BDA450 W22 Assignment 1

Due date: February 11th, 11:59pm

This is an INDIVIDUAL assignment. The work that you complete and submit must be your own, not the work of others, nor the work of a team. It is permissible to discuss the assignment in general terms, discuss general approaches to problems, help someone to understand a concept or give them a bit of advice if they have a problem, etc. However, you are not permitted to 'work together', nor to give/receive solutions (whether partial or complete) to/from other students.

Clear indicators that you are violating the rules include (but are not limited to):

- Looking at someone else's work and using that to come up with your own submission
- Sharing/sending/receiving/copying files to/from other students
- Using cut-and-paste or copy-and-paste (if both the source and destination are not your own creations)
- Etc.

If you are unclear at all about what is permissible, please ask your professor.

Submit your work in MySeneca. Include both your code, and the required output.

Late submissions <u>will not be accepted</u>.

Scenario

The physics of motion, especially aerodynamics, is a field where simulation has a long history. In class, we developed simple simulations of the motion of cars; here, we will develop a more sophisticated model of a projectile in flight, taking into account three dimensions and the impact of wind, drag, gravity, etc.

Your simulation will begin with a projectile (in this case, modelled after a baseball, but it could be anything) being launched at a certain angle with a certain speed. (Details of the parameters are provided below.) The goal of the projectile is to hit a target on the ground; you will simulate the flight of the projectile until it hits the ground, and then determine the distance from the target.

Details of the simulation

Your simulation will take place over flat ground, in three dimensions: x and y (coordinates over the horizontal plane of the ground) and z (the height above the ground).

The signature for your simulation should include the following parameters:

```
runsim(self, startingspeed, startingbearing, startingtrajectory,
windspeed, windbearing, targetx, targety, timeinterval):
```

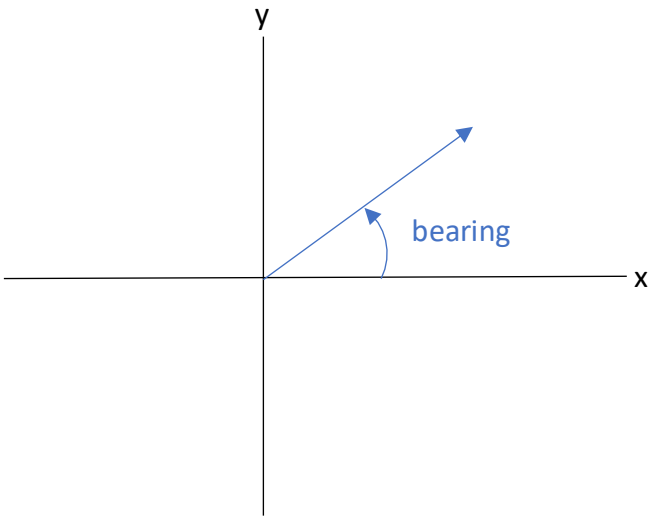
with the following meanings:

- startingspeed, startingbearing, and startingtrajectory are the initial values for the speed (in meters per second), bearing (in degrees from the x-axis) and trajectory (in degrees from horizontal) of the projectile at the time of launch. Bearing and trajectory are described in detail below.
- windspeed and windbearing are the velocity (in m/s) and bearing of the wind.
- targetx and targety are the x- and y-coordinates of the target. The target is on the ground, so it has a z-value of zero.

The launch point has coordinates of (0, 0, 2.2) (i.e, it starts from the origin in the x-y plane, but from a height of 2.2 meters).

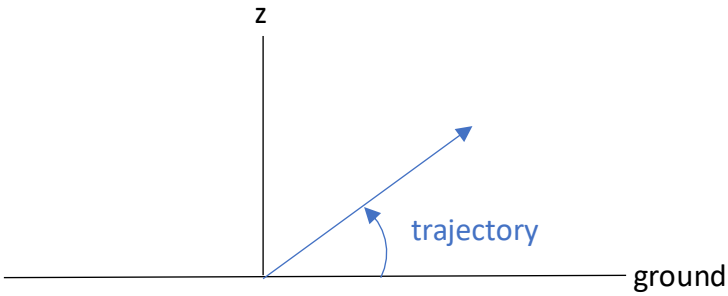
When considering the physics of motion, we need to consider both the quantity of movement/speed/acceleration/etc., as well as its direction. In a simple 1-dimensional scenario (like the car braking simulation), it is very easy—everything is either pointing to the left, or to the right. However, in 2- or 3-dimensional situations, it is more complex. In such cases, values with direction are typically represented using *vectors*; we will do so here as well, but without going into all of the details.

