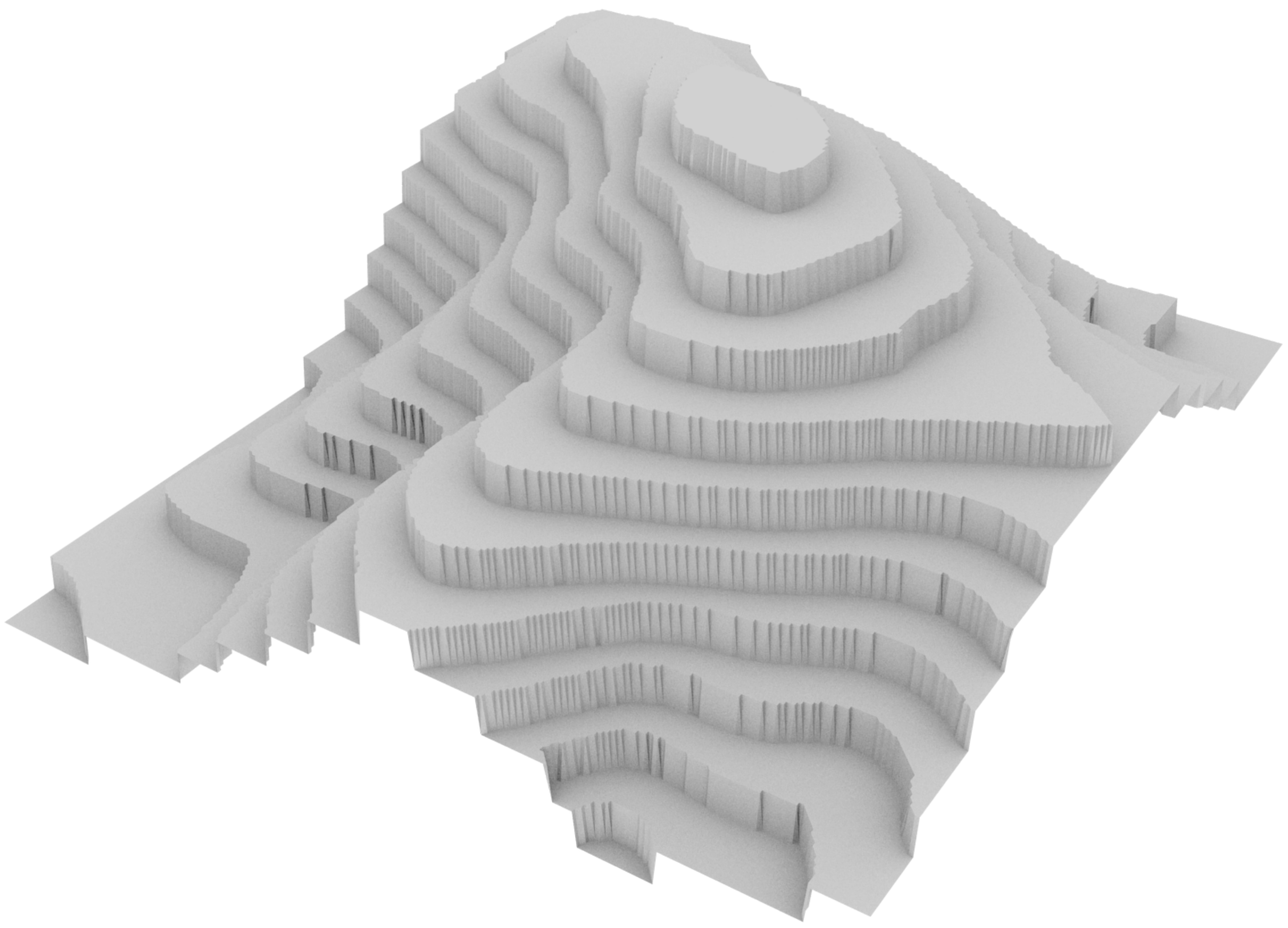
Assignment 01

Spatial interpolation

Deadline is **2020-12-02 10:00am**

Late submission? 10% will be removed for each day that you are late.

You're allowed for this assignment to work in a **group of 2** (and thus submit only one solution for both of you). You are free to form a group yourself; if you're looking for a partner let me know (Hugo), or let others know on Discord. If you prefer to work alone it's also fine.



