Building a Better Bomb Code

In this exercise you will use what you have learned so far about Python to write a program to solve a more realistic problem in aerodynamics – a flour bomb which has air resistance. You will also learn how to plot graphs with more than one curve, and about "*global*" variables.

The main goal of this exercise is to write a program similar to the previous exercises – which computed the trajectory of a bomb released from an airplane – but now taking into account the drag forces on the projectile as it falls.

Once we include the force of drag we must make some assumptions about the size and shape of the projectile. We will assume that the "bomb" is a small sack of flour (about one pound or so) which is dropped from a small airplane. Pilots do this sometimes for entertainment to see who can get closest to a target on the ground. A good first guess for the altitude and airspeed of this "Flour Bomb" is that it would be dropped from around 200 feet at about 75 knots.

Drag Force

The drag force exerted on an object moving through a fluid (such as air) is generally proportional to a quantity q, called the "dynamic pressure," which is the pressure that would be exerted on the front of the object by bringing a small parcel of the impinging fluid to rest. Since this is simply the conversion of the kinetic energy of the fluid to the potential energy of pressure exerted on the object, the dynamic pressure is given by[1]

$$q = \tfrac{1}{2}\rho v^2 \,, \tag{1}$$

where ρ is the density of the fluid and v is the velocity of the fluid. The drag force on the object is also proportional to the cross sectional area A the object presents to the fluid. There are various ways to define what is meant by the "cross sectional area," but disregarding such distinctions for the moment this means that the drag force, F_D can be written as

$$F_D = C_D \, q \, A \tag{2}$$

where C_D is a dimensionless constant of proportionality known as the "coefficient of drag." Using Eq. (1) then yields

$$F_D = \tfrac{1}{2} C_D A \rho v^2 \,. \tag{3}$$

Thus the drag force on the object is proportional to the cross sectional area of the object, the density of the fluid, and the square of the velocity of (or through) the fluid. Drag force

[1] Hoerner, Sighard F., "Fluid-Dynamic Drag" (Hoerner Fluid Dynamics, Albuquerque, NM, 1965)

always acts to resist motion, which means that it acts to slow down the object as it moves through the fluid.

Acceleration

The effect of the drag force on the bag of flour as it falls can be computed by applying Newton's law that $\mathbf{F} = m\mathbf{a}$. The acceleration of the bag will be $\mathbf{a} = \mathbf{F}/m$, where $\mathbf{F}$ is the sum of all forces acting on the bag and m is the mass of the bag. It will be useful to consider the forces (and accelerations) in the horizontal (x) and vertical (y) directions separately. (In other words, $\mathbf{F}$ is a "vector.") Solving first for the acceleration of the bag in the horizontal direction gives:

$$a_x = -\frac{C_D A \rho v v_x}{2m} \tag{4}$$

where the speed $v = \sqrt{v_x^2 + v_y^2}$. Note that every component of the right hand side of this equation is positive, except for v_x, which may be positive or negative. With the overall minus sign, this means that the acceleration in this direction will always be in the opposite direction of the velocity, and will act to slow it down in this direction. For example, if the velocity component v_x is negative (the bag is moving in the negative x-direction) then the a_x is positive, so that the drag force still acts to slow the object down. This is exactly as it should be, so we don't need to take special steps to get the direction of the acceleration correct.

In the vertical direction, the force of gravity is negative, because it always pulls the object downward, and combining gravity with the drag force gives

$$a_y = -g - \frac{C_D A \rho v v_y}{2m} \tag{5}$$

If the velocity component v_y is negative (which means that the object is falling downward) then the contribution due to the drag force is positive, again against the direction of motion. Drag always acts against the direction of motion, and this equation automatically incorporates that idea.

Computer Simulation of Varying Forces

Computing the path of the projectile is now more complicated than before, because the forces on it change with time. We will model the motion of the bag of flour as it falls by taking many small time-steps, each of size Δt, and at each new time we will compute the new acceleration on the bag, the new velocity of the bomb, and then the new position of the bomb. We must be careful about the order in which we compute these quantities. We don't want to put a new value in a variable until we no longer need the old value.

