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.

1.0.1 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

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

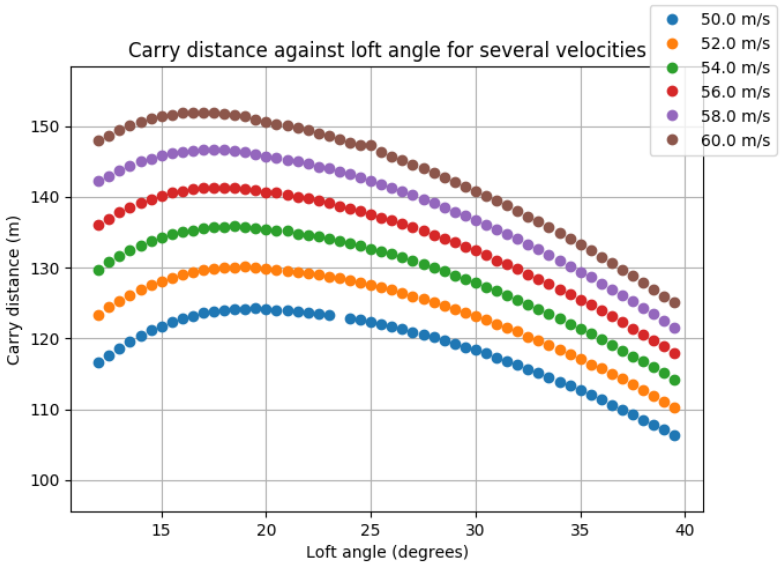


Figure 2: Effect of air density on carry distance and optimum loft angle, $\omega_{spin} = 300\,\mathrm{rad\,s^{-1}}$, $v_i = 50\,\mathrm{m\,s^{-1}}$

