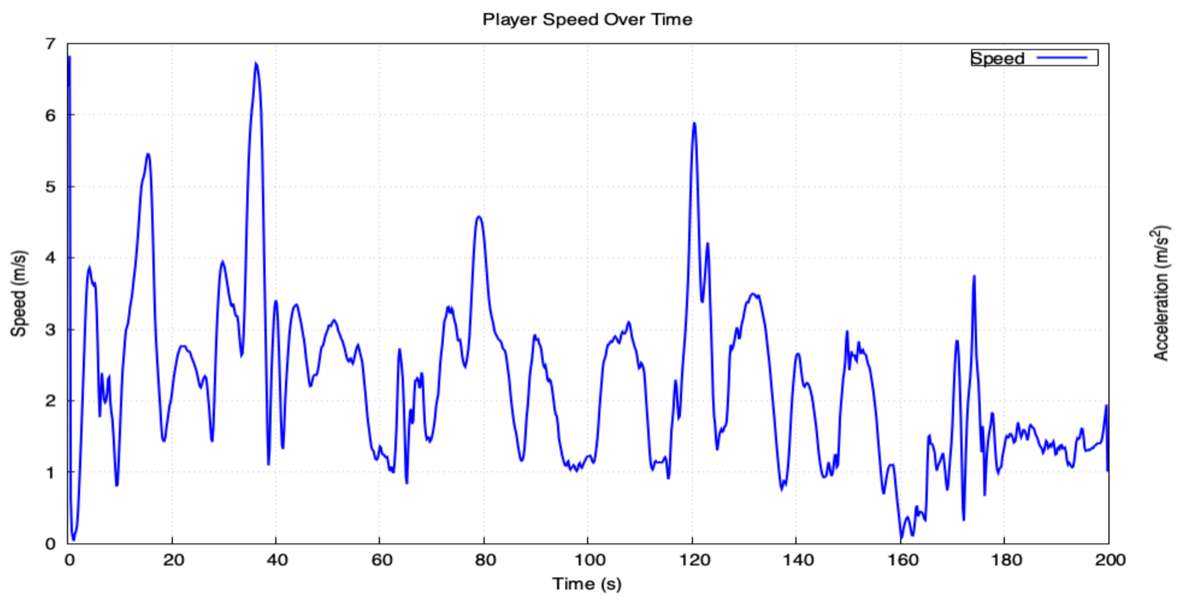
Student ID: 210899247

Question 1c
So, to model national and club teams I would define two separate classes, NationalTeam and ClubTeam, both derived from the base Team class.
The NationalTeam class would introduce a new member variable, continent of type string, while the ClubTeam class would introduce a new member variable, country of type string. Therefore both classes would inherit all other attributes and methods from Team. This structure reflects the different additional information needed for each team type while maintaining the shared functionality through inheritance.

Question 2c



The speed magnitude was computed and plotted against time. From the plot, we can see that the player's speed fluctuates throughout the 200 seconds, showing several peaks corresponding to bursts of movement. The maximum speed reached by the player was **6.82937 m/s**, as indicated by the highest point on the plot.

Question 2d
The resulting magnitued values were stored in a new valarray a_mag. Thus, the v_mag.size equals to 999 because it is calculated by applying centered finite difference once to the original data , which has 1001 points, thus removing two points.
a_mag.size() equals to 997 because it is calculated by applying centered finite differences a second time to the velocity data, which already has 999 points, again removing two points

Question 2e
The main sources of error are **truncation error** and **round-off error.** Truncation error comes from using finite difference approximations for derivatives, and can be reduced

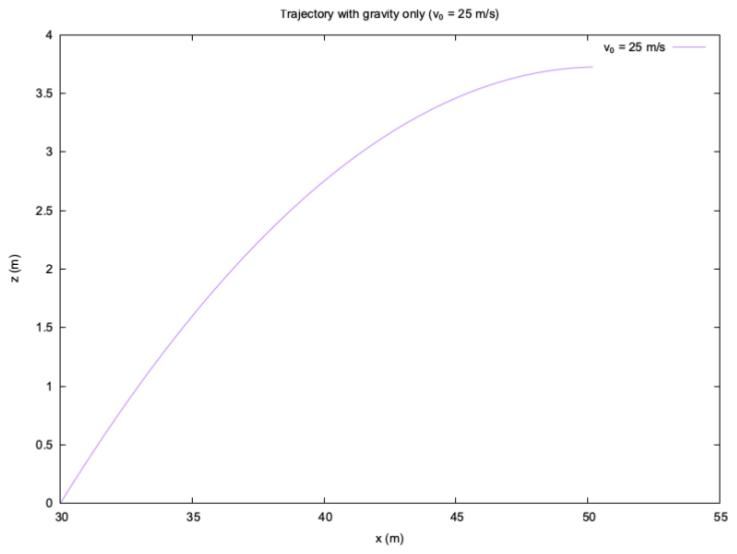
by using smaller Δt or higher-order schemes. Round-off error arises from limited floating-point precision and increases with the number of operations. In this case, truncation error is the dominant one. Float is not the best choice for the real-valued variables in this computation. Since operations involve many subtractions between similar values and square roots, it would be better to use double, which offers higher precision and reduces rounding errors.

