

Optimum Loft Angle for Greatest Carry Distance

Group D

Alison McIntosh

Emily Dark

Henry Archer

Kyle Stewart

Stuart Ballantyne

1 Response

The loft angle of the golf club which maximises the range of the golf ball trajectory is referred to as the optimum loft angle. This investigation consisted of two main parts: the impact of the club, and the flight of the golf ball. The golf ball considered is the Titleist Pro V1x, with characteristics outlined under (2.1) Assumptions.

Figure 1: Effect of club speed on carry distance and optimum loft angle, $\omega_{spin} = 300 \text{ rad s}^{-1}$

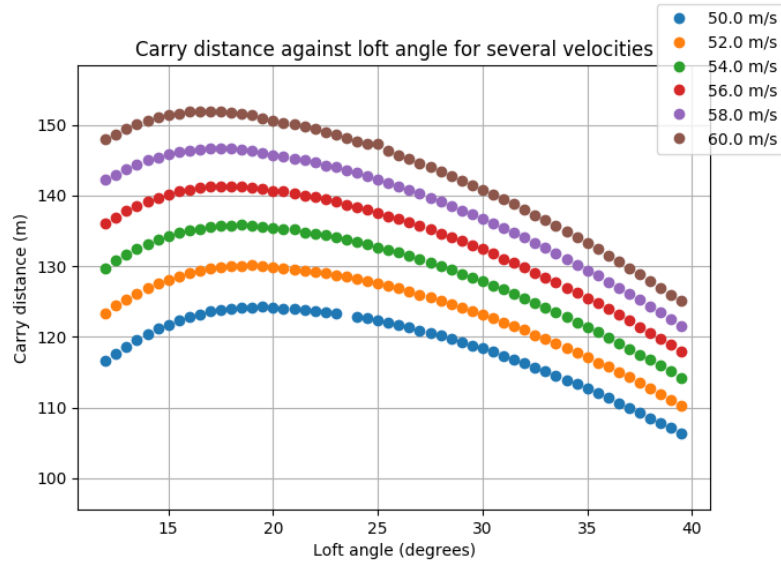


Figure 2: Effect of air density on carry distance and optimum loft angle, $\omega_{spin} = 300 \text{ rad s}^{-1}$, $v_i = 50 \text{ m s}^{-1}$

