University of BRISTOL

MATHEMATICAL AND DATA MODELLING - GROUP 7

DATA MODELLING - FOURTH PROJECT

# Ping Pong Physics

*Gus Breese, Matt Clifford, Charlie Coombes, Mark Fitzsimmons*

**Abstract**

*This report focuses on modelling the spin and trajectories of 2 types of table tennis ball; the pre-2000 38mm ball and the modern 40mm ball. The preliminary focus of the project was to verify claims that increasing the diameter of the ball reduced the maximum achievable spin from 150rps to 133rps. The report then goes on to explore how varying mass and thickness affects spin, as well as investigating the effects of air density on spin and trajectory. Equations are derived to describe the interactions of the ball between the table and bat, and are then utilised within a computer model to simulate the trajectory, bounce, and hitting of the ball given any input parameters. High speed footage is also taken and analysed to validate the conclusions drawn from the model, and is used to confirm the realism of the outputs of the model.*

Under the supervision of
Prof. Alan Champneys

July 30, 2022

# Contents

# 1 Introduction

Table tennis consists of a ball being hit and bounced across a court while utilising techniques such as topspin, backspin, sidespin and smashes. Originally a standard tournament ball was 38mm in diameter, however the game was judged to be played at too high a speed to be a good spectator sport. In 2000 the standard ball was increased to 40mm as this increase in size would slow the game down, making rallies last longer, and so improving the game for spectators.

This report will also investigate the effects of the radius, mass and thickness of the ball on spin, as well as how the initial velocity of the ball affects spin and trajectory. Eventually, a model will be created for the effect collisions with different surfaces, such as the bat and table, have on the ball's trajectory. For a ball to have spin it needs to come in contact with a surface that has friction, without this the ball would either slide over the surface or be bounced back with no spin. The rubber coat on a table tennis bat provides this friction which allows the ball to spin if the player slices the bat with an up or downward motion at an angle. Spin causes the ball to curve through the air, as well as affecting its bounce, making it harder for the opponent to return the ball. This is due to the boundary layer of air around the ball making the air on one side move faster than the other. Differing velocities lead to a variation in pressure on either side, where pressure is high on the side with decreased velocity and low on the side with increased velocity since velocity and pressure are inversely proportional as stated by Bernoulli's Principle [1]. The pressure differences create a lifting force is known as the Magnus effect.

The two other forces acting on a table tennis ball in motion are air drag and weight. As the ball accelerates drag is increased until it is equal to the weight at which point terminal velocity is reached. Due to the short range nature of the game this model assumes terminal velocity to not be reached by a ball under the circumstances.

Using this report statistics about the number of revolutions per second of different sized balls can be verified. As well as this trajectories can be predicted for varying input velocities and angular velocities.

## 1.1 Initial Assumptions

- All balls are exactly 38mm or 40mm.
- 38mm and 40mm balls are exactly 2.5g and 2.7g respectively. [2]
- All ball hits are clean in order to avoid unnecessary complications in calculations.
- The angular velocity within the vertical direction will remain unaltered within an collision with the table.
- The ball will always roll whenever it makes contact with the table.
- The angular velocity within a plane will remain constant given that the velocity in that plane equals 0

# 2 Validating the Initial Claims

The brief claimed there was a decrease in maximum spin from 150 rps to 133 rps when the tournament ball size was changed from 38mm to 40mm. To verify this, maximum spin is assumed to occur at the instant the ball leaves the bat, as from this point onwards, the ball should lose spin due to air resistance.
It is assumed that the player puts the same amount of force into spinning both balls, and the force is transferred into spinning the ball identically. To calculate how much spin is affected, the moment of inertia for each ball needs to be calculated. The equation for Moment Inertia is:

$$I = \int r^2 dm$$

Where $r$ is the radius of the sphere. For this derivation, the sphere is sliced into thin cylinders. These cylinders have axes in the $z$ direction, with radius $R$ and thickness $dz$. The equation for the moment of inertia of a cylinder:

$$I = \int \frac{1}{2} r^2 dm$$

Substituting terms for $dm$:

$$I = \int_{-R}^{R} \frac{1}{2}(\rho_b \pi r^2 dz) r^2 = \int_{-R}^{R} \frac{\rho_b \pi}{2}(R^2 - z^2)^2 dz$$

where:

- $\rho_b$ = Density of the ball
- $R$ = Radius of the cylinder
- $dz$ = Thickness of the cylinder
- $r$ = Distance to the axis $r^2 = (R^2 - z^2)$

Integrate and substitute:

$$I = \frac{\rho_b \pi}{2}\left[R^4 z - \frac{2}{3}R^2 z^3 + \frac{z^5}{5}\right]_{-R}^{R} = \rho_b \pi R^5 [1 - \frac{2}{3} + \frac{1}{5}] = \frac{8\rho_b \pi R^5}{15}$$

Whenever this is applied to a shell of thickness $r_2 - r_1$:

$$I = \frac{8\rho_b \pi r_2^5 - r_1^5}{15}$$

Substituting $\rho_b = \frac{3m}{4\pi r_2^3 - r_1^3}$ [3]

$$I = \frac{2m}{5}\left(\frac{{r_2}^5 - {r_1}^5}{{r_2}^3 - {r_1}^3}\right) \tag{1}$$

The thickness of each ball is needed to calculate $r_1$, this can be calculated from knowing that

$$Volume = \frac{mass}{density} \tag{2}$$

where the density of a table tennis ball's cellulose nitrate is 1375 $kg/m^3$ [4].
The volume of the balls is

$$Volume = \frac{4}{3}\pi(r_2^3 - r_1^3) \tag{3}$$

Equations 2 and 3 can be rearranged to find $r_1$ as

$$r_1 = r_2^3 - \sqrt[3]{\frac{3Volume}{4\pi}} \tag{4}$$

The thickness of the 38mm ball is 0.41mm and the 40mm ball is 0.39mm. This assumes that the gas inside the ball has no contribution to the mass. To investigate if this is a valid assumption, the difference in mass from a sealed 40mm ball when it is cut open, releasing the gas, can be measured. The difference was found to be 0.0002 grams, so can be safely assumed negligible.
Using

$$T = \frac{1}{2}I\omega^2 \tag{5}$$

The Kinetic energy required to spin the 38mm ball 150 rps is 261.5 Joules
The angular velocity (spin) can be calculate from the same kinetic energy input, and dividing by $2\pi$ to convert from radians per second to revolutions per second.

$$\omega = \sqrt{2T/I} \tag{6}$$

This gives 137 rps for the 40mm ball, verifying that the claim of a reduction to 133 rps is slightly exaggerated, but albeit reasonable.

Assuming that the tangential motion of the bat on the ball causes an instantaneous angular velocity change on the ball, the bat's velocity can be found with regards to the given radius and required angular velocity of the ball.

$$Velocity = r\omega \tag{7}$$

where $\omega$ is in radians per second.

These calculations are based upon a totally elastic collision, and so cannot be rendered as totally accurate. There would be energy lost between the ball and the bat during impact due to defamation. However, it is interesting to note that, if the 38mm and 40mm balls were struck with a bat velocity of 17.9 m/s, the maximum rps attainable by each ball stated at the onset of the question would differ from the results achieved within this model. With those metrics in place, the smaller and larger ball would obtain angular velocities of 150 rps and 142.4 rps respectively. Instead, for the 40mm ball to obtain its stated maximum rps, the bat would have to travel at a speed of 17.2 m/s.

## 2.1 Varying Balls

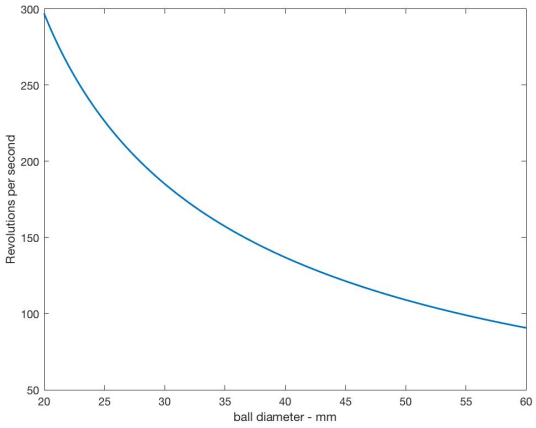



Figure 1: Constant energy, 0.2615kJ, transferred into spin, with varying ball diameter. Mass constant at 2.7g.

To investigate the effect that different types of ball have on the spin, varying parameters are run through eq. 6. Mass, thickness and diameter of the ball are shown below in figs. 1, 2 and 3. In actuality for the change in ball thickness, the mass of the ball also changes with how much material is needed, but mass is kept constant in the example of ball diameter changing to show solely the affect of change in diameter on the spin produced.
Figure 1 shows there is a clear difference in revolutions per spin between the 38mm and 40mm balls of roughly 10 rps which is less than stated in the brief. Figure 2 shows that from 0.1mm to 0.5mm, the spin is very sensitive to changes in thickness, with a 7.2% spin increase from 0.4mm to 0.3mm. This is important for ball manufacturers to consider since tournament balls are within this range at 0.41mm. Mass is the least sensitive, with a close to linear relationship with spin as shown in fig. 3, meaning the assumption of the change in mass to be negligible is fair.

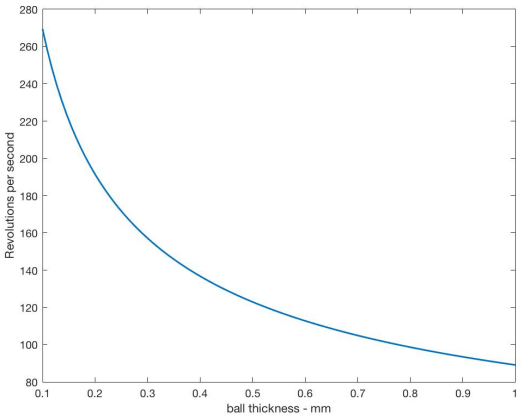



Figure 2: Constant energy, 0.2615kJ, transferred into spin, with varying ball thickness, which includes mass increase. Ball diameter constant at 40mm

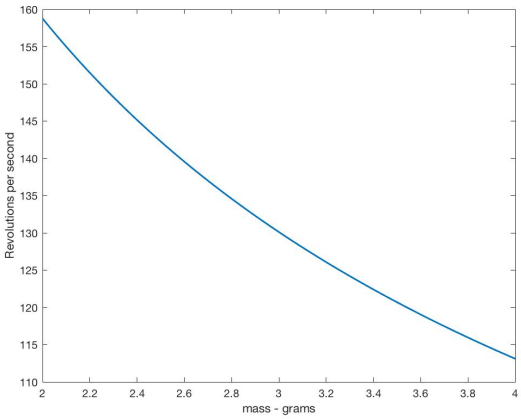



Figure 3: Constant energy, 0.2615kJ, transferred into spin, with varying ball mass. Ball diameter constant at 40mm.

# 3 Developing the Model

## 3.1 Frame of reference

The axis configuration for the model is shown in fig. 4, where $x$ axis is along the length of the table, $y$ along the width and $z$ along the height.

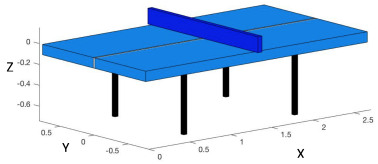
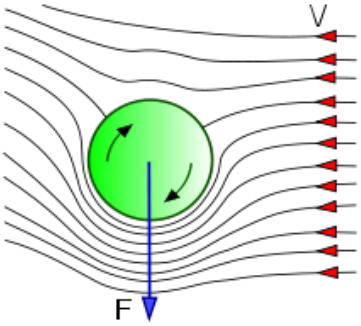



Figure 4: Axis labels for the configuration of the table

## 3.2 Modelling the Trajectory

There are 3 forces which are involved in the changing of the ball's trajectory during its flight in the air. These are gravity, drag, and the Magnus force.
The Magnus effect involves the force created by a spinning object flying through a viscous fluid. The force created will always be perpendicular to the velocity. This force is created due to the fact that on one side a boundary layer of a fluid is created, which collides with more of the passing fluid. This creates an area of higher pressure on one side of the ball, whilst a lower pressure area is created on the other side of the ball, as the fluid does not collide with the boundary layer.




