Hanze Hogeschool

Bio-informatica

Eindopdracht

Muniru, Sam & Bert

Supervised by

Dr. Supervisor's Name

October 9, 2024

Contents

Chapter 1

Introduction

Ever heard of the game FIFA by EA? It is the most popular football game in the world played by millions. So to get the correct trajectory for the ball is one of the most important aspects of the game. If the trajectory is not realistic it wouldn't be as good of a game as it is now. One of the ways to get the correct trajectory is calculating it using a model with different forces acting on the ball. Think of things like speed, gravity, the magnus effect and air resistance. By accurately modelling these forces, we can predict the path of a football. This is essential for things like video games but also for predicting paths of different things or projectiles.

So in this report the aim is to provide a model that models the trajectory of a football. This will be done using different physics formulas modelling the forces at play like speed, gravity, magnus effect and air resistance. Furthermore the specific parameters that must be chosen to accurately model these forces will be examined and explained.

With all this in mind the question is, how can the trajectory of a football be accurately modeled using the different forces at play?

1.1 Goal

As you might have guessed the goal is to model the trajectory of a football that is kicked by someone. This is done by calculating the different forces at play on any given moment. The forces chosen to model are velocity, acceleration and the Magnus force. With these three the ball's position can be calculated for any given moment. The formulas for these three will be explained further in the theory section. In the results different starting parameters will be used to showcase multiple paths of a football using tools like Plotly to visualize the trajectories. The expectation is that this model will show a realistic trajectory of a football considering the effects of the different forces.

1.2 Theory

To understand this model a little knowledge about basic physics can be useful, here is a small explanation about the forces. So everywhere around you there are different forces at play. One of the important forces is the air resistance experienced with most outside activities like running, cycling and even walking. For example with running you have to use energy to move forward, this is translated to force by your muscles. While moving you push the air out of the way and to do so the force you generate must be higher than the air resistance. If it is windy outside it takes more energy to move the same distance because the air exerts a higher force on you. So lets take a look at al the forces on a football. Air resistance is one of them of course, but also gravity and the Magnus force. All the forces on the ball are combined in different formulas. Velocity, Acceleration and a formula for the Magnus force. These three are split in a X-, Y-, Z-axis to calculate the forces in a 3D plane.

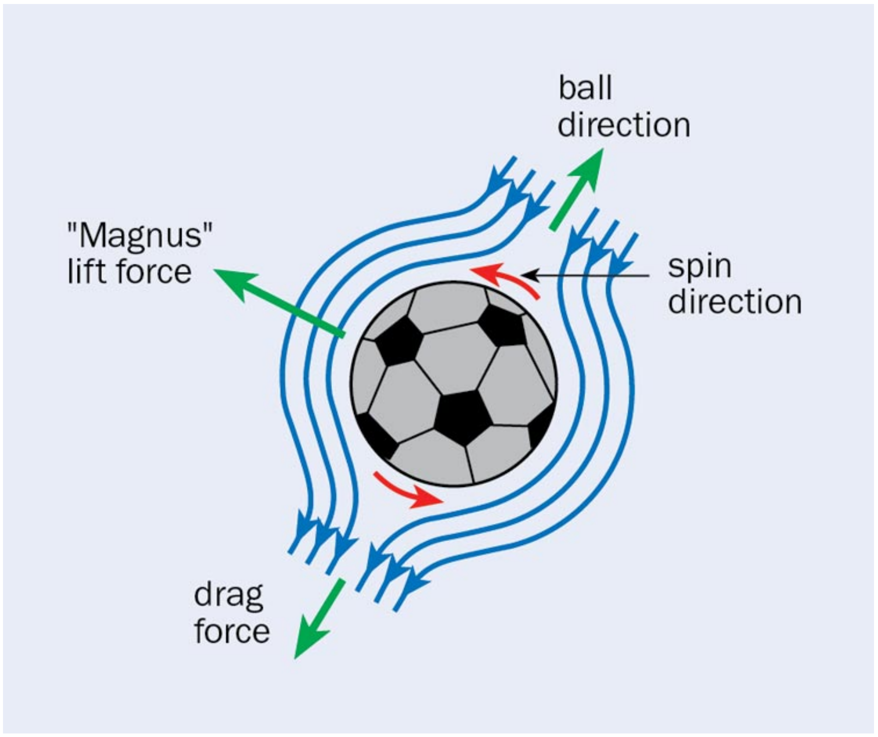

Figure 1.1: An image showing the different forces on a football (#fig:ball picture)