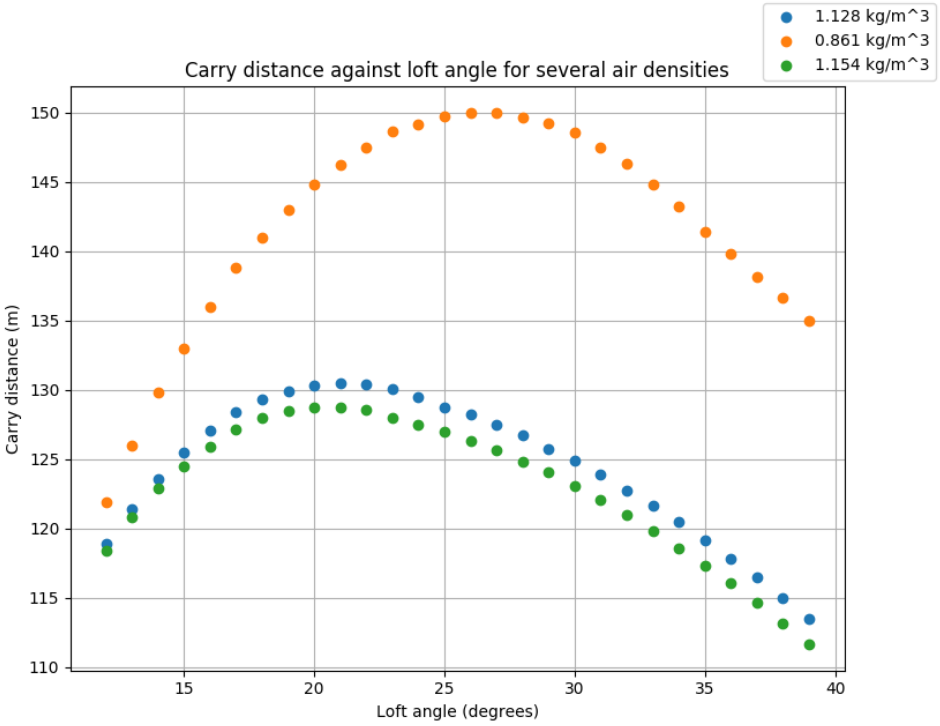
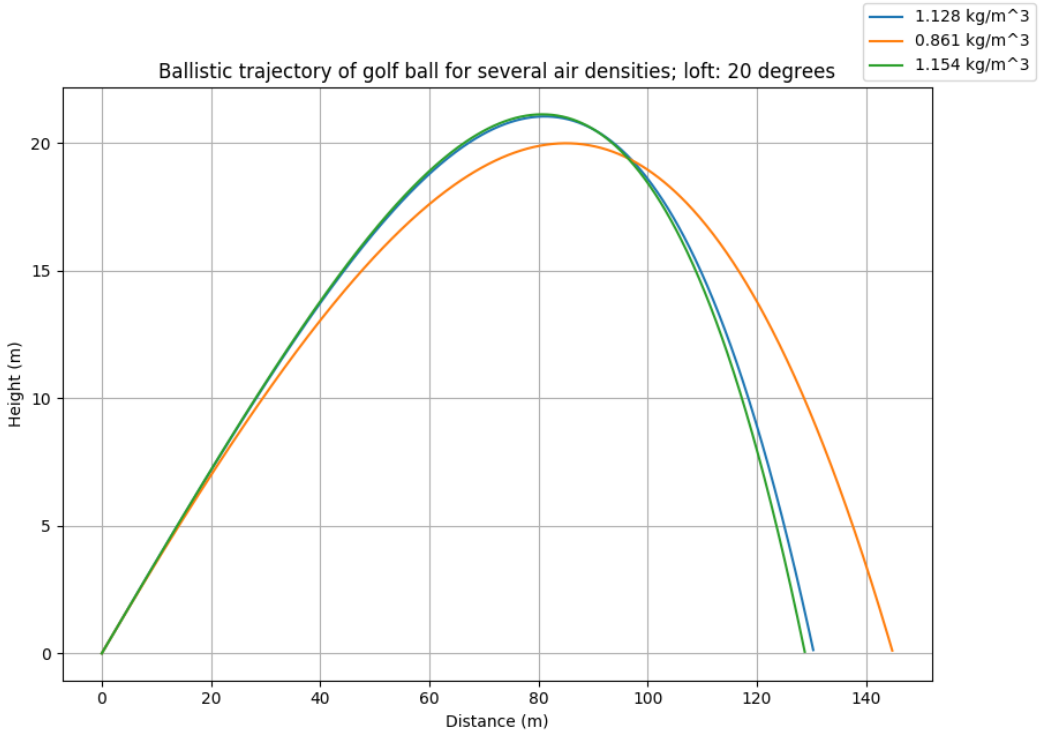


Figure 3: Effect of club speed on carry distance and optimum loft angle, $\omega_{spin} = 300\,\mathrm{rad\,s^{-1}}$, $v_i = 50\,\mathrm{m\,s^{-1}}$



2 Theory

2.1 Impact

2.1.1 Conservation of energy

The collision between the club head and golf ball is inelastic, meaning the kinetic energy of the system is not conserved, 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 and it is introduced in calculations. Further energy is lost as heat and sound.

$$Mv_{cfn} + mv_{bfn} = Mv_{ci}\cos\theta \tag{1}$$

$$Mv_{cfp} + mv_{bfp} = -Mv_{ci}\sin\theta \tag{2}$$

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

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

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

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)[4]

$$v_{bo} = v_{bf} = \sqrt{v_{bfn}^2 + v_{bfp}^2} \tag{5}$$

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

Lieberman and Johnson give values for e decreasing from approximately 0.76 for impact speeds of $37\,\mathrm{m\,s^{-1}}$ to values of around 0.72 for impact speeds of $50\,\mathrm{m\,s^{-1}}$. Applying a linear fit gives the empirical equation (7)[4].

$$e = 0.86 - 0.0029v_{impact}\cos\theta \tag{7}$$

Table 2: Something something table

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

2.2 Flight

2.2.1 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 [4].

2.2.2 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.

3 Theory and model

3.1 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 ($9.81\,\mathrm{m/s^2}$) and does not flucuate 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 neglible;

· the golf ball is a Titleist Pro V1x, with a mass of $45.93\,\mathrm{g}$, diameter of $42.67\,\mathrm{mm}$, a moment of inertia of $0.009\,145\,\mathrm{g}\cdot\mathrm{m}^2$, and 352 circular dimples.