If you imagine a ball travelling through the air, it might travel in any direction. In our simulation, the ground is flat, so the x-y plane (with a height (z) value of zero) represents the ground. Considering only the horizontal direction of travel, we indicate this direction by the degree from the x-axis, which we call the *bearing*. Looking straight down (so we don't see z/height), this looks like:



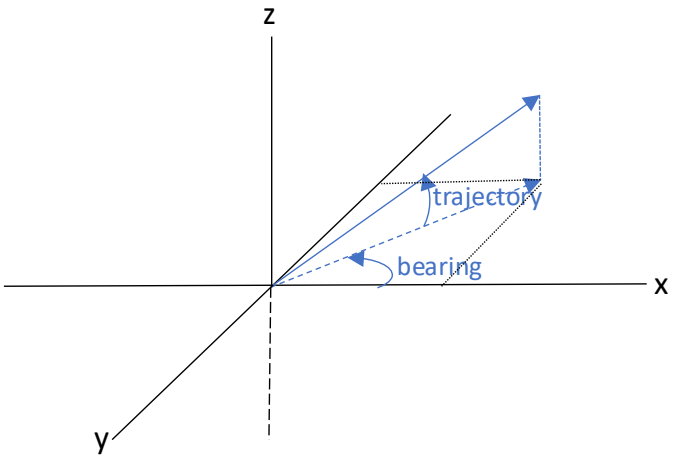
Note that positive bearings are pointing above the x-axis, and negative bearings point below the x-axis.

However, when considering a ball, you also need to consider the upward direction. We indicate the angle of travel as measured from the horizontal ground as the *trajectory*. Looking straight from the side:



Note that when considering movement/velocity/acceleration/etc., angles are always measured from the current position, not the starting position (or any other point).

Putting these two views together into a 3-dimensional diagram (I hope this is reasonably clear!):

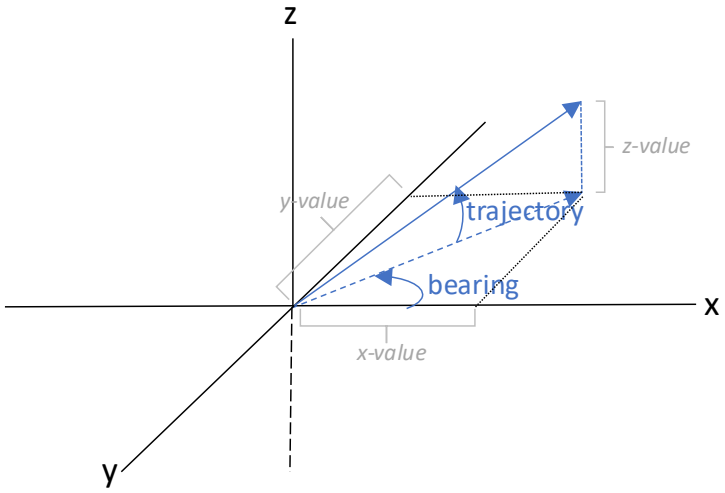


Using this approach, we can represent any value that has a direction: the bearing and trajectory represent the direction, and the *magnitude* (the length of the line) represents the quantity (e.g., the speed, or the acceleration, etc.)

If this isn't completely clear, I would encourage you to go through it carefully again (or ask for clarification), but I have provided code and equations to help with most of what you will need.

There are two primary operations that you will need to perform on these vectors:

- At each time step you will need to figure out, for example, the new coordinates of the projectile's position. Knowing the magnitude/bearing/trajectory doesn't allow you to directly calculate that! The provided function spherical_to_components() takes the magnitude, bearing, and trajectory as inputs, and returns the amount that applies in each dimension. For example, if you give it the distance, bearing, and trajectory moved, it will return how far the projectile moved in each of the x, y, and z dimensions.



- You will need to combine different quantities with direction. For example, when determining the speed/direction of the air flowing past the ball, you will need to combine the ball's movement through the (still) air with the speed of the wind. The function add_spherical_vectors() will do this calculation for you.
- Note that both of these require that the astropy module be installed/imported.

