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 rocket in flight, taking into account three dimensions and the impact of thrust, drag, gravity, etc.

Your simulation will begin with a rocket being launched at a certain angle with a certain amount of thrust. (Details of the parameters are provided below.) The rocket's engine will run for a certain amount of time, providing thrust, before stopping. The goal is for the rocket 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.

Writing in object oriented programming.

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,
thrustforce, thrusttime, targetx, targety, timeinterval):
```

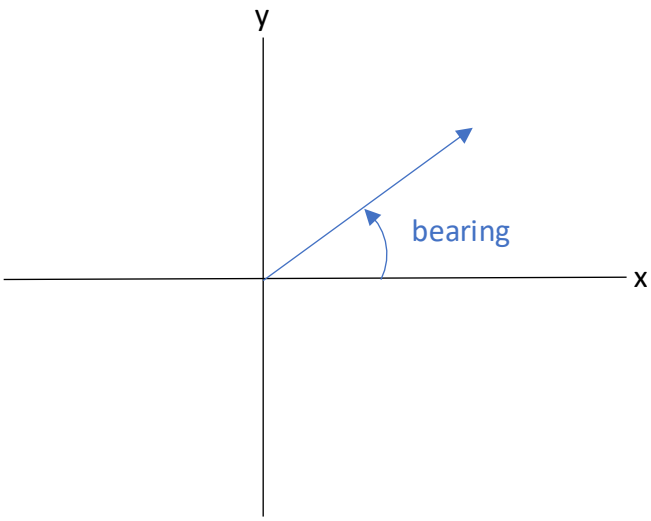
with the following meanings:

- 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.
- thrustforce is the strength of the force produced by the engine (in N), while thrusttime is the number of seconds for which the engine runs before stopping. (After stopping, the engine produces zero thrust.)
- 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.
- timeinterval is the length of each timestep, in seconds. A value of 0.1 is reasonable.

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).

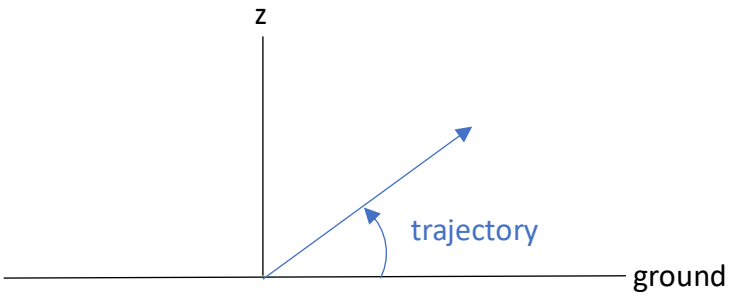
When considering the physics of motion, we need to consider both the amount of movement/speed/acceleration/etc., as well as its direction. In a simple 1-dimensional scenario (like the car braking simulation), it is very easy—everything is either pointing to the left, or to the right. However, in 2- or 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 an object 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 degrees from the x-axis, which we call the *bearing*. Looking straight down (so we don't see z/height), this looks like:



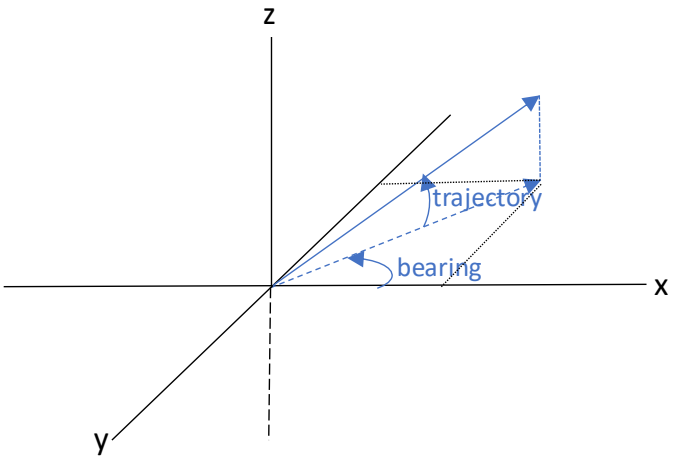
Note that positive bearings are pointing above the x-axis, and negative bearings point below the x-axis.

However, when considering an object flying through the air, 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.)

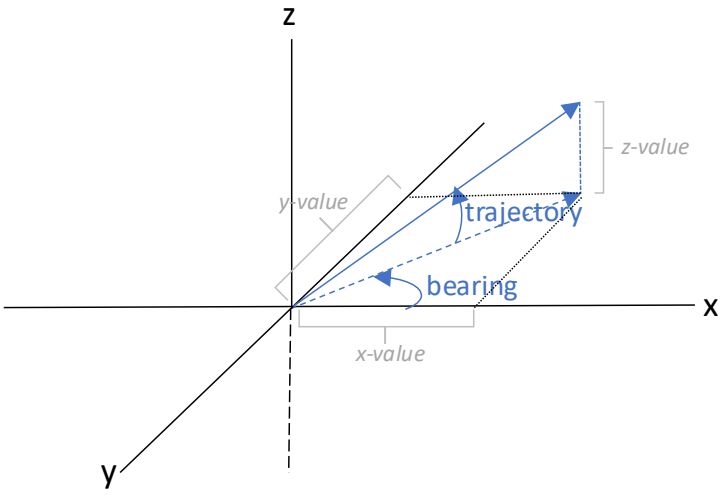
If this isn't completely clear, I would encourage you to go through it carefully again (or ask for clarification), but I have provided code and equations to help with most of what you will need.

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

- At each time step you will need to calculate the rocket's *movement*, i.e., figure out the new coordinates of the rocket's position. Note that position is a set of coordinates (x, y, and z), but the information that you have about the rocket's movement doesn't provide x/y/z directly—you have bearing/trajectory/velocity instead.

  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.



- At each time step, you will also need to determine how the rocket's *velocity* is changing (both the speed, and direction.) To do so, you will need to combine different forces/accelerations/etc. which are acting on the projectile. In this particular simulation, you will have the forces of thrust, drag, and gravity.

