Team 429

Problem B

FIFA penalty kicks

Abstract

In this work, we analyse motion of a football during penalty kick. We introduce forces acting on a ball due to presence of air and derive equations of motion for a kicked ball during penalty kick. We propose a simplified model of motion preceded by crucial assumptions. We obtain a family of trajectory curves as a solution of motion equations. Then we find extreme curves that are described by critical initial parameters which are setting ranges of possible values of initial conditions that are necessary for a ball to reach upper corner.

We develop tools allowing us to accurately predict ball trajectories and use them to produce predictions about shots in a set subspace of parameters. We find a method to emulate a goalkeeper and incorporate him into our numerical model to generate maps of the difficulty of different shots. We study the effect of the power, direction and rotation on the trajectory and determine how they affect the result of the penalty kick.

Contents

1 Introduction

1.1 Problem interpretation

In our paper we will model penalty kick as a simple projectile with drag force and lift force acting due to the rotation of the ball. Our goal is to determine how wide is the range of initial velocity and spin of the ball, that can result in a shot in the upper corner. We also aim to find such velocity and spin, that make it possible to avoid the goalkeeper and score a goal. The main issue in this part of the problem is to properly model the behaviour of the goalkeeper and combine it with previous results.

We focus on the aerodynamic approach to the problem, thus we do not consider the ball bouncing off the ground or the goal frame. All other necessary assumptions will be introduced together with particular models.

1.2 Football

In football, in certain situations, penalty kicks are executed. A penalty kick is taken at a distance of 11 meters from the center of the goal. The most desirable spots to shoot the ball at are the upper corners – these shots are usually the most difficult to save.

We define the upper corner of the goal as a square with sides 75 cm long. Such choice is motivated by intuition and symmetry of football's cross section. We take the radius of the ball to be 0.11 m [8]. Other necessary dimensions are provided in the problem statement.

The goalkeeper's job is to prevent the ball from coming inside the goal. The goalkeeper has a limited time to react and reach the ball, which is the main reason why the striker usually comes out on top. The rules prevent the goalkeeper from coming forward, which would grant him an advantage.

2 Notations

Symbol	Meaning	Value
g	gravitational acceleration	9.81 $\mathrm{m/s^2}$
ρ	fluid density	1.2 $\mathrm{kg/m^3}$(for air)
$\vec{v}$	velocity	
$\vec{\omega}$	angular velocity	
$\vec{F}_M$	force due to Magnus Effect	
$\vec{F}_D$	drag force	
c_M	lift coefficient	0.3 [4]
c_D	drag coefficient	0.25 [5] [3]
m	mass	0.4 kg [8]
I	moment of inertia	
a	radius of the ball	
t	time	
λ [1]	angular friction	

Table 1: Table of used symbols

3 Motion analysis

3.1 Introduction and assumptions

First, let us consider the simplest case of kicking a football in open space. At the moment after impact, the ball possesses a certain initial velocity $\vec{v}$ and spin (angular velocity) $\vec{\omega}$.

In general, both c_M and c_D are functions of the Reynolds number, thus they change depending on the fluid, that the ball is immersed in. However, we are going to assume constant values of those coefficients, for a given football and air of known, constant parameters.

3.2 Air drag and lift

When a ball is moving in the air we usually consider hydrodynamic forces, such as drag, as negligible, so taking into account only gravity seems reasonable. However during penalty kick the speed of the ball is great enough for the air drag and lift forces to be significant. Especially when the ball is rotating, one can notice that the trajectory starts to curve to the side. It happens due to the Magnus effect.

3.3 Magnus effect and drag force

When a sphere is rotating with angular velocity ω in uniform flow of air the difference of pressure is creating a lift force which is perpendicular to the direction of velocity and is given by:

$$\vec{F_m} = \pi\rho a^3 c_M |\vec{v}||\vec{\omega}|(\hat{v} \times \hat{\omega}) \tag{1}$$

where a is ball's radius, c_M is a constant and v is speed of the ball. The second force that is acting on the ball is air drag given by equation:

$$\vec{F_D} = -\frac{1}{2} c_D \rho \pi a^2 |\vec{v}|\vec{v} \tag{2}$$

3.4 Governing equations

The equations of motion in 3D, which take into account gravity, air drag and lift force, are given by:

$$\begin{cases} m\ddot{x} = \pi\rho a^3 c_M |\vec{\omega}|\dot{y} - \frac{1}{2} c_D \rho \pi a^2 |\vec{v}|\dot{x} \\ m\ddot{y} = -\pi\rho a^3 c_M |\vec{\omega}|\dot{x} - \frac{1}{2} c_D \rho \pi a^2 |\vec{v}|\dot{y} \\ m\ddot{z} = -g - \frac{1}{2} c_D \rho \pi a^2 |\vec{v}|\dot{z} \end{cases} \tag{3}$$

4 Accuracy of shots - simulation and analysis

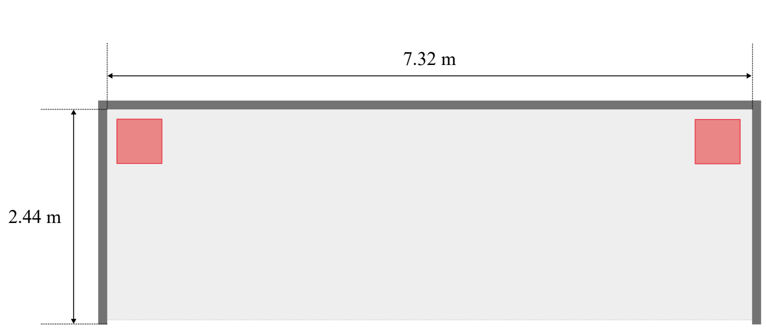
4.1 Simulated events

Knowing all the forces acting on the ball, we can effectively simulate its movement. Using Python and an RK4 algorithm [7] we constructed a simulation that incorporates gravity, drag and lift forces and rotation dissipation ($\dot{\omega} = -\lambda\omega$ [1]) to calculate the trajectory of the projectile given its initial velocity and rotation. We use a constant time step dt of 0.01 s and incorporate constants at values stated in Table 1.

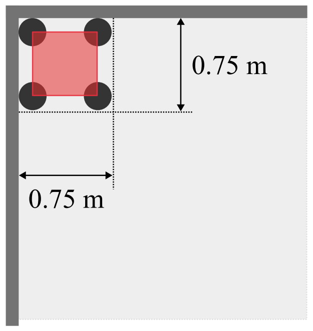
We run the simulation until the ball hits the ground (meaning we do not consider bounces) or reaches the goal line. At that point, we determine whether the ball:

- did not reach the goal (Fig. 2a)
- was a miss or hit the frame (Fig. 2b)
- was a shot on goal (Fig. 2c)
- and whether it hit the upper corner (which we define as explained at Fig. 1)

The Figure 2d shows the same shot as Fig 2c from an upper angle and demonstrates the curvature of the trajectory as a result of the Magnus effect.

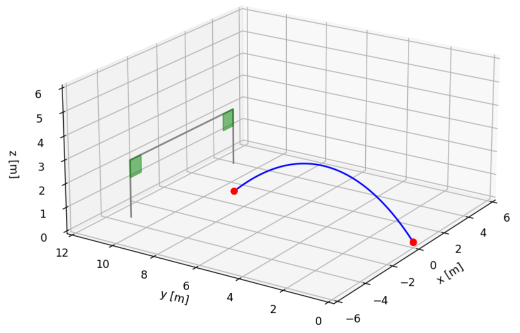


(a) We classify a shot as a goal when the ball's center reaches the goal line and ball's surface is not touching the frame or the ground).

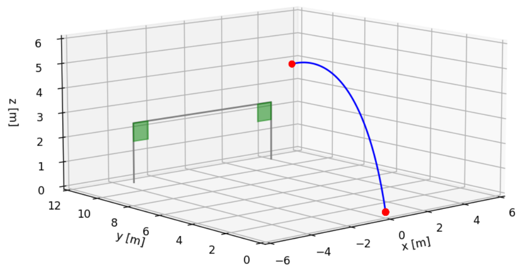


(b) We define a goal as an upper corner one, when the ball is entirely inside a 0.75 m square.

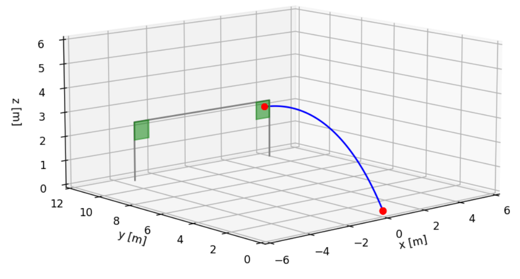
Figure 1: Dimensions of the goal and definitions of a standard and upper corner goal.



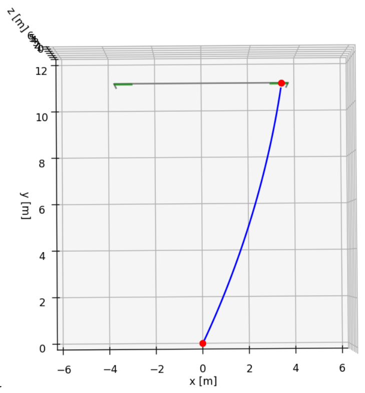
(a) A shot that was too weak and hit the ground before reaching the goal line



(b) A shot that was off the target and too strong.



(c) A shot that hit the corner. The trajectory resembles a parabola, but is changed by lift and drag forces.



(d) Same shot as in Fig. 2c, except viewed from the top. One can clearly see the effect of the Magnus force (the rotation was 70 rad/s along the z axis).

Figure 2: Simulations of penalty kick trajectories. Blue lines represent the trajectories, red dots starting and ending points and the green squares represent the top corners.