1.2.1 Magnus Force Implimentation

There are multiple forces that are interaction with the ball when u kick it. Things like gravity, acceleration and air resistance. But since the ball has a spin the magnus force comes in to play as well. This force deflects the ball in flight [2]. The formula for the magnitude of the acceleration created by magnus force is the following.

$$\overrightarrow{f} = \frac{8}{3}\pi\rho r^3 \cdot \overrightarrow{\omega} \cdot \overrightarrow{V}$$

The constant:

$$\lambda = \frac{8}{3}\pi\rho r^3$$

Equals:

$$\overrightarrow{f} = \lambda \cdot \begin{vmatrix} \overrightarrow{i} & \overrightarrow{j} & \overrightarrow{k} \\ \omega_x & \omega_y & \omega_z \\ V_x & V_y & V_z \end{vmatrix}$$

The ω is the spin on the ball when kicked aka the angular velocity. The r stands for radius and ρ is the density of the ball. V stands for the velocity relative to the surrounding air. In the model we will implement the velocity for the general displacement of the ball in like the formula below.

$$\frac{\delta V}{\delta t} = \sqrt{velocity_x^2 + velocity_y^2 + velocity_z^2}$$

Now we solve the magnus displacement for each axis, first for x:

$$\frac{\delta \lambda_x}{\delta t} = \lambda \cdot (\omega_y \cdot V_z - \omega_z \cdot V_y) \cdot e^- \mu$$

Now for y:

$$\frac{\delta \lambda_y}{\delta t} = \lambda \cdot (\omega_y \cdot V_z - \omega_z \cdot V_x) \cdot e^- \mu$$

And for z:

$$\frac{\delta \lambda_z}{\delta t} = \lambda \cdot (\omega_y \cdot V_x - \omega_x \cdot V_y) \cdot e^- \mu$$

1.2.2 Acceleration

The speed of the ball slows down over time due to all acting forces on it. These forces are magnus force, air resistance and gravity. Now to add them all together. First for the vertical axis:

$$\frac{\delta a_z}{\delta t} = \frac{-k \cdot V_z \cdot V + \lambda_z - m \cdot g}{m}$$

Here, k represents the air resistance, v_z the upward speed, V is the total speed on the ball, m is the total mass , g is the force of gravity and λ_z is the magnus.

Second the horizantal axis:

$$\frac{\delta a_x}{\delta t} = \frac{-k \cdot V_x \cdot V + \lambda_x}{m}$$

Here, V_x is the horizontal speed and λ_x is the horizontal magnus force.

Third is the sideways direction:

$$\frac{\delta a_y}{\delta t} = \frac{-k \cdot V_y \cdot V + \lambda_y}{m}$$

Here, V_y is the sideways speed and λ_y is the magnus force in the sideways direction.

Chapter 2

Methods

2.1 The software model

To achieve the goal of accurately modelling the path of a football, a combination of mathematical formulas, computational and visualization tools are utilized. The main piece of software that ties all of this together is R (version 4.4.1). R is a powerful tool for statistical computing and visualizing this in all sorts of graphs. Additionally a few extra R libraries are imported and used:

- deSolve (version 1.40) The deSolve package provides a set of functions for solving ordinary differential equations (ODEs) and partial differential equations (PDEs) in R. It is commonly used in fields such as biology, chemistry, physics, and engineering to model and simulate dynamic systems.
- Pracma (version 2.4.4) This package provides R implementations of more advanced functions in numerical analysis, with a special view on on optimization and time series routines. Uses Matlab/Octave function names where appropriate to simplify porting. Some of these implementations are the result of courses on Scientific Computing ("Wissenschaftliches Rechnen") and are mostly intended to demonstrate how to implement certain algorithms in R/S. Others are implementations of algorithms found in textbooks.
- Plotly (version 4.10.4) Plotly's R graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, and 3D (WebGL based) charts.