  The provided function add_spherical_vectors() will help you to do this; it allows different effects to be combined by adding their vectors. Details of the calculations are provided below.

Note that both of the provided functions require that the astropy module be installed/imported.

Difference equations

Note that you will be tracking both current position and current velocity of the rocket 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 (as velocity/bearing/trajectory) 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):

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

In addition to updating the position each timestep, you must update the speed/direction of the rocket, as forces act on it.

Thrust from the engine pushes the rocket straight forward (i.e., the force/acceleration from the thrust has the same bearing and trajectory as the rocket's current movement). The force of the thrust is passed to your program as an argument; the acceleration due to the thrust is:

$$thrust\_acceleration\_magnitude_t = thrust\_force_t / 5$$

where 5 is the mass of the rocket, in kg. The change in velocity due the thrust, then, is

$$thrust\_velocity\_vector_t = (thrust\_acceleration\_magnitude_t * timeinterval, rocket\_bearing_t, rocket\_trajectory_t)$$

Drag (i.e., the resistance of the air as the object moves through it) is effectively a force working in the exact opposite direction to the current movement of the rocket. The force of drag on the rocket, then, is:

$$drag\_force_t = 0.006 * rocket\_velocity_t^2$$

where 0.006 is a coefficient reflecting the density of air, the shape and the size of the rocket.

The acceleration due to the drag, then is:

$$drag\_acceleration\_magnitude_t = drag\_force_t / 5$$

where 5 is again the mass of the rocket.

Because drag works in the opposite direction to the travel of the rocket, an easy way to represent this as a vector is to use the same bearing/trajectory as the rocket, but to use a negative magnitude). The vector representing the change in velocity due to drag, then, is:

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

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

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

(Note that the gravitational effect doesn't change over time, so there is no *t* subscript.)

And finally the new velocity of the rocket:

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

Run and Output

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

Your simulation should output a text message indicating the distance from your rocket's final position to the target (where distance = $\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$). 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 rocket 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.
- A plot of trajectory 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 Rocket")
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')
```