This simple, but accurate simulation allows us to check any set of $\vec{v}$ and $\vec{\omega}$, which is a powerful tool used in further analysis. When facing a 6-dimensional parameter space it is nearly impossible to extensively cover every combination. To be able to visualize our results, we must limit ourselves to a 2-dimensional plot. We can do this by assuming a set $v := |\vec{v}|$ and $\vec{\omega}$. We also rarely add an ω_y component, as it is impractical to perform when shooting forward (it leads to interesting patterns – see Fig. 6b). We then define φ and θ as spherical angles describing the initial direction of $\vec{v}$ (where the x axis would point to the right, y towards the goal and z up). We produce plots by checking the results of penalties taken in different directions, plotting them linearly with the angles. The result is a spherical plot mapped onto a plane as explained on Figure 3. With this we can see what are the results of

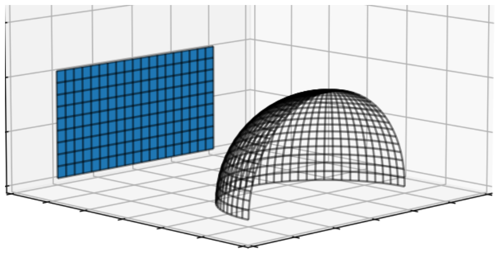


Figure 3: Key to understanding the subsequent plots: a unit sphere around the shooter (or, more accurately, the ball) is divided into cells based on the angles. These cells are projected onto a plane, to make it possible to visualize them in this paper. Perhaps however, a large colored sphere around the player could prove more useful to him. The quarter of a sphere represents possible directions of the initial velocity vector.

penalties taken without rotation, with a medium v =25 m/s, on Figure 4 depending on the direction the kick was taken. The range of the parameters φ, θ was decreased to only show shots that went near the goal. With nothing stopping us from experimenting with different

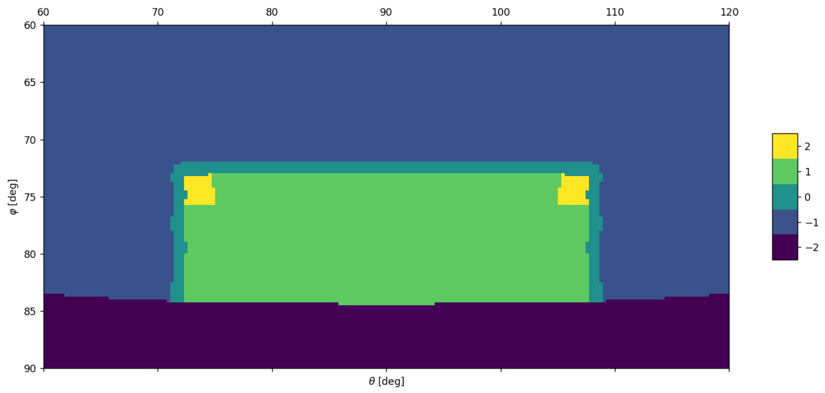
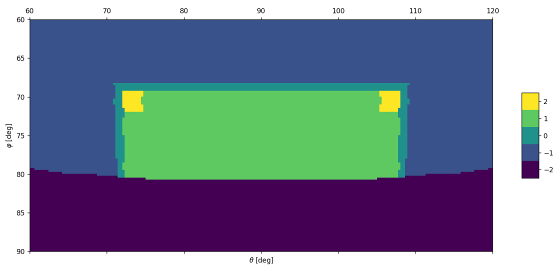
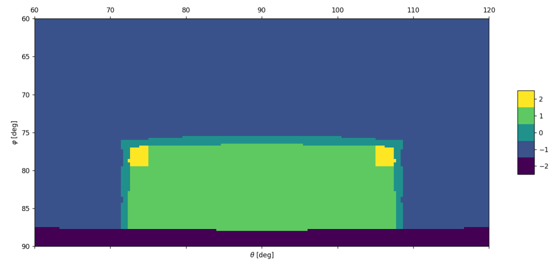


Figure 4: Results of a penalty taken with no spin. Each pixel represents one simulation and its color – the result. Result -2 means the ball hit the ground before reaching the goal line. Result -1 indicates the shot was a miss. Result 0 means the ball hit the frame. Result 1 means it was a shot on goal and 2 means it was an upper corner shot (Fig. 1). As can be seen, the shots needs to be at an angle $\theta < 85^o$ to even reach the goal (for v = 25 m/s)

rotations, we can simulate the results for a top and back spin, as well as spins along the z and y axis.

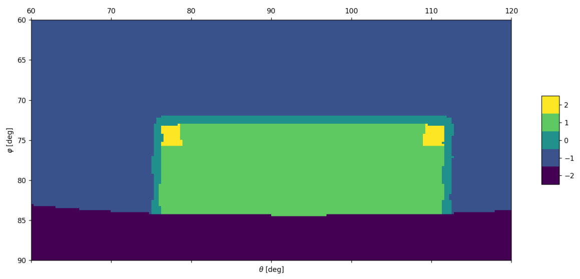


(a) In green and yellow are the angles one should aim at to shoot a goal with a topspin (of -70 rad/s along the x axis). Clearly in comparison to Fig. 4, the shooter has to aim higher.

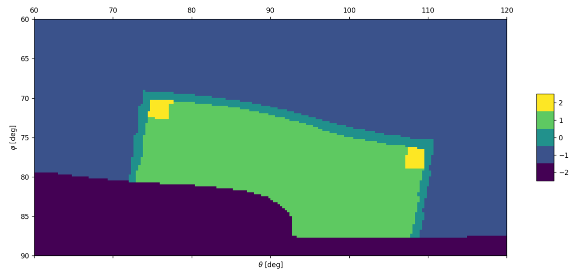


(b) Results of shots with a backspin (+70 rad/s along x axis). In contrast to the top spin, the ball needs to be aimed lower.

Figure 5: Result plots for different angles for 25 m/s shots with rotation along the x axis. Legend is the same as in Fig. 4.



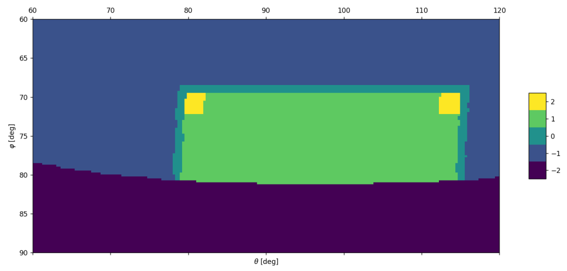
(a) Results of shots with perhaps the most famous 'sideways' rotation (70 rad/s along z axis). For this rotation and velocity of 25 m/s the shooter has to aim around 6 degrees to the right, compared to no rotation.



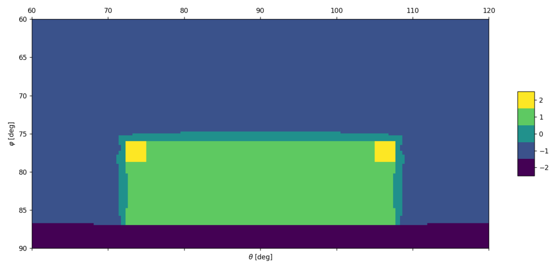
(b) Result for a spin -70 rad/s along the y axis. This type of spin is difficult to obtain when shooting towards the goal. We can see that it results in downward lift when shooting to the left and upward when shooting to the right. This would of course be reversed for a counterclockwise spin.

Figure 6: Results for sideways spins. For the sake of the plots, the chosen rotations were quite high. For a standard penalty kick the rotation would probably even slower and the effects even more minor.

As the figures suggest, applying a rotation requires a correction in initial velocities. However, it does not appear to make the shot any easier: if we think of accuracy as of the distribution of the initial angles, the area on the plots would be proportional to hitting the goal or respectively the corner. We can check if different types of shots make a larger effect on Figure 7



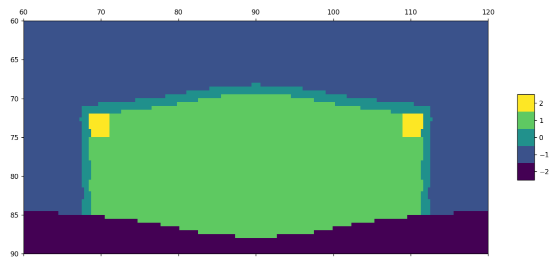
(a) A weaker shot with $v = 20$ m/s but a very large rotation of 100 rad/s along the z axis. The results are shifted by a larger margin but still remain in the expected shape.



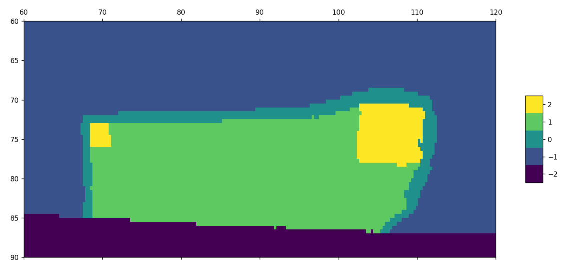
(b) A stronger shot with velocity of 35 m/s but no rotation. The obtained results are very regular. However, this part of this work is not taking into account the changes in accuracy resulting from increasing the power of the shot.

Figure 7: Comparison of result plots of a weaker but more curved shot and a stronger one with no rotation.

As can be seen above, the adding a rotation to the ball can change the direction one has to aim to hit the corner. However, there does not seem to be any practical benefits from doing so. We could however, add rotation to the ball that otherwise would miss, to make it land inside the goal. Let us devise an algorithm for guiding a shot to a point at some x and z at the goal line: let us define a point A as the final point of the trajectory for a shot without spin. Then, let $\vec{u}$ be a normalized projection of a vector, from the A_{xz} point to the desired point, onto the xz plane. Then $\vec{\omega_0} = (0, \omega, 0) \times \vec{u}$ would guide the ball toward the desired point. But, this would add rotation to the shots that were already accurate causing them to miss. To correct this we can modify ω_0 by a function like $e^{-\kappa^2}$ (where κ is proportional to the distance of the trial shot from the target). The result of such an algorithm can be seen on Figure 8. The magnitude of the correcting angular velocity is represented on Figure 9b.