[5]

Figure 5: An illustration of how a ball with topspin is effected by the Magnus effect. The high pressure boundary layer is located on above the ball, whilst the lower pressure below it. this creates a downward force.

Gravity will work negatively within the $z$ direction at a constant rate, whilst drag and the Magnus effect will have shifting values, depending on the angular and lateral velocities of the ball. These forces can be brought together to form Newtown's First Law equations as stated below.

$$R_x = ma_x - D_x - M_x \tag{8}$$

$$R_y = ma_y - D_y - M_y \tag{9}$$

$$R_z = ma_z - mg - D_z - M_z \tag{10}$$

R = the resultant force, m = the mass of the ball, a = the acceleration of the ball at the specified time, g = gravitational constant, D = drag force and M = the Magnus force. All of which are in the $x$, $y$ and $z$ directions. The drag and Magnus forces can be calculated using the following equations:

$$D = \frac{1}{2}\rho A v^2 C_d \tag{11}$$

$$M = \frac{C_m \rho V^3}{8}(\omega \times v) \tag{12}$$

$\rho$ denotes the density of the fluid in which the ball is travelling in, V is the volume of the ball, $C_m$ the Magnus coefficient, v the velocity of the fluid, A the area of the ball, $C_d$ the coefficient of drag for the ball and $\omega$ the angular velocity of the ball. Where $v_x$, $v_y$ and $v_z$ are the components of velocity, |v| is its modulus, $\omega_x$, $\omega_y$ and $\omega_z$ are the angular velocity components and g is the acceleration due to gravity. We can calculate the coefficients of acceleration for drag and the Magnus effect[6], which then become dependant on various

angular and lateral velocities. Considering that the surface area of the ball $A = \pi d^2$ with d=diameter of the ball:

$$K_d = \frac{C_d \rho \pi d^2}{8m} \tag{13}$$

$$K_m = \frac{C_m \rho \pi d^3}{8m} \tag{14}$$

Equations 11 and 12 are split into their component forms and simplified using eqs. 13 and 14. Initially the ball was modelled ignoring forces exerted by the bat and as stopping when it hits the table. Using these assumptions the balls motion was modelled in Matlab using the following equations:

$$\frac{d^2x}{dt^2} = -K_d v_x |v| - K_m(-\omega_y v_z + v_y \omega_z) \tag{15}$$

$$\frac{d^2y}{dt^2} = -K_d v_y |v| - K_m(-\omega_x v_z + v_x \omega_z) \tag{16}$$

$$\frac{d^2z}{dt^2} = -g - K_d v_z |v| - K_m(-\omega_x v_y + v_x \omega_y) \tag{17}$$

This model only looks at the ball in two dimensions but the equations are shown for all three as they are used further in the project. This was so as we could identify what our model was achieving with a simplistic motion model, with two degrees of lateral freedom and one of angular freedom. $K_d v_n |v|$ represents the drag force acting on the ball in the negative $x$ and $z$ directions. $K_m(\omega \times v)$ represents the Magnus force on the ball which is acting in the $(\omega \times v)$ direction. The velocity terms in the Magnus force are negative as they represent the velocity of the air not the ball. The velocity components all need to be found before these calculations can be carried out. To find these the differentials of the balls position vector are required:

$$\frac{dx}{dt} = v_x \tag{18}$$

$$\frac{dy}{dt} = v_y \tag{19}$$

$$\frac{dz}{dt} = v_z \tag{20}$$

Ode's 15 to 20 can be solved simultaneous numerically, to obtain the ball's trajectory over time.

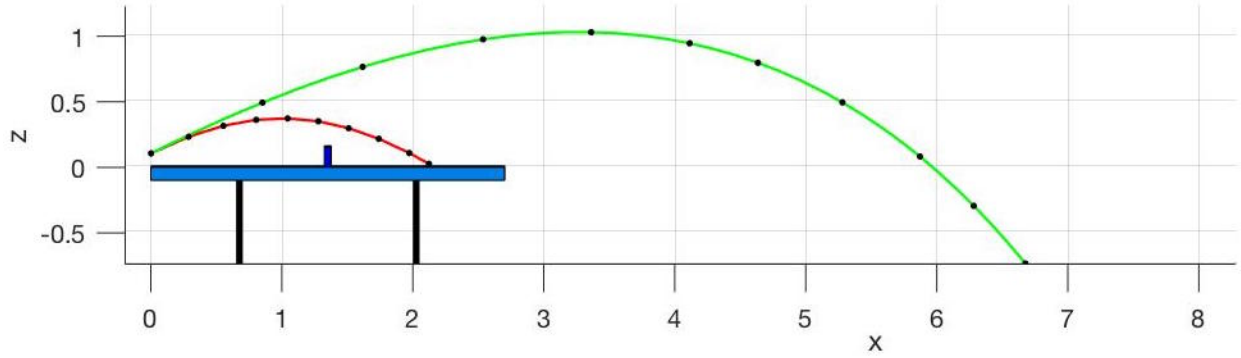



Figure 6: Red trajectory is with 50 revolutions per second top spin and green with out spin. Both balls start at position (0,0,0.1) m, with initial velocity (10,0,5) m/s.

For this simplified system the only spin that is considered is top spin, so that the trajectory can be easily modelled in a single plane. Figure 6 shows that the ball with top spin has an apex of vertical trajectory 65% lower than the ball with no spin, due to the Magnus force applying downwards. Resulting in a much shorter trajectory. This predicted trajectory is in conjunction with research [7] into shape of trajectories of different shots so suggests the first working model is a good basis for progression.

### 3.3 Modelling the Bounce

Having successfully modelled the trajectory of the ball in a single plane, the next aspect is to incorporate the interaction between the ball and the table. For this part of the model the following assumptions were decided:

- The collision between the ball and the table is instantaneous.
- The energy loss of the ball in the collision will be dealt with solely via the coefficient of Restitution, i.e. energy loss through sound and other mediums is ignored.
- There is no spin except top spin, so again the whole system can be plotted in a single plane.
- During the collision the ball either slides or rolls, both cannot occur.
- Coefficient of Restitution is constant.

When a ball bounces off a table, the impulse, S, can be described as follows:

$$S = F\Delta t = \mu N \Delta t = \mu m(v_{ez} - v_{iz}) \tag{21}$$

The exiting velocity, $v_{ez}$, in the vertical dimension taken at the centre of the ball can be expressed as:

$$v_{ez} = -k_v v_{iz} \tag{22}$$

where $F$ = Initial Force, $\mu$ = Coefficient of Friction, $N$ = Reaction Force, $t$ = Time, $k_v$ = Coefficient of Restitution and $v_{iz}$ = Incident velocity in the $z$ domain at the centre of the ball.
Impulse is therefore:

$$S = \mu m(-k_v v_{iz} - v_{iz}) = \mu m(1 + k_v)|v_{iz}| \tag{23}$$

Due to the fact that the impulse of the friction force acts in the $x$ and $y$ directions, to apply in both directions we must multiply by $cos\theta$. Since friction acts against the movement it must also be negative.

$$S_x = -\mu m(1 + k_v)|v_{iz}|cos\theta \tag{24}$$

Using trigonometry:

$$cos\theta = \frac{v_{bix}}{\sqrt{v_{bix}^2 + v_{biy}^2}} \tag{25}$$

Where $v_b$ = the velocity at the contact point on the table of the ball. Substituting this back into eq. 24:

$$S_x = \frac{-\mu m(1 + k_v)|v_{iz}|v_{bix}}{\sqrt{v_{bix}^2 + v_{biy}^2}} \tag{26}$$

Therefore, to use the impulse to find the velocity and spin after the collision the following relationship can be rearranged:

$$S = mv_2 - mv_1 = mv_{ex} - mv_{ix} \tag{27}$$

$$mv_{ex} = S + mv_{ix} = mv_{ix} - \frac{\mu m(1 + k_v)|v_{iz}|v_{bix}}{\sqrt{v_{bix}^2 + v_{biy}^2}} \tag{28}$$

$$v_{bix} = v_{ix} + \omega_{iy} r_{cb} = v_{ix} - \omega_{iy} r \tag{29}$$

Where $r_{cb}$ is the distance between the centre of the sphere and the point of contact with the table, which must therefore be a negative value since it is downwards. Substituting this in, and cancelling out the mass gives:

$$mv_{ex} = S + mv_{ix} = mv_{ix} + \frac{\mu m(1 + k_v)|v_{iz}|(\omega_{iy} r - v_{ix})}{\sqrt{(v_{ix} - \omega_{iy} r)^2 + (v_{iy} + \omega_{ix} r)^2}} \tag{30}$$

The equation for angular impulse is as follows:

$$S_{yr} = I(\omega_{ex} - \omega_{ix}) \tag{31}$$

Where

$$I = \frac{2rm}{3} \tag{32}$$

This is the moment of inertia for a thin shelled hollow sphere, the moment of inertia in eq. 1 is an over complication and the difference between eq. 1 and 32 is only 1.9%, so for further calculations the thickness of the shell is assumed negligible. Rearranging the angular impulse gives:

$$\omega_{ex} = \frac{S_{yr}}{I} + \omega_{ix} \tag{33}$$

Substituting in for the angular impulse, moment of inertia and $v_{biy}$ then cancelling out the mass gives the final equation used to find the exiting angular velocity at the contact point of the ball.

$$\omega_{ex} = \omega_{ix} + \frac{3\mu(1+k_v)|v_{iz}|(-v_{iy} - \omega_{iy}r)}{2r\sqrt{(v_{ix} - \omega_{iy}r)^2 + (v_{iy} + \omega_{ix}r)^2}} \tag{34}$$

The following equations are used to derive the velocity at the contact point.

$$v_{bex} = v_{ex} - \omega_{ey}r \tag{35}$$

Cancelling m from eq. 30 to find $v_{ex}$ and using the $y$ component form of eq. 34 the above equation can be written as:

$$v_{bex} = -\frac{\mu(1+k_v)|v_{iz}|(v_{ix} - w_{iy}r)}{\sqrt{(v_{ix} - w_{iy}r)^2 + (v_{iy} + w_{ix}r)^2}} + v_{ix} - w_{iy}r - \frac{3\mu(1+k_v)|v_{iz}|(v_{ix} - w_{iy}r)}{2\sqrt{(v_{ix} - w_{iy}r)^2 + (v_{iy} + w_{ix}r)^2}} \tag{36}$$

Simplifying gives the final equation used to find the exciting velocity of the ball at the contact point:

$$v_{bex} = v_{bix}(1 - \frac{2.5\mu(1+k_v)|v_{iz}|}{\sqrt{(v_{ix} - w_{iy}r)^2 + (v_{iy} + w_{ix}r)^2}}) \tag{37}$$

Assuming the ball rolls when it bounces rather than slips, $v_{bex} = 0$. For this condition to be satisfied:

$$\frac{\mu(1+k_v)|v_{iz}|}{\sqrt{(v_{ix} - w_{iy}r)^2 + (v_{iy} + w_{ix}r)^2)}} = \frac{1}{2.5} = 0.4 \tag{38}$$

Using this the equations for exciting velocity and angular velocity are derived.

$$v_{ex} = 0.4w_{iy}r + 0.6v_{ix} \tag{39}$$

$$v_{ey} = 0.6v_{iy} - 0.4w_{ix}r \tag{40}$$

$$w_{ex} = 0.4w_{ix} - 0.6\frac{v_{ix}}{r} \tag{41}$$

$$w_{ey} = 0.4w_{iy} + 0.6\frac{v_{ix}}{r} \tag{42}$$

Initially it was thought that the trajectory of the ball after the bounce could be modelled simply as the trajectory before the bounce with the direction of the $z$-velocity reversed and decreased by a factor. However, after further inspection this was found to be extremely inaccurate. Angular velocity had not been taken into account at all when first modelling the bounce which was one of the major inaccuracies. It is also an invalid assumption to say the $z$-velocity will be decreased by a constant factor on every bounce and only the $z$-velocity will be changed. This is why it is necessary to use impulse calculations to find the exiting velocity and angular velocity as all components can be taken into account. To do this velocity and angular velocity of the ball as it hits the table are needed, these can be found using data saved from the equation of motion ODEs described previously. Using these values and constants, such as the coefficients of restitution and friction acquired from research, the trajectory of the ball after a bounce can be modelled.