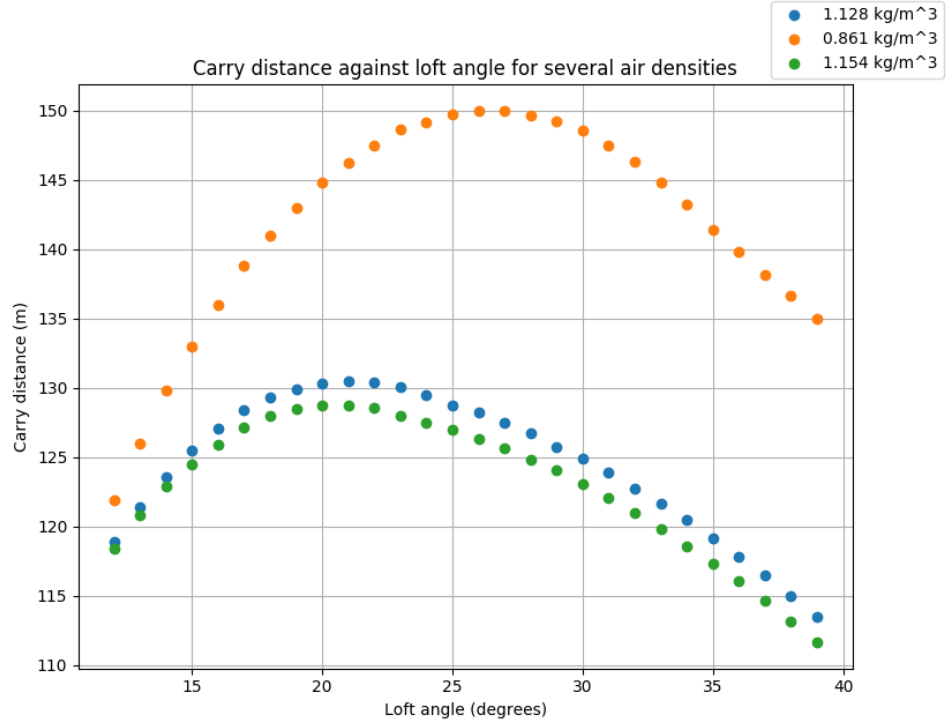
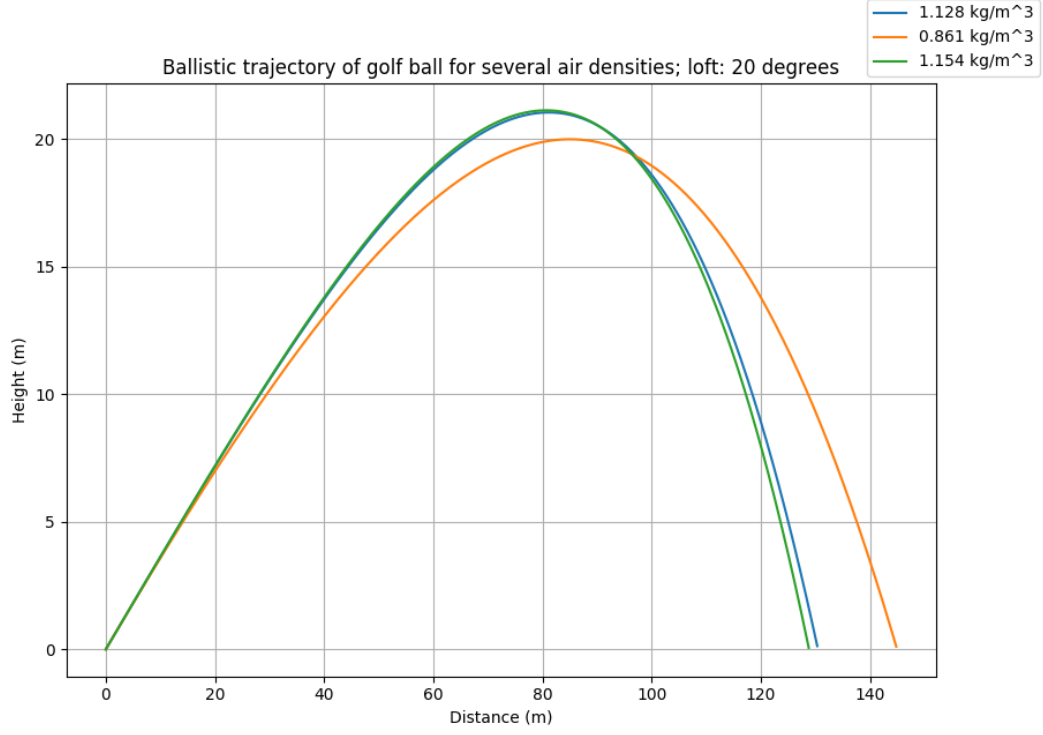


Figure 3: Effect of club speed on carry distance and optimum loft angle, $\omega_{spin} = 300 \text{ rad s}^{-1}$, $v_i = 50 \text{ m s}^{-1}$



1.1 Impact

1.1.1 Conservation of energy

The collision between the club head and golf ball is inelastic, meaning the kinetic energy of the system is not conserved throughout, but rather some of the energy is converted into different forms. One such form is elastic energy in the deformation of the golf ball. This has significant effect on the initial velocity of the golf ball, as the more the ball is deformed, the less kinetic energy the ball will have after the collision. The coefficient of restitution e is the ratio of the final and initial velocities between the golf ball and club head after the collision.

1.1.2 Conservation of linear momentum

Before the club head strikes the ball, the momentum of the system is entirely linear.

1.1.3 Conservation of angular momentum

After the collision, angular momentum exists in the form of backspin, which has great effect on lift.

1.2 Flight

1.2.1 Atmospheric conditions at different golf courses

Table 1: Atmospheric conditions at each golf course

Location	Temperature (K)	Humidity (%)	Altitude (m)	Pressure (kg/m^3)
Renaissance, East Lothian				1.13
La Paz, Bolivia				0.83
Sentosa, Singapore				1.15

1.2.2 Backspin of golf ball

When the club head strikes the ball, the golf ball slides up the face of the club head. Friction between these two surfaces causes the ball to rotate and when the ball leaves the face of the club head it is in a pure rolling state[1].

1.2.3 Dimpling

The air flow around a smooth ball is layered and quickly separates from the ball and creates a large drag. The air flow around a golf ball with dimples creates a layer of turbulence and delays the separation of air from the ball, therefore creating less drag.

2 Theory and model

2.1 Assumptions

The following assumptions are made throughout the report and model:

- golf course is level and has no effect on trajectory;
- height of the tee is negligible;
- gravitational field strength is constant (9.81 m/s^2) and does not fluctuate with height;
- driver is roughly a flat plate and strikes the ball precisely at the center, with no draw or fade;

- the mass of the club head is significantly greater than that of the shaft, so we consider the shaft's influence to be negligible;
- the golf ball is a Titleist Pro V1x, with a mass of 45.93 g, diameter of 42.67 mm, a moment of inertia of $0.009\,145\text{ g} \cdot \text{m}^2$, and 352 circular dimples.