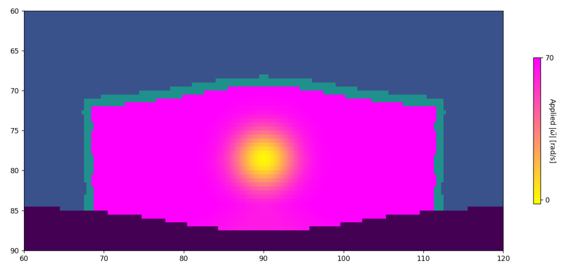


(a) Result plot of shots with rotation guiding them into the center of the goal. Such a technique is expanding the 'goal area' which in theory should increase the chance of scoring. However, it is not as easy to perform, because it uses information from trial (no rotation) shots. It suggests however, that it is a good idea to rotate the ball inwards, at least accuracy-wise.

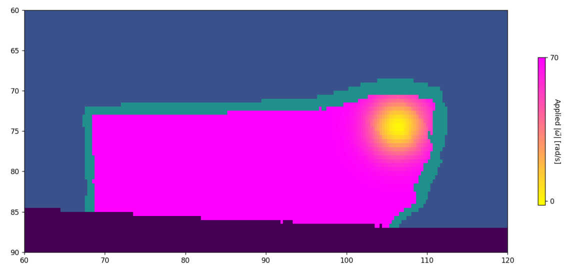


(b) Results plot of shots guided into the center of the upper right corner. In this instance, it became apparent it is necessary to limit the rotation, as not to misguide already accurate shots. The magnitude of the rotation is dictated by an exponent function of the square of the displacement. This can be seen more accurately on figure 9b

Figure 8: Plots showing how a non-uniform rotation distribution among shots can impact the collective results. The shots were guided towards a target with a maximal angular velocity of 70 rad/s.



(a) Magnitude of added angular velocities based on distance from the middle of the goal.



(b) Magnitude of added angular velocities based on distance from the upper right corner. This allows for a much larger solid angle of shots hitting the corner but requires matching the angular velocities, which could prove difficult.

Figure 9: A plot similar to the result plots above, but for shots on goal it displays their angular velocity (magnitude) instead of simply whether they were accurate.

5 Theoretical model

5.1 Analysis of motion in xy plane

Let us consider a spinning ball in a plane, in which angular velocity ω is perpendicular to the plane. This assumption is made because it is very difficult for a football player to give the ball rotation, direction of which significantly varies from being perpendicular to the xy plane. Another assumption that we made is that we decided to model drag force as linear function of velocity:

$$\vec{F_D} = -\frac{1}{2}c_D\rho\pi a^2|\vec{v}| \tag{4}$$

With that assumption our model is analytically solvable and this is minor deflection comparing to drag force described as dependent on squared velocity. In the figure 10 decomposition of lift force and drag force is shown.

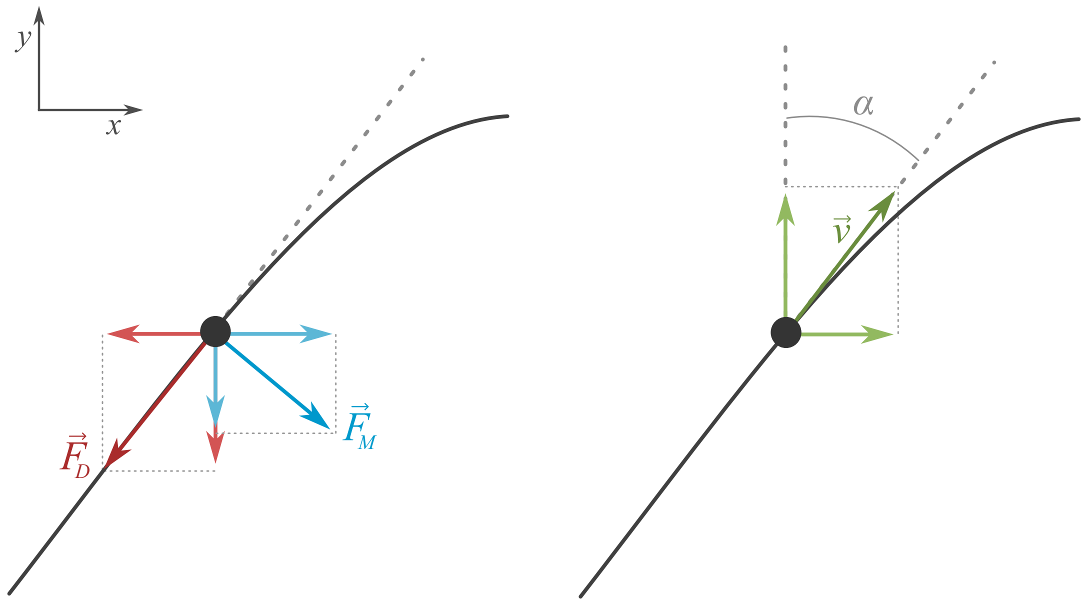


Figure 10: Decomposition of drag force and lift force in xy plane.

Simple geometrical relations are given as:

$$\begin{cases} F_{Mx} = F_M \cos\alpha = F_M \frac{v_y}{|\vec{v}|} = \pi\rho a^3 c_M |\vec{\omega}| v_y \\ F_{My} = -F_M \sin\alpha = -F_M \frac{v_x}{|\vec{v}|} = -\pi\rho a^3 c_M |\vec{\omega}| v_x \end{cases} \tag{5}$$

$$\begin{cases} F_{Dx} = F_D \sin\alpha = F_D \frac{v_x}{|\vec{v}|} = -\frac{1}{2} c_D \rho\pi a^2 v_x \\ F_{Dy} = F_D \cos\alpha = F_D \frac{v_y}{|\vec{v}|} = -\frac{1}{2} c_D \rho\pi a^2 v_y \end{cases} \tag{6}$$

Let us set $c_1 = \pi\rho a^3 c_M |\vec{\omega}|/m$ and $c_2 = \frac{1}{2} c_D \rho\pi a^2/m$. One can write down equations of motion in the xy plane:

$$\begin{cases} \ddot{x} = c_1\dot{y} - c_2\dot{x} \\ \ddot{y} = -c_1\dot{x} - c_2\dot{y} \end{cases} \tag{7}$$

For initial conditions: $\dot{y}(0) = v_0$, $\dot{x}(0) = 0$, $x(0) = y(0) = 0$ the solutions of this system of equations are given by:

$$\begin{cases} x(t) = e^{-c_2 t}(c_1 e^{c_2 t} - c_1 \cos(c_1 t) - c_2 \sin(c_1 t)) \frac{v_0}{c_1^2 + c_2^2} \\ y(t) = e^{-c_2 t}(c_2 e^{c_2 t} - c_2 \cos(c_1 t) + c_1 \sin(c_1 t)) \frac{v_0}{c_1^2 + c_2^2} \end{cases} \tag{8}$$

5.2 General equations of motion

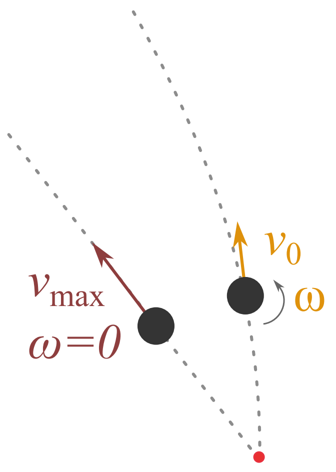
Let us discuss how equations (8) would change if we added another dimension of motion along axis z. Having angular velocity ω perpendicular to plane xy, the only connection between z axis and xy plane would be the term v_z in velocity magnitude:

$$|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{9}$$

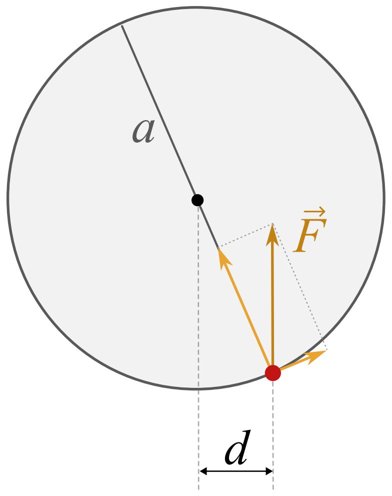
That term would contribute to formula for drag force and lift force. However, one can assume that kicking a ball during penalty kick is characterised by significant magnitude of v_y which will be much bigger than v_z in the term (9). Therefore we can omit the term v_z and notice that motion on xy plane can be considered as independent of motion along z axis. With gravitational force: $\ddot{z} = -g$ we can finally write down equations of motion in 3D:

$$\begin{cases} x(t) = e^{-c_2 t}(c_1 e^{c_2 t} - c_1 \cos(c_1 t) - c_2 \sin(c_1 t))\frac{v_0}{c_1^2+c_2^2} \\ y(t) = e^{-c_2 t}(c_2 e^{c_2 t} - c_2 \cos(c_1 t) + c_1 \sin(c_1 t))\frac{v_0}{c_1^2+c_2^2} \\ z(t) = v_{0z} t - \frac{1}{2} g t^2 \end{cases} \tag{10}$$

5.3 Initial velocity and angular velocity



(a)



(b)

Figure 11: (a) direct kick with maximal linear velocity and no rotation and an indirect one resulting with angular velocity ω.(b) Impulse Ft acting on the ball with offset d. Tangential component of this force is responsible for rotation and the radial one for linear velocity.