Impulse is equal to the change in momentum meaning the time of impact of the ball on the table doesn't need to be taken into account. This is a good method to find the velocities as contact time is assumed instantaneous which is not necessarily true. However using the experimental data discussed later in the report it can be seen that this is a valid assumption.

In the instant the ball is in contact with the table it is possible for the ball to either slip or roll before rebounding. In real situations both would occur, however for simplicity only one is considered at a time. Rolling occurs when there is friction between the ball and the table, which leads to imparted spin. For example, if a ball with no angular velocity collided with a table and rolled due to friction it would depart with topspin. The other case is slip, where only the normal reaction force is acting on the ball so its horizontal

speed theoretically remains constant. As the collision is inelastic the departing velocity would be thought to decrease from the incident, the coefficient of restitution being the ratio of these two velocities. However, as spin depends on velocity, shown by eqs. 41 and 42 if the incident velocity is significant enough the ball can leave with larger angular velocity than was incident.

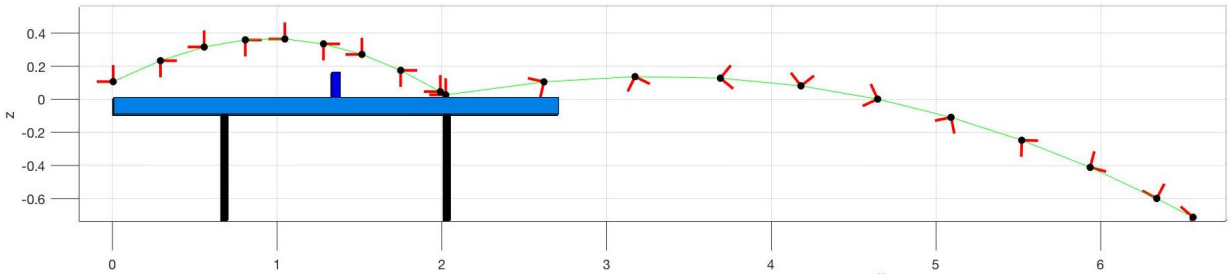



Figure 7: Trajectory of a bounce with topspin. Initial position is (0,0,0.1), initial velocity (10,0,5) m/s, and topspin (0,50,0) revolutions per second.

The red lines in fig. 7 represent the direction of spin at equal time intervals along the trajectory. Topspin requires an initial angular velocity about the $y$ axis and only $x$ and $z$ components of velocity re input to keep the ball in one plane. Theoretically a ball hit with only topspin will move in one plane so this is an accurate representation of its trajectory even thought realistically it would be extremely hard to input angular velocity about only one axis. The initial angular velocity is imparted by the torque about the center of the ball from the motion of the bat moving over it. The ball is also assumed to be hit from a stationary position slightly above the table edge. A moving ball initially would add slight complications but are negligible when modelling the bounce on the table. An increase in velocity after the bounce can be seen as the intervals become further apart, this is due to the topspin imparted on the ball by the bounce. Angular velocity also increases after the bounce as shown by the red lines. The model does not account for a rally so the ball carries on until it hits the ground. Modelling rallies could be an extension to improve the project in the future.

## 3.4 Modelling Spin

So far the model has only taken into account topspin. To make the model more applicable to real life situations, backspin and sidespin need to be added. Backspin is similar to topspin but the direction of angular velocity is reversed causing the ball to curve upwards rather than downwards. This is due to the Magnus effect. Implementing this in Matlab is trivial.

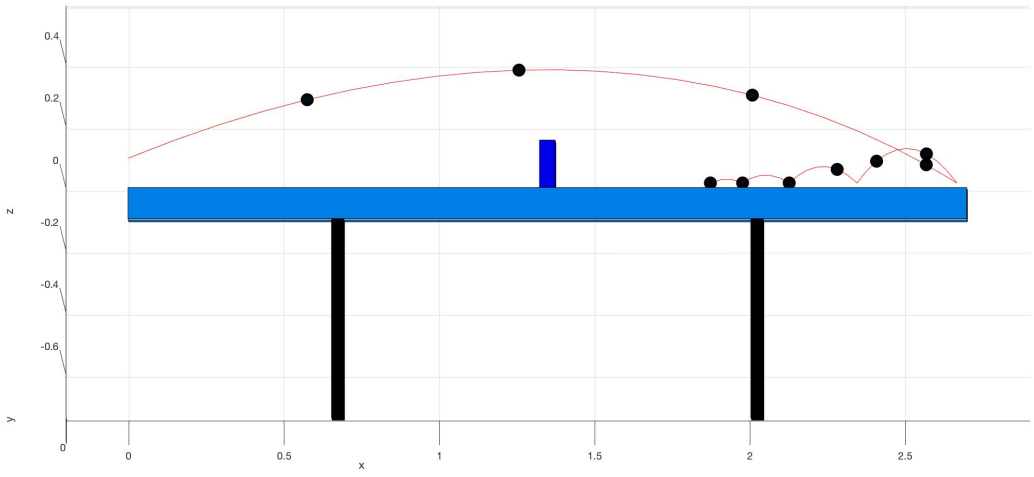



Figure 8: Trajectory of ball with backspin. Initial position is (0,0,0.1), initial velocity (5,0,2) m/s, and side spin (0,-10,0) revolutions per second.

It should be noted that Fig. 8 has the lines showing spin removed for clarity, however angular velocity values before and after the first bounce can be output, and a reduction from (0,-9.9,0) before the bounce to (0,-1.4,0) is shown. Negative angular velocity about the $y$ axis is necessary to provide backspin. This is gained from the torque acting on the ball's centre provided by the angle of attack of the bat moving under the ball. Compared to a topspin shot, a much lower velocity is required to hit the ball and allow it to bounce on the table, otherwise gravity and the drag force cannot force the ball down quickly enough for it to bounce on the table. Interestingly after the bounce the ball's $x$-velocity reverses direction. This is due to the backspin on the ball essentially reversing the momentum. The angular velocity decreases due to the low incident velocity needed to keep the ball in the range of the table. The velocity of the example shot is greatly decreased after the bounce as shown by the decreased distance between intervals, due to loses in kinetic energy.

Often a table tennis player will hit the ball with a combination of top or backspin and sidespin as this results in a curved trajectory meaning the ball is harder to return. Sidespin is more complex to include in the model as it involves modelling two planes in which the ball is moving rather than one. In order to do this the $y$-component forces and velocity need to be taken into account. This means adding the other ODE's for the motion of the ball travelling in the $x$-$y$ plane, which were shown by eq. 16 and 19.

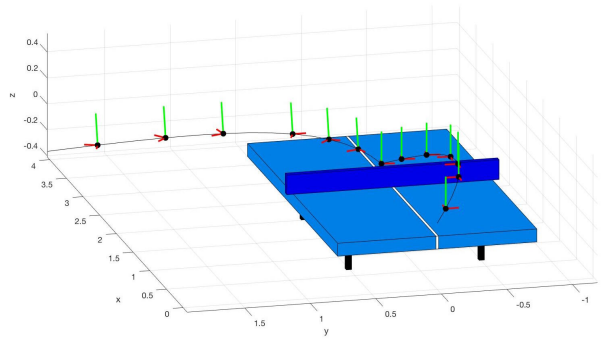


Figure 9: Initial position (0,0,0.1), initial velocity (6,-4,2) m/s, and side spin (0,0,50) revolutions per second.

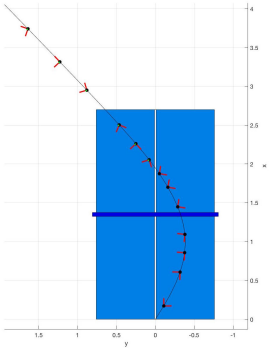


Figure 10: Same conditions as fig. 9, but shown from a bird's eye view to show the trajectory of the ball in the $x/y$ plane.

Figures 9 and 10 show the motion of a ball with sidespin, with angular velocity in the $z$ axis. The trajectory

simulates a serve. The magnitude of the angular velocity is represented by the length of its green vector attached to the ball, which fig. 9 shows has little reduction throughout the ball's flight.

As shown in the figure the ball veers across the table before and after the bounce due to the Magnus effect. If the angular velocity about the $z$ axis was positive the ball would veer across the table in the opposite $y$ direction. The trajectory is unaffected by the bounce due to the assumption that the $\omega_{iz} = \omega_{ez}$. The velocity after the bounce appears to increase as the distance between intervals increases in Figure 10, which is largely down to acceleration due to gravity as the ball goes over the edge of the table.

### 3.5 Spin Degradation

Using Tavares spin decay model [8][9], which looks at the spin decay of a golf ball:

$$\frac{d\omega}{dt} = \frac{-r^2 \rho A C_M v^2}{I} \tag{43}$$

Where $r$ is the radius of the ball, $I$ is the moment of inertia and $C_M$ is the coefficient of moment.
Tavares has found $C_M$ experimentally, by measuring the time dependent spin for a golf ball to be $C_M \approx 0.012S$, where $S$ is the spin parameter $S = r\omega/v$. There is no other data for this available, so in our model is it assumed to be the same value for the table tennis ball. Whilst this is a large assumption, a table tennis ball is far smoother than a golf ball and thus experiences less friction with the air, therefore despite being significantly lighter one would still expect to see minimal loss of spin due to air resistance, as is seen in the golf ball.
Equation 43 is applied in each dimensional component of the angular velocity and numerically solved using eqs. 15 to 20, which supply all the information required for the trajectory.

### 3.6 Simulating Spin

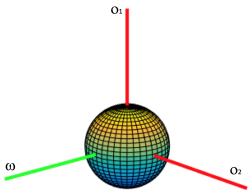



Figure 11: Computer representation of table tennis ball, with angular velocity vector in green, with the two orthogonal vectors in red.

For the simulation, animating the ball spinning is crucial for clarity. To represent the ball spinning, it's axes can be plotted. Finding two orthogonal vectors, $o_1$ & $o_2$, to the angular velocity makes a set of axis for the ball, as shown in fig. 11. The combination of these three vectors forms an orthogonal matrix.
Since the time step and angular velocity of the ball is known at each time step, this can be used to determine how many radians the ball has spun, $\theta$, over the time step/ frame in the animation. The rotational matrix, $R$, about the angular velocities' unit vector, $u$, shown below in eq. 44, rotates the two orthogonal vectors the correct amount for animation to the next time step. $o* = Ro$, where $o*$ is the next time step's orthogonal vector.

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix} \tag{44}$$

These 3 vectors can then be plotted from the balls centre at each point along it's trajectory, giving a spinning effect.
A video clip of the working simulation can be viewed at [10].