2.2 Impact

The trajectory of the golf ball is directly influenced by two parameters which arise from the impact of the club head with the golf ball. These are:

- backspin of the ball (angular velocity $\vec{\omega}$);
- translational velocity of the ball (velocity \vec{v}).

These two parameters depend on loft angle and the initial velocity of the club. The laws of conservation of linear (1-2) and angular momentum were applied to the system and a coefficient of restitution used.

$$Mv_{cfn} + mv_{bfn} = Mv_{ci} \cos \theta \quad (1)$$

$$Mv_{cfp} + mv_{bfp} = -Mv_{ci} \sin \theta \quad (2)$$

v_{bfn} and v_{bfp} can be expressed in terms of v_{ci} , the initial club speed, as equations (3-4).

$$v_{bfn} = (1 + e)v_{ci} \frac{\cos \theta}{1 + \frac{m}{M}} \quad (3)$$

$$v_{bfp} = -v_{ci} \frac{\sin \theta}{1 + \frac{m}{M} + \frac{mr^2}{I}} \quad (4)$$

These two vector components can be added to yield v_{bo} and ϕ_{bo} , the velocity and its direction for the ball on departure; equations (5-6)

$$v_{bo} = v_{bf} = \sqrt{v_{bfn}^2 + v_{bfp}^2} \quad (5)$$

$$\phi_{bo} = \theta + \tan^{-1} \frac{v_{bfp}}{v_{bfn}} \quad (6)$$

Lieberman and Johnson give values for e decreasing from approximately 0.76 for impact speeds of 37 m s^{-1} to values of around 0.72 for impact speeds of 50 m s^{-1} . Applying a linear fit gives the empirical equation (7).

$$e = 0.86 - 0.0029v_{\text{impact}} \cos \theta \quad (7)$$

Table 2: Something something table

Initial club speed (ms^{-1})	Ball velocity on departure (ms^{-1})	Angle of departure ($^{\circ}$)	Backspin ($rads^{-1}$)
44.7			
51.4			
58.0			

2.3 Flight conditions

The main atmospheric factor which affects the golf balls flight is air density which is dependent on pressure, temperature and humidity of the air molecules. When the air density increases there will be more resistance against the ball during its flight, thus the maximum carry distance will be less. As the temperature of the air increases, the air molecules have greater kinetic energy, space out more and as thus occupy a larger volume, resulting in a decrease in air density. When the air pressure is increased, air density also increases as there are more collisions between particles occur. The relative humidity of air is a measure of the water vapour relative to temperature and is the percentage of water vapour that could potentially be held in the air at that given temperature.

When this relative humidity increases then since humid air is lighter than dry air due to the moist air having more vapour and less Nitrogen and Oxygen, which have a greater molar mass, then the air density of humid air will be less than for dry air. Therefore, the greater the humidity, the lower the air density. Relative humidity is given by equation (8):

$$\phi = \frac{P_w}{P'_w} \times 100 \quad (8)$$

where P_w is the pressure for water vapour and P'_w is the equilibrium vapour pressure, that is the maximum pressure it could be in that given temperature value. This value of P'_w can be obtained by Antoine's equation (9):

$$P'_w = e^{\frac{A-B}{C+T}} \quad (9)$$

where T is the temperature in K and A , B , and C are component specific constants for the given medium. Through the use of Dalton's law for partial pressures, the molar fraction of elements which compose the atmosphere can be readily calculated using equation (10):

$$x_w = \frac{P_w}{P_T} \quad (10)$$

where x_w is the molar fraction of water vapour and n_T is the total number of moles which is obtained using the ideal gas law (11):

$$P_TV = n_T RT \quad (11)$$

where P_T is the total pressure, which is found using the barometric formula (12):

$$P_T = P_0 e^{\frac{-Mg}{RT_0} h} \quad (12)$$

where P_0 and T_0 are, respectively, pressure and temperature at sea level (103 125 Pa, 288.15 K), g is the gravitational field strength, L is the teemperature lapse rate (0.0065 K m⁻¹). At higher altitudes, the air molecules can spread out further resulting in a decrease in air density.

