Object oriented software engineering: Spatial Algorithms

Lecture and Workshop 3

Gary Watmough, Peter Alexander, Guillaume Goodwin

Week 1 Task – General Feedback

- Our solution is now on Learn under week 1 LD solution.
  - NOT CORRECT!!!! – Just one way of doing this.
- The method should be placed in the PointsField Class.
- Figures need captions
  - Some produced lots of figures but there was not enough information in the caption we need to know:
    - Sorted or unsorted
    - X or Y
    - Time and the units of the time if presenting time.
  - Avoid placing comments in the driver instead of captions, do both!

Week 1 Task – Commented Code

- The purpose of commented code:
  1. Help other users understand what the code is doing
  2. Remind yourself what the code is doing
- Look over your code again and ask the following questions:
  - In 2 years will I understand without having to look at the code?
  - Would someone who has not been in the class and who maybe has less coding experience understand in general what is happening?
  - Would another user know which bits would need changing for particular reasons?

Comparing methods: potential problem

```
Data = random list of numbers
timestart
Data.sortmethod1
timeend

timestart
Data.sortmethod2
timeend
```

Feedback on learning diary from week 2

- How was it?
- Solution will follow once all are uploaded.

5

Any problems

- Office hours:
  - Today: Wednesday: 09:00 – 11:00
  - Others: contact me by email gary.watmough@ed.ac.uk

6

Week by week guide

~~1. Handling spatial data:~~

~~2. Divide and Conquer~~

3. Grid data and arrays

a) Handling, traversing and searching raster data. Point and focal functions.

4. Problem solving

a) Flow, Nearest Neighbour Analysis

7

This week – intended learning outcomes

- be familiar with a range of algorithms used to manipulate and analyse spatial data
- develop python classes suited to the representation and analysis of spatial data
- undertake spatial data input/output in standard formats

8

Writing to files

9

Important first step this week and for CWK

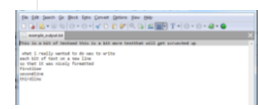
```
# writing to files
# Open file with 'w' - write option
myfile=open('H:\\Teaching\\GGGA\\2018\\week3_flow\\Gary_Run_thru\\example_output.txt', 'w')

# writelines writes a sequence of characters
myfile.writelines("This is a bit of text")
myfile.writelines("and this is a bit more text")
myfile.writelines("that will get scrunched up")
# you can write format lines (this is a newline character)
myfile.writelines("\n")
myfile.writelines("\n what I really wanted to do was to write")
myfile.writelines("\neach bit of text on a new line")
myfile.writelines("\nso that it was nicely formatted")

lines=["\nfirstline", "\nsecondline", "\nthirdline"]
myfile.writelines(lines)

myfile.close()
```

- 'w' is the write option
- Writelines adds text
- Need to use '\n' for newline



10

Handling multiple objects

11

This week

- Look at Arrays
  - What are they
  - How they work
  - How to define one in Python
  - Manipulating an array (array operations)
- Raster
  - Creating a raster class
  - Traversing a raster
- Focal functions and neighbourhood analysis

12

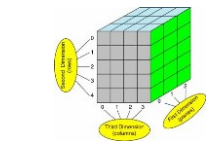
Handling multiple objects

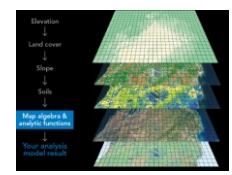
- So far in the course we have used lists and dictionaries to group objects or values
- Groups are useful because
  - You can group anything in any combination
  - They are dynamic
- BUT…
  - Groups can be awkward to use when you have more than 1-dimension in the data (a grid)
  - You need lists of lists

13

Arrays

- Way of representing groups of values
- Can be 1-dimensional (like a list)
  - 1 3 5 6 8 2 1 9 13 2 56 12
- 2-dimensional (eg raster grid)
  12 45 23 96
  45 97 43 88
  83 32 21 47
  43 94 41 12
- 3-dimensional or more (raster stack)





Arrays

- Not part of standard python
- Need to import numerical Python Modules to use them
- Numerical Python or NumPy
  - Comes as part of most standard distributions
  - Use indexes to access individual items in similar way to lists

15

How Arrays Work

- Arrays are objects
- Have an attribute array.shape() which is a tuple describing the number and size of dimensions
- When you create an array object you create an object with a fixed number of slots to hold data of the same type
- Each slot is referenced using:
  - A reference to the array (usually a variable)
  - The numeric position of the 'slot' starting at zero (eg like lists)

Index	Value
0	245
1	457
2	632
3	534
4	835
5	154
6	332
7	825

16

Defining Arrays

Note on coding: We build code up in this section, so each new chunk should be included in the editor window underneath the preceding code. Do not over write or delete previous chunks.