### 3.7 Comparison with Experimental Results

Using knowledge of the Magnus effect and implementation in Matlab, trajectories of a table tennis ball with varying spin and velocity were predicted. By comparing the predictions to the trajectories viewed in the high speed camera footage it can be seen that the model is an accurate prediction method for real life ball trajectories. In both cases top spin causes the ball to curve downwards sooner than it would have just due to gravity proving there is another downwards force acting on the ball.

When including the bounce in the model, the ball is predicted to experience a change in trajectory and angular velocity after departing the table depending on input spin. When the ball is hit with top spin it is found to increase in angular velocity and velocity after the bounce in the example shown. However the exiting angular velocity depends on incident velocity which must be sufficient enough to cause an increase in spin, if not the angular velocity will decrease. With back spin it is found that angular velocity decreases

after the bounce, often so much the ball reverses direction. With side spin the ball continues to curve in two planes after the bounce but with slightly decreased angular velocity. All these trajectory predictions are verified by the experimental data which shows top spin to increase angular velocity after impact, back spin to decrease angular velocity after impact and side spin to also decrease angular velocity after impact.

# 4 Discussion of results

## 4.1 Affecting spin by varying parameters

As can be seen by the increase in tournament ball diameter, an increase in radius causes the balls velocity to decrease due to the increase in moment of inertia. In turn, this results in a decrease in angular velocity as was verified by the calculations at the start of the report. With a diameter of 38mm the maximum angular velocity is assumed to be 150 rps, and using the energy required to achieve this, the maximum angular velocity for a ball of diameter 40mm is calculated to be 137 rps. The claims are only true for a human player as a machine could put a much greater degree of spin on a ball.

The change in ball diameter also caused a slight increase in mass of the ball, from 2.5g to 2.7g which will also increases the moment of inertia. Therefore, increasing mass will decrease the angular velocity. Increasing the thickness of a ball increases the moment of inertia as well due to the larger difference between $r_1$ and $r_2$. Again, this decreases the angular velocity.

These results show that increasing mass, radius or thickness of the ball will decrease the spin imparted leading to a better spectator sport, justifying the change in 2000. Unfortunately video footage was only taken using a 40mm ball so the variation in spin could not be verified using experimental data, however mathematically these findings are justified.

## 4.2 Optimal bat speed for maximising spin

Working backwards from the maximum spin that was provided, the optimal velocity of the bat could be found. For the 38mm ball this was found to be 17.9 m/s and for the 40mm ball 17.2 m/s. These calculations assume an elastic collision which is unrealistic due to energy loses to the surroundings. The velocity should be expected to be slightly less than those predicted. Therefore in order to achieve maximum spin, assuming this is 133 rps and 150 rps for the 40mm and 38mm balls respectively, the actual velocities required are likely to be slightly higher than those calculated.

Looking at the experimental data, the amount of spin that was found to be applied to top spin shots varies within arange of spin results from 64.5rps to 101rps were found. Though these shots were played by an experienced member of the University of Bristol table tennis team, there is a significant skill gap between this player, and the very best in the world. Taking this into consideration a maximum angular velocity of 133rps in a tournament rally is highly believable.

## 4.3 Spins effect on trajectory

As previously spoken about, the change in trajectory due to spin is actually due to the Magnus effect. The variation in pressure around the ball causes it to curve in the direction of the high to low pressure gradient. For topspin this is downwards in the $z$ direction, for backspin this is upwards in the $z$ direction and for sidespin this can be either way in the $y$ direction.
Topspin trajectories can be viewed in figs. 6 and 7 showing how the ball curves downwards. A backspin trajectory can be viewed in fig. 8 and a sidespin trajectory can be viewed in figs. 9 and 10. These trajectories are verified when viewing the footage from the high speed camera.

## 4.4 Environments effect on spin

Table tennis players report that different environments have a different effect of the spin maintained on the ball. Losses in spin can lead to less desirable interactions with the table. To investigate this, different air densities can be input into the model and the spin of the ball can then be analysed. Temperature is the primary factor that affects air density, however there are numerous other factors that can also affect it to varying degrees. The higher the temperature, the lower the air density. Air densities of 1 $kg/m^3$ to 1.6 $kg/m^3$ are used as these are the typical limits of real life situations [11].

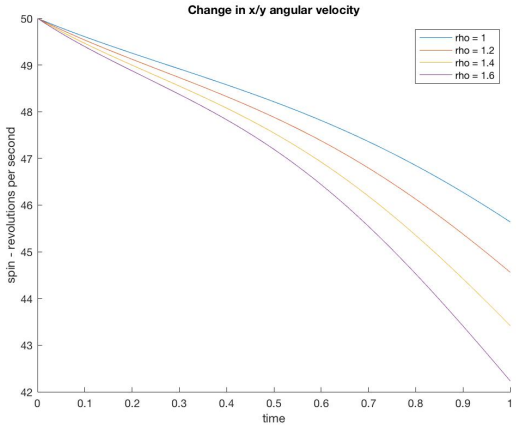



Figure 12: Angular velocity change in $x$ direction. Initial position (0,0,0.1), initial velocity (10,10,10) m/s, and side spin (50,50,50) revolutions per second. NB - Graph is identical for $y$ direction. Air densities in are $kg/m^3$

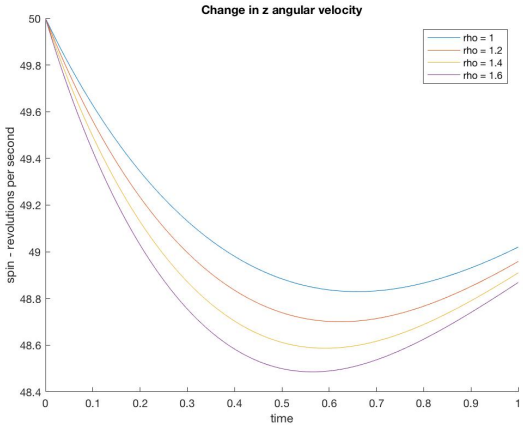



Figure 13: Angular velocity change in $z$ direction. Initial position (0,0,0.1), initial velocity (10,10,10) m/s, and side spin (50,50,50) revolutions per second. Air densities are in $kg/m^3$

Figures 12 and 13 show that with an increase in air density, the greater the change in spin over the ball's flight. Figure 12 shows that an air density increase from 1 $kg/m^3$ to 1.6 $kg/m^3$ results in 59.8% increased spin reduction after 1 second, and fig. 13 shows a 37.6% increase in peak spin reduction after about 0.55 seconds. In fig. 13, the ball's spin starts to increase after about 0.55 seconds, the time of peak spin reduction varies slightly for different air densities. This is due to the ball reaching the apex of its vertical arc and starting to descend due to gravity. This acceleration in the $z$ direction, increases the $z$ angular velocity. Figure 13 also shows that the change in $z$ angular velocity is fairly small, with only a 2.2% reduction for 1 $kg/m^3$ and a 3.0% reduction for 1.6 $kg/m^3$, compared to the 10.2% and 15.8% change in the $x/y$ direction. So the effect of different air densities for spin in the $z$ axis will predominantly affect the trajectory of the ball, as opposed to spin degradation.

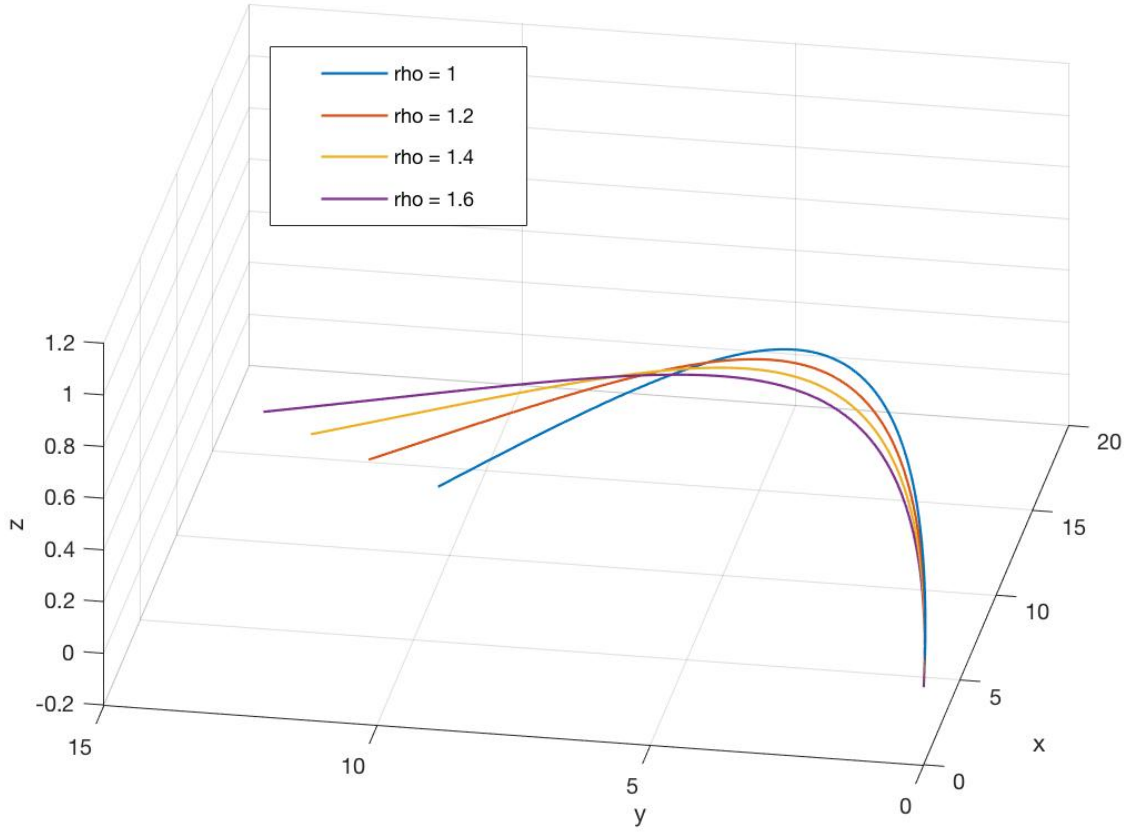



Figure 14: Trajectory change with air density ($kg/m^3$) variation. Initial position (0,0,0.1), initial velocity (10,0,5) m/s, and side spin (0,0,50) revolutions per second. Ball's flight is for 1 second. Air densities are in $kg/m^3$

The higher the air density, the more 'curl' the ball achieves, with end position distance increase of 27.2% from 1 $kg/m^3$ to 1.6 $kg/m^3$ air density. Similar results are obtained in the $x$ and $y$ spin axis.

