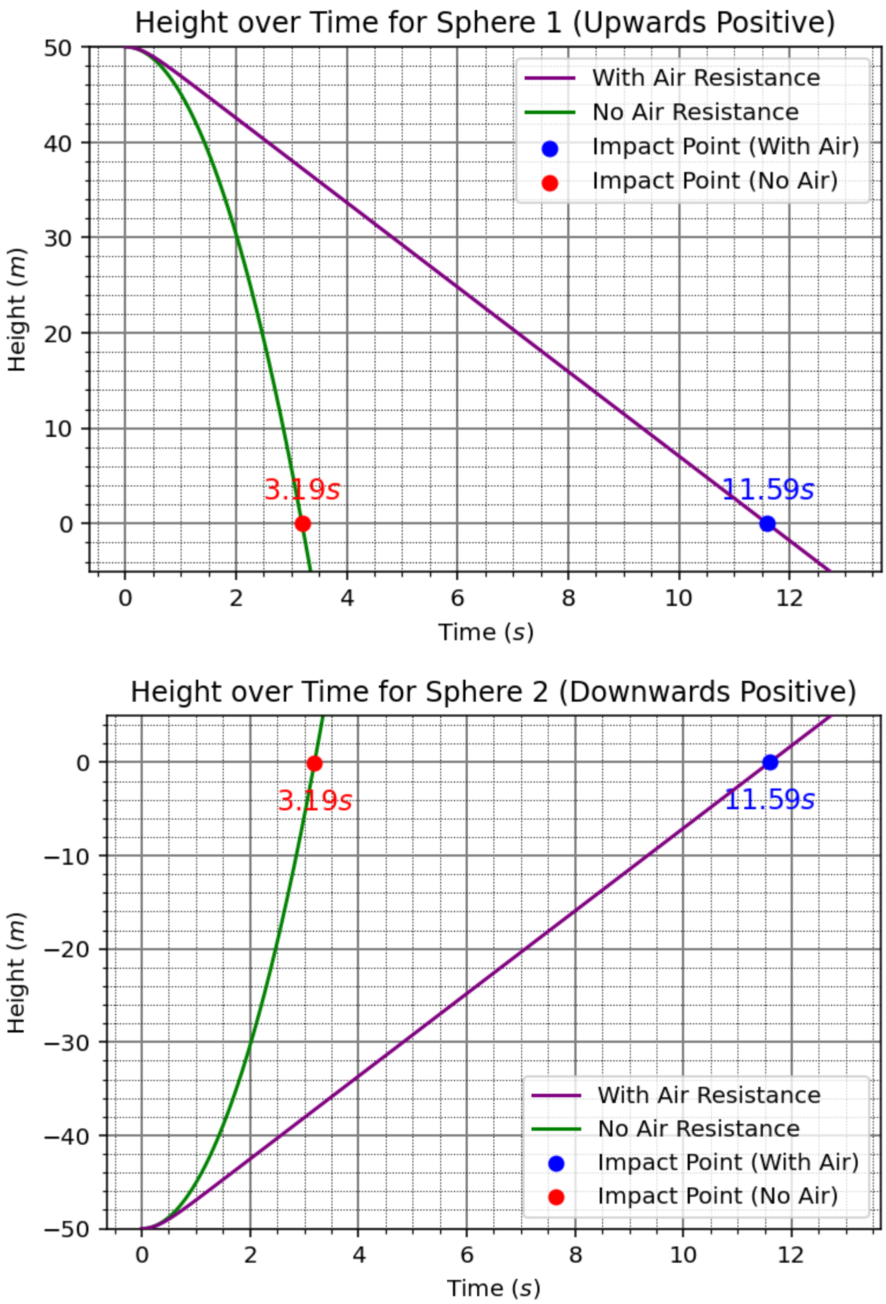

Figure 2.0: Simulations of height both with and without air resistance

2.1. Part b

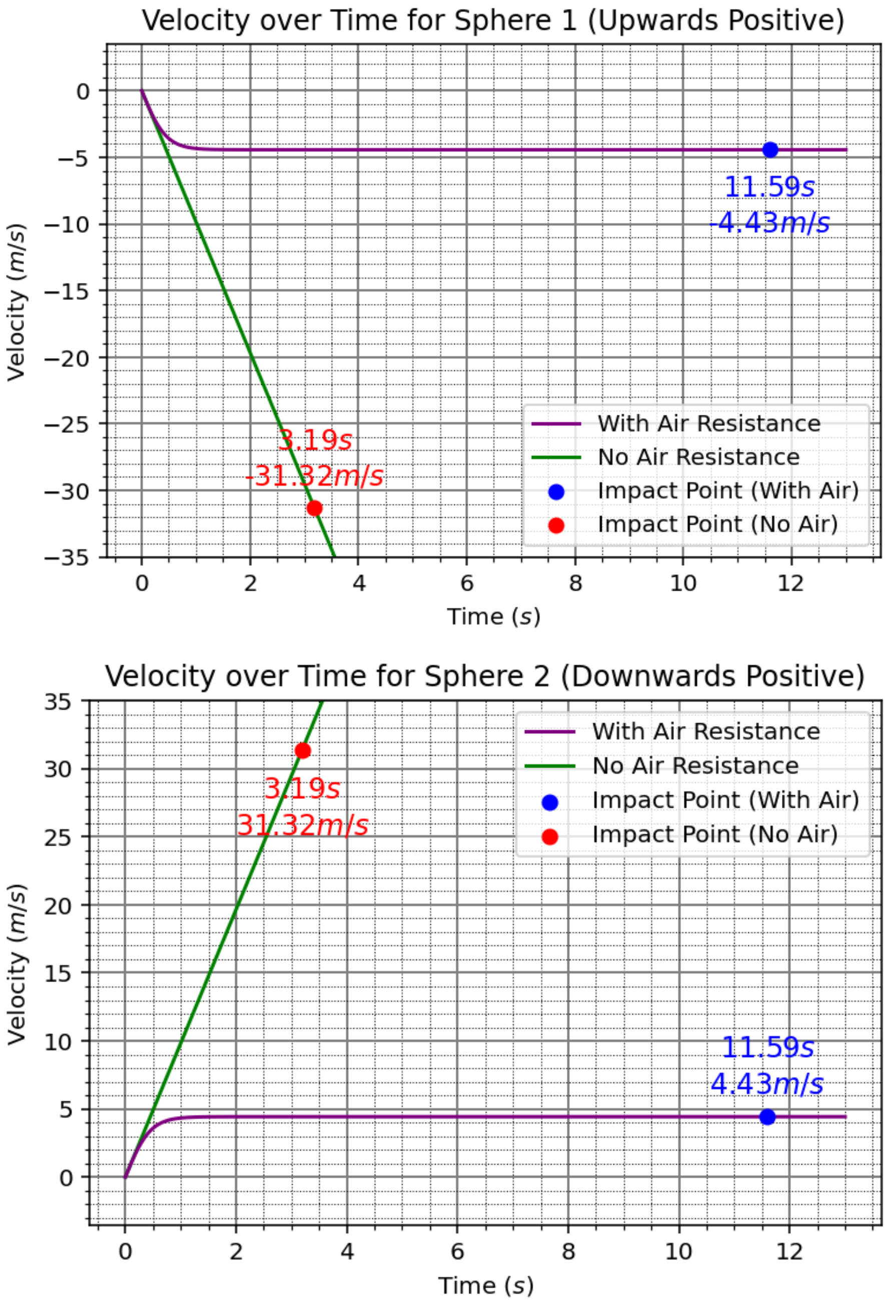

Figure 2.1: Simulations of velocity both with and without air resistance

2.2. Part c

We can see from the graph that terminal velocity v_T is achieved and the size is $4.43ms^{-1}$. This corresponds well with results of Equation 11

$$v = -\sqrt{\frac{0.1kg \cdot 9.825ms^{-2}}{5.00 \cdot 10^{-2}kgm^{-1}}} \approx -4.43ms^{-1} \tag{11}$$