Question 2f
To make the motion appear smooth on a 120 Hz display using data sampled at 5 Hz, I would use spline interpolation on the x(t) and y(t) arrays. The original data points are spaced 0.2 s apart, so interpolating at intervals of 1/120 s is required to match the frame rate. Spline interpolation produces smooth and continuous trajectories between the known points, avoiding sharp transitions and visual artifacts. This provides a realistic reconstruction of the player's movement at the required resolution.

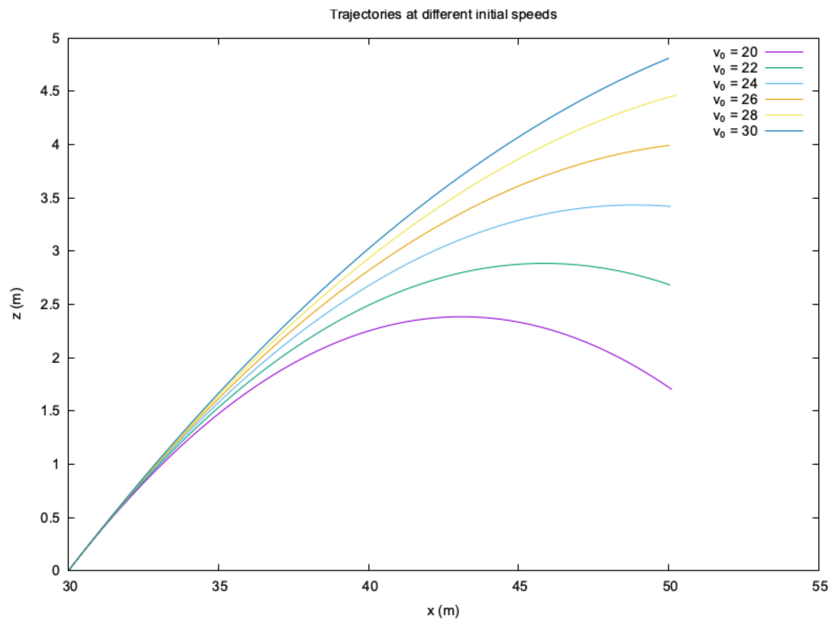
Queston 3b
The interpolating polynomial has degree 3 because it is constructed from 4 distinct data points, and a Lagrange polynomial passing through n points has degree n−1. In this case, the data table contains speed–power values at four speeds, resulting in a unique polynomial of degree 3 that interpolates all of them. When evaluating the function on the provided speed data, interp_coeffs() is the preferred method. This is because the polynomial coefficients only need to be computed once, and then the resulting polynomial can be efficiently evaluated at each speed value. In
contrast, Lagrange_N() constructs the entire expression symbolically for every evaluation, which is computationally more expensive.

Question 3e
The error for the new formula is of order $\mathcal{O}(1/N^6)$. This is because the four-point Newton–Cotes rule has a local truncation error of order h^7, which leads to a global error of order $h^6 = 1/N^6$ when applied as a composite rule. This indicates higher accuracy compared to lower-order methods, provided the integrand is smooth.

Question 4b
The output is stored in simple_v25.dat and visualised on the (x,z) plane using gnuplot, as shown on the figure. As could be seen on the plot, the shot is over.