The deSolve library is used the solve the multiple differential equations in the model. These are mainly used to calculate the acceleration, velocity and position based on the different forces acting on the football. With a function in the deSolve library the model is run over a defined time and stored in a data frame. To visualize all the positions of the ball the library Plotly is used. This is a library to make interactive plots in R.

2.2 Model configuration

The model uses a lot of different parameters and constants as seen in the multiple formulas and the table below. Every parameter has a initial value and the constants have a constant value. In the table below every one of those is shown with a small description and the initial or constant value. The parameters used to simulate a still football being kicked are v, angle.z, angle.y, angle.x and state_straight. These parameters are used as a start point in the model, otherwise the ball will not move.

Most of these parameters are self-explanatory and chosen based on typical physical characteristics of a football. And the others are mostly some constants and coefficients that are needed for the formulas. These are fixed and always the same, so these can be found online when searching for the formulas and the constants. All the parameters as well as the starting values are entered in the piece of R code shown in the attachments These can later be used in the code that functions as the model, which will be discussed in the results.

Variable/Parameter	Description	Value/Formula
r	Radius of the football	0.11 m
rho	Air density	1.20 kg/m^3
cw	Drag coefficient of the football	0.5
cm	Magnus coefficient	1
m	Mass of the football	0.445 kg
k	Drag force coefficient	$\frac{1}{2} \cdot cw \cdot \rho \cdot (\pi \cdot r^2)$
mu	Friction coefficient	0.1
g	Gravitational acceleration	9.81 m/s^2
km	Another coefficient	$cm \cdot 8 \cdot \rho \cdot r^3$
f	Force	10 N
lambda	Magnus constant	$\frac{8}{3}\pi \cdot r^3$
e	Euler's number	$e \approx 2.71828$
v	Initial velocity of the football	30
angle.z	Initial angle of the football's trajectory (z-axis)	25 degrees
angle.y	Initial angle of the football's trajectory (y-axis)	30 degrees
angle.x	Initial angle of the football's trajectory (x-axis)	7 degrees
state_straight	Initial state of the football	See code
times	Time steps	0 to 10 with step size 0.01

Chapter 3

Results

3.1 The Model

In this section, the results of the simulation are shown. The model shows the trajectory of a football based on a calculation taking different forces, like drag, Magnus and gravitational forces into account.

In the piece of code below the formulas of these different forces are used to form the model. These are put in a R function simply named model. This function is used with the deSolve library to solve the differential equations and calculate the acceleration, velocity and angular velocity of the football on every time step. With these parameters every possible position can be calculated.

```r
# Formulas in a function named model
model <- function(t, y, parms){
  with(as.list(c(y, parms)), {
    # velocity & positioning
    velocity <- c(velocity.x, velocity.y, velocity.z)
    position <- c(position.x, position.y, position.z)
    angular <- c(angular.x, angular.y, angular.z)

    # Model for acceleration
    V <- sqrt(velocity.x^2 + velocity.y^2 + velocity.z^2)

    # Magnus force components
    lambda.z <- lambda *
      (angular.y*velocity.x- angular.x * velocity.y) * e^-mu
    lambda.x <- lambda *
      (angular.y*velocity.z- angular.z * velocity.y) * e^-mu
    lambda.y <- lambda *
```

```
      (angular.x*velocity.z- angular.z * velocity.x) * e^-mu

    # Accelerations
    az <- (-k * velocity.z * V + lambda *
             (angular.y*velocity.x- angular.x * velocity.y)
           * e^-mu - m * g) / m
    ax <- (-k * velocity.x * V + lambda *
             (angular.y*velocity.z- angular.z * velocity.y)
           * e^-mu) / m
    ay <- (-k * velocity.y * V  + lambda *
             (angular.x*velocity.z- angular.z * velocity.x)
           * e^-mu) / m

    acc <- c(ax, ay, az)

    # Return the derivatives
    return(list(c(acc, velocity, angular)))
  }
  )
}
```

The code chunk below will run the model.

```r
# performing simulation
out <- as.data.frame(ode(times = times,
                         y = state_straight,
                         parms = parameters,
                         func = model))

# filtering results
output <- subset(out, position.z >= 0)
```