The molar fraction of water vapour can be obtained through equation (13)

$$x_w = \frac{n_w}{n_T} \quad (13)$$

From here, it was a case of getting the number of moles for each major component of the atmosphere - nitrogen, oxygen, argon, and water vapour - using equation (14):

$$n_T - n_w = n_O + n_N + n_{Ar} \quad (14)$$

and considering the fraction of each component of air:

$$n_N = 0.7808(n_T - n_w) \quad (15)$$

$$n_O = 0.20195(n_T - n_w) \quad (16)$$

$$n_{Ar} = 0.0093(n_T - n_w) \quad (17)$$

By the consideration of the molar masses, the density of air can be calculated by equation (18):

$$\rho = \frac{((M_N P_N) + (M_O P_O) + (M_{Ar} P_{Ar}))}{RT} \quad (18)$$

where P is the partial pressure of an element, found by equation (19):

$$P = \frac{n_x RT}{V} \quad (19)$$

where n_x is the number of moles for a given element. Consequently, this allowed the air density to be found for each course based on environmental conditions.

2.4 Flight

2.4.1 Simple golf ball

A simple golf ball experiencing only weight may be modelled by the following system of differential equations;

$$a_x = \frac{\partial v_x}{\partial t} = 0 \quad (20)$$

$$a_y = \frac{\partial v_y}{\partial t} = -g \quad (21)$$

where g is the gravitational field strength.

Assuming the initial velocity is v_0 and the launch angle is θ , equations (20-21) can be solved to give equations (22-23):

$$v_x = v_0 \cos \theta \quad (22)$$

$$v_y = v_0 \sin \theta - gt \quad (23)$$

Integrating equations (22-23) with respect to time yields the displacement as a function of time to give equations (24-25):

$$x = v_0 t \cos \theta \quad (24)$$

$$y = v_0 t \sin \theta - \frac{1}{2}gt^2 \quad (25)$$

Figure 4: Golf ball trajectory, no drag or lift

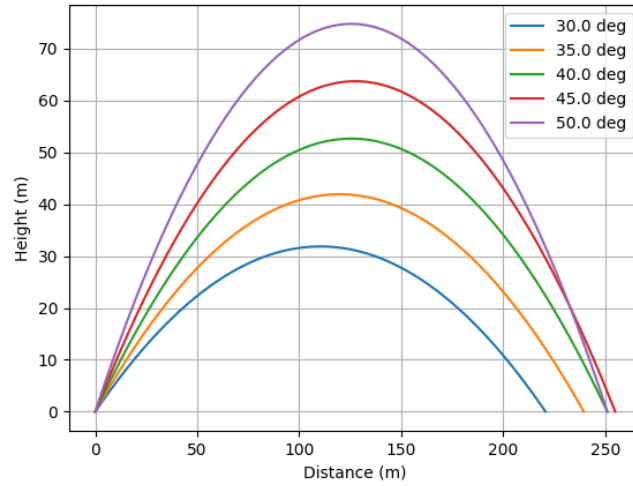
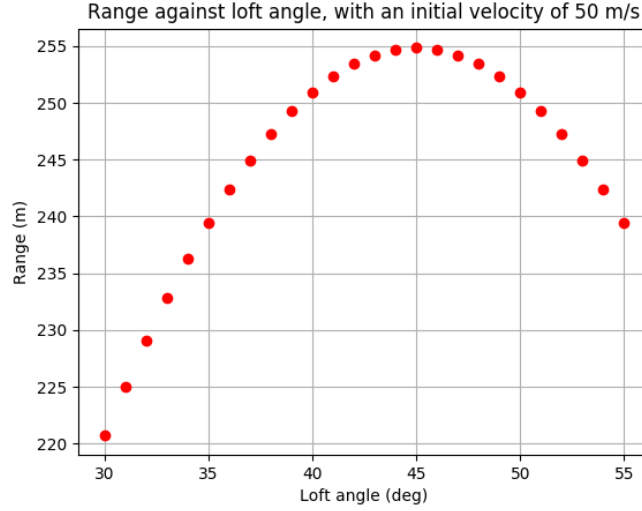


Figure 5: Range as a function of loft angle, no drag or lift



Figures (1-2) show the maximum range is when the loft angle at 45.0° , as predicted by the equations of projectile motion.

2.4.2 Smooth golf ball experiencing drag

The drag equation (26)

$$\vec{F}_d = \frac{1}{2}AC_d\rho_{air}|\vec{v}|\vec{v} \quad (26)$$

where ρ_{air} is the density of air; A is the reference area, which in the case of a smooth sphere of radius r , is the cross-sectional area πr^2 ; C_d is the coefficient of drag, which is dependent on the Reynolds number; and \vec{v} is the flow velocity relative to the golf ball. In this case, we assume the air is stationary and the golf ball is moving through the air with velocity \vec{v} .

Applying equation (26) to equations (20-21), we get

$$\frac{\partial v_x}{\partial t} = -k|v_x|v_x \quad (27)$$

$$\frac{\partial v_y}{\partial t} = -g - k|v_y|v_y \quad (28)$$

where $k = \frac{1}{2}AC_d\rho_{air}$. These equations already do not have a closed-form solution and require numerical methods.