3.2 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.

Table 3: Something something table

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

3.3 Flight conditions

The main atmospheric factor affecting the golf balls flight is air density which is dependent on air pressure, temperature and humidity. When air density increases, there is increased resistance against the ball during its flight, thus the maximum carry distance is less. Air molecules have greater kinetic energy as the temperature of the air increases, causing them to occupy a larger volume. This results in a decrease in air density. When the air pressure is increased, air density also increases as 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. The air density of humid air is less than dry air. Relative humidity is given by equation (19):

$$\phi = \frac{P_w}{P'_w} \times 100 \tag{8}$$

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) [5]:

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

where T is the temperature in K and A, B, and C are 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 calculated using equation (10):

$$x_w = \frac{P_w}{P_T} \tag{10}$$

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):

$$P_T V = n_T R T \tag{11}$$

where P_T is the total pressure, which is found using the barometric formula (12) [2]:

$$P_T = P_0 e^{\frac{-Mg}{RT_0}h} \tag{12}$$

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 temperature lapse rate ($0.0065\,\mathrm{K\,m^{-1}}$). 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)

$$x_w = \frac{n_w}{n_T} \tag{13}$$

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

$$n_T - n_w = n_O + n_N + n_{Ar} \tag{14}$$

and considering the fraction of each component of air:

$$n_N = 0.7808(n_T - n_w) \tag{15}$$

$$n_O = 0.20195(n_T - n_w) \tag{16}$$

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

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

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

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

$$P = \frac{n_x RT}{V} \tag{19}$$

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. [3]

3.4 Flight

3.4.1 Simple golf ball

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

$$a_x = \frac{\partial v_x}{\partial t} = 0 \tag{20}$$

$$a_y = \frac{\partial v_y}{\partial t} = -g \tag{21}$$

where g is the gravitional 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):

$$v_x = v_0 \cos\theta \tag{22}$$

$$v_y = v_0 \sin\theta - gt \tag{23}$$

Integrating equations (22-23) with respect to time yields the displacement as a function of time to give equations (24-25) [4]:

$$x = v_0 t \cos\theta \tag{24}$$

$$y = v_0 t \sin\theta - \frac{1}{2} g t^2 \tag{25}$$

Figure 4: Golf ball trajectory, no drag or lift

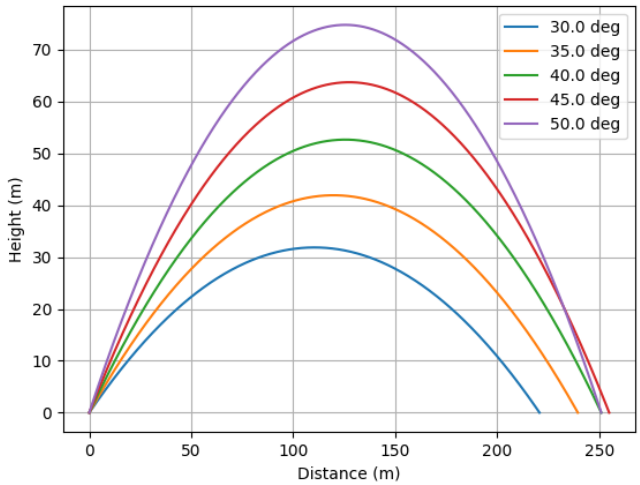
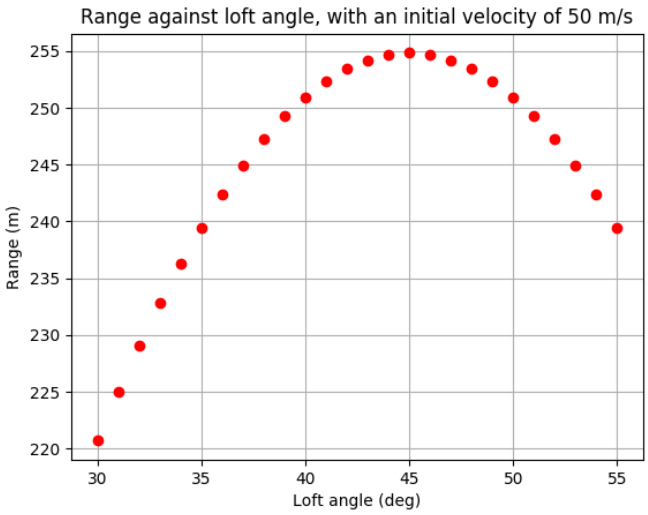