2.3. Ekstra

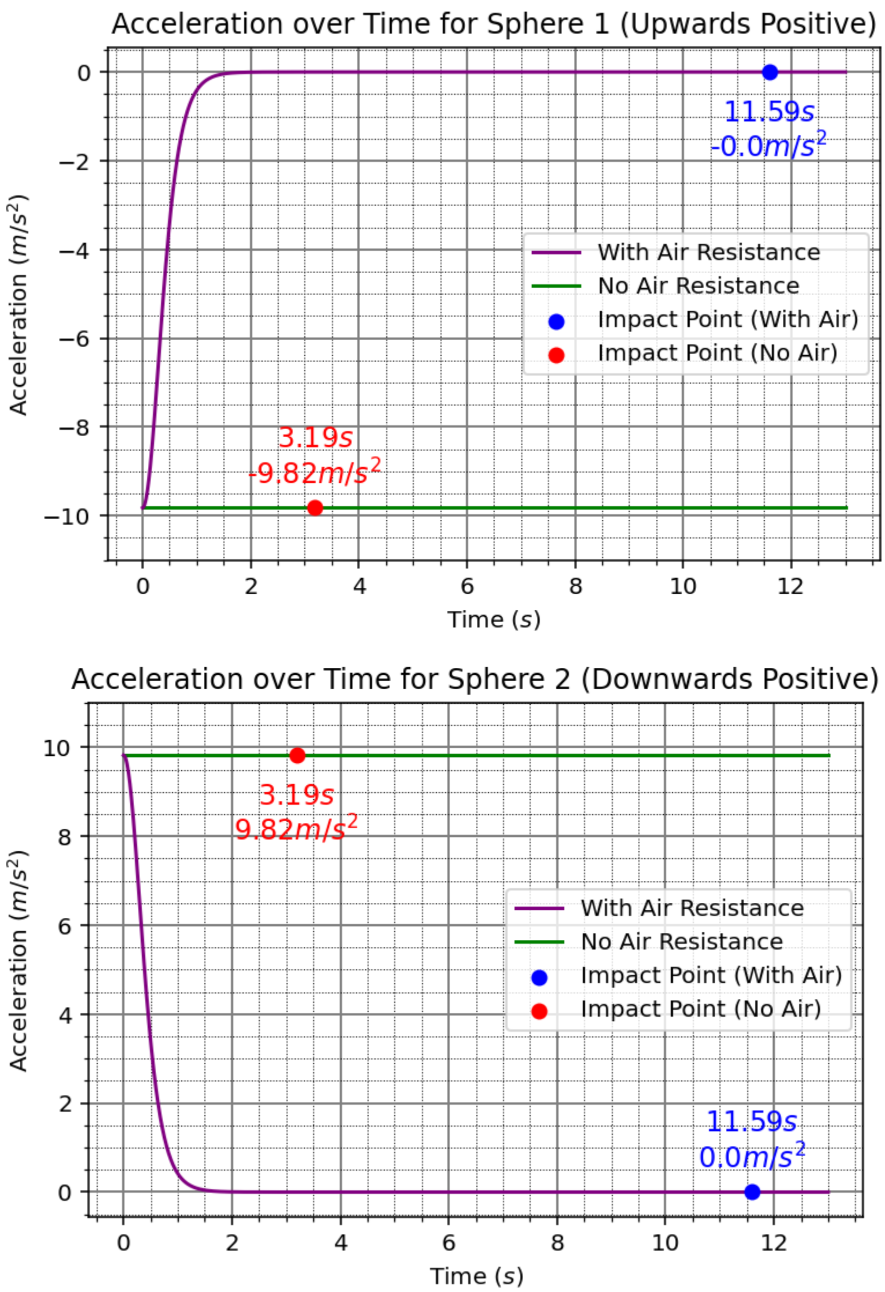

Figure 2.2: Simulations of acceleration both with and without air resistance

2.4. Source code

Python

```python
import numpy as np
import matplotlib.pyplot as plt

class FallingSphere:
    def __init__(self, name="Sphere", resolution=1000,
                 start_time=0.0, end_time=10.0, distance_from_ground=100,
                 speed_towards_ground=0.0, g=9.825, mass=1.0, C=0.05,
                 positive_direction="upwards"):
        """
        Initialize the FallingSphere simulation.
        g = 9.825 is used as this is the actual value for Oslo
        """
        if positive_direction not in ["upwards", "downwards"]:
            raise ValueError(
                "direction must be either 'upwards' or 'downwards'")

        if distance_from_ground <= 0:
            raise ValueError(
                "Height above ground must be greater than zero.")

        if g < 0 or (g == 0 and speed_towards_ground <= 0):
            raise ValueError(
                "Gravity must be positive"
                "or zero with initial speed towards ground")

        if mass <= 0:
            raise ValueError(
                "Mass needs to be positive")

        if C < 0:
            raise ValueError(
                "C needs to be zero (for no resistance) or positive")

        if start_time >= end_time:
            raise ValueError(
                "End time needs to be larger than start time")

        # Controls positive direction and how calculations change
        self.positive_direction = positive_direction
        self.sign = 1 if positive_direction == "upwards" else -1

        # General properties of object
        self.name = name
        self.resolution = resolution

        # Time based parameters
        self.t = np.linspace(start_time, end_time, resolution) # All time values
        self.dt = self.t[1] - self.t[0]  # Time step size

        # Constants
        self.g = g
        self.mass = mass
        self.C = C

        # Initial conditions with positive direction in mind
        self.h0 = self.sign * distance_from_ground
        self.v0 = self.sign * speed_towards_ground
        self.a0 = self.sign * -g

        # Arrays for simulation (with and without air resistance)
        self.h = np.zeros(resolution)
        self.v = np.zeros(resolution)
        self.a = np.zeros(resolution)
        self.h_noair = np.zeros(resolution)
        self.v_noair = np.zeros(resolution)
```

```
        self.a_noair = np.zeros(resolution)

        # Initial conditions stored in array
        self.h[0] = self.h0
        self.v[0] = self.v0
        self.a[0] = self.a0

        # Impact times
        self.t_impact = None
        self.t_noair_impact = None

    def calculate(self):
        """
        Compute the motion of the sphere over time.
        """
        # For loop starts at index 1 as initial value at index 0 is set
        for i in range(1, self.resolution):
            self.h[i] = self.h[i - 1] + self.v[i - 1] * self.dt
            self.v[i] = self.v[i - 1] + self.a[i - 1] * self.dt
            self.a[i] = self.sign * (
                -self.g + (self.C * (self.v[i] ** 2)) / self.mass)

        # Without air resistance (vectorized calculations, does not need looping)
        self.h_noair = self.h[0] + (
            self.v0 * self.t + 0.5 * self.sign * -self.g * (self.t ** 2))
        self.v_noair = self.sign * -self.g * self.t
        self.a_noair = self.sign * -self.g * np.ones_like(self.t)

    def find_impact_times(self):
        """
        Find the time indices when the sphere impacts the ground.
        """
        self.t_impact = np.argmin(np.abs(self.h))
        self.t_noair_impact = np.argmin(np.abs(self.h_noair))

        # Checks if simulation is long enough for impact
        if abs(self.h_noair[self.t_noair_impact]) > 0.5:
            raise ValueError(
                "Object does not hit the ground, within current timeframe")
        elif abs(self.h[self.t_impact]) > 0.5:
            raise ValueError(