One can model an impact of the foot on the ball by a force acting at some angle α for time t (11). We can consider friction as negligible [1] and assume that due to elastic deformation of a ball we can describe impact in such a way. Let us call v_{max} the maximal velocity which will be result of a central kick:

$$Ft = mv_{max} \tag{11}$$

When the kick is not central the ball gains both linear velocity v_0 and angular velocity ω:

$$\begin{cases} Ft\sin\alpha = Ft\frac{d}{a} = mv_0 \\ Fat\cos\alpha = Fat\sqrt{1-\frac{d^2}{a^2}} = I\omega = \frac{2}{5}ma^2\omega \end{cases} \tag{12}$$

The result is a relation between v_0 and ω with given v_{max}:

$$\omega = \frac{5}{2}\frac{\sqrt{v_{max}^2 - v_0^2}}{a} \tag{13}$$

This dependence will let us calculate optimal initial velocity and angular velocity for the biggest deflection during flight of the ball.

6 Further calculations

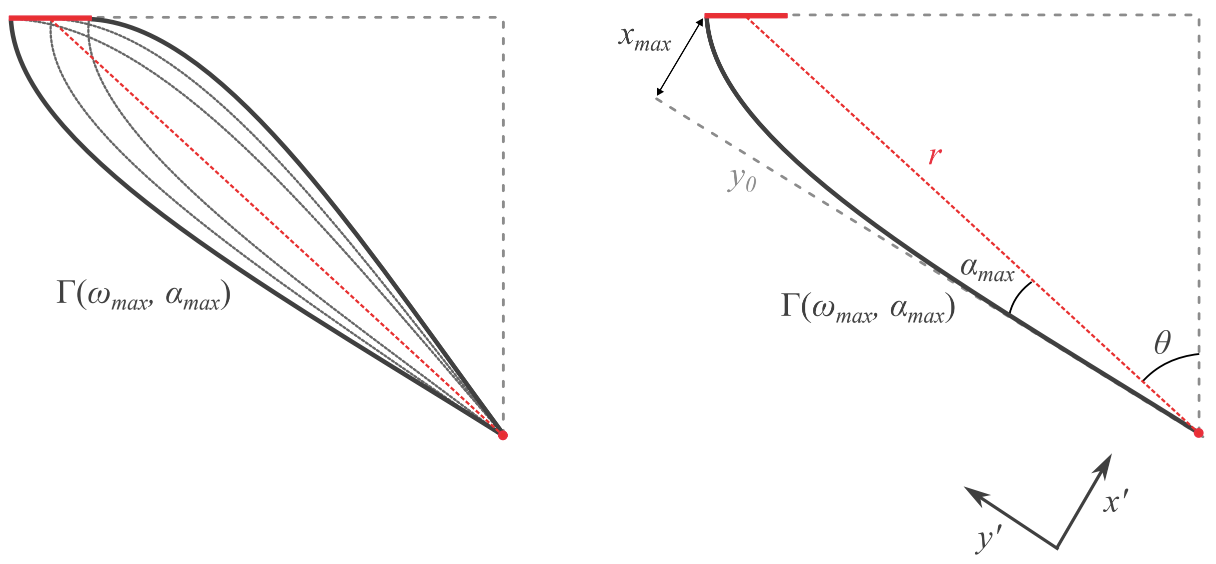
Having equations of motion and dependence between initial velocity and spin we can propose a method to determine their range. Firstly we will calculate v_0 and ω for the maximal deflection in transverse direction during flight. Then we can consider family of curves starting in one point (penalty mark) and ending in chosen area (upper corner of goal) and calculate boundaries of ranges of ω, v_0 and α.

6.1 Calculating the greatest deflection due to the spin

In the picture 12 we can see a scheme of family of curves. Our goal is to find a curve that will maximize deflection in x' direction for a given v_{max}. In the picture such a curve is labeled as Γ and symbolizes the extreme curvature of a possible trajectory of the ball. This curve corresponds to the boundary values of parameters α and ω that let us put a ball in the upper corner. To maximize deflection of a ball's trajectory one can use equations with y' axis pointed in direction marked by angle α. Let us set $\omega = \frac{2}{5}\frac{\sqrt{v_{max}^2 - v_0^2}}{a}$ in constants c_1 and c_2. For given v_{max} equations of motion $(x(t), y(t))$ now depend solely on time and initial linear velocity v_0. So the desired v_0 that maximizes curvature comes from solving the system of equations:

$$\begin{cases} x_{max}^2 + y_0^2 = r^2 \\ y'(t_0) = y_0 \\ \frac{dx'(t_0,v)}{dv} = 0 \end{cases} \tag{14}$$

In the figure 13 relation between deflection of curve in x is shown dependent on v_0 represented as fraction of v_{max}. The maximum is easily noticeable.



$\Gamma(\omega_{max}, \alpha_{max})$
x_{max}
y_0
r
α_{max}
$\Gamma(\omega_{max}, \alpha_{max})$
θ
x'
y'

Figure 12: (left) family of curves which end in upper corner, Γ is an extreme curve.(right) geometry of Γ.

That peak represents curve which is deflected by the greatest angle α and has maximum angular velocity ω. Their values are the upper boundary of whole family of curves that end in upper corner of the goal:

$$\begin{cases} \alpha_{max} = \arctan \frac{x_{max}}{y_0} \\ \omega_{max} = \frac{2}{5} \frac{\sqrt{v_{max}^2 - v^2}}{a} \end{cases} \tag{15}$$

Where x_{max}, y_0 and v are solutions of system 14. So for a given v_{max} ranges of v_0, α and ω are represented:

$$\begin{cases} v_0 \in (0, v_{max}) \\ \alpha \in (\theta - \alpha_{max}, \theta + \alpha_{max}) \\ \omega \in (0, \omega_{max}) \end{cases} \tag{16}$$

6.2 Movement along z axis

As said earlier, movement on xy plane is independent of axis z. Therefore, previous solutions should be combined with simple projectile motion along z. To find maximal and minimal initial velocity along z we considered equation of motion:

$$z(t) = v_{0z}t - \frac{1}{2}gt^2 \tag{17}$$

The time of motion is determined by a path in xy, so to find critical value we should use the shortest time of motion which is associated with the shortest path in xy - straight kick in the corner with maximal velocity v_{max} in xy. Time of the motion and critical values of v_z are solutions of system of equations:

$$\begin{cases} r = v_{max}t \\ h_{min} = v_{zmin}t - \frac{1}{2}gt^2 \\ h_{max} = v_{zmax}t - \frac{1}{2}gt^2 \end{cases} \tag{18}$$

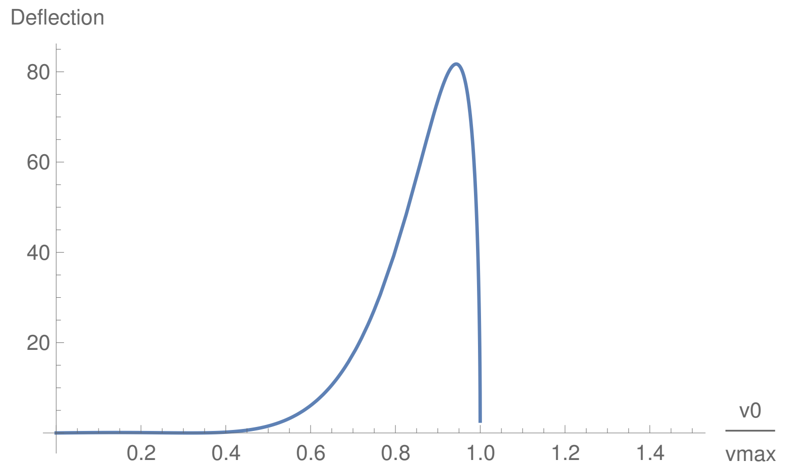


Figure 13: Dependence of x_{max} on fraction $\frac{v_0}{v_{max}}$. Maximum corresponds to extreme curve.

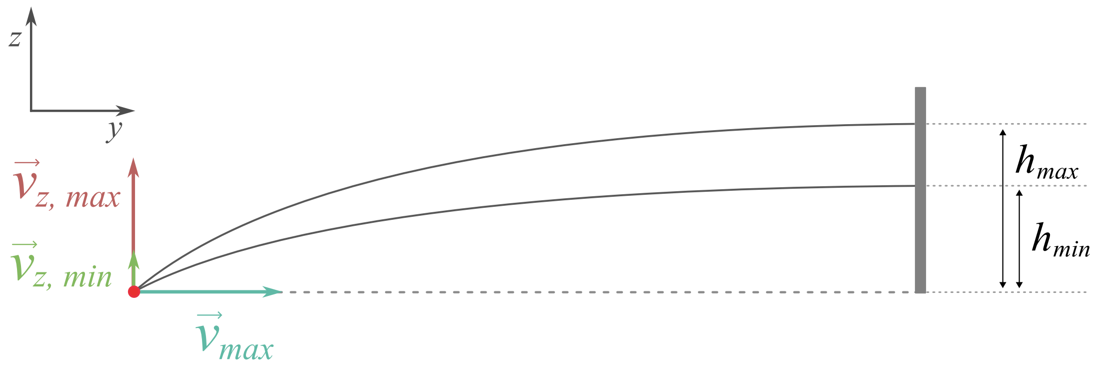


Figure 14: Geometry of simple projectile motion. Minimal and maximal trajectories.