Figure 6: Golf ball trajectory when experiencing drag but no lift, $C_d = 0.5$

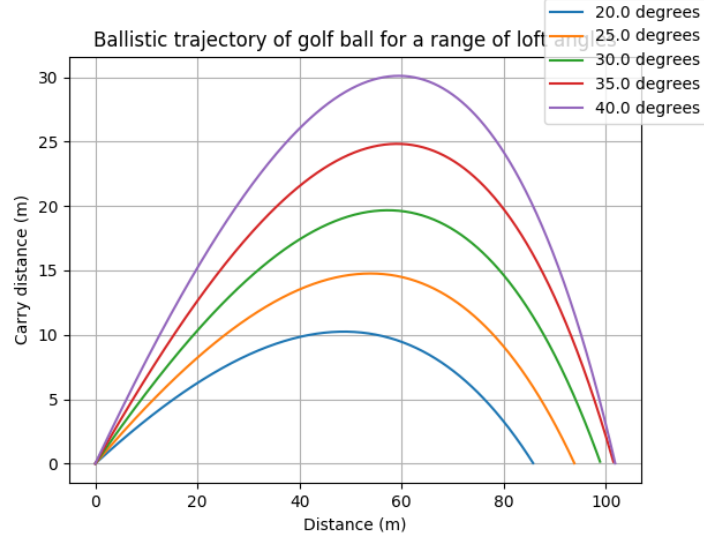


Figure 7: Range as a function of loft angle, $C_d = 0.5$



Figure (4) shows that the maximum range is achieved at 40.2° , for an initial velocity of 50 m s^{-1} . Additionally, the range is significantly decreased when drag was added to the model. The greatest range with drag is about 114m versus the previous range of 255 m at 45.0° . However, C_d is not constant and depends on the Reynolds number, which is proportional to the velocity of the golf ball. The Reynolds number

is given by:

$$R = \frac{2vr}{\nu} \quad (29)$$

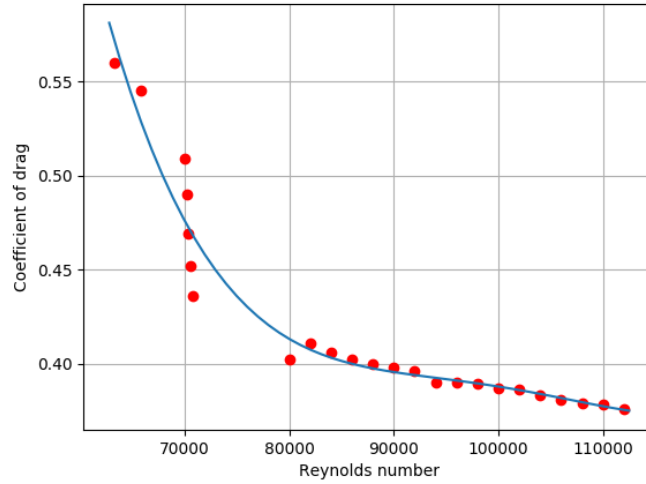
where ν is the kinematic viscosity of air.

Using data from [whereever we found the Reynolds number data], we used Python to construct a quartic fit as shown in Figure (5).

A quartic approximation is best as any greater degree would begin to oscillate too much at the edges (Runge's phenomenon). Likewise, a lesser degree would be too linear to give any meaningful relation.

One of the limitations of this approach is that we did not have any data above Reynolds number 112,000 (38.39 m s^{-1}) or below 63,300 (21.70 m s^{-1}), so the approximation would not hold for those conditions, and may in fact be drastically worse due to the chaotic behaviour of the polynomial at the aforementioned values. To get around this, the coefficient of drag is fixed at 0.8 for Reynolds numbers less than 53,000 and fixed at 0.37 for Reynolds numbers greater than 120,000.

Figure 8: Coefficient of drag as a function of Reynolds number with curve of best fit



Applying the approximation for the coefficient of drag, the optimum loft angle decreases down to about 34.5° , shown in figures (6-7).