                "Object does not hit the ground with air resistance, "
                "within current timeframe")


    def display_results(self):
        """
        Print the time and velocity at impact for both cases.
        """
        print(
            f'The sphere hits the ground at '
            f'{round(self.t[self.t_impact], 3)}s, '
            f'reaching a speed of {round(self.v[self.t_impact], 3)}m/s'
            f'with {self.positive_direction} positive direction')
        print(
            f'and without air resistance at '
            f'{round(self.t[self.t_noair_impact], 3)}s, '
            f'reaching a speed of {round(self.v_noair[self.t_noair_impact], 3)}m/s '
            f'with {self.positive_direction} positive direction')

    def plot_height(self):
        """
        Plot height vs. time for both cases.
        """
        plt.figure(0)

        max_val = np.ceil(abs(self.h0) / 5.0) * 5
```

```python
        y_step = max_val * 0.05
        min_val = 2 * y_step

        if self.sign == 1:
            plt.ylim(-min_val, max_val)
            plt.text(
                self.t[self.t_impact],
                self.h[self.t_impact] + 1 * y_step,
                f'{round(self.t[self.t_impact], 2)}$s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.h_noair[self.t_noair_impact] + 1 * y_step,
                f'{round(self.t[self.t_noair_impact], 2)}$s$',
                color="red", fontsize=12, ha="center")
        else:
            plt.ylim(-max_val, min_val)
            plt.text(
                self.t[self.t_impact],
                self.h[self.t_impact] - 2 * y_step,
                f'{round(self.t[self.t_impact], 2)}$s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.h_noair[self.t_noair_impact] - 2 * y_step,
                f'{round(self.t[self.t_noair_impact], 2)}$s$',
                color="red", fontsize=12, ha="center")

        plt.plot(
            self.t, self.h,
            color="purple", label="With Air Resistance", zorder=3)
        plt.plot(
            self.t, self.h_noair,
            color="green", label="No Air Resistance", zorder=2)

        plt.scatter(
            self.t[self.t_impact], self.h[self.t_impact],
            color="blue", zorder=4, label="Impact Point (With Air)")
        plt.scatter(
            self.t[self.t_noair_impact], self.h_noair[self.t_noair_impact],
            color="red", zorder=3, label="Impact Point (No Air)")

        plt.grid(which="both", linestyle="-", color="gray", linewidth=1)
        plt.minorticks_on()
        plt.grid(which="minor", linestyle=":", color="black", linewidth=0.5)

        plt.title(
            f'Height over Time for {self.name} '
            f'({self.positive_direction.capitalize()} Positive)')
        plt.legend()
        plt.xlabel(r'Time ($s$)')
        plt.ylabel(r'Height ($m$)')
        plt.show()

    def plot_velocity(self):
        """
        Plot velocity vs. time for both cases.
        """
        plt.figure(1)
        max_val = np.ceil(abs(self.v_noair[self.t_noair_impact]) / 5.0) * 5
        y_step = max_val * 0.05
        min_val = 2 * y_step
        if self.sign == 1:
            plt.ylim(-max_val, min_val)
            plt.text(
                self.t[self.t_impact],
```

```
                self.v[self.t_impact] - 2 * y_step,
                f'{round(self.t[self.t_impact], 2)}$s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_impact],
                self.v[self.t_impact] - 3.5 * y_step,
                f'{round(self.v[self.t_impact], 2)}$m/s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.v_noair[self.t_noair_impact] + 2.5 * y_step,
                f'{round(self.t[self.t_noair_impact], 2)}$s$',
                color="red", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.v_noair[self.t_noair_impact] + 1 * y_step,
                f'{round(self.v_noair[self.t_noair_impact], 2)}$m/s$',
                color="red", fontsize=12, ha="center")
        else:
            plt.ylim(-min_val, max_val)
            plt.text(
                self.t[self.t_impact],