Let `x` and `y` be the position of the bomb, in meters, and `vx` and `vy` be it's velocity in the horizontal and vertical directions, in meters/second. Let `ax` and `ay` be the acceleration of the bomb in the horizontal and vertical directions, in m/s^2, and let `mass` be the mass of

the bomb, in kg. Then for each time step the acceleration in both directions is given by something like:

```
speed = (vx**2 + vy**2)**0.5
ax = -Cdrag*Area*airden*speed*vx/(2.0*mass)
ay = -Cdrag*Area*airden*speed*vy/(2.0*mass)
ay = ay - g
```

Here `airden` is ρ, the density of the air, `Cdrag` is C_D, the coefficient of drag, `Area` is the cross sectional area of the bag, and `g` is the acceleration due to gravity (9.807 m/s^2).

Given the accelerations above, the new values of the velocities in both directions are

```
vx = vx + ax*dt
vy = vy + ay*dt
```

Here `dt` is Δt, the size of our time steps. With the new velocities we can then compute the new positions:

```
x = x + vx*dt
y = y + vy*dt
```

This series of computations is to be repeated (with `t` increased by `dt` on each step) until the bomb strikes the ground (`y` becomes less than or equal to zero) or until the time in the air reaches 30 seconds.

Terminal Speed

As the bomb falls under the force of gravity it picks up speed, and therefore the drag force acting upon it increases. Eventually the drag force will match the force of gravity. When this equilibrium is reached the bomb will no longer accelerate. The speed at which this happens – the speed at which the drag force exactly matches the force of gravity – is called the "terminal speed" of the bomb, because it is the final velocity attained by the falling object. In the computer simulation, the signal that the terminal speed has been reached is that the acceleration is at or near zero.

Coefficient of Drag

The exact value of the coefficient of drag of a bag of flour is unknown. The coefficient of an infinite flat plate should be 1.0, but the coefficient of drag of a perfect cube is $C_D = 1.05$, and for a perfect sphere it is 0.47. Presumably the coefficient of drag for a flour bomb is between these last two values, but we cannot know this for sure. Because we do not know the exact value for the coefficient of drag, we will compute the trajectory of the bomb for a number of different values of C_D.

If we could drop several flour bombs of known size and mass from an airplane at a known altitude and airspeed, then measuring where the bomb lands would let us estimate

from our model what is the actual coefficient of drag of a bag of flour. Once C_D is known we can compute where subsequent flour bombs will land. f

Plotting Multiple Curves

In an earlier exercise you learned how to plot a graph in Python using the MatPlotLib module. Plotting more than one curve on the same graph is almost as easy. First, begin with

```
import matplotlib.pyplot as plt
plt.figure()
```

early on, before you do any calculations, to set things up. This just starts the creation of the graph. Then, within your nested loops, your script would repeatedly clear and fill arrays for x and y values and then call

```
plt.plot(xray,yray)
```

to add another curve to the graph. Finally, after all of the loops have finished, add the plot title and axes labels and then save and/or show the plot as before. That's all there is to it.

Global Variables

As you have already learned, you can pass values to a function via the list of arguments. Functions can also make use of the values of variables which are defined outside of the body of the function. Variables defined outside of any function are said to have a "global scope." In contrast, variables which are defined inside the body of a function only exist inside that function, and are said to have "local" scope.

One thing you have to be careful about is assigning a value to a variable with global scope from inside a function. This will cause an error, unless you have declared the global variable to be global in that function, using the `global` statement. For example, the function `dropit()` from the previous exercise could be set up to keep a count of how many times it has been used, like so:

```
Ndropit = 0

def dropit(Altitude, Vx0) :
       global Ndropit
       Ndropit = Ndropit + 1
       ...
```

Declaring the variable `Ndropit` to be global inside the function is required to be able to modify its value. Try commenting out the `global` statement and see what happens.

String Variables

In previous assignments you have already been introduced to character string "constants," sometimes called "literals." In fact, you used the string literal "Hello, World!" in your first Python program.

Almost any text enclosed between double quotes is a string literal. Single quotes also work. It is also possible to assign string values to variables, and to ask for string input instead of numerical values, and use string variables in `print()` statements.

The `input()` function actually returns a character string value, which is why you have to then feed the result to the `float()` function, which converts whatever argument it is given to a floating point number. So to put someone's name into a variable as a character string you could simply say something like:

```
their_name = input("What is your name? ")
```

You will easily find lots of information about Python strings online.*

Assignment

The overall goal of this exercise is to simulate the fall of an object dropped from an airplane, now including air resistance. Your code will step the object forward in very small increments of time until it hits the ground. Your script will repeat this calculation for a variety of different values of the coefficient of drag. The results will be reported both in a table and in a graph with multiple curves.