## 4.5 Experimental Results and Interaction with the Table

In order to gain experimental data a high speed camera was used to capture a high level tennis player performing a variety of shots [12].

| Shot Type | Spin Before Bounce | Spin After Bounce |
|---|---|---|
| Top Spin Rally | 64.5 | 72.7 |
| | 66.7 | 74.1 |
| | 67.1 | 72.5 |
| | 101 | 104.2 |
| Maximum top spin | -31.2 | 100 |
| | -47.6 | 92.5 |
| Serve Backspin | 57.1 | 50.6 |
| | 60.6 | 56.3 |
| | 61.2 | 56.3 |
| Serve Sidespin | 57.1 | 56.3 |
| | 54.1 | 53.3 |
| | 56.3 | 55.4 |

Unfortunately, only spin could be accurately recorded by the cameras, as to record ball velocities would have required 2 cameras so that displacement in all directions could be viewed. However, it can be assumed that a mid rally shot has a higher velocity than a serve, due to the players being able to impart more of an impulse on a ball travelling towards them than a ball that is, relatively, stationary. The shots with the

largest top spin were themselves returns of shots with top spin. In the case of every top spin shot, the ball gained spin off the table whereas every back spin shot resulted in the ball losing spin. When looking at the equation for the exit angular velocity of the ball in the $y$ plane:

$$\omega_{ey} = 0.4\omega_{iy} + \frac{0.6v_{ix}}{r} \tag{45}$$

It can clearly be seen that whether the spin is increased or not by the interaction with the table is primarily determined by the $x$ component of the velocity, i.e. the velocity of the ball down the table. If this velocity is high enough then the ball will gain spin, if it is low then the ball will lose spin. This is determined by a critical point, where:

$$\frac{0.6V_{ix}}{r} = 0.6\omega_{iy}, v_{ix} = \omega_{iy}r \tag{46}$$

thus:

$$v_{ix} = \omega_{iy}r, \omega_{ey} = \omega_{iy}, \omega_{iy} \text{ is unaffected by the bounce} \tag{47}$$

$$v_{ix} > \omega_{iy}r, \omega_{iy} \text{ is increased in the bounce} \tag{48}$$

$$v_{ix} < \omega_{iy}r, \omega_{iy} \text{ is decreased in the bounce} \tag{49}$$

This is backed up by what can be seen in the experimental results. Since in every top spin shot both the $\omega_{iy}$ and $v_{ix}$ are positive the spin can be increased if the velocity is great enough, however for a backspin shot the $\omega_{iy}$ and $v_{ix}$ are of opposite sign therefore the spin will always become more positive, so the ball must lose backspin. This is reflected in the experimental results as in every topspin shot the ball gains some spin from the bounce, and in every backspin shot the ball loses spin.

The assumption that $\omega_{iz}$ is unaffected in the bounce is also largely backed up by the results, as in every sidespin shot the ball lost only 1 or 2 rps, which can be considered to be negligible in the effect that this has on the trajectory of the ball. This loss of spin is likely to be because perfect sidespin isn't imparted on the ball, and small amounts of top spin or sidespin in the $x$ plane are likely to cause result in tiny losses of energy during the impact.

# 5 Further Work

## 5.1 Collisions between the ball and the bat

One of the most challenging tasks set within the brief was to model how the ball would bounce off of the bat whenever a shot has been played. Within this collision it was assumed that the ball would slide during the contact, as opposed to the collision with the table, where the ball was assumed to roll. The reason for this is that, although the bat has a higher coefficient of friction than the table, the velocity differential between the bat and the ball is likely to be significantly greater than between the ball and the table, which is conducive to the ball slipping rather than rolling. The velocities of the bat have to be taken into account when deriving the equations of motion for the ball. A further assumption would be that the ball would be played off of the "middle" of the bat; in other words, the player would make perfect contact with the ball. Furthermore, the experimental tests with the high speed camera showed that the impact time between the bat and the ball is less than 0.5ms [12]. This suggests that there is little deformation in the rubber and sponge on the bat, which leads to the idea of different bat sponges contributing to impact shock wave dissipation, rather than having an affect on spin produced as suggested in the brief. Therefore, any deformation to the rubber and sponge are ignored for the purposes of this model.

There are a number of factors that make the modelling of the collision between the bat and the ball significantly more complicated than that between the ball and the table, namely that the bat itself has velocity, as well as a variable angle of attack, which means that the spin on the ball can have very different effects on the output velocity depending on the conditions of the collision. Taking this into consideration, one way to greatly simplify the problem is to assume that the bat can only rotate on the $y$ axis, meaning that the angle has no effect on the side spin of the ball.

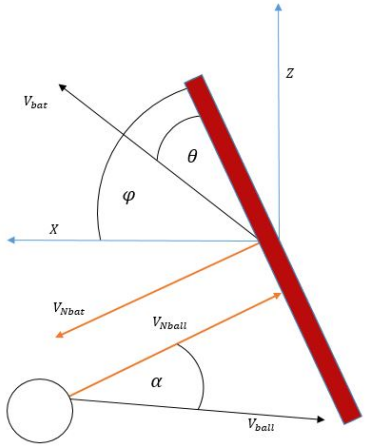



Figure 15: Diagram illustrating the important velocity components of the ball and bat

In order to calculate the impulse of the collision it is necessary to know both the angle between the direction of the velocity of the bat, and the bat itself as shown by the angle $\theta$ in fig. 15. Further to this, it is necessary to know the angle between the velocity of the ball and the component of the ball's velocity that is perpendicular to the bat, as is depicted by $\alpha$. The angle of attack of the bat is shown by $\phi$. Since in this collision it is being assumed that the bat has no acceleration, this means that the component of the balls velocity that is normal to the surface of the bat is affected only by the coefficient of restitution during the collision. Taking into account the velocity of the bat, the effective velocity of the ball perpendicular to the surface of the bat can be taken as:

$$V_{iN} = V_{ball}cos(\alpha) - V_{bat}sin(\theta) \tag{50}$$

And the normal component of the exit velocity is:

$$V_{eN} = -k_v V_{iN} \tag{51}$$

Therefore, similarly to the impact with the table, impulse can therefore be calculated to be:

$$S_x = \frac{\mu m(1+k_v)|V_{iN}|V_{bix}}{\sqrt[2]{V_{bix}^2 + V_{biy}^2}} \tag{52}$$

$$S_y = \frac{\mu m(1+k_v)|V_{iN}|V_{biy}}{\sqrt[2]{V_{bix}^2 + V_{biy}^2}} \tag{53}$$

Where $V_{bix}$ and $V_{biy}$ are the $x$ and $y$ velocities of the point of the ball that is in contact with the bat. The derivations of the equations for the exit velocities is where the limitation that the bat can only rotate on the $y$ axis is most helpful, as it means that varying the angle of attack only has an effect on the $x$ and $z$ components of the velocity, and the $y$ component of the angular velocity. In terms of the $x$ velocity if the bat is held perpendicular to the table then it can only be affected by the coefficient of restitution, as the bat is not accelerating, however if any other angle is applied to the bat then any incoming topspin or backspin will have a resultant $x$ velocity component. Therefore, taking into account the tangential velocity of the topspin, the resulting $x$ velocity can be described by:

$$V_{ex} = -k_v(V_{ix} - V_{pix})sin(\phi) + \omega_{iy}rcos(\phi) \tag{54}$$

Where any velocity with the subscript p, such as $V_{pix}$ refers to the velocity of the bat. The $y$ velocity is unaffected by the angle of attack, so can be calculated as:

$$V_{ey} = V_{piy} - V_{iy} - \frac{\mu(1+k_v)|V_{iN}|(\omega_{iz}r + V_{iy} - V_{piy})}{\sqrt[2]{((V_{iy} - V_{piy}) - \omega_{iz}r)^2 + ((V_{iz} - V_{piz}) - \omega_{iy}r)^2}} \tag{55}$$

The resultant $z$ component of the velocity is affected by $\omega_{iy}$, which is already included in the impulse equation, as well as the vertical component of the normal reaction if the bat has any angle of attack that isn't perpendicular to the table. The resultant $z$ component can therefore be found by:

$$V_{ez} = V_{piz} - V_{iz} - \frac{\mu(1+k_v)|V_{iN}|(V_{piz} - V_{iz} - \omega_{iy}r)sin(\phi)}{\sqrt[2]{((V_{iy} - V_{piy}) - \omega_{iz}r)^2 + ((V_{iz} - V_{piz}) - \omega_{iy}r)^2}} - k_v(V_{ix} - V_{pix})cos(\phi) \tag{56}$$

Similarly to the assumption that $\omega_{iz}$ doesn't change in the collision with the table, if the bat is perpendicular to the table then $\omega_{ix}$ will remain unchanged in the collision, however if some angle is applied to the bat the spin on the $x$ axis should be affected as it is no longer acting perpendicular to the contact surface. The exit spin on the $x$ axis can therefore be described by:

$$\omega_{ex} = \omega_{ix} - \frac{3\mu(1+k_v)|V_{iN}|(\omega_{ix}r + V_{iy} - V_{piy})}{2r\sqrt[2]{((V_{iy} - V_{piy}) - \omega_{iz}r)^2 + ((V_{iz} - V_{piz}) - \omega_{iy}r)^2}}cos(\phi) \tag{57}$$

The other side spin, on the $z$ axis, is similarly affected by the $y$ component of the velocity, and is affected most by the angle of attack when the bat is perpendicular to the table:

$$\omega_{ez} = \omega_{iz} - \frac{3\mu(1+k_v)|V_{iN}|(\omega_{iz}r + V_{iy} - V_{piy})}{2r\sqrt[2]{((V_{iy} - V_{piy}) - \omega_{iz}r)^2 + ((V_{iz} - V_{piz}) - \omega_{iy}r)^2}}sin(\phi) \tag{58}$$

The spin on the ball on the $y$ axis, i.e. top spin, is affected by both the $z$ and $x$ velocity of the ball. The reason that neither side spin is affected by the $x$ velocity, while the top spin is affected by both the $x$ and $z$ velocities is because the bat is limited to rotation in the $y$ plane, resulting in the $y$ components having an extra degree of freedom. For the side spin to be affected by the $x$ velocity would require the bat to be able to rotate on the $z$ plane. Rotation about the $x$ plane would be relatively useless to the model as it would have no effect on the angle between the direction of velocity of the ball and the surface of the bat. Taking into account both the $x$ and $z$ velocities we can find $\omega_{iy}$ to be:

$$\omega_{ey} = \frac{(3\mu(1+k_v)|V_{iN}|(-\omega_{iy}r - V_{iz} + V_{piz}))sin(\phi) + (3\mu(1+k_v)|V_{iN}|(-\omega_{iy}r + V_{pix} - V_{ix}))cos(\phi)}{2r\sqrt[2]{((V_{iy} - V_{piy}) - \omega_{iz}r)^2 + ((V_{iz} - V_{piz}) - \omega_{iy}r)^2}} \tag{59}$$

Testing these equations with various inputs offers very promising results. Hitting through a ball that has no spin with a closed bat imparts topspin on the ball even with no $z$ velocity on the bat, similarly if the bat is open as it strikes the ball then backspin will be imparted. Using these equations to test the results from the very start of the paper, achieving the maximum spin for 38mm and 40mm balls, 133rps of spin can be achieved on a 40mm ball that only has a slightly negative velocity in the $z$ direction by hitting through the ball with a slightly closed angle of 66.4° to the $z$ axis, with a bat velocity of (1, 0, 16.6)m/s. Applying these same conditions to the 38mm ball and 141rps of topspin is achieved, which is very close to the proposed figure of 150rps. When other factors are taken into consideration such as the fact that the material from which the balls are made has changed, and that the model is ignoring the fact that the ball is likely to both slip and roll during the contact with the bat, this seems to be a very good result, which also serves the purpose of validating some of the assumptions made early in the paper.

One of the principal ways to check the usefulness of this model is to measure various output velocities in comparison to the angle of attack of the bat. For this, it was decided that the bat would be considered within 2 different contexts. Firstly, whenever $\phi$ was measured between $\frac{\pi}{2}$ and $\pi$, the player would be playing a topspin shot, where the incident velocity of the ball $v_i = (-5, -1, -3)^T$, the velocity of the bat $v_p = (2, 1, 5)^T$, and the angular velocity $\omega_i = (-10, -25, 10)^T$. The graphs for the angular and lateral velocities are given below. Note that the ball's velocity is considered to be negative before the collision. Therefore although the ball's angular velocity in the $y$ plane is negative, from the perspective of the ball it is travelling with topspin.