Overview: from sample points to raster

The aim of this assignment is to implement (part of) four interpolation methods used for modelling terrains:

1. nearest neighbour [NN]
2. IDW (version with a search circle) [IDW]
3. linear interpolation in TIN (based on a Delaunay triangulation) [TIN]
4. ordinary kriging (with a search circle) [kriging]

You have to create a Python program that reads *(x,y,z)* points and that outputs a raster with different resolutions.

What you are given to start

Code is in the `/hw/01/` folder of the GitLab repository of the course.

We give you the skeleton of the Python program:

1. `geo1015_hw01.py` is the main(). You are *not* allowed to modify this file!
2. `mycode_hw01.py` is where *all* your code should go. The 4 functions defined there must be completed, and you are not allowed to change the input parameters. You are of course allowed to add any other functions you want in that unit (please do not include any other unit you code, add your code to that single file). You are only allowed to import modules from the Python standard library (anything you installed through `pip` is thus not allowed, except *scipy*, *numpy*, and *startin*). Notice that the functions we have defined do not return anything, instead they write the output raster to the disk; you cannot change this behaviour.
3. `params.json`: a very simple JSON file that defines what the input file is, which interpolation methods to execute, with what parameters, and where to store the output.
4. `variogram.py` is a small script to decide the theoretical variogram function that should be fitted to experimental variogram. This Python script is used as a stand-alone module, and independently from the other Python modules. You use it to interactively decide which function best fits your data, and you hard-code this value in the kriging function (only for the file `samples.xyz`). Skim the code to understand what it does and play with the parameters in the theoretical variogram section (eg the number of bins and the nugget/sill/range of the different functions).
5. `samples.xyz`: an example of an input file with sample points. All input files that the program reads will be like this one: a CSV file with headers (x y z), each line is one point, and the delimiters between *x y z* is a space.
6. `example_output.asc`: an example of an .asc output file that your program will need to create (QGIS reads it, among others).
7. the folder `other-samples` contains some areas with part of a .tif file we took somewhere, and we generated randomly samples from it in `samples.xyz`

Notice that the file `params.json` doesn't need to be validated, and the input/output and parameters are also valid. In other words, one can't put an output file at `/home/elvis/temp/myoutput.asc` if that folder doesn't exists, or give a negative value for the search circle of IDW.