To complete this exercise you must do the following:

1. **Introduction:** As before, your script should print a brief introduction that describes what it does.
2. **Inputs:** Your program should first read in a line of information describing the flour bomb. In particular, the cross sectional area of the bomb (in square centimeters) and the mass (in grams), in that order. These should be converted to square meters and kilograms, respectively

   Next, your script should read the conditions under which the bomb is to be dropped. In particular, the altitude (in feet) and the airspeed (in knots), in that order, on a single line. These should be converted to meters, and meters per second, respectively. All internal calculations will now use the metric system.

   You will also need to know the density of air, ρ, but instead of reading in a value you should simply use the value $\rho = 1.29$ kg/m^3, which is the density of air under "standard" atmospheric conditions.

   Finally, read in a character string label for the experiment. Your script should read this in last, but it should print it out in the first line of the final report.

* For example, see `https://www.w3schools.com/python/python_strings.asp`

3. **Coefficient of Drag:** Your script should simulate the dropping of the bomb from the same initial altitude and airspeed for 51 evenly-spaced values for the coefficient of drag, C_D, ranging from zero to 1.25. It is important to include zero, because we already know the answer for $C_D = 0.0$ from the previous programs (or simple pencil and paper calculations). It is also a good idea to extend the values beyond what is expected, to be sure that we really get the true value within the range of values studied. We want to use many values, so that we can pin down the true value of C_D as accurately as possible, but at the same time we want the whole table to fit on one page.

4. **Computation:** For each value of the coefficient of drag, your script will compute the complete trajectory of the falling object until it hits the ground (or until 30 seconds have passed). You may again assume that the bomb is dropped from level flight.

   You script must do this using a function, called `stepit()`, which takes as input parameters ("arguments") the coefficient of drag, C_D, then the initial altitude in meters, and then the initial forward speed, in meters per second, in that order. The function must return the results as a list which contains the time, then the position of the impact, then the velocity at impact, and then the acceleration at impact. Each of the position, velocity, and acceleration will be lists containing two values, the first being the horizontal value and the second being the vertical value (so each represents the components of a "vector"). Thus the function will return a list containing several lists.

   Your `stepit()` function should not print anything during the drop. The function should use the same variable names internally for position, velocity, and acceleration as used above, to make it easier for the instructor to read.

   To compute the drag force your function also needs to know the cross sectional area of the bag of flour and its mass, as well as the density of air. These are to be obtained using "global" variables.

5. **Output:** A typical printer page has 66 lines on it, so we want to use less than 66 different values for C_D. If we divide the range 0.0 to 1.25 into 50 intervals we get a regular and orderly spacing between values. If there are 50 *intervals*, then there should be 51 *values* of C_D.

   The results of the computations which will be of interest are the time and position of impact, as well as the velocities and accelerations at impact (to determine whether the terminal speed has been reached). All of this information should be displayed in a well-organized table, with one line for each value of C_D. The value of C_D should be the first item on each line, followed by the time of flight, then position, velocity and acceleration information. For each of position, velocity, and acceleration, display first the horizontal value and then the vertical value. These distances should be converted back into units of feet.

   Your table should have a width of no more than 80 characters, and each column in the table should have a title at the top naming the quantity listed

and its units. The numerical values shown in the table should not have excessive digits past the decimal point.

Additional information about the simulation should be displayed above the table, starting with the character string label for the experiment. After that, report the size and mass of the bag and the altitude and airspeed of the drop. Don't forget to include units for all numbers.

6. **Graph:** Your script should also produce a single graph, created using the MatPlotLib `pyplot` module, showing the paths of *all* 51 trajectories which your code computed. Python will automatically change the colors between curves, so the result should be quite colorful. Don't forget that your graph must have a proper title and proper labels on the axes. Each curve should be produced in the `stepit()` function, which should call the `plt.plot()` function as illustrated previously. The units for lengths can be metric.

Representative values you could use to test your code are an area of 105 cm^2, a mass of 453 g (that's one pound), an altitude of 200 ft, and an airspeed of 75 kts. These represent a friendly competition for a local flying club flying Cessna 150's.

Remember that you can check this script by comparing the results for $C_D = 0$ to results from your previous scripts (or from hand calculations).

If you are familiar with calculus then you may like to know that your program is finding the numerical solution to a differential equation ($\mathbf{F} = m\mathbf{a}$) by what is known as Euler's method. If you are not yet familiar with calculus then it may surprise you to know that you are doing calculus anyway.