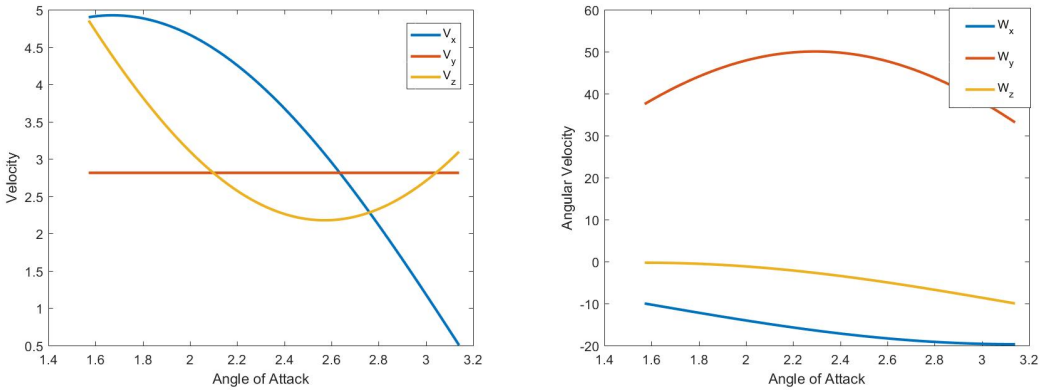



Figure 16: Graphs showing the various lateral(left) and angular(right) velocities of a ball after colliding with a bat, enacting a topspin forehand shot, using the parameters given above.

The $z$ velocity decreases, up to the point where $\phi$ equals 2.5rad. This is the point where the restitution term, $(k_v(V_{ix}-V_{pix})cos(\phi))$, of the $z$ velocity is greater than that of the impulse term $(\frac{\mu(1+k_v)|V_{iN}|(V_{piz}-V_{iz}-\omega_{iy}r)}{\sqrt[2]{((V_{iy}-V_{piy}))}}\sin\phi)$.
The $x$ velocity decreases as the normal velocity of the bat within the $x$ plane decreases, resulting in the restitution force having a smaller effect on the $x$ component of the velocity. As previously discussed, the $y$ velocity of the ball will remain constant regardless of the angle of attack of the bat. The critical angular velocity to consider is that in the $y$ plane, which represents topspin and backspin. The most optimal $\phi$ value for topspin, with the specified inputs is approximately 2.3rad. It is important to note that for all $\phi$ tested the angular velocity of the ball is increased.
The second style of shot that will be analysed is the "cut". In this case, the bat will be moving forward in the $x$ direction, but negatively in the $z$ plane, whilst the ball has a positive $z$ velocity and contains backspin. Consider that $v_i = (-5,-1,1)^T$, $v_p = (2,1,-5)^T$ and $\omega_i = (-10,25,-10)^T$

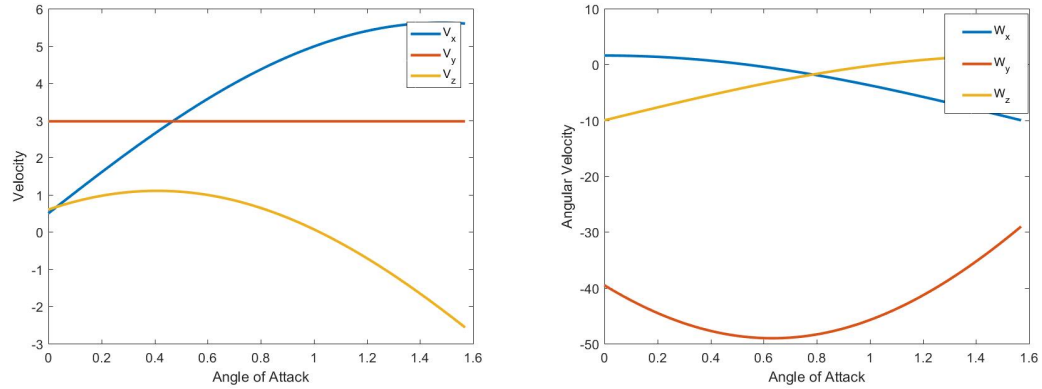



Figure 17: Graphs showing the various lateral(left) and angular(right) velocities of a ball after colliding with a bat enacting a "cut" shot using the parameters provided above.

The values for the $x$ velocities are the easiest to interpret as they largely follow our natural intuition. As the angle of attack increases, the $x$ component of the restitution force will increase as the bat comes closer to being perpendicular with the table. When the angle of attack is very low, the ball continues to have a small negative $x$ velocity, in this case the resultant $x$ velocity is equal to 0 when the bat is at an angle of 0.21rad. This makes logical sense, despite the bat travelling in the opposite direction, as the ball will slip off the bat at such an angle that the friction and restitution forces applied on the ball are insufficient to reverse it's direction of travel. Meanwhile within the $z$ plane the velocity barely increases past 1 $ms^{-1}$, but for low values of $\phi$ the rate of change of the $z$ velocity is low. The reduced velocity values as $\phi$ increases are due to the magnitude of the normal velocity's $z$ component increasing with $\phi$. In truth, the results are expected as there would rarely be a cut shot at which the bat is held at a greater angle than 30° between the bat and the table. The backspin of the ball will too increase regardless of the angle of attack of the bat.

An interesting aspect to note from varying the value of $\phi$ under constant initial conditions is that for

both the topspin and backspin shot there is respectively a maxima and minima in the parabola of the $\omega_{ez}$ values. This is because the resultant $\omega_{ez}$ is dependant on both the $x$ and $z$ components of the incoming velocity, meaning that the optimal angle of attack differs, assuming one is looking to maximise the spin, depending on the initial conditions of the shot.

### 5.2 Real life applications

During the project, meetings were arranged with the owner of a start up company, Paul, who is designing an automated table tennis serve machine. The machine fires balls with a random spin to help the user practice returning shots. The machine comes with an android app that it runs from. Paul provided us with a sensor which would read the speed of the bat at the moment it came into contact with the ball, the sensor is designed for badminton however, so Paul was struggling to extract any meaningful data from it. We had the intention of using that sensor to accurately measure bat speeds to verify the outputs our model provided in relation to the collision of the bat, but this proved too complicated to implement given our short time scale. However, we are discussing the possibility of integrating our model with his machine and app, to improve it's functionality. It would help calibrate the spin and velocity that his machine is putting on the ball, since you know the starting height and where the ball lands, as currently the machine doesn't accurately know this. Furthermore, since bounce position can be calculated from given position, velocity and angular velocity, this could be integrated into the app to let the user select where on the table they would like the machine to fire, as well as the bounce characteristics that occur with different types of spin. The simulation model has been made more user friendly for these purposes, with the addition of a GUI [10].

## 6 Conclusion

Overall, the model does a very good job of describing each of the constituent parts of a table tennis shot, and could be utilised to find solutions to the problems set out in the project brief. The model is particularly good at simulating the interaction between the ball and the table. This is because it requires certain conditions for the ball to gain spin from the bounce, and the experimental results from the high speed camera show this to be a good representation of the system. In terms of the trajectory model, the Magnus effect has been implemented along with drag and gravity. While there is little information in the public domain about the Magnus coefficient of a table tennis ball, using the value suggested in [6] produced very good results. The most complex part of the system definitely lies in the modelling of the collision between the bat and the ball. While some major simplifications had to be made, the overall results were very promising, and was even useful in terms of verifying results achieved earlier in the project, namely the maximum spin that a person could put on a 38mm and 40mm ball.

In terms of solving the problems laid out in the project brief, the model attempts to explore them all to the greatest possible extent within the constraints of complexity and available time. Figures 1, 2 and 3 demonstrate the sensitivity of spin to each of the parameters of the ball. The bat speeds required to achieve the supposed maximum spin for both ball sizes was calculated using eq. 7, and this was later verified using the bat model. The bat model takes into account the 'grippiness' of the bat through the incorporation of the coefficient of friction, which means it can be adjusted for different bats, however the elasticity of the rubber has been ignored for the purposes of simplifying the model. While this is certainly a drawback, the model runs well enough without it such that elasticity is clearly a refinement rather than a crucial element. The effect of the spin on the trajectory is fully described by the Magnus effect, and as such this is included in the equations of motion for the ball. The effects of the different types of spin on the trajectory can be seen in figs. 7 and 10. The loss of spin due to air resistance is determined by eq. 43. Unfortunately the coefficient of moment used is borrowed from a golf ball, though this is likely to have only a very small effect on the results, as in the case of both balls the spin degradation should be very small. Every force acting on the ball during its flight, except gravity, is affected by the density of the air, making it relatively easy to compare the effects of varying air density on the flight of the ball, as displayed in fig. 14, and the effects of varying air density on spin degradation can be seen in figs. 12 and 13. Finally, the interaction between the ball and the table is dealt with from eqs. 21 to 42. The collision between the ball and the table behaves with a great deal of realism, and the only obvious improvements would be to include the deformation of the ball, and loss of spin in the $z$ axis.

# 7 Limitations and Improvements

When considering contact between the ball and the table it was assumed that the ball would only ever roll. In reality, the contact is likely to be a combination of rolling and slipping which is significantly more complicated, as $v_{bex}$ does not necessarily equal 0, which is the basis for the exit velocity and spin calculations. To improve the model an Eulers approach could be taken in order to split the contact up into slip and stick, which would result in a different equation to 38 and therefore different exit velocity and angular velocity equations. Alternatively, Stronge's approach [13] could be considered, however this is a very involved method, and would require some highly complex mathematics. Another limitation in the assumptions of the bounce condition is that the collision is instantaneous, this is also a limitation for modelling the collision with the bat. Due to the deformation of the ball and bat surface on impact the equations of motion become much more complex, and factors such as contact time must be taken into account. However it would be possible to go about this a number of ways, such as modelling the bat as a rigid flat surface on a spring assuming the deformation affects the surface as a whole, though this approach would require the inclusion of damping and would still not be wholly accurate. The interaction of the ball with the table or floor only takes into account a flat surface. Bounces that hit the edges of the table are not taken into consideration, which result in an unpredictable change in velocity after the bounce due to the sharp edge, this would only be something to include in the perfect simulation.

# References


[1] Brian E Faulkner and F Marty Ytreberg. Understanding bernoulli's principle through simulations. *American Journal of Physics*, 79(2):214–216, 2011.

[2] Yuza N Iimoto Y, Yoshida K. Rebound characteristics of the new table tennis ball; differences between the 40mm (2.7 g) and 38mm (2.5 g) balls. *Table Tennis Sciences*, 5:233–243, 2002.

[3] Eric W. Weisstein. Moment of Inertia - Spherical Shell. `"http://scienceworld.wolfram.com/physics/MomentofInertiaSphericalShell.html`.

[4] Engineering Toolbox. Density of solids. `http://www.engineeringtoolbox.com/density-solids-d_1265.html`, 2017.

[5] Patrick Brandon. What is the magnus effect? `http://ffden-2.phys.uaf.edu/211_fall2010.web.dir/Patrick_Brandon/what_is_the_magnus_effect.html`, 2017.

[6] Qiang Huang Member IEEE Weimin Zhang Zhangguo Yu Xiaopeng Chen, Ye Tian. Dynamic model based ball trajectory prediction for a robot ping-pong player.

[7] Emil Vassilev. Trajectories of different types of spin. `https://www.linkedin.com/pulse/20140920022202-82311677-virtual-tennis-academy-lesson-1-slice-topspin-in-tennis`, 2014.

[8] K. Shannon G. Tavares and T. Melvin. Golf ball spin decay model based on radar measurements. *Science and Golf*, 3, 1998.

[9] Alan M. Nathan. The effect of spin-down on the flight of a baseball. 2008.

[10] Youtube clip of simulation. `https://www.youtube.com/watch?v=Ijy7N31rks0`, 2017.