Difference equations

Note that you will be tracking both current position and current velocity of the ball throughout the simulation (where velocity is a vector with magnitude (speed), bearing, and trajectory). To update the projectile's position at each time step *t,* you will need to calculate the change in all of the x, y, and z dimensions. Taking the velocity at a particular time, you can break it down into the x, y, and z components using the provided function. The difference function for the x-dimension, then, is (and other dimensions are similar):

$$x\text{-}position_t = x\text{-}position_{t-1} + xvelocity_{t-1} * timeinterval$$

To determine the headwind (i.e., the total velocity of the air moving past the ball), you must combine the velocity due to the ball moving through still air, plus the velocity of the wind. The vector for the speed through still air is the same as for the velocity of the ball, but in the opposite direction. An easy way to calculate this is to set the bearing and trajectory the same as the ball's current velocity, but set the magnitude to the negative of the ball's magnitude. For example, if the ball's (magnitude, bearing, trajectory) is (10, 20, 25), the velocity of the air is (-10, 20, 25). The total headwind, then (noting that the wind does not change over time in this simulation, and the trajectory of the wind is 0):

$$headwind\_vector_t = airvelocity\_vector_{t-1} + wind\_vector$$

The force of drag on the ball, then, is:

$$drag\_force_t = 0.003 * headwind\_magnitude_t^2$$

where 0.003 is a coefficient reflecting the density of air, the shape and the size of the ball.

The acceleration due to the drag, then is:

$drag\_acceleration\_magnitude_t = drag\_force_t / 0.2$

where 0.2 is the mass of the ball.

The vector representing the change in velocity due to drag is:

$drag\_velocity\_vector_t = (drag\_acceleration\_magnitude_t * timeinterval, headwind\_bearing_t, headwind\_trajectory_t)$

The vector representing the change in velocity due to gravity (which pulls straight down at a constant rate!):

$gravity\_velocity\_vector = (9.8*timeinterval, 0, -90)$

And finally the new velocity of the ball:

$ball\_velocity\_vector_t = ball\_velocity\_vector_{t-1} + drag\_velocity\_vector_t + gravity\_velocity\_vector$

Run and Output

Noting that the ball starts as coordinates (0, 0, 2.2), the simulation should run until the ball hits the ground (i.e, z = 0).

Your simulation should output a text message indicating the distance from your ball's final position to the target (where distance = $\text{sqrt}((x_1-x_2)^2 + (y_1-y_2)^2)$). It should also output the following plots:

- x-position vs. time (showing distance travelled over time)
- x-position vs. z-position (showing the path traveled, from a side-view)
- x-position vs. y-position (showing the path traveled, from a top-down-view). This plot should also show the position of the target (see below).
- A 3D plot showing the path of the ball in the x, y, and z dimensions simultaneously. This plot should also show the position of the target (see below).
- A plot of velocity vs. time.

Your plots should be titled/labelled properly. You should also plot the position of the target as indicated above. A code sample demonstrating both of these:

```
plt.title("X Position vs. Y Position")
plt.xlabel("x (Distance) (m)")
plt.ylabel("y (Distance) (m)")
plt.plot(self.xpos, self.ypos)
plt.plot(targetx, targety, 'ro')
plt.axis('square')
```

(Note that the plt.axis('square') makes the scales on both axes the same, which better portrays the path.)

A code sample for the 3D plot:

```
ax = plt.axes(projection='3d')
plt.title("3D Path of Ball")
ax.set_xlabel("x (m)")
ax.set_ylabel("y (m)")
ax.set_zlabel("z (m)")
plt.plot(self.xpos, self.ypos, self.zpos)
plt.plot(targetx, targety, 0, 'ro')
```

Note that the you can rotate the 3D plot while viewing it by clicking and dragging.