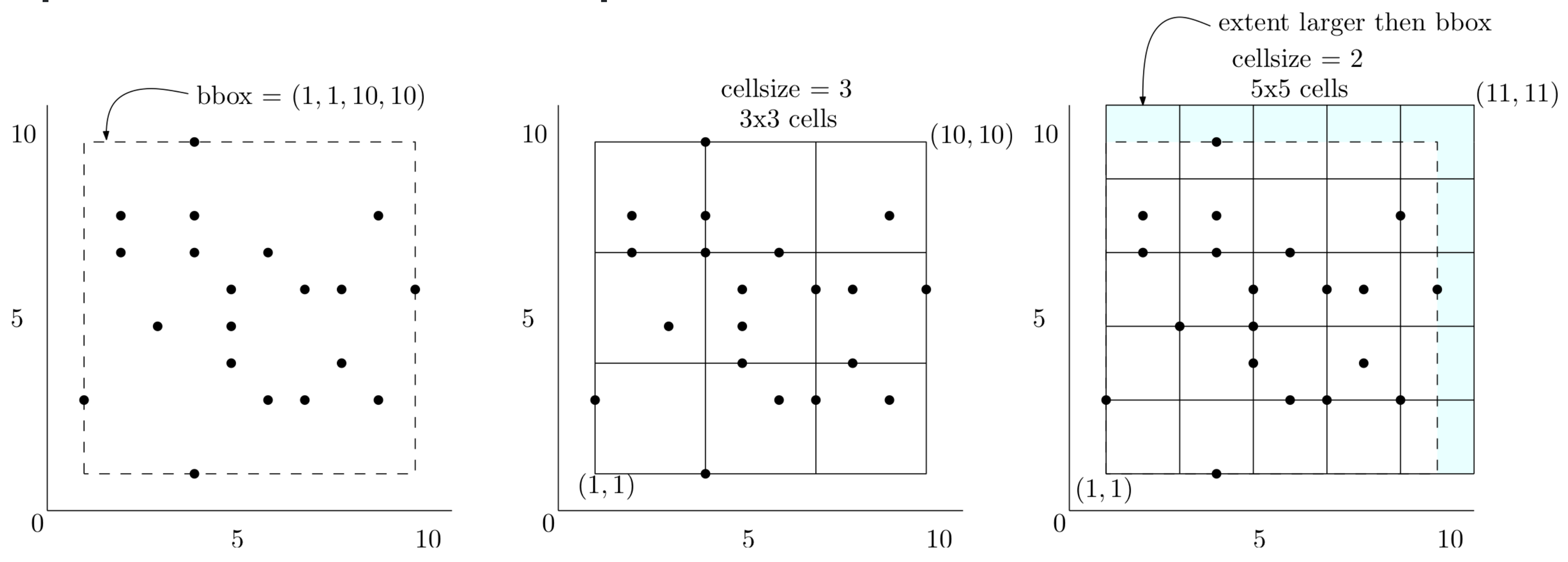
The output format

Your output file has to be an Esri ASCII raster format (.asc), see its full specifications.

It's a simple ASCII/text file that can be easily created with Python, and QGIS reads it natively. To ensure that your output is valid, test it with at least QGIS.

In the given code/files, there is an example of a valid file (`example_output.asc`).

Specifications for the output raster



You need to create a raster that covers at least the bounding box (bbox; it must be axis-aligned). As shown in the figure above, you need to calculate the bbox, and then you need to start at the lower-left corner and add cells (based on the `cellsize` parameters given as input) and fill the raster with the appropriate number of cells so that all input sample points are inside (or directly on the boundary) of the output raster.

In the middle case in the figure, 3x3 cells "covers" all the input points (since x=10 and y=10 are exactly on the boundary). For the case on the right in the figure, 4x4 cells would not be sufficient, and thus the output raster is larger than the other one (and that for the same input sample points), and has 5x5 cells (of size 2).

The lower-left corner of the bbox of an input dataset must always be respected, and you have to start "adding" the cells from that point.

Observe also that the output raster need not be a square grid, it can be a rectangle (ie the number of columns in the raster does not need to be the same as the number of rows).

Good to know

- `nodata_value` should be set to -9999

- you can ignore all CRS, you just read the points and output a .asc file
- the coordinates of the input files will always be in meters
- For all 4 interpolation methods, if the centre of a given cell is outside the convex hull of the input dataset, then `no_data` should be assigned to this cell. You are allowed to use a function from the allowed libraries to perform this.
- the time it takes for your program to terminate has no influence on the marks
- however, speeding things up is a good idea. You can do this by using any functions/packages from the Python libraries *scipy*, *numpy*, and *startin* (the only ones allowed besides the Python standard library)
- the file `mycode_hw01.py` gives you hint about what to use, and how
- if you use startin for the TIN, you are **not** allowed to use directly the builtin function for the interpolation (`dt.interpolate_tin_linear(x, y)`), you have to code it yourself
- you are however allowed to use `dt.interpolate_nn(x, y)` in startin for the nearest neighbour queries (instead of the kd-tree in scipy; but it's slower)
- you do not need to code yourself the Delaunay triangulator, use SciPy Delaunay or *startin* (your choice)
- for the kriging interpolation, we will only assess the result with the input `samples.xyz`, since you need to manually fit the variogram
- the simple version of kriging that we saw in the lesson is not numerically stable, so try different parameters (eg the search radius) if you get a weird result
- you will have problems with kriging if there are no sample points in the search radius

What to submit and how to submit it

You have to submit a total of 5 files:

1. the Python file `my_code_hw01.py` where your name and that of your colleague are clearly identified at the top.
2. the 4 output rasters as the file `params.json` defines them: `nn.asc`, `idw.asc`, `tin.asc`, and `kriging.asc`

Those 5 files need to be zipped and the name of the file is the studentIDs of the members separated by a "_", for example `5015199_4018169.zip`.

Oh, and **no** report is necessary.

Upload the ZIP file to this surfdrive page.

[last updated: 2020-11-11 14:52]

The teaching materials (book + videos) are available under the Creative Commons Attribution 4.0 International (CC BY 4.0) licence.