17

Defining 1-dimensional arrays

- There are several ways to create arrays
- Can create a 1-dimensional array from a list

```
import numpy as np

mylist=range(10)
print(mylist)
print("The item at index position 5 is: " + str(mylist[5]))

myArray=np.array(mylist)

print(myArray)
print("The item at index position 5 is: " + str(myArray[5]))
myArray[3:7] #also can slice the array just like in a list.

```

You can reference individual items just like in a list

18

Defining 2-dimensional arrays

```
####### defining a 2-d array ###################
# the same technique can be used for a 2-d array,

mylist2 = range(21,31)
print(mylist)
print(mylist2)


my2Darray=np.array([mylist, mylist2])
print(myArray)
print(my2Darray)
```

- The structure of arrays:
  - A 2-D array is an array of 1D arrays
  - A 3-D array is an array of 2D arrays, which themselves are arrays of 1-D arrays

19

Task

- Play around with the code for a couple of minutes,
- Can you make a 3-d?
- What happens if mylist2 has a different length to mylist?

20

Defining Sub-arrays

```
81
82 mylist3=range(41,51)
83
84 my2Darray=np.array([mylist, mylist2, mylist3])
85 print(my2Darray)
86
87 subArray=my2Darray[1]
88 print(" ")
89 print(subArray)
90 print(" ")
91 print(subArray[3])
92 #alternatively:
93 print(my2Darray[1][3]) #referencing the same element
94 print(my2Darray[1,3]) # same element
95
```

2D array is an array of 1D arrays

So: a single element of a 2D array is a 1D array

Conventional way to access an element:

my2Darray[i,j]

21

Reshaping Arrays

```
103 mylist = range(30)
104 myArray = np.array(mylist)
105 print(myArray)
106
107 print("shape of array is"+str(myArray.shape))
108
109 b=np.reshape(myArray, (15,2)) #.reshape is a method in numpy
110 print(b)
111 print("shape of array is"+(str(b.shape)))
112
113 c=np.reshape(myArray, (5,3,2))
114 print(c)
115 print("shape of array is"+str(c.shape))
116
117 c.shape=(30) # diretly modify the shape of the array
118 print(c)
119
120 ##################################################
```

Add comments to your code to describe what each reshape is doing

What do the numbers refer to in the reshape argument?

Directly modify the shape attribute of the array

22

Other ways to define arrays

```
122 # can initialise an array to contain all zeros or all ones:
123
124 import numpy as np
125 myArray=np.zeros((10))
126 print(myArray)
127
128 myArray=np.ones((3,3))
129 print(myArray)
130
131 myArray=np.arange(5) # array with 0,1,2,3,4,
132 print(myArray)
133
134 myArray=np.empty((4,4))#populates array with undefined values.
135 print(myArray)
136
```

```
In [2]: import numpy as np
   ...: myArray=np.zeros((10))
   ...: print(myArray)
   ...:
   ...: myArray=np.ones((3,3))
   ...: print(myArray)
   ...:
   ...: myArray=np.arange(5) # array with 0,1,2,3,4,
   ...: print(myArray)
   ...:
   ...: myArray=np.empty((4,4))#populates array with undefined values.
   ...: print(myArray)
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
[0 1 2 3 4]
[[ 4.67296746e-307   1.69111804e-306   7.56685915e-307   1.86921279e-306]
 [ 1.11256817e-306   1.86011362e-306   1.42417221e-306   1.11260619e-306]
 [ 8.90894053e-307   1.86919378e-306   1.86807792e-306   1.37962456e-306]
 [ 1.69111861e-306   1.78828164e-306   1.37961777e-306   7.56599887e-307]]
```

23

Array Operations

- Allows us to perform the same operation on groups of numbers without explicitly iterating –

```
141 myArray=np.arange(6)
142 print(myArray)
143
144 myArray=myArray+3
145 print(myArray)
146
147 myArray=myArray*myArray
148 print(myArray)
149
150 b=myArray
151 c=myArray+b
152 print(c)
153
```

```
154 #perform more complex operations
155 #good start is to look at the options when typing np.
156 np.
157 ###
158 ##
159 # T
160
161 imp
162 imp
163
164 ran
165 col
166
167 my2
168
```

abs
absolute
absolute_import
add
add_docstring
add_newdoc
add_newdoc_ufunc
add_newdocs
alen
all

24

Any Questions?

Break time

25

Raster datasets

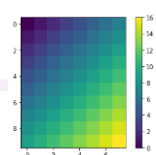
Important data input for spatial analysis

Has everyone worked with raster's before?