6.3 General results

Summarizing, we can vary 4 parameters of initial conditions: velocity along z axis, velocity in xy plane determined by v_0 and α and magnitude of angular velocity which pointed in z direction due to our assumptions. We exactly know what are the boundaries of ranges of these parameters and how to calculate them. The parameter that determines all of these ranges of initial conditions is v_{max} – velocity after the direct (central) kick in the ball resulting in a straight motion in xy plane. As final results we can present all four sets of initial conditions we are interested in:

$$\begin{cases} v_0 \in (0, v_{max}) \\ \alpha \in (\theta - \alpha_{max}, \theta + \alpha_{max}) \cup (-\theta - \alpha_{max}, -\theta + \alpha_{max}) \\ \omega \in (0, \omega_{max}) \\ v_z \in (v_{z,min}, v_{z,max}) \end{cases} \tag{19}$$

Value of α can be an element of two symmetrical sets. They represent right and left corner of the goal. The values of boundaries of these sets were calculated and plotted in dependence of v_{max} 15 16 17.

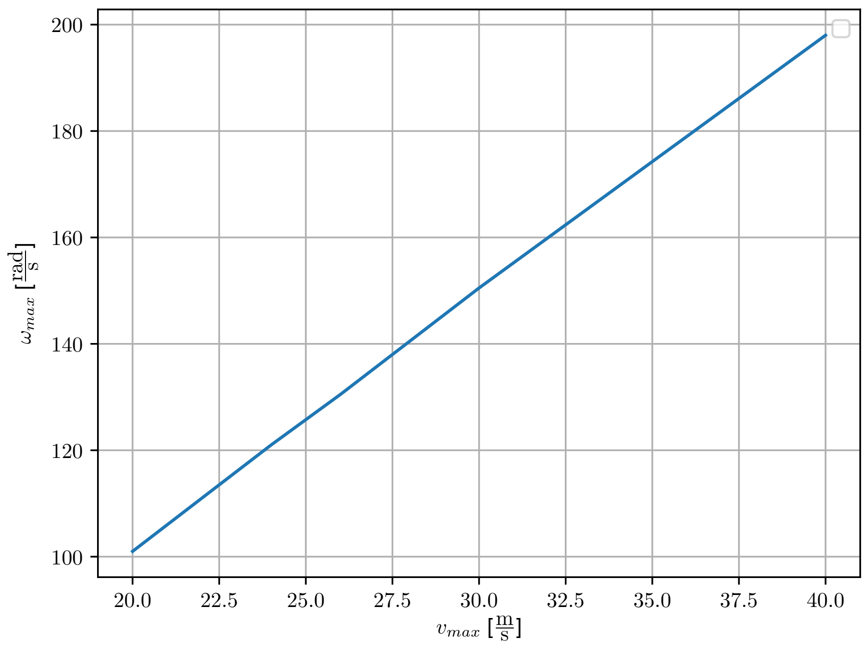


Figure 15: Upper boundary of initial ω in dependence on v_{max}.

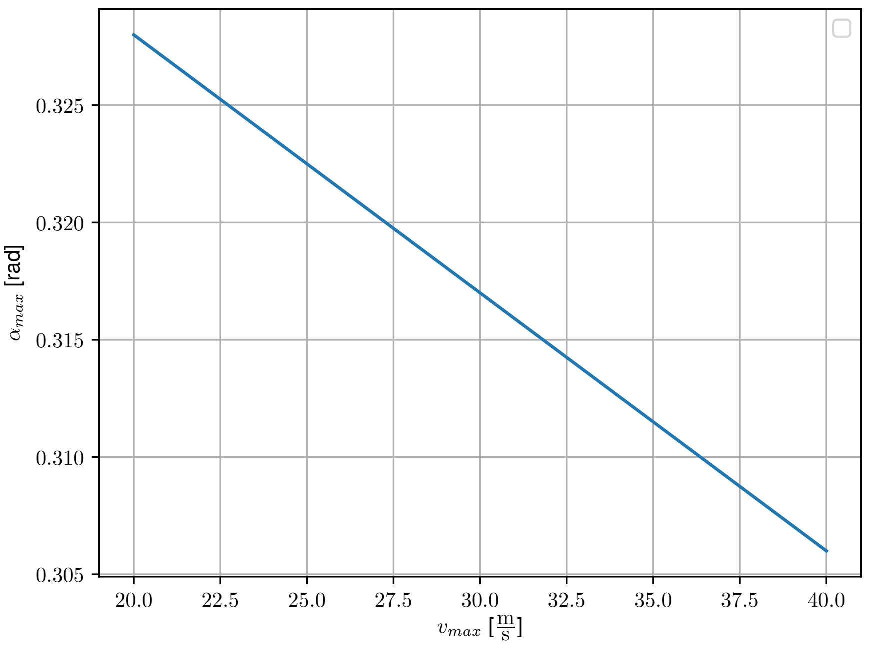


Figure 16: Maximal angle α in dependence on v_{max}.

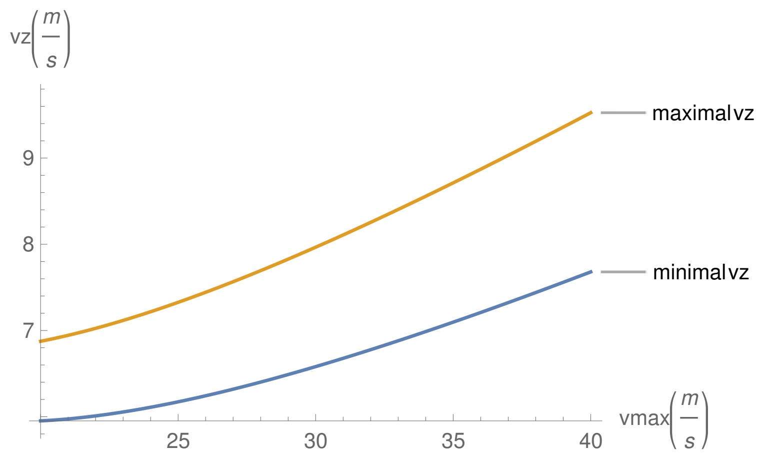


Figure 17: Maximal and minimal z component of initial velocity in dependence on v_{max}.

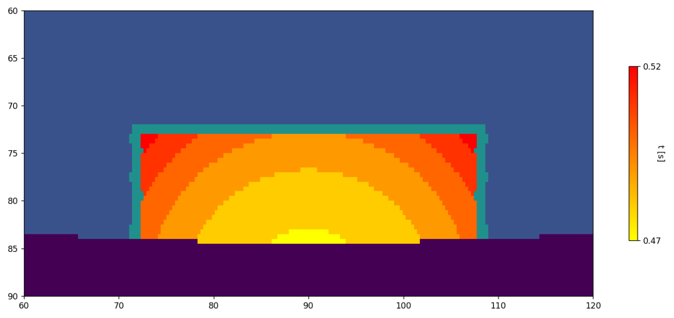
7 The goalkeeper problem

7.1 Introduction and assumptions

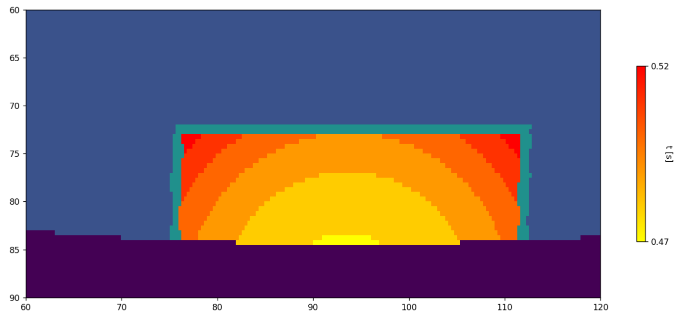
It is logical to assume, that the goalkeeper would stand in the middle of the goal to have the most highest of reaching the ball on either side. We assume he is positioned exactly at the goal line, as stated in FIFA rules. In reality goalkeeper's movements are based on predictions of where the player is going to shoot, made even before the striker approaches the ball. There is not much physics can offer to avoid it – the penalties will always be a guessing game. It is possible to beat the goalkeeper even when he correctly solves the left-right dilemma.

7.2 Measuring the difficulty of a shot

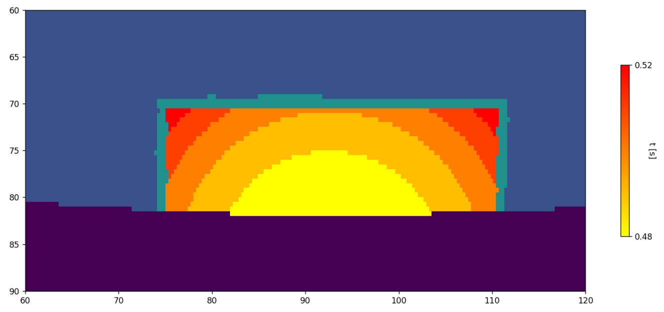
To find the best chance of avoiding the goalie, we need some measurement of how difficult it is to save the shot. One idea could be to measure the time it takes for the ball to reach the goal line. This can be done quite simply and using a plot similar to the one used before (9), we can plot t for different directions of initial velocity, but set velocity magnitude and rotation vector. The results obtained are presented on Figures 18



(a) No rotation



(b) Rotation along the z axis



(c) Rotation along the z axis and a top spin

Figure 18: Plots showing the time it takes for a shot to reach the goal. It seems that sideways rotation does not affect it much, but a top spin shortens the time.

The time is lowest in the middle, but that is also easiest to defend for the goalie. Thus, we need a way to incorporate the him into our model. We propose a simple model: assume the goalkeeper starts in the middle of the goal, and can move with an oblique throw in any direction at some velocity v.

