COMP 5070
Statistical Programming for Data Science
Bushfire detection

Assessable Exercise 1 DUE by 11:00pm (CST), Sunday, March 27

- This exercise is a part of the continuous assessment that is worth 25% of your overall grade.
- Your code should be submitted as a single .py file using LearnOnline. Do not hardcode any paths on your computer in the code, as I should be able to load and run your code.
- The exercise is out of 100 marks. To obtain the maximum available marks you should aim to:

    1. Code the requested program (60%).
    2. Use a clear coding style (10%). Code clarity is an important part of your submission. Thus, you should choose meaningful variable names and adopt the use of comments - you don't need to comment every single line, as this will affect readability - however you should aim to comment at least each section of code.
    3. Have the code run successfully (10%).
    4. Output the information in a presentable manner as decided by yourself (10%).
    5. Document code limitations including, but not limited to, the requested functionalities (10%).

This assessable exercise can be openly discussed within the group online and you are welcome to share tips and tricks (not entire programs, however).

Having said that, the ground rules are:

- If you use another person's code in your file, please note the source and how much of the code is not yours.
- If you submit a program cobbled together by other peoples' code with no, or little, original input from yourself, you will automatically receive a zero mark. The idea is to develop your own programming style with (or without) the help of others. Any code used should support your approach to how you write the program, not replace your own efforts.

If you're unsure at any point, you're welcomed to check with me.

Bushfire detection

Bushfires are very common in Australia. Even after the main fire has gone, there is still a very high risk of a new fire to re-ignite from remaining hotspots.

In this exercise you should write a code to simulate an automatic drop searching the area for a hotspot. Here are rules and useful comments for your task:

1. Your search area has a rectangular shape with some *width* and *depth*. Please see example code provided at the end. Coordinates of the drone and hotspot are measured between 0 and *width* for X coordinates; and between 0 and *depth* for Y coordinates.
2. The hot spot position is generated at random in the beginning of the code. The drone does not know where the hotspot is, so you can not cheat and take hotspot coordinates from the map. The drone should find the hotspot.
3. We observe drone over fixed time intervals of 1 second. So, you will use a loop with 1 second time increments. Number of iterations in the loop can be relatively large, however it is finite as the drone will run out of fuel/battery. You can make it 1200 seconds (20 minutes). Obviously, larger search area would require more time/fuel.
4. We assume that drone constantly moves (that is, never stops). The drone moves with constant speed, e.g., 1 m/s.
5. Drone can change direction freely. There is no inertia. One full second drone goes North at 1 meter per second, then it changes direction and next full second it goes South at the same speed.
6. Drone can observe the world around with some radius, e.g., R = 10 meters. Hence, if at some point in time the drone location happens to be within 10 meters from the hotspot – that means you found the hotspot and search stops.
7. The drone is autonomous. You cannot control its actions. You just set a program and then wait for the result – drone finds the hotspot or dies without fuel/power.
8. You can use any strategy for a search, and it does not need to be complex. Just a random flying there and back is a strategy too.
9. The drone has to solve two problems at the same time: (1) it should remain within the search area; it can go a little bit out if it is necessary for your strategy but then it should return back; (2) drone should find the hotspot as quick as possible.
10. The output of your program will be a list of tuples. Each tuple has a format (X,Y) with coordinates of the drone at that time moment. If you run a search for 1200 iterations (seconds), then the list will have 1201 tuples with the drone coordinates starting with the drone initial position (0,0). This list is a path of the drone, and it will be plotted by the code provided. If the drone is lucky to find the hotspot, then the loop stops, and the list will be shorter.
11. Feel free to change any parameters above. For example, for some strategies it might be beneficial to have starting point not (0,0) but the centre of the search area (width/2, depth/2). You can change everything. Just be reasonable and don't make it too easy –

for example, don't make observation radius too large, so the drone can see everything from any position.

Hints and supporting code

- To calculate coordinates of the drone after the move, you will need to use some basic Trigonometry (google for "solving right triangles" or "sin cos in right triangles"). You don't really need right triangles, but the presentation is easier to follow.
- To code your solution, you will need to import library "math". Beware, it uses Radians for angles.
- To calculate the distance between drone's and hotspot's coordinates (to check if the drone sees the hotspot) you need to use Euclidian distance (google for "Euclidian distance formula") and then use "math" library again.

```
# example of the code
import math
import random

# input parameters for drone
V = 1                    # drone speed per second
R = 10                   # radius how far drone can see
drone_XY = [(0.0, 0.0)]  # initial coordinates of the drone

# input parameters for map and hotspot
width, depth = 50, 50
hotspot_XY = (width * random.random(), depth * random.random())

# your main code here



# example of your output – list of tuple with coordinates
drone_XY = [(0.0, 0.0),
            (0.7, 0.7),
            (0.7, 1.7),
            (0.7, 2.7),
            (1.7, 2.7)]

# code to plot drone trajectory
import matplotlib.pyplot as plt
xs, ys = zip(*drone_XY)

fig, ax = plt.subplots()
ax.set_xlim(0, width)
ax.set_ylim(0, depth)
ax.plot(xs, ys, 'o--', lw=2, color='black', ms=3)
ax.plot(hotspot_XY[0], hotspot_XY[1], 'o', lw=2, color='red', ms=10)
```