[11] Wikipedia. Density of air. `https://en.wikipedia.org/wiki/Density_of_air`, 2017.

[12] Youtube clip of slow motion footage. `https://www.youtube.com/watch?v=wzJqh4aJvYs`, 2017.

[13] W. J. Stronge. *Impact Mechanics*. Cambridge University Press, 2000.

# Appendix

## Main execution function

```
function runMain3d(posX,posY,posZ,velX,velY,velZ,wX,wY,wZ)

global lengthT; global width; global stepTime;
lengthT = 2.7; width = 1.52;
%converting to radians per second
wX=wX*2*pi; wY=wY*2*pi; wZ=wZ*2*pi;
%------------------------------------------------------------------------
%model calculations
contact = 0; contact2 = 0; countB = 0;
%calcuating first trajectory until bounce
[posB,velBB,omegaBB,path1,omegaPath1,contact] =
↪  trajectory([posX,posY,posZ],[velX,velY,velZ],[wX,wY,wZ]);
path = path1; omegaPath = omegaPath1;
%carry on bouncing until off the table
while contact == 1 && countB < 4
    %velocity and angular velocity change after the bounce
    [velAB,omegaAB] = bounce(velBB,omegaBB);
    %calculating second trajectory until bounce
    [posB,velBB,omegaBB,path1,omegaPath1,contact] = trajectory(posB,velAB,omegaAB);
    %concatinating both trajectories
    path = [path;path1]; omegaPath = [omegaPath;omegaPath1];
    countB = countB + 1;
end
%------------------------------------------------------------------------
if (max(max(omegaPath))) ~= 0
    omegaPathN = omegaPath/(4*(max(max(omegaPath))));
else
    omegaPathN = omegaPath;
end
%working out angular velocity vectors
X = path(:,1) + omegaPathN(:,1);
Y = path(:,2) + omegaPathN(:,2);
Z = path(:,3) + omegaPathN(:,3);

%initialising vectors for for loop
vec1 = zeros(length(path)+1,3); vec2 = zeros(length(path)+1,3);
%rotate orthogonal vectors by how much the ball has spun
for i = 1:length(path)
    if sum(omegaPathN(1,:)) ~= 0 && i == 1
        %find orthogonal vectors to angular velocity
        [a1 a2] = (findOrth(omegaPathN(i,:)));
        a1=a1'; a2=a2';
        vec1(i,:) = a1; vec2(i,:) = a2;
    end
    if sum(omegaPathN(1,:)) == 0 && i == (length(path1)+1)
        %find orthogonal vectors to angular velocity
        [a1 a2] = (findOrth(omegaPathN(i,:)));
        a1=a1'; a2=a2';
        vec1(i,:) = a1; vec2(i,:) = a2;
    end
    theta = sum(omegaPath(i,:))*stepTime;
    vec1(i+1,:) = rot(omegaPath(i,:),vec1(i,:),theta);
    vec2(i+1,:) = rot(omegaPath(i,:),vec2(i,:),theta);
end
```

```
%scale down to nice plotting
scale = 10; vec1 = vec1/scale; vec2 = vec2/scale;

%translate vectors to be in the ball's frame of reference
X2 = path(:,1) + vec1([1:length(path)],1);
Y2 = path(:,2) + vec1([1:length(path)],2);
Z2 = path(:,3) + vec1([1:length(path)],3);

X3 = path(:,1) + vec2([1:length(path)],1);
Y3 = path(:,2) + vec2([1:length(path)],2);
Z3 = path(:,3) + vec2([1:length(path)],3);

%animate
plot3d(path,X,Y,Z,X2,Y2,Z2,X3,Y3,Z3)
```

### Calculate trajectory using ODE45

```
function [posOut,velOut,omegaOut,path,omegaPath,contact] =
↪  trajectory(posIn,velIn,omegaIn)
%INPUTS:  posIn    - starting position (m), 3 X 1 vector
%         velIn    - starting velocity(m/s), 3 X 1 vector
%         omegaIn  - starting angular velocity(revolutions/s), 3 X 1 vector

%setting coefficients
%Diameter of ball in meters
D=0.04;
R=D/2;
a=pi*R^2;
%Drag coefficient estimate
CDrag=0.4;
%Magnus coefficient estimate
CMagnus=1;
%density of the air kg/m^3
rhoAir=1.29;

%Ping-pong ball mass in kg
P.m=0.0027;
P.g=9.8015;  %acc due to gravity m/s^2
%degradation rate of spin - set to zero for no degradation
deg = 1;
% P.alphax=deg;
% P.alphay=deg;
% P.alphaz=deg;

I = inertiaB(R,0.0027);

P.alpha=(-R*rhoAir*a*R*0.012)/I;

%Force coefficients, drag and Magnus
P.Kd=CDrag*rhoAir*pi*D^2/(8*P.m);
P.Km=CMagnus*rhoAir*pi*D^3/(8*P.m);

%Rules for bouncing
P.restitution=0.7;  % coefficient of restitution; '1' for a perfect bounce
```

```
%------------------------------------------------------------------------

%initial position
x0=posIn(1);y0=posIn(2);z0=posIn(3);
%initial velocity
vx0=velIn(1);vy0=velIn(2);vz0=velIn(3);
%initial angular velocity
omegax0=omegaIn(1);omegay0=omegaIn(2);omegaz0=omegaIn(3);

%initial conditions
X0=[x0;y0;z0;vx0;vy0;vz0;omegax0;omegay0;omegaz0];
tstart=0;
tfinal=2;  %in sec
global stepTime; stepTime = 0.03;

options1=odeset('events', 'on', 'RelTol', 1e-7, 'AbsTol', 1e-8);

[tt,state,te,oute,ie] = ode45('RHS', [tstart:stepTime:tfinal],X0,options1,P);

%encoding bounce data for output
if isempty(oute) == 1
    posOut = [state(end,1);state(end,2);state(end,3)];
    velOut = [state(end,4);state(end,5);state(end,6)];
    omegaOut = [state(end,7);state(end,8);state(end,9)];
else
    posOut = [oute(1);oute(2);oute(3)];
    velOut = [oute(4);oute(5);oute(6)];
    omegaOut = [oute(7);oute(8);oute(9)];
end
contact = 0;
if isempty(oute) == 0
    if oute(3) > 0
        contact = 1;
    end
end
%encoding trajectory data for output
path = [state(:,1),state(:,2),state(:,3)];
omegaPath = [state(:,7),state(:,8),state(:,9)];
end
```

## ODE's and event situations

```
function [out1, out2, out3] = RHS(t,state,options,P)

global lengthT; global width;

%nice variable names
x=state(1);y=state(2);z=state(3);
vx=state(4);vy=state(5);vz=state(6);
omegax=state(7);omegay=state(8);omegaz=state(9);

%the speed
absv=sqrt(vx^2+vy^2+vz^2);

%stop condition when ball hits the surface
```

```
stop = 1;
%for table
if state(3) < 0.02 && state(1) < ((lengthT) +0.095) && abs(state(2)) < width/2 %takes
↪  into account the radius of the ball
    pos=[state(1);state(2);state(3)];
    vel=[state(4);state(5);state(6)];
    angvel=[state(7);state(8);state(9);];
    stop = 0;
end
%for floor
if state(3) < -0.72
    pos=[state(1);state(2);state(3)];
    vel=[state(4);state(5);state(6)];
    angvel=[state(7);state(8);state(9);];
    stop = 0;
end
if nargin < 3 | isempty(options)
%Right hand side of the vector field
out1 = [vx; vy; vz;...
    -P.Kd*vx*absv+P.Km*(-omegay*vz+vy*omegaz);...
    -P.Kd*vy*absv+P.Km*(-omegax*vz+vx*omegaz);...
    -P.g-P.Kd*vz*absv-P.Km*(-omegax*vy+vx*omegay);...
    P.alpha*omegax*vx; P.alpha*omegay*vy; P.alpha*omegaz*vz;];
else
    switch(options)
        case 'events'        % used only if 'Events' is 'on'
            out1 = stop;     % Detect height = 0
            out2 = 1;        % Stop the integration
            out3 = -1;       % Negative direction only, namely decreasing through z=0
    end
end
end
```

## Calculating moment of inertia of the ball

```
function [inertia]= inertiaB(r2,mass)
% calculate moment of intertia of a shell (ball)
% mass in kilograms

% calculating parameters
density = 1375;
Volume=mass/density;
r1 = (r2^3 - (Volume/(4*pi/3)))^(1/3);
% moment of inertia equation - derived from triple integral
inertia=(2*mass/5)*((r2^5 - r1^5)/(r2^3 - r1^3));
```

## Calculate how bounce affect velocity and angular velocity

```
function [velOut, omegaOut] = bounce(velIn,omegaIn)
%function that takes the velocity and angular velocity just before the ball
%bounces, and outputs the velocity and angular velocity after the ball
```

```
%exits the bounce.
%INPUTS:  velIn    - starting velocity(m/s), 3 X 1 vector
%         omegaIn  - starting angular velocity(revolutions/s), 3 X 1 vector

eq = 1; %choose which formula to use
%setting constants
%coefficient of restitution
k=0.7;
%coefficient of friction
mu=0.4;
%radius of ball
r=0.02;

%converting rps to radians per second
omegaIn = omegaIn*2*pi;
%extracting input data
v_ix = velIn(1); v_iy = velIn(2); v_iz = velIn(3);
w_ix = omegaIn(1); w_iy = omegaIn(2); w_iz = omegaIn(3);

if eq == 1
    %velocity change equations
    v_ex = 0.4*w_iy*r + 0.6*v_ix;
    v_ey = -0.4*w_ix*r + 0.6*v_iy;
end
if eq == 2
    r1 = 1.96; r2 = 0.02;
    a = ((r2^5-r1^5)/(r2^3-r1^3));
%     N = (2*(r2^5 - r1^5))/((2*(r2^5 - r1^5))+(5*(r2^3 - r1^3)));
    N = a/(a+((5/2)*r2));
    %velocity change equations
    v_ex = (1 - N)*v_ix + N*w_iy*r2;
    v_ey = (1 - N)*v_iy - N*w_ix*r2;
end
%velocity change z equation
v_ez = -k*v_iz;
%angular velocity change equations
w_ex = 0.4*w_ix - (0.6*v_iy)/r;
w_ey = 0.4*w_iy + (0.6*v_ix)/r;
w_ez = w_iz;

%encoding data for output
velOut = [v_ex; v_ey; v_ez];
omegaOut = [w_ex; w_ey; w_ez];
%converting back to rps
omegaOut = omegaOut/(2*pi);
```

## Find orthogonal vectors

```
function [vecB1 vecB2] = findOrth(omega)
nOmega = omega/(norm(omega));
orth = null(nOmega(:).');
vecB1 = orth(:,1);
vecB2 = orth(:,2);
```

### Rotate orthogonal vectors

```
function vecA = rot(omega,vecB,theta)
%rotates about unit vector in 3d
u = omega/norm(omega);
uX = u(1); uY = u(2); uZ = u(3);
%rotation matrix
R = [cos(theta)+(uX^2)*(1-cos(theta)),uX*uY*(1-cos(theta))-uZ*sin(theta),...
    uX*uZ*(1-cos(theta))+uY*sin(theta);
    uX*uY*(1-cos(theta))+uZ*sin(theta),cos(theta)+(uY^2)*(1-cos(theta)),...
    uY*uZ*(1-cos(theta))-uX*sin(theta);
    uX*uZ*(1-cos(theta))-uY*sin(theta),uY*uZ*(1-cos(theta))+uX*sin(theta),...
    cos(theta)+(uZ^2)*(1-cos(theta))];
%times together
vecA = [R(1,1)*vecB(1) + R(1,2)*vecB(2) + R(1,3)*vecB(3),...
    R(2,1)*vecB(1) + R(2,2)*vecB(2) + R(2,3)*vecB(3),...
    R(3,1)*vecB(1) + R(3,2)*vecB(2) + R(3,3)*vecB(3)];
%normalise
vecA = vecA/norm(vecA);
```