Using known oblique throw formulae we can calculate the velocity required to reach a certain point r in a given time t. For an oblique throw from the goalkeepers position (at height z_0) to the ball's position at t, along a new vector ϱ, we can calculate the v components along ϱ and z:

$$\begin{cases} v_\varrho = \frac{\varrho}{t} \\ v_z = \frac{1}{t}(z_t - z_0 + \frac{1}{2}gt^2) \\ v(\vec{r}, t) = \sqrt{v_\varrho^2 + v_z^2} \end{cases} \tag{20}$$

Then we assume the goalkeeper can simulate the trajectory in his mind and calculate the required v for each point (for the sake of realism we assume it takes him a standard 200 ms to react). He then chooses the point with the lowest required velocity. We can then assign the goalkeeper his skill as the maximum velocity he can achieve. Thus, if the minimal required velocity is larger than his achievable, the goal goes through. To visualise the goalkeeper's thought process, see Figure 19:

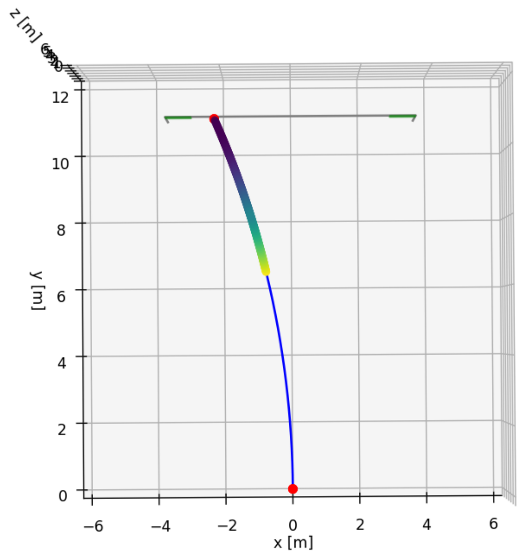
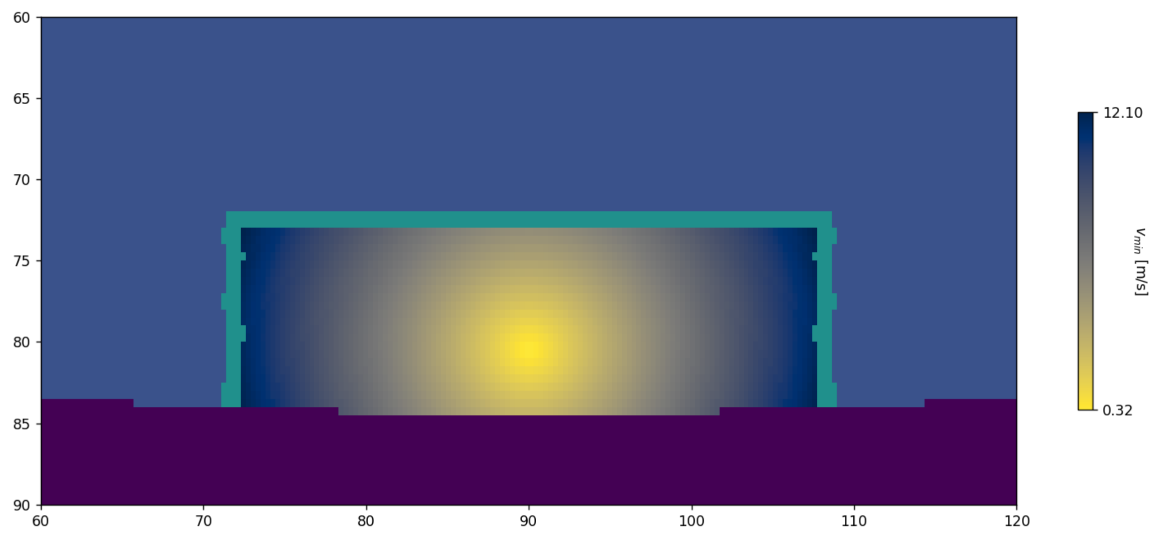
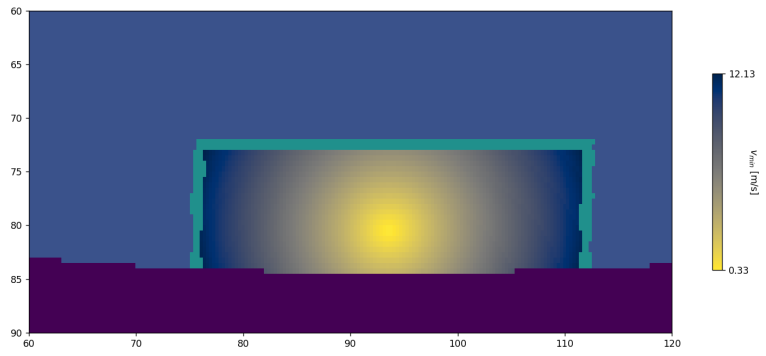


Figure 19: Trajectory of the ball and a color map of $v(\vec{r}, t)$ for its last part. It turns out the easiest point to catch the ball is at the end of the trajectory, even for a very curved path.

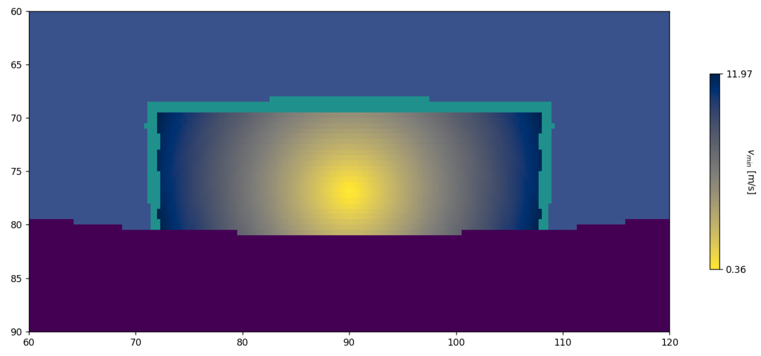
With these tools available, we can check the minimal v (marked v_{min}) of several trajectories and determine the most difficult one to save. To do this we generate a similar plot as before, but this time plotting the v_{min} as a function of ϕ and θ (for given v and $\vec{\omega}$). Several of these plots can be seen in Figure 20.



(a) The required goalkeeper velocity for shots without rotation. These results match our football knowledge: the most deadly shots are to the sides, especially when sent just under the bar. This also correctly evaluates that a bottom corner shot can be harder than a shot to the side. For a given goalkeeper, we should estimate his maximal achievable velocity and check which angles are beyond his reach.

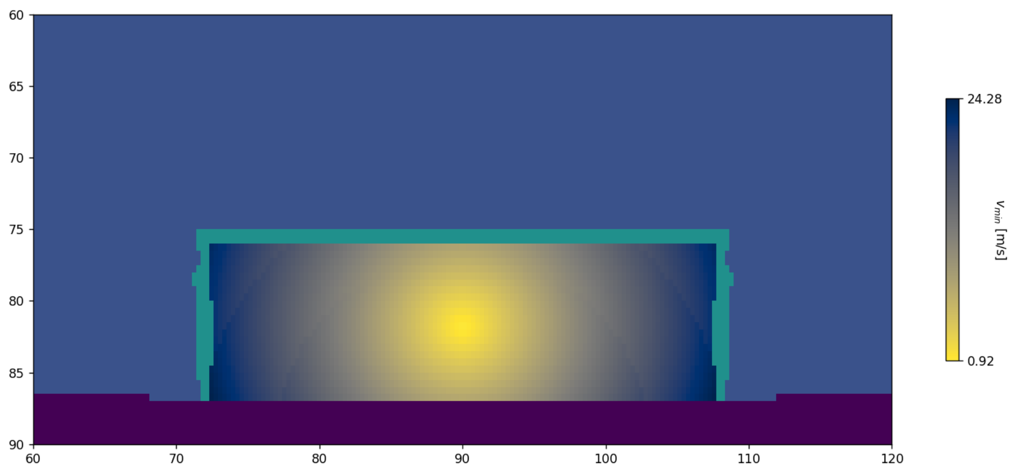


(b) For a rotation along the z axis, no major changes to the velocities are seen on the plot.

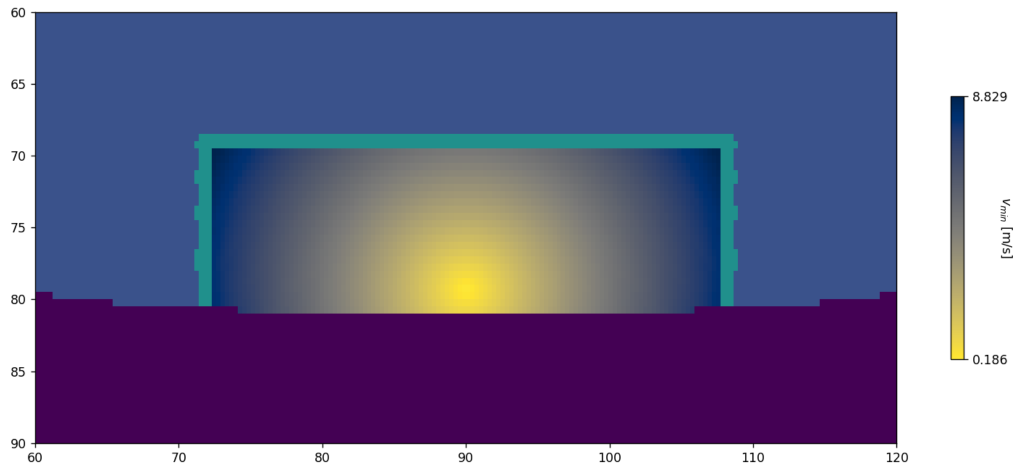


(c) Shooting the ball with a top spin changes the maximal v_{min} slightly.

Figure 20: Plots estimating the difficulty of saving a given shot (at $v = 25$ m/s), by calculating the velocity required of the goalkeeper. There is no surprise they look like simple gradient images, for this is in alignment of our current knowledge about penalties. This is an indication that our simple and intuitive model of the goalkeeper matches reality in a meaningful degree.



(a) Strong shot without rotation, $v = 35$ m/s. The scale is much higher than for the weaker shots, showing how much the strength of the shot matters. It is also apparent that for stronger shots the lower shots are more effective.