Figure 5: Range as a function of loft angle, no drag or lift



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

3.4.2 Smooth golf ball experiencing drag

The drag equation (26)

$$\vec{F}_d = \frac{1}{2} A C_d \rho_{air} |\vec{v}| \vec{v} \tag{26}$$

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

$$\frac{\partial v_x}{\partial t} = -k|v_x|v_x \tag{27}$$

$$\frac{\partial v_y}{\partial t} = -g - k|v_y|v_y \tag{28}$$

where $k = \frac{1}{2} A C_d \rho_{air}$. 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, $C_d = 0.5$

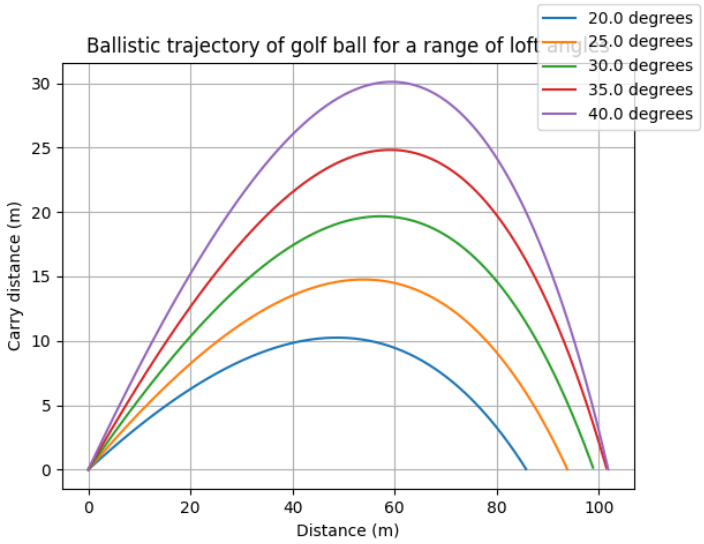


Figure 7: Range as a function of loft angle, $C_d = 0.5$

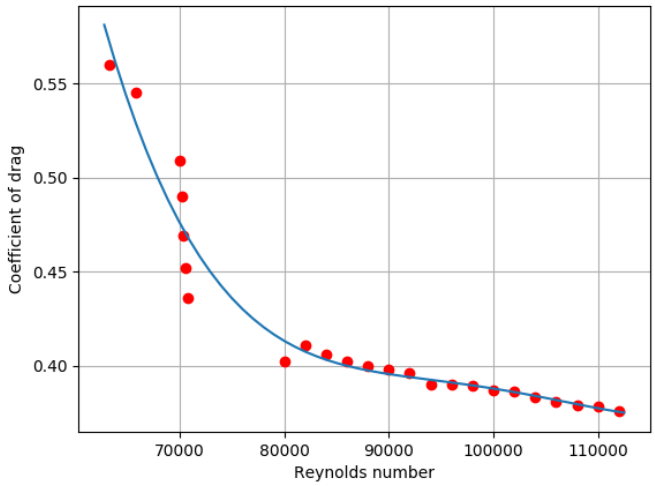
Figure (4) shows that the maximum range is achieved at 40.2°, for an initial velocity of $50\,\mathrm{m\,s^{-1}}$. Additionally, the range is significantly decreased when drag was added to the model. The greatest range with drag is about 114 m versus the previous range of 255 m at 45.0°. However, C_d 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 equation (29)[1]:

$$R = \frac{2vr}{\nu} \tag{29}$$

where ν is the kinematic viscosity of air.