### Animate balls path

```
function plot3d(path,X,Y,Z,X2,Y2,Z2,X3,Y3,Z3)
global lengthT; global width; global stepTime;
%making real size ball
r=0.02;
phi=linspace(0,pi,30);
theta=linspace(0,2*pi,40);
[phi,theta]=meshgrid(phi,theta);
x=r*sin(phi).*cos(theta);
y=r*sin(phi).*sin(theta);
z=r*cos(phi);

err=0.2;
[minX maxX minY maxY] = getLim(path,lengthT,width,err);
%stop motion animation
for i = 1:length(path)
    hold off
    plot3(path(:,1),path(:,2),path(:,3),'r');
    hold on
    %plotting table surface
    plotTable(lengthT,width,0.1)
    grid on
    axis equal
    a=path(i,1);b=path(i,2);c=path(i,3);
    surf(x+a, y+b, z+c)
    %plot angular velocity and spin
    plot3([path(i,1) X(i)],[path(i,2) Y(i)],[path(i,3) Z(i)],'g','LineWidth',2)
    plot3([path(i,1) X2(i)],[path(i,2) Y2(i)],[path(i,3) Z2(i)],'r','LineWidth',2)
    plot3([path(i,1) X3(i)],[path(i,2) Y3(i)],[path(i,3) Z3(i)],'r','LineWidth',2)
    %set axis scale
    set(gca,'DataAspectRatio',[1 1 1],...
        'PlotBoxAspectRatio',[1 1 1],...
        'XLim',[minX maxX],...
        'YLim',[minY maxY],...
```

```
        'ZLim',[-0.74 (max(path(:,3))+err)])
    pause(stepTime) %same as time step in ode45, so gives real time animation
    xlabel('x');ylabel('y');zlabel('z');
end
```

### Plot real size table tennis table

```
function plotTable(a,b,c)
%a = length
%b = width
%c = hieght

% plotting table
vertic = [0 -b/2 -c;a -b/2 -c;a b/2 -c;0 b/2 -c;0 -b/2 0;a -b/2 0;a b/2 0;0 b/2 0];
face = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
patch('Vertices',vertic,'Faces',face,'FaceVertexCData',[0 0.5 0.9],'FaceColor','flat')

lin = 0.01;
vertic = [0 -lin -c;a -lin -c;a lin -c;0 lin -c;0 -lin 0;a -lin 0;a lin 0;0 lin 0];
face = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
patch('Vertices',vertic,'Faces',face,'FaceVertexCData',[1 1 1],'FaceColor','flat')

% plotting net
w = 0.025; h = 0.1525; l = 0.05;
verticN = [a/2-w -b/2-l 0;a/2+w -b/2-l 0;a/2+w b/2+l 0;a/2-w b/2+l 0;...
    a/2-w -b/2-l h;a/2+w -b/2-l h;a/2+w b/2+l h;a/2-w b/2+l h];
faceN = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
patch('Vertices',verticN,'Faces',faceN,'FaceVertexCData',[0 0 0.9],'FaceColor','flat')

% plotting legs
w2=0.02; f=-0.74; leg = 3;
for i = [-1 1]
    for j = [1 3]
        verticN = [j*a/4-w2 i*b/leg-w2 f;j*a/4+w2 i*b/leg-w2 f;j*a/4+w2 i*b/leg+w2
        ↪  f;j*a/4-w2 i*b/leg+w2 f;...
            j*a/4-w2 i*b/leg-w2 -c;j*a/4+w2 i*b/leg-w2 -c;j*a/4+w2 i*b/leg+w2
            ↪  -c;j*a/4-w2 i*b/leg+w2 -c];
        faceN = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
        patch('Vertices',verticN,'Faces',faceN,'FaceVertexCData',[0 0
        ↪  0],'FaceColor','flat')
    end
end
```

### Get limits of 3d plot

```
function[minX maxX minY maxY] = getLim(path,lengthT,width,err)

if abs(min(path(:,2))) > (width/2)
```

```
    minY = min(path(:,2)) - err;
else
    minY = -((width/2) + err);
end
if abs(max(path(:,2))) > (width/2)
    maxY = max(path(:,2)) + err;
else
    maxY = (width/2) + err;
end

if abs(min(path(:,1))) > (0)
    minX = min(path(:,1)) - err;
else
    minX = -(0 + err);
end
if abs(max(path(:,1))) > (lengthT)
    maxX = max(path(:,1)) + err;
else
    maxX = (lengthT) + err;
end
```

### Making GUI for easy use

```
function varargout = GUI(varargin)
% GUI MATLAB code for GUI.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI.M with the given input arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI

% Last Modified by GUIDE v2.5 28-Mar-2017 11:52:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GUI_OutputFcn, ...
```

```
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI is made visible.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI (see VARARGIN)

% Choose default command line output for GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
posX = get(handles.edit1,'string');
posX = str2double(posX);
posY = get(handles.edit2,'string');
posY = str2double(posY);
posZ = get(handles.edit3,'string');
posZ = str2double(posZ);
velX = get(handles.edit15,'string');
velX = str2double(velX);
velY = get(handles.edit8,'string');
velY = str2double(velY);
velZ = get(handles.edit9,'string');
```

```
velZ = str2double(velZ);
wX = get(handles.edit10,'string');
wX = str2double(wX);
wY = get(handles.edit11,'string');
wY = str2double(wY);
wZ = get(handles.edit12,'string');
wZ = str2double(wZ);

runMain3d(posX,posY,posZ,velX,velY,velZ,wX,wY,wZ)
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a double


% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a double


% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a double


% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a double


% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double


% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%        str2double(get(hObject,'String')) returns contents of edit14 as a double


% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
posX = get(handles.edit1,'string');
posX = str2double(posX);
posY = get(handles.edit2,'string');
posY = str2double(posY);
posZ = get(handles.edit3,'string');
posZ = str2double(posZ);
velX = get(handles.edit15,'string');
velX = str2double(velX);
velY = get(handles.edit8,'string');
velY = str2double(velY);
velZ = get(handles.edit9,'string');
velZ = str2double(velZ);
wX = get(handles.edit10,'string');
wX = str2double(wX);
wY = get(handles.edit11,'string');
wY = str2double(wY);
wZ = get(handles.edit12,'string');
wZ = str2double(wZ);
frames = get(handles.edit14,'string');
frames = str2double(frames);
runMainStop(posX,posY,posZ,velX,velY,velZ,wX,wY,wZ,frames)



function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double


% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
↪  get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


% --------------------------------------------------------------------
function Untitled_2_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

### Stop frame for alternate plot

```
function runMainStop(posX,posY,posZ,velX,velY,velZ,wX,wY,wZ,frames)
global lengthT; global width; global stepTime;
lengthT = 2.7; width = 1.52;
wX=wX*2*pi; wY=wY*2*pi; wZ=wZ*2*pi;
%------------------------------------------------------------------------
%model calculations
contact = 0; contact2 = 0; countB = 0;
%calcuating first trajectory until bounce
[posB,velBB,omegaBB,path1,omegaPath1,contact] =
↪  trajectory([posX,posY,posZ],[velX,velY,velZ],[wX,wY,wZ]);
path = path1; omegaPath = omegaPath1;
%carry on bouncing until off the table
while contact == 1 && countB < 4
    %velocity and angular velocity change after the bounce
    [velAB,omegaAB] = bounce(velBB,omegaBB);
    %calculating second trajectory until bounce
    [posB,velBB,omegaBB,path1,omegaPath1,contact] = trajectory(posB,velAB,omegaAB);
    %concatinating both trajectories
    path = [path;path1]; omegaPath = [omegaPath;omegaPath1];
    countB = countB + 1;
end
%------------------------------------------------------------------------
%scaling angular velocity, for nice plotting
if (max(max(omegaPath))) ~= 0
    omegaPathN = omegaPath/(4*(max(max(omegaPath))));
else
    omegaPathN = omegaPath;
end
%working out angular velocity vectors
X = path(:,1) + omegaPathN(:,1);
Y = path(:,2) + omegaPathN(:,2);
Z = path(:,3) + omegaPathN(:,3);
%making equal spaced plot points in time
t=length(path); stop = [];
for i = 1:frames
    stop = round([stop ((t*i)/frames)]);
```

```
end
%initialising vectors for for loop
vec1 = zeros(length(path)+1,3); vec2 = zeros(length(path)+1,3);
%rotate orthogonal vectors by how much the ball has spun
for i = 1:length(path)
    if sum(omegaPathN(1,:)) ~= 0 && i == 1 %(length(path1))
        %find orthogonal vectors to angular velocity
        [a1 a2] = (findOrth(omegaPathN(i,:)));
        a1=a1'; a2=a2';
        vec1(i,:) = a1; vec2(i,:) = a2;
    end
    if sum(omegaPathN(1,:)) == 0 && i == (length(path1)+1)
        %find orthogonal vectors to angular velocity
        [a1 a2] = (findOrth(omegaPathN(i,:)));
        a1=a1'; a2=a2';
        vec1(i,:) = a1; vec2(i,:) = a2;
    end
    theta = sum(omegaPath(i,:))*stepTime;
    vec1(i+1,:) = rot(omegaPath(i,:),vec1(i,:),theta);
    vec2(i+1,:) = rot(omegaPath(i,:),vec2(i,:),theta);
end

%scale down to nice plotting
scale = 10; vec1 = vec1/scale; vec2 = vec2/scale;

%translate vectors to be in the ball's frame of reference
X2 = path(:,1) + vec1([1:length(path)],1);
Y2 = path(:,2) + vec1([1:length(path)],2);
Z2 = path(:,3) + vec1([1:length(path)],3);

X3 = path(:,1) + vec2([1:length(path)],1);
Y3 = path(:,2) + vec2([1:length(path)],2);
Z3 = path(:,3) + vec2([1:length(path)],3);

%making real size ball
r=0.02;
phi=linspace(0,pi,30);
theta=linspace(0,2*pi,40);
[phi,theta]=meshgrid(phi,theta);
x=r*sin(phi).*cos(theta);
y=r*sin(phi).*sin(theta);
z=r*cos(phi);

err=0.2;
[minX maxX minY maxY] = getLim(path,lengthT,width,err);

%plotting specified points in time
hold off
for i = stop
    plot3(path(:,1),path(:,2),path(:,3),'k');
    hold on
    %plotting table surface
    plotTable(lengthT,width,0.1)
    grid on
    axis equal
    a=path(i,1);b=path(i,2);c=path(i,3);
    surf(x+a, y+b, z+c)
    %plot angular velocity and spin
    plot3([path(i,1) X(i)],[path(i,2) Y(i)],[path(i,3) Z(i)],'g','LineWidth',2)
```

```
    plot3([path(i,1) X2(i)],[path(i,2) Y2(i)],[path(i,3) Z2(i)],'r','LineWidth',2)
    plot3([path(i,1) X3(i)],[path(i,2) Y3(i)],[path(i,3) Z3(i)],'r','LineWidth',2)
    %set axis scale
    set(gca,'DataAspectRatio',[1 1 1],...
        'PlotBoxAspectRatio',[1 1 1],...
        'XLim',[minX maxX],...
        'YLim',[minY maxY],...
        'ZLim',[-0.74 (max(path(:,3))+err)])
    pause(stepTime) %same as time step in ode45, so gives real time animation
    xlabel('x');ylabel('y');zlabel('z');
end
```