(b) Weak shot without rotation. Here the scale only reaches 8.8 m/s, which means that for a weaker shot to pass the keeper it needs to be accurate. The plot also shows that shots under the bar are more effective in comparison.

Figure 21: Plots showing goalkeeper's required velocity to save shots with different initial velocities. They show how strong the relation is between the difficulty of the shot and the magnitude of v.

7.3 Summary

In summary, the biggest impact on the difficulty of the shot comes from the ball's speed and from how far from the goalkeeper we hit. We can deduce that the rotation plays a secondary role, as long as we're dealing with a deterministic goalkeeper. It could however make an impact by confusing the keeper as to the ball's path.

This result corresponds with our solution of the first part of the problem. As a kick in the upper corner is the most difficult for the goalkeeper to save, the initial conditions that will minimize chance for that were proposed by us in chapter 6.

8 Conclusions

One of the greatest strengths of the theoretical model is the ability to compute the maximal deflection from the initial path and determine in explicit form ranges of initial conditions for a ball to reach upper corner. However this model was only simplification of governing equations and is applicable under few conditions such as fixed direction of rotation or linear drag.

The simulations offer a broad insight to the penalty kick problem. Partially, through visualization of the analysed phenomena, but also by confirming predictions of the theoretical model. The acquired plots have proven important in justifying our intuition, as well as developing the theory.

Avoiding the goalkeeper required us to make some assumptions about how we model him. We found a model that matches the real behavior and developed tools to check which shots should be the most difficult to defend. We confirmed our predictions by finding out that strong shots to the sides of the goal, especially the upper corners, have the best chance of avoiding the keeper.

9 Appendix

9.1 Wide application range of the simulation

The numerical model of the ball's dynamics has a wider range of applications than just penalty kicks. Even more, the effects of rotation are more amplified in free kicks, or perhaps other sports such as baseball.

The power of created simulation can be appreciated through reconstruction of notable football events such as the famous shot of Roberto Carlos in 1997.

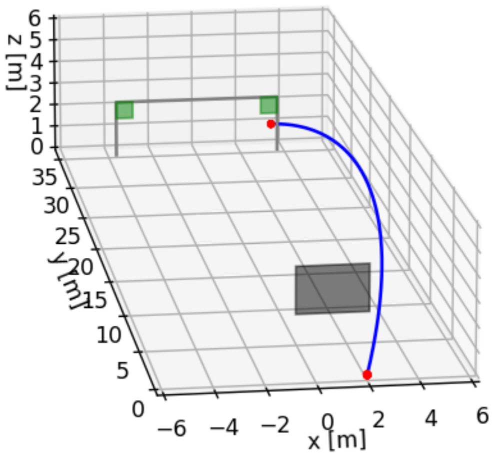


Figure 22: The famous, curved, 30 meter shot by Roberto Carlos.

Here we can see how shooting the penalty in the style of Panenka affects the time the ball takes to reach the goal (see Fig. 18):

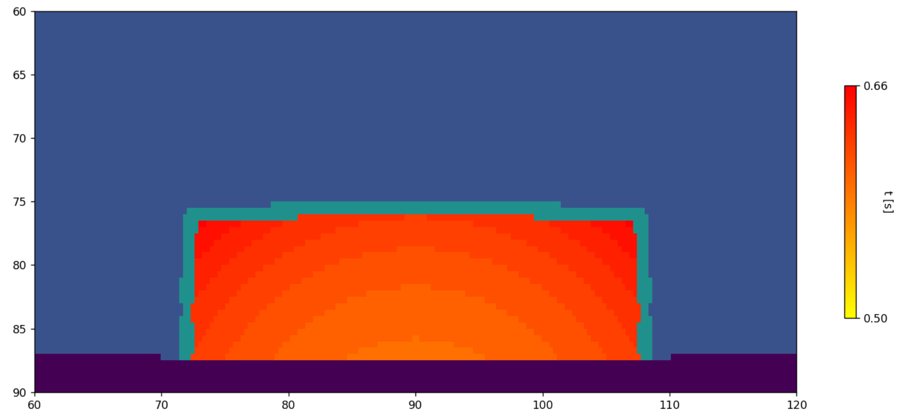


Figure 23: The slow paced 'Panenka' penalty can easily fool the goalkeeper

9.2 Python code

9.2.1 Single trajectory

```
import numpy as np, matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib.patches import Rectangle
import mpl_toolkits.mplot3d.art3d as art3d

#constants in SI units
dt = 0.01 #can be 0.001
maxt = 3
g = 9.81
m=0.4
R = 0.11
C_L = 0.3 #Lift coefficient
C1 = C_L*R**3*np.pi*1.2 #effective lift coeff.
C_d = 0.25 #drag coeff.
C2 = C_d*1.2*np.pi*R**2/2 #eff. drag
lmb = 8*np.pi*1.6e-5*R*3/(m*2) #lambda for omega dissipation,
#uses Reynolds number

goalx = 3.66 #goal size / 2
goalz = 2.44
window = 0.75 #window square size
nsteps = int(maxt/dt)


###########################################################
```

```
#EASY CHANGE:
distance = 11 #of the goal. can be changed i. e.
#for free kicks (ball starts at 0. 0.)

xoffset = 0
#initial velocity:
v0 = 20 #m/s
theta=90/180*np.pi #up to 180 deg
phi = 90/180*np.pi

o = np.array([0., 0., 70.]) #\omega in rad/s
####################################################



v = v0*np.array([np.sin(theta)*np.cos(phi),
np.sin(theta)*np.sin(phi), np.cos(theta)])

r0 = np.array([0., xoffset, R])
r=r0
rs = np.empty((nsteps, 3))
vs = np.empty((nsteps, 3))
os = np.empty((nsteps, 3))

def Fdrag(v):
    return -C2*np.linalg.norm(v)*v

def Fmagnus(v, o):
    cr7 = np.cross(o, v)
    SIUU = np.linalg.norm(cr7)
    if SIUU == 0: return 0
    return C1*np.linalg.norm(v)*np.linalg.norm(o)*cr7/SIUU

def fr(v):
    return v

def fv(v, o):
    fg = np.array([0., 0., -g])

    f = fg + Fdrag(v)/m + Fmagnus(v, o)/m
    return f
def fo(o):
    return -lmb*o

def SimRK4(r, v, o):
    #Runge Kuttas method of the 4th order
    kr1 = dt*fr(v)
    kv1 = dt*fv(v, o)
    ko1 = dt*fo(o)

    kr2 = dt*fr(v + kv1/2)
    kv2 = dt*fv(v + kv1/2, o + ko1/2)
    ko2 = dt*fo(o + ko1/2)

    kr3 = dt*fr(v + kv2/2)
    kv3 = dt*fv(v + kv2/2, o + ko2/2)
    ko3 = dt*fo(o + ko2/2)

    kr4 = dt*fr(v + kv3)
    kv4 = dt*fv(v + kv3, o + ko3)
    ko4 = dt*fo(o + ko3)

    nr = r + (kr1+2*kr2+2*kr3+kr4)/6
    nv = v + (kv1+2*kv2+2*kv3+kv4)/6
    no = o + (ko1+2*ko2+2*ko3+ko4)/6

    return nr, nv, no
def ShotEvaluator(r):
    #-2 error
    #-1 hit the ground
    # 0 hit the frame
    # 1 hit the goal but not
    # 2 hit the upper corner!!
    w=-2
    if r[1]<distance and r[2]<R:
        w = -1
    if (abs(r[0])>=goalx-R or r[2]>=goalz-R) and r[1] >= distance:
        w=0
    if abs(r[0])<goalx-R and r[1] >= distance and r[2]<goalz-R:
        w=1
    if w==1 and abs(r[0])>goalx-window+R and r[2]>goalz-window+R:
        w = 2
    if w==-2: print('Shot evaluation error')
    return w


"""
goalieZ = goalz/2
def MinimalGoalkeeperVelocity(r, t):
    t = t-0.2 #goalie reaction time
    vy = (r[2] - goalieZ +g*t**2/2)/t
    vx = np.linalg.norm(np.array([r[0], distance-r[1]]))/t
    return np.linalg.norm(np.array([vy, vx]))
goalvs = np.empty(nsteps)
mini = int(0.25/dt) #when the scatter starts
i=0
while i<nsteps and r[2] >= 0.9*R and r[1] <= distance:
    rs[i], vs[i], os[i] = r, v, o
    if i>=mini:
        goalvs[i] = MinimalGoalkeeperVelocity(r, dt*(i))
    r, v, o = SimRK4(r, v, o)
    i+=1

goalvs=goalvs[mini:i-1]
rs, vs, os = rs[:i-1], vs[:i-1], os[:i-1]
rT = rs.T
ax = plt.axes(projection='3d')
ax.plot3D(rT[0][:mini], rT[1][:mini], rT[2][:mini], 'blue')
ax.scatter(rT[0][mini:], rT[1][mini:], rT[2][mini:],
c = plt.cm.viridis(goalvs/np.amax(goalvs))) """
#^use this to emulate a goalkeeper


########
i=0
while i<nsteps and r[2] >= 0.99*R and r[1] <= distance:
    rs[i], vs[i], os[i] = r, v, o
    r, v, o = SimRK4(r, v, o)
    i+=1


rs, vs, os = rs[:i-1], vs[:i-1], os[:i-1]
rT = rs.T
ax = plt.axes(projection='3d')
ax.plot3D(rT[0], rT[1], rT[2], 'blue')

##########
GOAL = np.array([[-goalx, distance, 0.], [-goalx, distance, goalz],
[goalx, distance, goalz], [goalx, distance, 0.]])
GOAL = GOAL.T
ax.plot3D(GOAL[0], GOAL[1], GOAL[2], 'gray')
ax.set_xlabel('x [m]')
ax.set_ylabel('y [m]')
ax.set_zlabel('z [m]')
########### Here you can expand the viewing box, but adjust the aspect!
```

```
BOX = 12
ax.set_xlim3d([-BOX/2, BOX/2])
ax.set_ylim3d([0, BOX])
ax.set_zlim3d([0, BOX/2])
ax.set_box_aspect(aspect = (1,1,0.5))


p = Rectangle((-goalx, goalz-window),
window, window, color='green', alpha=0.5)
ax.add_patch(p)
art3d.pathpatch_2d_to_3d(p, z=distance, zdir="y")

p = Rectangle((goalx-window, goalz-window),
window, window, color='green', alpha=0.5)
ax.add_patch(p)
art3d.pathpatch_2d_to_3d(p, z=distance, zdir="y")


##### Here you can add a wall
# p = Rectangle((r0[0]-1.5, r0[2]), 3, 2,
# color='black', alpha=0.5)
# ax.add_patch(p)
# art3d.pathpatch_2d_to_3d(p, z=8.5, zdir="y")

u, v = np.mgrid[0:2*np.pi:20j, 0:np.pi:10j]
x = np.cos(u)*np.sin(v)*R+r0[0]
y = np.sin(u)*np.sin(v)*R+r0[1]
z = np.cos(v)*R+R
ax.plot_wireframe(x, y, z, color="r")

x = np.cos(u)*np.sin(v)*R+r[0]
y = np.sin(u)*np.sin(v)*R+r[1]
z = np.cos(v)*R+r[2]
ax.plot_wireframe(x, y, z, color="r")

plt.show()
```