Figure 9: Trajectory of golf ball with drag considering Reynolds number, $v_i = 50 \text{ m s}^{-1}$

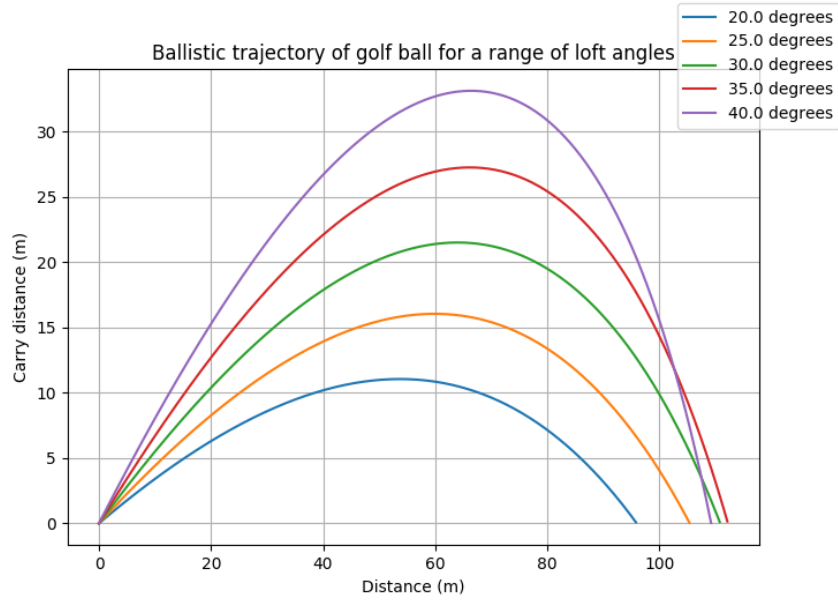
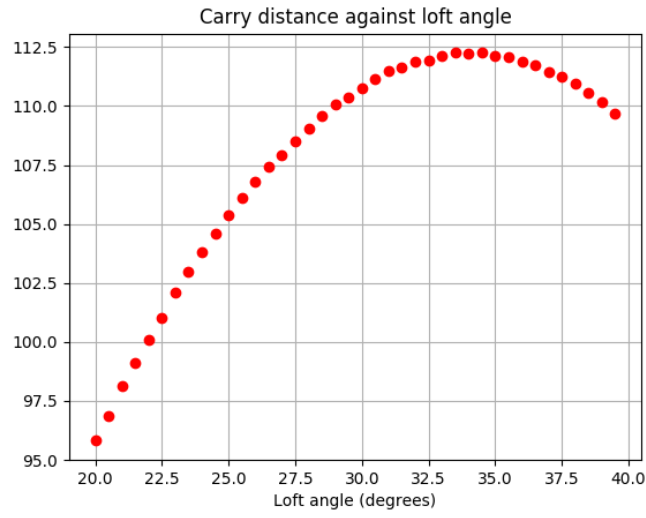


Figure 10: Range against loft angle for golf ball with drag considering Reynolds number, $v_i = 50 \text{ m s}^{-1}$



2.4.3 Smooth golf ball experiencing lift

Lift on a golf ball is caused by the Magnus effect, which is dependent on the backspin of the ball. The equation is similar to the drag equation, however the direction of the force is perpendicular to both angular velocity and translational velocity, unlike in the drag equation.

$$F_l = \frac{1}{2}\rho_{air}C_l|v|^2(\hat{\omega} \times \hat{v}) \quad (30)$$

where C_l is dependent on the spin parameter of the ball according to [whoever found this equation].

$$C_l = -3.25S^2 + 1.99S \quad (31)$$

The spin parameter is given by the ratio of the magnitude of tangential velocity to the magnitude of translational velocity.

$$S = \frac{r|\omega|}{|v|} \quad (32)$$

References

- [1] A. Raymond Penner. The physics of golf: The optimum loft of a driver. *American Journal of Physics*, 69(5):563–568, May 2001.

The model was written using Python version 3.8, on a Linux machine, although earlier versions (i.e. 3.5) should work. It requires the following Python packages: NumPy, SciPy, and Matplotlib. Use `./model3d.py -h` for a list of parameters. It also supports 3d plots, though these are experimental. The source code is also hosted as a git repository at <https://github.com/s-ballantyne/gdp>.