Using data from *Alam 2011*[1], Python was used to construct a quartic fit of drag coeffiecient as a function of Reynolds Number, 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 limitions of this approach is that there is no data above Reynolds number $112,000$ ($38.39\,\mathrm{m\,s^{-1}}$) or below $63,300$ ($21.70\,\mathrm{m\,s^{-1}}$), so the approximation does 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 overcome this, the drag coefficient was 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: Drag coefficient as a function of Reynolds number with curve of best fit



Applying the approxmation 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, $v_i = 50\,\mathrm{m\,s^{-1}}$

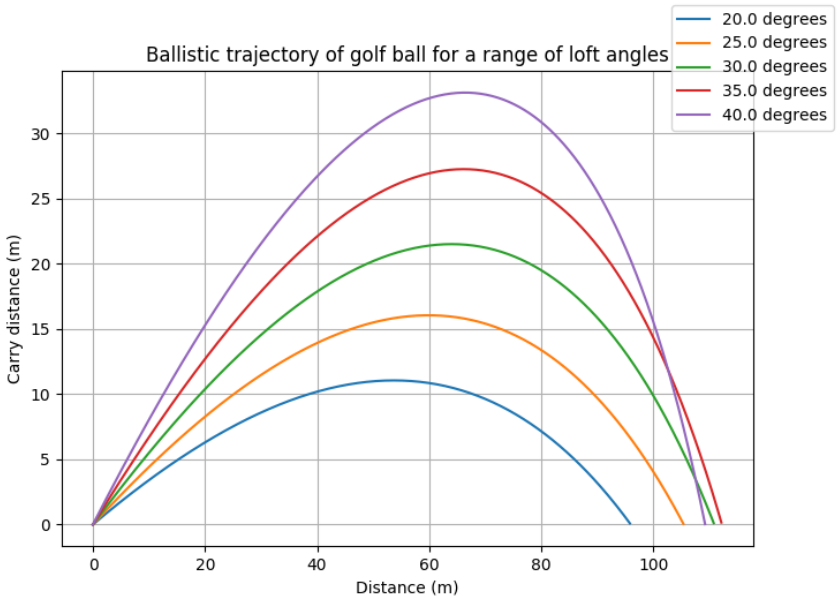
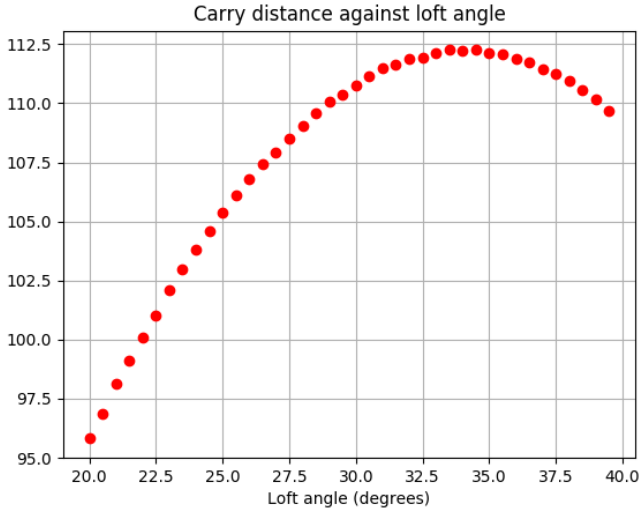


Figure 10: Range against loft angle for golf ball with drag considering Reynolds number, $v_i = 50\,\mathrm{m\,s^{-1}}$



3.4.3 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.

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

where C_l is dependent on the spin parameter of the ball according to [**?**].

$$C_l = -3.25S^2 + 1.99S \tag{31}$$

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

$$S = \frac{r|\omega|}{|v|} \tag{32}$$

References

[1] Firoz Alam, Tom Steiner, Harun Chowdhury, Hazim Moria, Iftekhar Khan, Fayez Aldawi, and Aleksandar Subic. A study of golf ball aerodynamic drag. *Procedia Engineering*, 13:226–231, 2011.

[2] Mário N. Berberan-Santos, Evgeny N. Bodunov, and Lionello Pogliani. On the barometric formula. *American Journal of Physics*, 65(5):404–412, May 1997.

[3] Roger Legg. Properties of humid air. In *Air Conditioning System Design*, pages 1–28. Elsevier, 2017.

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

[5] Denis Roizard. Antoine equation. In *Encyclopedia of Membranes*, pages 1–3. Springer Berlin Heidelberg, 2014.