9.2.2 Multiple trajectiories

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

#constants in SI units
dt = 0.01 #can be 0.001
maxt = 3
g = 9.81
m=0.4
```

```
R = 0.11
C_L = 0.3 #Lift coefficient
C1 = C_L*R**3*np.pi*1.2 #effective lift coeff.
C_d = 0.25 #drag coeff.
C2 = C_d*1.2*np.pi*R**2/2 #eff. drag
lmb = 8*np.pi*1.6e-5*R*3/(m*2)
#lambda for omega dissipation, uses Reynolds number

goalx = 3.66 #goal size / 2
goalz = 2.44
window = 0.75 #window square size
nsteps = int(maxt/dt)
r0 = np.array([0., 0., R])
distance=11

def Fdrag(v):
    return -C2*np.linalg.norm(v)*v

def Fmagnus(v, o):
    cr7 = np.cross(o, v)
    SIUU = np.linalg.norm(cr7)
    if SIUU == 0: return 0
    return C1*np.linalg.norm(v)*np.linalg.norm(o)*cr7/SIUU

def fr(v):
    return v
def fv(v, o):
    fg = np.array([0., 0., -g])

    f = fg + Fdrag(v)/m + Fmagnus(v, o)/m
    return f
def fo(o):
    return -lmb*o

def SimRK4(r, v, o):
    kr1 = dt*fr(v)
    kv1 = dt*fv(v, o)
    ko1 = dt*fo(o)

    kr2 = dt*fr(v + kv1/2)
    kv2 = dt*fv(v + kv1/2, o + ko1/2)
    ko2 = dt*fo(o + ko1/2)

    kr3 = dt*fr(v + kv2/2)
    kv3 = dt*fv(v + kv2/2, o + ko2/2)
    ko3 = dt*fo(o + ko2/2)

    kr4 = dt*fr(v + kv3)
    kv4 = dt*fv(v + kv3, o + ko3)
    ko4 = dt*fo(o + ko3)

    nr = r + (kr1+2*kr2+2*kr3+kr4)/6
    nv = v + (kv1+2*kv2+2*kv3+kv4)/6
    no = o + (ko1+2*ko2+2*ko3+ko4)/6

    return nr, nv, no

def ShotEvaluator(r):
    #-2 error
    #-1 hit the ground
    # 0 hit the frame
    # 1 hit the goal but not
    # 2 hit the upper corner!!
    w=-2
    if r[1]<distance and r[2]<R:
        w = -1
    if (abs(r[0])>=goalx-R or r[2]>=goalz-R) and r[1] >= distance:
        w=0
    if abs(r[0])<goalx-R and r[1] >= distance and r[2]<goalz-R:
        w=1
    if w==1 and abs(r[0])>goalx-window+R and r[2]>goalz-window+R:
        w = 2
    if w==-2: print('Shot evaluation error')
    return w

def Shoot(r0, v0, o0):
    #performs one simulation and returns final position
    r, v, o = r0, v0, o0
    i=0
    while i<nsteps and r[2] >= R and r[1] <= 11:
        r, v, o = SimRK4(r, v, o)
        i+=1
        if(i>=nsteps): print("time out")
    return r

#this can be used to emulate a goalkeeper
goalieZ = goalz/2
mini = int(0.25/dt)
def MinimalGoalkeeperVelocity(r, t):
    t = t-0.2 #goalie reaction time
    vy = (r[2] - goalieZ +g*t**2/2)/t
    vx = np.linalg.norm(np.array([r[0], distance-r[1]]))/t
    return np.linalg.norm(np.array([vy, vx]))

def Shoot2(r0, v0, o0):
    r, v, o = r0, v0, o0
    i=0
    minv = 1000
    while i<nsteps and r[2] >= R and r[1] <= 11:
        r, v, o = SimRK4(r, v, o)
        if i>=mini:
            a=MinimalGoalkeeperVelocity(r, dt*i)
            if a < minv:
                minv = a #finds minimal velocity
                #in the entire trajectory,
                #which can be useful for placing
                #goalie outside of the goal
        i+=1
        if(i>=nsteps): print("time out")
    return r, minv

#parameters for the results plot


precision = 4 #int

"""
o0 = np.array([0., 0., 0.])
v_power = 25



numphis = 50*precision
LEFT_phi = 60#deg
RIGHT_phi = 120
UP_theta = 60
DOWN_theta = 90
phis = np.linspace(LEFT_phi/180*np.pi,
RIGHT_phi/180*np.pi, numphis)[::-1]
numthetas = 60
thetas = np.linspace(UP_theta/180*np.pi,
DOWN_theta/180*np.pi, numthetas)
results = np.empty((numphis, numthetas))

correction = np.empty((numphis, numthetas, 3))
```

```
def corvector(r):
    r2 = r
    r2[1]=0
    v1 = np.array([0., 0., goalz/2])-r2
    #v1 = np.array([goalx-window/2, 0., goalz-window/2])-r2
    #u = np.array([np.cos(phi)*np.sin(theta),
    # np.sin(phi)*np.sin(theta), np.cos(theta)])
    u = np.array([0., 1., 0.])
    v2 = np.cross(u, v1)
    v2 = v2/np.linalg.norm(v2)

    return v2*(1-np.exp(-np.linalg.norm(v1)**2/window**2))
"""
#above can be used for correcting the trajectory



v_power = 35
o0 = np.array([0., 0., 0.])

r0 = np.array([0., 0., R])
numphis = 50*precision
LEFT_phi = 60#deg
RIGHT_phi = 120
UP_theta = 60
DOWN_theta = 90
phis = np.linspace(LEFT_phi/180*np.pi,
RIGHT_phi/180*np.pi, numphis)[::-1]
numthetas = 60
thetas = np.linspace(UP_theta/180*np.pi,
DOWN_theta/180*np.pi, numthetas)
results = np.empty((numphis, numthetas))
results2 = np.zeros((numphis, numthetas))
#results 2 can be anything
alphas = np.zeros((numphis, numthetas))

OMEGA = 70 #if you want to add corrections
baset=5 #this has to be something that
#doesnt disturb the cbar
for i in range(numphis):
    for j in range(numthetas):
        phi = phis[i]
        theta = thetas[j]
        v0 = v_power*np.array([np.cos(phi)*np.sin(theta),
        np.sin(phi)*np.sin(theta), np.cos(theta)])
        r, t = Shoot2(r0, v0, o0)
```

```
            w = ShotEvaluator(r)
            results[i][j] = w
            if w==-2:
                t=baset
            elif w==-1:
                t=baset
            elif w==0:
                t=baset
            elif w>0:
                alphas[i][j] = 1
                #we make the results which is the
                # base for the results plot
                #and then overwrite it with more data,
                # like the time of the shot
            results2[i][j] = t

fig = plt.figure(frameon=False)
im = plt.imshow(results.T)
im.set_extent(np.array([LEFT_phi,
RIGHT_phi, DOWN_theta, UP_theta]))

ticks=[np.amin(results2), np.amax(results2)]
aspect=5

im2 = plt.imshow(results2.T)
im2.set_cmap('cividis_r')
im2.set_alpha(alphas.T)

im2.set_extent(np.array([LEFT_phi, RIGHT_phi,
DOWN_theta, UP_theta]))
cbar = plt.colorbar(im2, shrink=0.5, ticks=ticks)
cbar.set_label(r'$v_{min}$ [m/s]', rotation=270)
plt.show()
```

References

[1] Z. Zhu, B. Chen, S. Qiu, R. Wang and X. Qiu, *Simulation and Modeling of Free Kicks in Football Games and Analysis on Assisted Training*

[2] T. Asai, M. J. Carre, T. Akatsuka and S. J. Haake, *The curve kick of a football I: impact with the foot*

[3] M. J. Carre, T. Asai, T. Akatsuka and S. J. Haake, *The curve kick of a football II: flight through the air*

[4] T. Asai, K. Seo, O. Kobayashi and R. Sakashita, *Fundamental aerodynamics of the soccer ball*

[5] NASA, *Drag on a soccer ball* – `https://www.grc.nasa.gov/www/k-12/airplane/socdrag.html`

[6] G.Irenson, *Beckham as physicist?*

[7] Kamalu J. Beamer, *Projectile Motion Using Runge-Kutta Methods*

[8] Official FIFA website
https://www.fifa.com/