3.2 Visualization

The code below will produce the figure that will produce a visualization of the model from three perspectives.

```r
# set up a 2x2 grid
par(mfrow = c(2, 2))

# 2 dimensional fig of trajectory from side
plot(output$position.y,
     output$position.z,
     type = "l",
     ylab = "< Down | Up >",
     xlab = "< Player | Goal >",
     main = "Trajectory from the side")

segments(x0 = 30, y0 = min(output$position.z),
         x1 = 30, y1 = 2.44, col = "red")

# 2 dimensional fig of trajectory
plot(output$position.x,
     output$position.z,
     xlim = c(-6, 6),
     type = "l",
     ylab = "< Down | Up >",
     xlab = "< Left | Right >",
     main = "Trajectory from the front")
# first line from (-3.66, 0) to (-3.66, 2.44)
segments(x0 = -3.66,
         y0 = 0,
         x1 = -3.66,
         y1 = 2.44,
         col = "red")
```

```r
# second line from (-3.66, 2.44) to (3.66, 2.44)
segments(x0 = -3.66,
         y0 = 2.44,
         x1 = 3.66,
         y1 = 2.44,
         col = "red")

# third line from (3.66, 2.44) to (3.66, 0)
segments(x0 = 3.66,
         y0 = 2.44,
         x1 = 3.66,
         y1 = 0,
         col = "red")

# 2 dimensional fig of trajectory
plot(x = output$position.x,
     y = output$position.y,
     xlim = c(-6, 6),
     type = "l",
     ylab = "< Player | Goal >",
     xlab = "< Left | Right >",
     main = "Trajectory from above")
 # first line from (-3.66, 30) to (3.66, 30)
segments(x0 = -3.66,
         y0 = 30,
         x1 = 3.66,
         y1 = 30,
         col = "red")
```

In this figure 3.1, three perspectives of the trajectory are shown: the side, front, and above perspectives. A goal is lined out in red, standing 30 meters from the shot. In the perspectives form the front and above, it is visible that that the ball begins by traveling in a straight line before skewing to the left. This is the Magnus effect.

```r
knitr::include_graphics(c('../images/3d_plot.png', "../images/3d_plot2.png"))
```