Question 4c



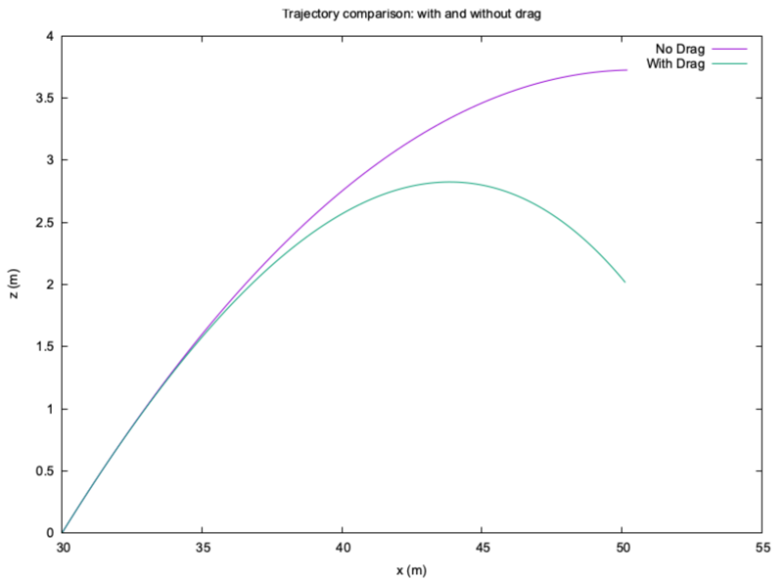
Among the shots tested, only the one with v0 = 20 m/s was classified as good, passing under the bar. All other shots with v0 in {22, 24, 26, 28, 30} m/s went over. This matches my physical and mathematical intuition: increasing the initial speed raises both the maximum height and range, leading to overshooting the goal height when the launch angle is fixed.

Question 4d

The equation when introducing the drag force:

$$\vec{Y}'(t) := \begin{bmatrix} x'(t) \\ y'(t) \\ z'(t) \\ v_x'(t) \\ v_y'(t) \\ v_z'(t) \end{bmatrix} = \begin{bmatrix} v_x(t) \\ v_y(t) \\ v_z(t) \\ -\frac{1}{2m}C_d\rho A|\vec{v}|v_x \\ -\frac{1}{2m}C_d\rho A|\vec{v}|v_y \\ -g - \frac{1}{2m}C_d\rho A|\vec{v}|v_z \end{bmatrix} =: \vec{f}(t, \vec{Y})$$

The plot



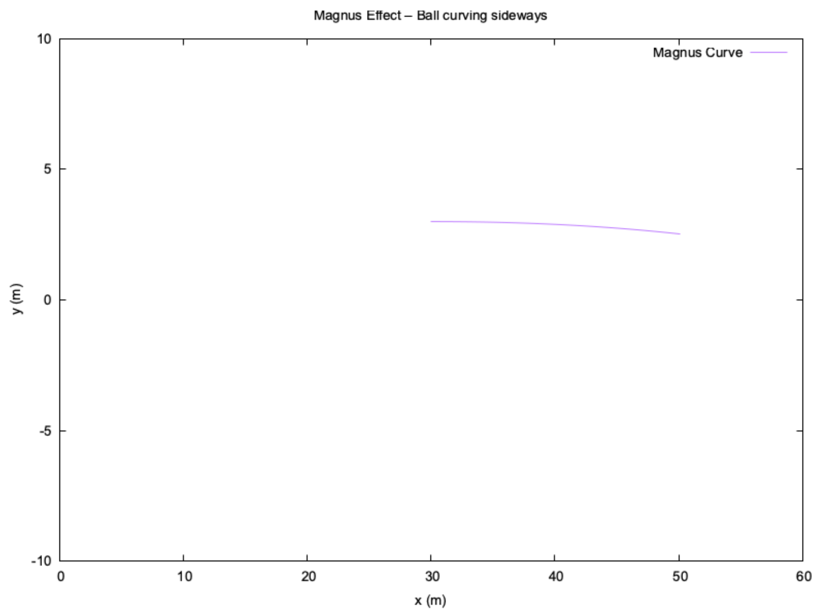
With drag enabled, the ball crosses the goal line under the bar, confirming a good shot. The trajectory is flatter and shorter compared to the no-drag case, as expected. Time to cross the goal line increases to 1.13 s, indicating that air resistance slows the ball down.

Question 4e

Motion of the football including drag and magnus forces:

$$\vec{Y}'(t) := \begin{bmatrix} x'(t) \\ y'(t) \\ z'(t) \\ v_x'(t) \\ v_y'(t) \\ v_z'(t) \end{bmatrix} = \begin{bmatrix} v_x(t) \\ v_y(t) \\ v_z(t) \\ -\frac{1}{2m}C_d\rho A|\vec{v}|v_x + \frac{S}{m}\omega_z v_y \\ -\frac{1}{2m}C_d\rho A|\vec{v}|v_y - \frac{S}{m}\omega_z v_x \\ -g - \frac{1}{2m}C_d\rho A|\vec{v}|v_z \end{bmatrix} =: \vec{f}(t, \vec{Y})$$

Question 4f



The shot is good. The ball crosses the goal line 6.18369 m from the left post at y = -3.66(-GOAL_W/2), confirming the Magnus effect curves the ball inward.