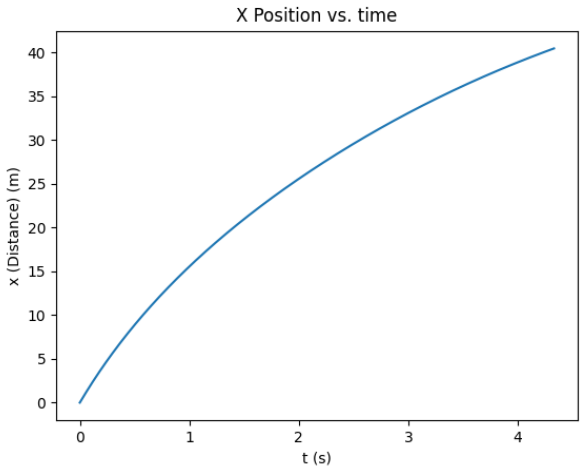
Submit your results with a wind speed of 20, a wind bearing of 170, and a target position of (10, 20). Choose whatever values you want for the launch velocity, but try (via trial and error) to find values that will have your ball land as close as possible to the target, ideally within 10 m.

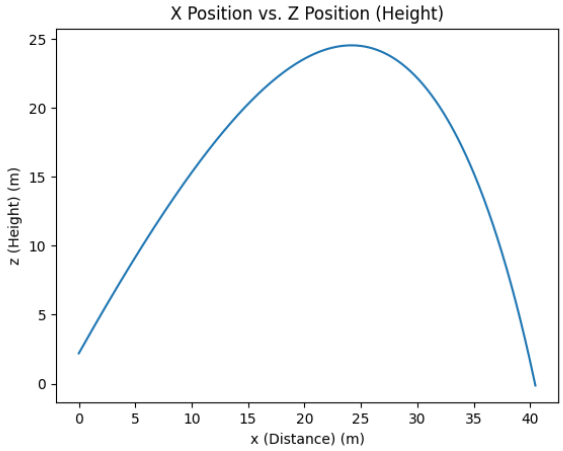
Sample output

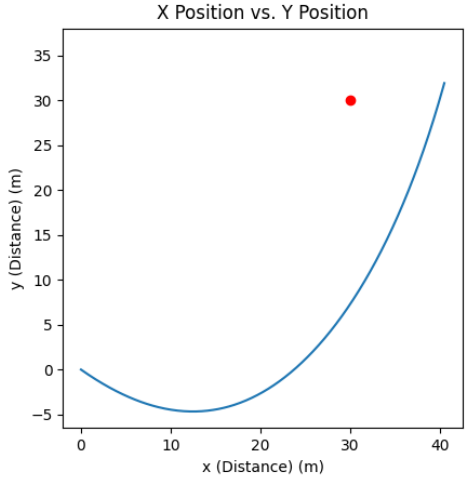
Sample output, when starting with these arguments:

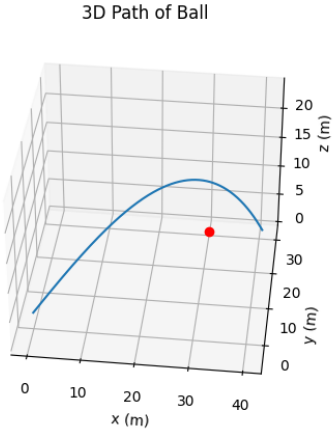
```
sim.runsim(40, -35, 50, 25, 90, 30, 30, .01)
```

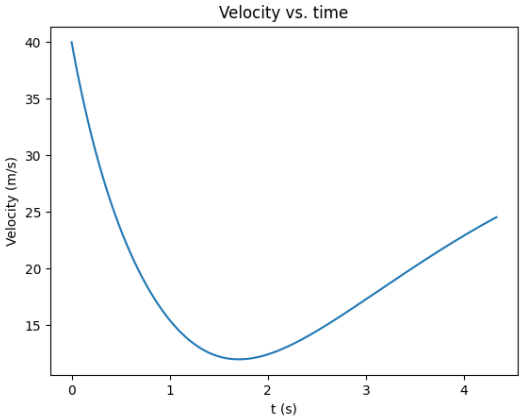
Distance from target: 10.63




X Position vs. Z Position (Height)
z (Height) (m)
x (Distance) (m)
0
5
10
15
20
25
30
35
40


X Position vs. Y Position
y (Distance) (m)
x (Distance) (m)
−5
0
5
10
15
20
25
30
35
40


3D Path of Ball
z (m)
y (m)
x (m)
0
5
10
15
20
30
40


Velocity vs. time
Velocity (m/s)
t (s)
40
35
30
25
20
15
0
1
2
3
4