Alex Camilleri – GEOG5003M - White Star Line – Development notes

I began the development process by breaking the code down into compartments: *GUI, Data reading, Iceberg assessment, Results.* I used a notebook to write out some quick ideas and sketch out a possible interface layout.

GUI

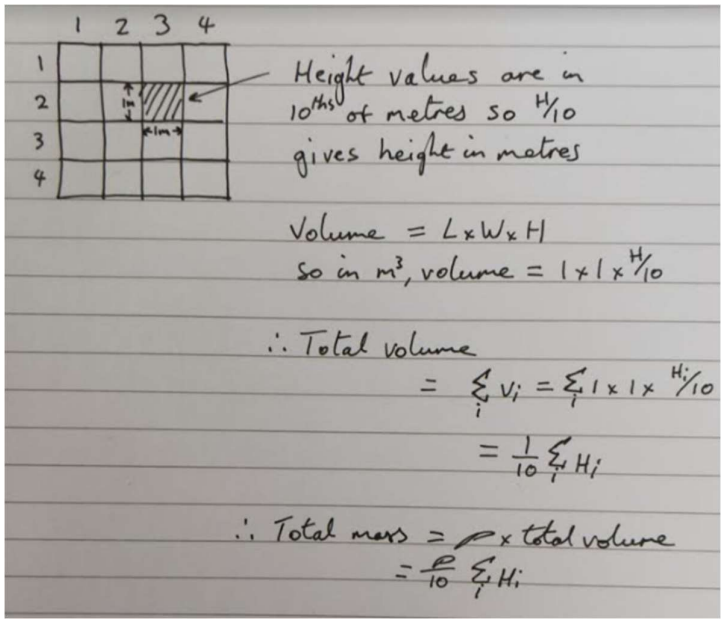
This was a great project to really get to grips with using Tkinter. Although my original idea was for the GUI to pose as a kind of dashboard one might find at the helm of a ship, with a big button saying "SCAN ICEBERG!" that displayed the radar image in the same window, this didn't end up in the final version. When placing the radar/lidar image in a column in the GUI, the column would stretch which sent the buttons out of alignment. I decided not to pursue this and instead have the lidar/radar images display in a separate window using matplotlib.

Data reading

I decided to use the numpy library for this program rather than lists of various dimensionality. Firstly because I wanted to branch out and make use of a common large data analysis libraries, but also because I was interested in the potential increase in efficiency. I had read that many libraries are not written purely in Python, but make use of C and Fortran which are "lower-level" languages (closer to machine language) and so are faster. This led me to viewing the code from more of a computer-science perspective rather than just as someone learning how to use the Python language to achieve a task. Numpy arrays are apparently more efficient because they store homogenous data-types in contiguous memory locations, while Python lists are the complete opposite of this, so are much less efficient for the processor to access while executing code. Numpy arrays make it simple to explore and index information. For my purposes, the numpy.where({condition}) function was invaluable for indexing the locations of pixels of ice and using them to find the height values in the lidar file.

Iceberg assessment

I sorted out the mathematics behind the iceberg assessment quite early on. It was important to do this first rather than try and write it straight into code. It's much easier for me at this stage to translate a well-founded external set of ideas into code, rather than try to purely code it out from the beginning. Designing a model before building it is easier with more chance of "success", than improvising from the start, which could go either way.



Results

My initial idea for saving the results was to have the save button visible but to have a global variable inside the scan function which would change to "True" when the scan function had completed. The scan button would have a conditional statement executing a csv.writer only if the global variable was "True". In the end, I couldn't quite get this to work, so I opted to put the csv.writer at the end of the scan function, then display feedback to the user on the GUI that the report had been saved.

Issues overcome

This process has taught me the meaning of “feature creep”. I found as I was coding I would get lots of ideas about how to make the program intricate in various interesting ways but at all times I had to keep myself to a pretty strict schedule, focussing only on what was asked. As a result, I am slightly disappointed with this program because it is not what exactly what I envisioned but it’s own thing, reflective of problems I overcame, rather than elements I intentionally *designed*. I also made a strong attempt at reading and assessing multiple icebergs, but did not have enough time to integrate it into the final program. I really increased my understanding of indexing of lists and numpy arrays so the time was not wasted either way.

During development of the scan_iceberg function I found that my code was running unexpectedly slowly. When running the function it would take around 70-80 seconds to complete. I spent a large part of the development process this way because I neglected to address the issue early on. I was more interested in using the available time to get the broad picture of the program down. This ended up being a mistake, because the time taken to run the code during testing and building was frustrating. Eventually I found that I had written code to read the lidar_file into a numpy array inside the for loop that searched for values in the lidar_array. A redundant activity as the numpy array only needed to be created once.

```
8   def scan_iceberg_slow(radar_file, lidar_file):
9       start = time.time()
10
11      with open(radar_file, 'r'):
12          radar_array = numpy.loadtxt(radar_file, delimiter = ",")
13          radar_array = radar_array.astype('int32')
14      ice_positions = numpy.where(radar_array >= 100)
15
16      volume = 0
17      for i in range(len(ice_positions[0])):
18          x = ice_positions[0][i]
19          y = ice_positions[1][i]
20          with open(lidar_file, 'r'): # this was the first version I wrote, very slow
21              lidar_array = numpy.loadtxt(lidar_file, delimiter = ",")
22              lidar_array = lidar_array.astype('int32')
23          height_metres = lidar_array[x][y]/10
24          volume = volume + height_metres*1*1
25
26      mass_above_water = volume*900
27      total_mass = mass_above_water*10
28      total_volume = volume*10
29      end = time.time()
30      print(end-start) # This version: 74.85s
```

After moving these lines of code *outside* the for loop, I found the run time improved significantly. From 75 seconds to 0.2! This has helped me to understand the importance of thinking the code through to make sure there are no redundant steps.

```
33  def scan_iceberg_fast(radar_file, lidar_file):
34      start = time.time()
35
36      with open(radar_file, 'r'):
37          radar_array = numpy.loadtxt(radar_file, delimiter = ",")
38      with open(lidar_file, 'r'): # the second version with the lidar_array creation outside the loop
39          lidar_array = numpy.loadtxt(lidar_file, delimiter = ",") # much faster
40      ice_positions = numpy.where(radar_array >= 100)
41
42      volume = 0
43      for i in range(len(ice_positions[0])):
44          x = ice_positions[0][i]
45          y = ice_positions[1][i]
46          height_metres = lidar_array[x][y]/10
47          volume = volume + height_metres*1*1
48
49      mass_above_water = volume*900
50      total_mass = mass_above_water*10
51      total_volume = volume*10
52      end = time.time()
53      print(end-start) # This version: 0.22s
54
```

Sources:

Python documentation: https://docs.python.org/3/

Tkinter documentation: https://tkdocs.com

Matplotlib documentation: https://matplotlib.org/

Geeks for Geeks: https://www.geeksforgeeks.org/why-numpy-is-faster-in-python/