TASK 1

Both the **lecture3\_focal.py** and the **task1\_w3.py** are commented and running.

The two algorithms are both applying a **boxcar** **filter** to a raster. This filer assigns to the cell at the center of the search box a value that is the average of the values within the search box. The effect of this filter is to reduce the differences between the cells, smoothing the raster surface.

Both algorithms I uploaded use the **raster\_test2.txt** for the calculations. The main difference between these two is that **task1\_w3.py** performs the boxcar filter on all the cells of the original raster, while **lecture3\_focal.py** ignores the ones that are at a distance less or equal to “buffer” from the boundaries.

The downside of using the **task1\_w3** is that not all the cells in the output raster are calculated using the same filter searching area. And this can be misleading if the user is not well aware of it.

Instead, in the **lecture3\_focal.py** all the cells in the output rater are calculated using the full-size filter search box. As mentioned before the ones that will not have the full search box are excluded from the calculation (value=zero). However, using the **raster\_test2.txt** in **lecture3\_focal.py**, I noticed that the “zero” value was disrupting the color scale, impeding to really appreciate the result. This because the range of values from ~2000 to ~3000 m was after going from 0 to ~3000. Therefore the results were too homogeneous in color.

My first solution was to produce a raster without the buffer area “**focalMean=np.zeros((rows-search\*2,cols-search\*2))”**. However, this means to have as output a raster variable in sized depending on the buffer, which in case of further operations might complicate things. Two other possible solutions that I could think of are : (1) assign to the buffer area the minimum value found in the raster, so that the range of values will not be altered, (2) exclude zero from the color scheme (not sure if this is possible).

(!) I’ve not quite understood the code on page 55. I get what we want to do but not how we are trying to achieve it using the ‘factor’.

TASK 2

I’m comparing the algorithm in the file **Lecture3\_binning\_example\_modal** and **Lecture3\_modal\_2**.

In the **example\_modal** the algorithm has two nested loops. The first goes over all the cells of the raster. Per each cell, the second loop checks if the value of the cell is equal to one of the values in the category list “**if array2D[i,j] == categories[v]”**. In the example of the 25 million cells and 25 categories, the algorithm can have a total number of:

* minimum of 50 million loop iterations (25million for the raster + (25million \* 1 for the category)) – if we hypothesize a raster with all the same value and that this value is stored in the [0] position of the list;
* maximum of 650 million loop iterations (25million for the raster + (25million \* 25 for the category)) – if we hypothesize a raster with all the same value and this value stored in the [24] position of the list;

In the **modal\_2** the algorithm has a loop that goes over the dataset once, and for each cell there is an if – else loop that goes over a dictionary (I’m not analyzing here the additional functionality modalcount). In the example of the 25 million cells and 25 categories, the number of reiterations will be similar to the ones of the **example\_modal**. Except for the fact that the majority of these iterations is undertaken by the if – else loop:

* minimum of 25million loop iteration for the raster + 25million \* 1 over the dictionary;
* maximum of 25 million loop iterations for the raster + 25million \* 25 over the dictionary;

Without having timed the algorithms I think the **modal\_2** is faster since it uses an in build functionality to go over the dictionary.

I like more the **modal\_2** because it is returning a dictionary with each category directly linked with its frequency, meanwhile, the **example\_modal** is returning a list of values and I have to remember which category is associated with the different indexes [i]. In addition, **example\_modal** apparently does not work well with numbers. It can count them only if the number is expressed as ‘text’ in the category list (see below). If the number is stored as a number in the category list it won’t be detected. I’m unsure why this happens, I think this is **np.array** that is converting numbers to text. **Modal\_2** instead just work with both text and numbers without changing numbers’ nature.

Case 1 – **example\_modal** does not work with numbers

aList=['A', 'B', 'A', 'A', 'B', 'C', 'A', 99, 99]

categories=['A', 'B', 'C',**99**]

this are the bin value

[ 400. 200. 100. 0.]

Case 2 – **example\_modal** it works with numbers

aList=['A', 'B', 'A', 'A', 'B', 'C', 'A', 99, 99]

categories=['A', 'B', 'C', **'99'**]

this are the bin value

[ 400. 200. 100. 200.]