                self.v[self.t_impact] + 2.5 * y_step,
                f'{round(self.t[self.t_impact], 2)}$s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_impact],
                self.v[self.t_impact] + 1 * y_step,
                f'{round(self.v[self.t_impact], 2)}$m/s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.v_noair[self.t_noair_impact] - 2 * y_step,
                f'{round(self.t[self.t_noair_impact], 2)}$s$',
                color="red", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.v_noair[self.t_noair_impact] - 3.5 * y_step,
                f'{round(self.v_noair[self.t_noair_impact], 2)}$m/s$',
                color="red", fontsize=12, ha="center")


        plt.plot(
            self.t, self.v,
            color="purple", label="With Air Resistance", zorder=3)
        plt.plot(
            self.t, self.v_noair,
            color="green", label="No Air Resistance", zorder=2)

        plt.scatter(
            self.t[self.t_impact], self.v[self.t_impact],
            color="blue", zorder=4, label="Impact Point (With Air)")
        plt.scatter(
            self.t[self.t_noair_impact], self.v_noair[self.t_noair_impact],
            color="red", zorder=3, label="Impact Point (No Air)")

        plt.grid(which="both", linestyle="-", color="gray", linewidth=1)
        plt.minorticks_on()
        plt.grid(which="minor", linestyle=":", color="black", linewidth=0.5)

        plt.title(
            f'Velocity over Time for {self.name} '
            f'({self.positive_direction.capitalize()} Positive)')
        plt.legend()
        plt.xlabel(r'Time ($s$)')
        plt.ylabel(r'Velocity ($m/s$)')
```

```python
        plt.show()

    def plot_acceleration(self):
        """
        Plot acceleration vs. time for both cases.
        """
        plt.figure(2)
        max_val = np.ceil(self.g / 2.0) * 2 + 1
        y_step = max_val * 0.05
        min_val = y_step
        if self.sign == 1:
            plt.ylim(-max_val, min_val)
            plt.text(
                self.t[self.t_impact],
                self.a[self.t_impact] - 2 * y_step,
                f'{round(self.t[self.t_impact], 2)}$s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_impact],
                self.a[self.t_impact] - 3.5 * y_step,
                f'{round(self.a[self.t_impact], 2)}$m/s^2$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.a_noair[self.t_noair_impact] + 2.5 * y_step,
                f'{round(self.t[self.t_noair_impact], 2)}$s$',
                color="red", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.a_noair[self.t_noair_impact] + 1 * y_step,
                f'{round(self.a_noair[self.t_noair_impact], 2)}$m/s^2$',
                color="red", fontsize=12, ha="center")
        else:
            plt.ylim(-min_val, max_val)
            plt.text(
                self.t[self.t_impact],
                self.a[self.t_impact] + 2.5 * y_step,
                f'{round(self.t[self.t_impact], 2)}$s$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_impact],
                self.a[self.t_impact] + 1 * y_step,
                f'{round(self.a[self.t_impact], 2)}$m/s^2$',
                color="blue", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.a_noair[self.t_noair_impact] - 2 * y_step,
                f'{round(self.t[self.t_noair_impact], 2)}$s$',
                color="red", fontsize=12, ha="center")
            plt.text(
                self.t[self.t_noair_impact],
                self.a_noair[self.t_noair_impact] - 3.5 * y_step,
                f'{round(self.a_noair[self.t_noair_impact], 2)}$m/s^2$',
                color="red", fontsize=12, ha="center")

        plt.plot(
            self.t, self.a,
            color="purple", label="With Air Resistance", zorder=3)
        plt.plot(
            self.t, self.a_noair,
            color="green", label="No Air Resistance", zorder=2)

        plt.scatter(
            self.t[self.t_impact], self.a[self.t_impact],
            color="blue", zorder=4, label="Impact Point (With Air)")