Listing 1: Python model)

```

1  #!/usr/bin/env python3
2
3  import numpy as np
4  import argparse
5
6  from scipy.integrate import odeint as integrate
7  from matplotlib import pyplot as plot
8  from numpy.linalg import norm
9  from mpl_toolkits.mplot3d import Axes3D
10
11  parser = argparse.ArgumentParser()
12
13  # Ball parameters
14  constants = parser.add_argument_group("Constants")
15  constants.add_argument("-m", "--mass", default=0.04593, help="Mass of ball (kg)")
16  constants.add_argument("-r", "--radius", default=0.04267 / 2, help="Radius of ball (m)")
17
18  constants.add_argument("-g", "--gravity", type=float, default=9.81, help="For when we get a Mars base (m/s/s)")
19  constants.add_argument("-d", "--density", type=float, default=1.225, help="Density of air (kg m-3)")
20  constants.add_argument("--viscosity", type=float, default=1.46e-5, help="Kinematic viscosity of air")
21
22  # Initial parameters
23  initialparams = parser.add_argument_group("Initial parameters")
24  initialparams.add_argument("-vi", "--velocity", type=float, default=50, help="Initial velocity (m/s)")
25  initialparams.add_argument("-yi", "--height", type=float, default=0, help="Initial height (m)")
26
27  initialparams.add_argument("-sp", "--spin", type=float, default=0, help="Spin (z)")
28  initialparams.add_argument("-spy", "--spiny", type=float, default=0, help="Spin (y)")
29  initialparams.add_argument("-spx", "--spinx", type=float, default=0, help="Spin (x)")
30
31  # Loft angle
32  parser.add_argument("-li", "--loftinitial", type=float, default=10, help="Loft angle (initial)")
33  parser.add_argument("-lf", "--loftfinal", type=float, default=20, help="Loft angle (final)")
34  parser.add_argument("-st", "--step", type=float, default=1, help="Loft angle (step)")
35
36  # Debugging
37  parser.add_argument("-v", "--verbose", action="store_true")
38
39  # Parse args
40  args = parser.parse_args()
41
42  # Input validation
43  assert args.loftfinal > args.loftinitial, "Final loft angle must be greater than initial loft angle!"
44  assert args.step != 0, "Step must be non-zero!"
45  assert ((args.loftfinal - args.loftinitial) / args.step).is_integer(), "Step size must divide the change in loft angle"
46
47  assert args.mass != 0, "Mass must be non-zero."
48  assert args.radius != 0, "Radius must be non-zero."
49  assert args.viscosity != 0, "Kinematic viscosity must be non-zero."
50  assert args.density != 0, "Density of air must be non-zero."
51
52  g = args.gravity
53  density = args.density
54
55
56  # Coefficient of drag from Reynolds number, based on degree four polynomial.
57  def re_to_cd(re):
58      # Clamp output value as it is only an approximation
59      if re > 120000:
60          return 0.370
61      elif re < 53000:
62          return 0.8
63
64      # Array of coefficients
65      coeffs = np.array([
66          9.46410458e-20, -3.80736984e-14,
67          5.72048806e-09, -3.81337408e-04,
68          9.92620188e+00
69      ])

```

```

70
71     # Return value of polynomial approximation
72     return np.polyval(coeffs, re)
73
74
75 # Linear velocity to Reynolds number (Re = velocity * diameter / k. viscosity)
76 def reynolds(velocity, radius):
77     return 2 * radius * velocity / args.viscosity
78
79
80 # Linear velocity to drag coefficient
81 def sphere_cd(velocity, radius):
82     cd = re_to_cd(reynolds(velocity, radius))
83     return cd
84
85
86 # Drag equation
87 # F_d = 1/2 * air density * ref. area * coefficient * |velocity| * v
88 def drag(density, area, cd, velocity):
89     return -0.5 * density * area * cd * norm(velocity) * velocity
90
91
92 # Lift equation
93 # F_l = 1/2 * air density * ref. area * coefficient * |v|^2 * (what x vhat)
94 def lift(density, area, cl, velocity, rvelocity):
95     if cl == 0:
96         return np.array([0, 0, 0])
97
98     S = 0.5 * density * area * cl
99
100     # Cross product of angular velocity and linear velocity, for direction of spin
101     rxv = np.cross(rvelocity, velocity)
102     rxv /= norm(rxv)
103
104     # Magnitude of spin is considered in coefficient of lift
105     return S * norm(velocity) ** 2 * rxv
106
107
108 # Simple golfball, no drag, no lift, smooth
109 class BasicGolfball:
110     def __init__(self):
111         # Properties
112         self.mass = args.mass
113         self.radius = args.radius
114
115         # Coordinates
116         self.x = 0
117         self.y = args.height
118         self.z = 0
119
120         self.vx = 0
121         self.vy = 0
122         self.vz = 0
123
124         # Rotational velocities
125         self.rvx = 0
126         self.rvy = 0
127         self.rvz = 0
128
129     # Reference area, for a sphere this is the cross-section.
130     def area(self):
131         return np.pi * self.radius ** 2
132
133     # Set initial velocity
134     def set_velocity(self, v, theta):
135         self.vx = v * np.cos(theta)
136         self.vy = v * np.sin(theta)
137
138     # Set spin
139     def set_spin(self, spin):
140         self.rvx, self.rvy, self.rvz = spin
141
142     # Get all coordinates
143     def coords(self):
144         return np.array([self.x, self.y, self.z, self.vx, self.vy, self.vz, self.rvx, self.rvy, self.rvz])
145
146     # Set all coordinates [x, y, z, vx, vy, vz, rvx, rvy, rvz]
147     def set_coords(self, coords):
148         self.x, self.y, self.z, self.vx, self.vy, self.vz, self.rvx, self.rvy, self.rvz = coords
149
150     # Returns numpy array of position coordinates