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)

```
#!/usr/bin/env python3

import numpy as np
import argparse

from scipy.integrate import odeint as integrate
from matplotlib import pyplot as plot
from numpy.linalg import norm
from mpl_toolkits.mplot3d import Axes3D

parser = argparse.ArgumentParser()

# Ball parameters
constants = parser.add_argument_group("Constants")
constants.add_argument("-m", "--mass", default=0.04593, help="Mass of ball (kg)")
constants.add_argument("-r", "--radius", default=0.04267/2, help="Radius of ball (m)")

constants.add_argument("-g", "--gravity", type=float, default=9.81, help="For when we get a Mars base (m/s/s)")
constants.add_argument("-d", "--density", type=float, default=1.225, help="Density of air (kg m^-3)")
constants.add_argument("--viscosity", type=float, default=1.46e-5, help="Kinematic viscosity of air")

# Initial parameters
initialparams = parser.add_argument_group("Initial parameters")
#initialparams.add_argument("-vi", "--velocity", type=float, default=50, help="Initial velocity (m/s)")
initialparams.add_argument("-yi", "--height", type=float, default=0, help="Initial height (m)")

#initialparams.add_argument("-sp", "--spin", type=float, default=0, help="Spin (z)")
#initialparams.add_argument("-spy", "--spiny", type=float, default=0, help="Spin (y)")
#initialparams.add_argument("-spx", "--spinx", type=float, default=0, help="Spin (x)")

# Loft angle
parser.add_argument("-li", "--loftinitial", type=float, default=10, help="Loft angle (initial)")
parser.add_argument("-lf", "--loftfinal", type=float, default=35, help="Loft angle (final)")
parser.add_argument("-st", "--step", type=float, default=5, help="Loft angle (step)")

# Debugging
parser.add_argument("-v", "--verbose", action="store_true")

# Ball speed calculations
parser.add_argument("--clubmass", type=float, default=0.2, help="Mass of club head (kg)")
parser.add_argument("--vclub", type=float, default=51.4, help="Club speed (m/s)")
parser.add_argument("--inertia", type=float, default=9.145e-6, help="Inertia of golf ball")

# Parse args
args = parser.parse_args()

# Input validation
assert args.loftfinal > args.loftinitial, "Final loft angle must be gretaer than initial loft angle!"
assert args.step != 0, "Step must be non-zero!"
assert ((args.loftfinal - args.loftinitial) / args.step).is_integer(), "Step size must divide the change in loft angl

assert args.mass != 0, "Mass must be non-zero."
assert args.radius != 0, "Radius must be non-zero."
assert args.viscosity != 0, "Kinematic viscosity must be non-zero."
assert args.density != 0, "Density of air must be non-zero."

g = args.gravity
density = args.density

# Ball speed from club speed and loft angle
def ball_speed(theta):
    theta = np.radians(theta)
    e = 0.86 - 0.0029 * args.vclub * np.cos(theta)

    bfn = (1 + e) * args.vclub * np.cos(theta) / (1 + args.mass / args.clubmass)
    bfp = args.vclub * np.sin(theta) / (1 + args.mass / args.clubmass + (args.mass * args.radius**2 / args.inertia))
    return np.sqrt(bfn**2 + bfp**2)

# Spin
```

```
70  def ball_spin(theta):
71      theta = np.radians(theta)
72      bfp = args.vclub * np.sin(theta) / (1 + args.mass / args.clubmass + (args.mass * args.radius**2 / args.inertia))
73
74      return args.mass * bfp * args.radius / args.inertia
75
76  # Coefficient of drag from Reynolds number, based on degree four polynomial.
77  def re_to_cd(re):
78      # Clamp output value as it is only an approximation
79      if re > 120000:
80          return 0.370
81      elif re < 53000:
82          return 0.8
83
84      # Array of coefficients
85      coeffs = np.array([
86                9.46410458e-20, -3.80736984e-14,
87                5.72048806e-09, -3.81337408e-04,
88                9.92620188e+00
89              ])
90
91      # Return value of polynomial approximation
92      return np.polyval(coeffs, re)
93
94
95  # Linear velocity to Reynolds number (Re = velocity * diameter / k. viscosity)
96  def reynolds(velocity, radius):
97      return 2 * radius * velocity / args.viscosity
98
99
100 # Linear velocity to drag coefficient
101 def sphere_cd(velocity, radius):
102     cd = re_to_cd(reynolds(velocity, radius))
103     return cd
104
105
106 # Drag equation
107 # F_d = 1/2 * air density * ref. area * coefficient * |velocity| * v
108 def drag(density, area, cd, velocity):
109     return -0.5 * density * area * cd * norm(velocity) * velocity
110
111
112 # Lift equation
113 # F_l = 1/2 * air density * ref. area * coefficient * |v|^2 * (what x vhat)
114 def lift(density, area, cl, velocity, rvelocity):
115     if cl == 0:
116         return np.array([0, 0, 0])
117
118     S = 0.5 * density * area * cl
119
120     # Cross product of angular velocity and linear velocity, for direction of spin
121     rxv = np.cross(rvelocity, velocity)
122     rxv /= norm(rxv)
123
124     # Magnitude of spin is considered in coefficient of lift
125     return S * norm(velocity)**2 * rxv
126
127 # Simple golfball, no drag, no lift, smooth
128 class BasicGolfball:
129     def __init__(self):
130         # Properties
131         self.mass = args.mass
132         self.radius = args.radius
133
134         # Coordinates
135         self.x = 0
136         self.y = args.height
137         self.z = 0
138
139         self.vx = 0
140         self.vy = 0
141         self.vz = 0
142
143         # Rotational velocities
144         self.rvx = 0
145         self.rvy = 0
146         self.rvz = 0
147
148     # Reference area, for a sphere this is the cross-section.
149     def area(self):
150         return np.pi * self.radius**2
```