```

```python
        plt.scatter(
            self.t[self.t_noair_impact], self.a_noair[self.t_noair_impact],
            color="red", zorder=3, label="Impact Point (No Air)")

        plt.grid(which="both", linestyle="-", color="gray", linewidth=1)
        plt.minorticks_on()
        plt.grid(which="minor", linestyle=":", color="black", linewidth=0.5)

        plt.title(
            f'Acceleration over Time for {self.name} '
            f'({self.positive_direction.capitalize()} Positive)')
        plt.legend()
        plt.xlabel(r'Time ($s$)')
        plt.ylabel(r'Acceleration $(m/s^2)$')
        plt.show()

    def run_simulation(self):
        """
        Run the full simulation.
        """
        self.calculate()
        self.find_impact_times()
        self.display_results()
        self.plot_height()
        self.plot_velocity()
        self.plot_acceleration()

# Run the simulation
if __name__ == "__main__":
    sim_1 = FallingSphere(
        "Sphere 1",
        distance_from_ground = 50, end_time = 13.0, mass = 0.1, C = 0.05)
    sim_1.run_simulation()

    sim_2 = FallingSphere(
        "Sphere 2",
        distance_from_ground = 50, end_time = 13.0, mass = 0.1, C = 0.05,
        positive_direction = "downwards")
    sim_2.run_simulation()
```

Code 2.3: Simulation of falling spheres, over engineered to a high degree