```



```

151     def position(self):
152         return np.array([self.x, self.y, self.z])
153
154     # Returns numpy array of velocity at the current position
155     def velocity(self):
156         return np.array([self.vx, self.vy, self.vz])
157
158     # Returns numpy array of acceleration at the current position
159     def acceleration(self):
160         return np.array([0, -g, 0])
161
162     # Returns numpy array of rotational velocity (spin) at the current position
163     def rvelocity(self):
164         return np.array([self.rvx, self.rvy, self.rvz])
165
166     # Returns numpy array of rotational acceleration at the current position
167     def racceleration(self):
168         return np.array([0, 0, 0])
169
170     # Returns numpy array of differential eqns to be solved by odeint
171     def differentials(self):
172         d = np.zeros(9)
173
174         d[0:3] = self.velocity()
175         d[3:6] = self.acceleration()
176
177         d[6:9] = self.racceleration()
178
179         return d
180
181     # (Internal) Updates coordinates and returns list of equations to solve (for odeint)
182     def __eqns(self, t, coords):
183         self.set_coords(coords)
184
185         if args.verbose:
186             print(t, self.velocity(), self.rvelocity(), self.acceleration(), self.racceleration())
187
188         return self.differentials()
189
190     # Solve for trajectory over given interval
191     def solve(self, t0, t1, dt=0.01):
192         interval = np.linspace(t0, t1, int((t1 - t0) / dt))
193         res = integrate(self.__eqns, self.coords(), interval, tfirst=True)[: , :3]
194
195         out = np.array([e for e in res if e[1] >= 0])
196         return out
197
198
199 # Simple golf ball but with drag
200 class DragGolfball(BasicGolfball):
201     def __init__(self):
202         BasicGolfball.__init__(self)
203
204     # Coefficient of drag from velocity & radius
205     def cd(self):
206         return sphere_cd(norm(self.velocity()), self.radius)
207
208     def acceleration(self):
209         fd = drag(density, self.area(), self.cd(), self.velocity())
210         return BasicGolfball.acceleration(self) + fd / self.mass
211
212
213 # Golfball with lift and drag
214 class LiftGolfball(DragGolfball):
215     def __init__(self):
216         DragGolfball.__init__(self)
217
218     # Returns spin factor
219     def spinf(self):
220         v = norm(self.velocity())
221         w = self.radius * norm(self.rvelocity())
222         return w / v
223
224     # Returns coefficient of lift based on spin factor
225     def cl(self):
226         s = self.spinf()
227         return -3.25 * s ** 2 + 1.99 * s
228
229     def acceleration(self):
230         fl = lift(density, self.area(), self.cl(), self.velocity(), self.rvelocity())
231         return DragGolfball.acceleration(self) + fl / self.mass

```

```

232
233 # Spin decreases by about 1% every second
234 def racceleration(self):
235     return -0.01 * self.rvelocity()
236
237
238 if __name__ == "__main__":
239     # Figure 1
240     plot.figure()
241     for theta in np.arange(args.loftinitial, args.loftfinal, args.step):
242         ball = LiftGolfball()
243         ball.set_velocity(args.velocity, np.radians(theta))
244         ball.set_spin([args.spinx, args.spiny, args.spin])
245
246         res = ball.solve(0, 10)
247         x, y, z = res.T
248
249         plot.plot(x, y, label=format(theta, ".1f") + " degrees")
250
251     plot.xlabel("Range (m)")
252     plot.ylabel("Height (m)")
253     plot.title("Ballistic trajectory of golf ball for several loft angles")
254     plot.grid(True)
255     plot.legend()
256
257     # Figure 2
258     plot.figure()
259     xdata = []
260     ydata = []
261     for theta in np.arange(10, 45, 1):
262         ball = LiftGolfball()
263         ball.set_velocity(args.velocity, np.radians(theta))
264         ball.set_spin([args.spinx, args.spiny, args.spin])
265
266         res = ball.solve(0, 10)
267         x, y, z = res.T
268
269         xdata.append(theta)
270         ydata.append(x[-1])
271
272     plot.plot(xdata, ydata, 'ro')
273
274     plot.grid(True)
275     plot.xlabel("Loft angle (m)")
276     plot.ylabel("Carry distance (m)")
277     plot.title("Carry distance against loft angle for v-i = " + format(args.velocity, ".1f") + " m/s")
278
279     # Show figures
280     plot.show()

```