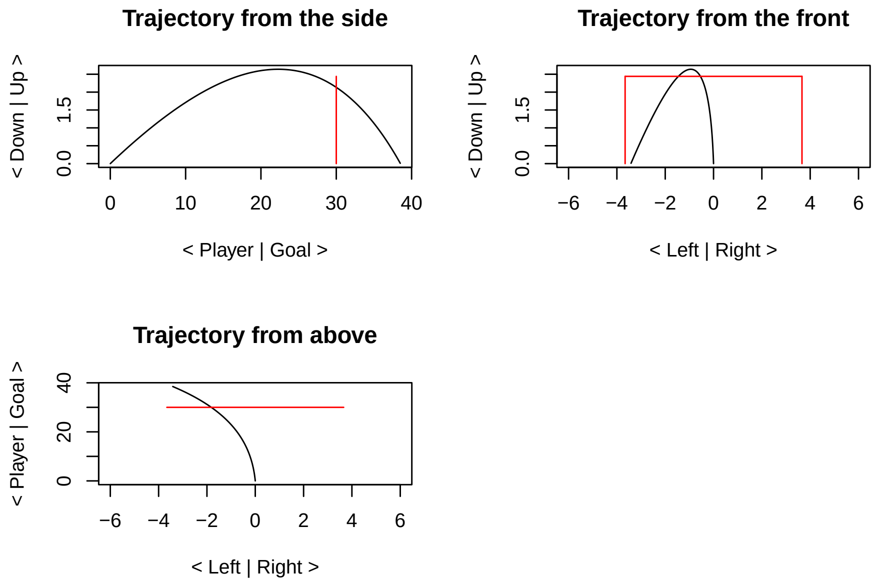

Figure 3.1: The trajectory of a football in the x, y and z space

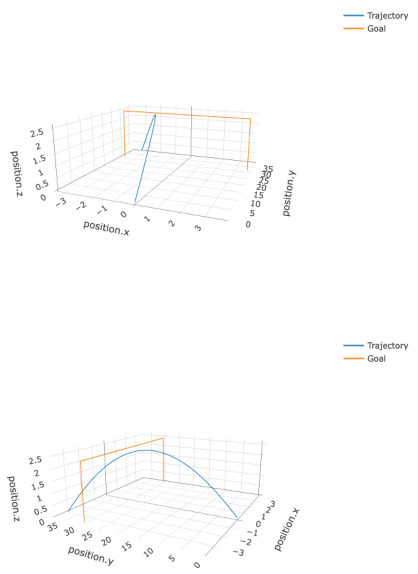

Figure 3.2: A 3d plot showing the same trajectory as seen in figure 3.1

Chapter 4

Discussion and Conclusion

4.1 Discussion

The original paper used for this project was confusing, and there were uncertainties in the math, with powers added without explanation. Our powers and coefficients differ from those used in the paper. To deal with this was a challenge to say the least. Nevertheless, the model we presented still shows a realistic trajectory of the football, accounting for different forces such as drag, gravity,and the Magnus effect. Improvements could still be made, such as refining the constant we use for the Magnus force. Overall, we consider this a somewhat valid model that functions as we expected. As is visible in Figure 1.

4.2 General conclusion and perspective

Our goal aimed at realistically simulating the trajectory of a football when kicked. With all forces acting on the ball it was quite the challenge, as mentioned before. Overall the end result is a success. We can produce a realistic trajectory, without a bounce, in 3d space. In the future we could expand op on our project we could add a bounce. Furthering the capabilities of our model. Making the model not only suited for simulating a shot but also include passes.

Bibliography

[1] Soetaert, K., Petzoldt, T., and Woodrow Setzer, R.: *Solving in R: package deSolve*, J. Stat. Softw., 33, 1-25, 2010.

[2] Che,Y., Abo Keir,M.(2022). *Study on the training model of football movement trajectory drop point based on fractional differential equation. Applied Mathematics and Nonlinear Sciences,7(1) 425-430.* https://doi.org/10.2478/amns.2021.2.00095.

[3] Natuurkunde.nl, *De vrije schop van David Beckham*, 2010. Available at: https://www.natuurkunde.nl/artikelen/807/de-vrije-schop-van-david-beckham.

Chapter 5

Attachments

```r
# variables & parameters
r <- 0.11
rho <- 1.20
cw <- 0.5
cm <- 1
parameters <- c(m = 0.445, k = (1/2) * cw * rho * (pi * r^2),
                mu = 0.1, g = 9.81,
                km = cm * 8 * rho * r^3, f = 10,
                lambda =  8/3*pi*r^3, e = exp(1))

# state of start
v <- 30
angle.z <- 25
angle.y <- 30
angle.x <- 7
state_straight <- c(velocity = c(x =6,
                                 y = 37.5,
                                 z = 11.5),
                    position = c(x = 0, y = 0, z = 0),
                    angular = c(x = 2,
                                y = -2,
                                z = 14))

# time steps
times <- seq(0, 10, by = 0.01)

# model
model <- function(t, y, parms){
  with(as.list(c(y, parms)), {
    # velocity & positioning
    velocity <- c(velocity.x, velocity.y, velocity.z)
    position <- c(position.x, position.y, position.z)
    angular <- c(angular.x, angular.y, angular.z)

    # Model for acceleration
    V <- sqrt(velocity.x^2 + velocity.y^2 + velocity.z^2)

    # Magnus force
    lambda.z <- lambda *
      (angular.y*velocity.x- angular.x * velocity.y) * e^-mu
    lambda.x <- lambda *
      (angular.y*velocity.z- angular.z * velocity.y) * e^-mu
    lambda.y <- lambda *
      (angular.x*velocity.z- angular.z * velocity.x) * e^-mu

    az <- (-k * velocity.z * V + lambda *
             (angular.y*velocity.x- angular.x * velocity.y)
           * e^-mu - m * g) / m
    ax <- (-k * velocity.x * V + lambda *
             (angular.y*velocity.z- angular.z * velocity.y)
           * e^-mu) / m
    ay <- (-k * velocity.y * V  + lambda *
             (angular.x*velocity.z- angular.z * velocity.x)
           * e^-mu) / m

    acc <- c(ax, ay, az)
    return(list(c(acc, velocity, angular)))
  }
  )
}
```