Sample output

(In the sample output below, note the interesting features in the velocity vs. time plot, in particular:

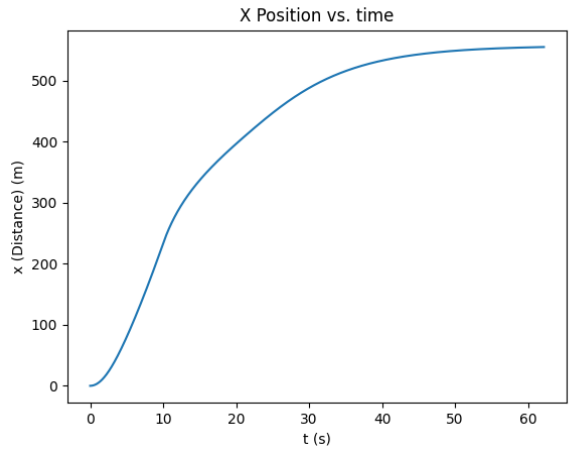
- For the first 10 seconds or so, velocity is increasing as the engine fires. The line curves, however, because the faster the rocket goes, the more drag resists it.

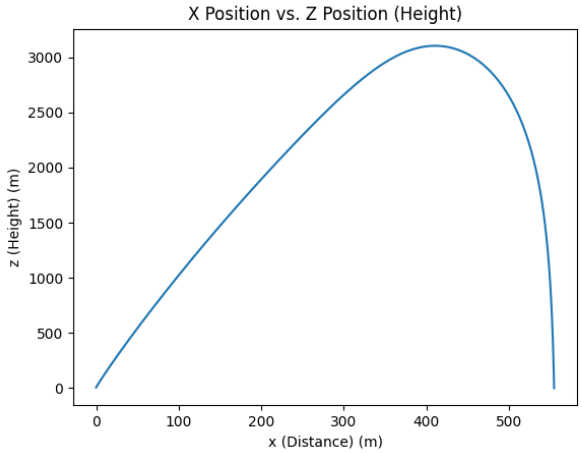
- After 10 seconds, the engine stops; the rocket keeps moving upwards, but its speed slows as gravity is the primary force.
- After about 20 seconds, the rocket reaches the top of its flight, and starts to speed up again as it begins to fall back to Earth.)

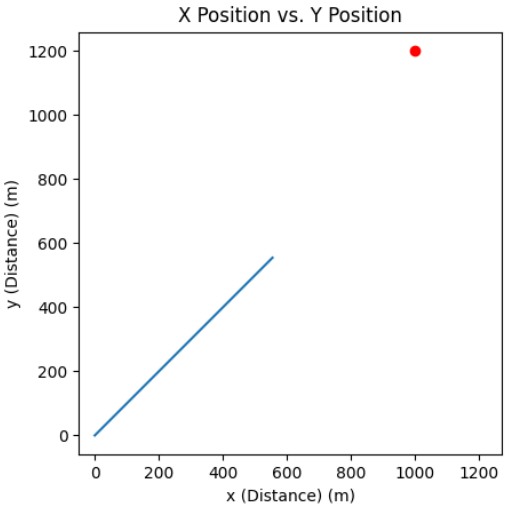
Sample output, when starting with these arguments:

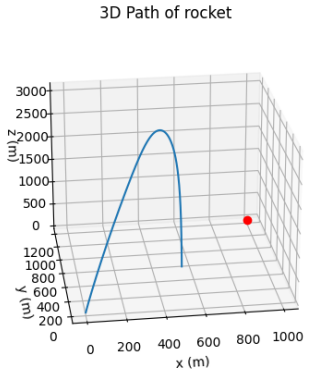
```
sim.runsim(5,45, 85, 500, 10, 1000, 1200, .01)
```

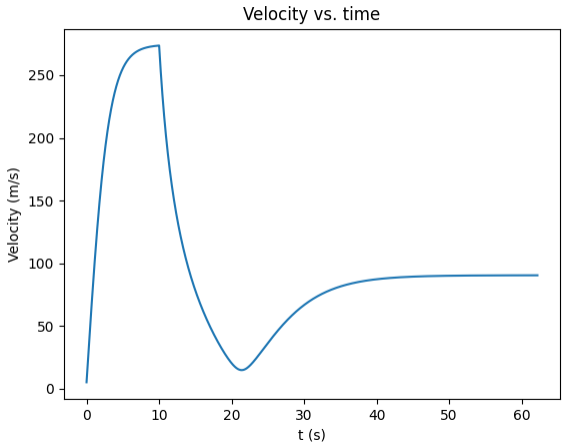
Distance from target: 783.78

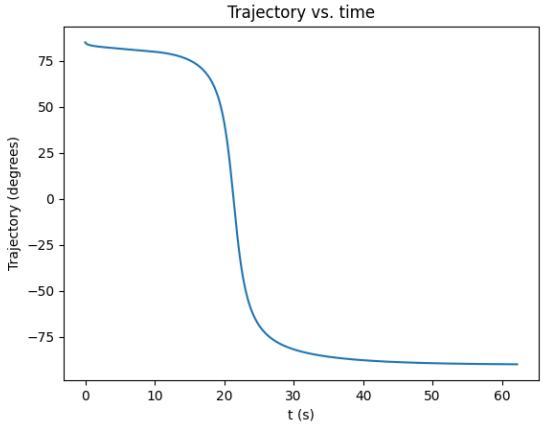





X Position vs. Y Position
y (Distance) (m)
x (Distance) (m)
0
200
400
600
800
1000
1200


3D Path of rocket
z (m)
y (m)
x (m)


Velocity vs. time
Velocity (m/s)
t (s)


Trajectory vs. time
Trajectory (degrees)
75
50
25
0
−25
−50
−75
0
10
20
30
40
50
60
t (s)