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

```
232 class LiftGolfball(DragGolfball):
233     def __init__(self):
234         DragGolfball.__init__(self)
235 
236     # Returns spin factor
237     def spinf(self):
238         v = norm(self.velocity())
239         w = self.radius * norm(self.rvelocity())
240         return w / v
241 
242     # Returns coefficient of lift based on spin factor
243     def cl(self):
244         s = self.spinf()
245         return -3.25 * s**2 + 1.99 * s
246 
247     def acceleration(self):
248         fl = lift(density, self.area(), self.cl(), self.velocity(), self.rvelocity())
249         return DragGolfball.acceleration(self) + fl / self.mass
250 
251     # Spin decreases by about 1% every second
252     def racceleration(self):
253         return -0.01 * self.rvelocity()
254 
255 
256 if __name__ == "__main__":
257     # Initial conditions
258     for density in [1.128, 0.861, 1.154]:
259         plot.figure()
260         for theta in np.arange(args.loftinitial, args.loftfinal, args.step):
261             ball = LiftGolfball()
262             ball.set_velocity(ball_speed(theta), theta)
263             ball.set_spin([0, 0, ball_spin(theta)])
264 
265             res = ball.solve(0, 10)
266             x, y, z = res.T
267 
268             plot.plot(x, y, label="Loft angle: " + format(theta, ".1f"))
269 
270         plot.legend()
271         plot.grid(True)
272         plot.xlabel("Distance (m)")
273         plot.ylabel("Height (m)")
274         plot.title("Ballistic trajectory for air density " + format(density, ".3f") + " kg/m^3")
275 
276         plot.figure()
277         xdata = []
278         ydata = []
279         for theta in np.arange(5, 30, 0.5):
280             ball = LiftGolfball()
281             ball.set_velocity(ball_speed(theta), theta)
282             ball.set_spin([0, 0, ball_spin(theta)])
283 
284             res = ball.solve(0, 10)
285             x, y, z = res.T
286 
287             xdata.append(theta)
288             ydata.append(x[-1])
289 
290         plot.plot(xdata, ydata, 'o', label="Air density: " + format(density, ".3f"))
291         plot.legend()
292         plot.grid(True)
293         plot.xlabel("Loft angle (m)")
294         plot.ylabel("Carry distance (m)")
295 
296     plot.show()
```