26

Plotting raster data

```
159
160 import numpy as np
161 import matplotlib.pyplot as mp
162
163 rows =10
164 cols =8
165
166 my2Darray = np.zeros((rows,cols)) #create array of 10x8 all zeros
167
168 for i in range (rows):
169     for j in range (cols):
170         my2Darray[i,j]=i+j #use two loops to create new values for the array
171
172 mp.imshow(my2Darray) #imshow = image show)
173 mp.colorbar() #add default colour bar
```



27

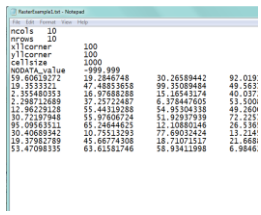
Introduction to working with raster data

- Raster surface is a continuous grid of evenly spaced points
- Each point has some value
- To use raster as spatial data we need:
  - Number of columns
  - Number of rows
  - Cellsize
  - Minimum x and y coordinate
  - Nodata values

Ideally stored as a 2-D array

28

ArcGIS Ascii Raster Format



```
ncols   10
nrows   10
xllcorner       100
yllcorner       100
cellsize        1000
NODATA_value    -999.999
59.60619272     19.2846748      30.26589442     92.019
19.3533321      47.48853658     99.35089484     49.563
2.355480353     16.97688288     15.16543174     40.037
2.298712689     37.25722487     6.378447605     53.500
12.96229128     55.44319288     54.95304338     49.260
30.72197948     51.97606724     51.92937939     72.225
95.09563511     65.24644625     12.10880146     26.536
30.40689342     10.75513293     77.69032424     13.214
19.37982789     45.66774308     18.71071317     21.668
53.47098335     63.61581746     58.93411998     6.9846
```

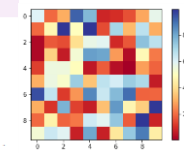
29

Reading in raster data to Python

```
from RasterHandler import readRaster
import matplotlib.pyplot as mp

raster = readRaster("RasterExample1.txt")

mp.imshow(raster.getData(), cmap='RdYlBu')
mp.colorbar()
mp.show()
```

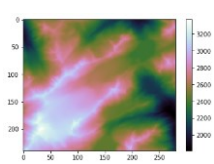


Cmap = colour map: see codes here: https://matplotlib.org/users/colormaps.html

30

Task: Reading in raster data

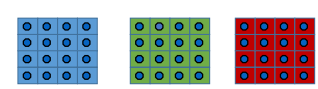
- Display the second raster file in an appropriate colour scheme for a Digital Elevation Model (DEM)



31

Introduction to working with raster data

- Satellite imagery is a collection of 2-D arrays



32

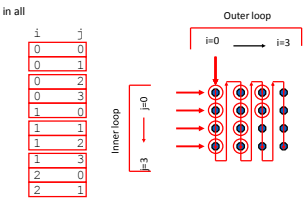
A raster class in Python

- Need to represent raster's in python and allows us to perform particular operations
- We can store the grid array and grid information as instance variables
- Use methods to pass back these instance variables
  - This means that the data can be accessed but not changed from outside the class which is good programming practice.
- This is what the raster.py module that you have been given is doing

33

Raster traversing (iterating)

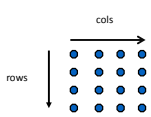
- To visit every point in the grid we have to access each array location.
- In a 4x4 array there are 16 locations in all
- So we need two nested loops



34

Raster Traversing (iterating through every cell)

- To visit every point in the grid we have to access each array location.
- In a 4x4 array there are 16 locations in all
- So we need two nested loops




```
import numpy as np
import matplotlib.pyplot as mp

rows =4
cols =4

my2Darray = np.zeros((rows,cols))

for i in range (rows):
    for j in range (cols):
        my2Darray[i,j]=i+j

for i in range(rows):
    for j in range (cols):
        print(str(i)+", "+str(j)+", "+str(my2Darray[i,j])+"\n")
```

35

Raster traversing

- As we saw earlier, arrays allow more powerful analysis
- Task, run the following code and comment, we will discuss before we move on

```
224 rastersum=0
225 for i in range(rows):
226     for j in range(cols):
227         rastersum=rastersum+my2Darray[i,j]
228 print (rastersum)
```

36

Built in method?

- Numpy
- print(np.mean(my2Darray))
- print(np.sum(my2Darray))

37

Any Questions?

Break time

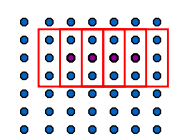
38

Raster Analysis

39

Focal functions

- A common spatial analysis task is to search the local area around a cell and calculate functions on this local area
- For example, in the search area, what is the:
  - Mean or sum
  - Maximum or minimum
  - Modal value (useful for categorical data)
- Focal function can be a box or a moving window



40

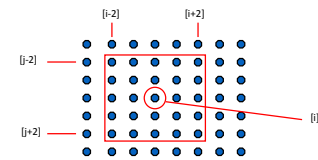
Focal functions: how do we engineer this?

- Think of the focal box as a mini sub-raster within the main raster
- We have to traverse this sub-raster in the same way as our main raster
- We are traversing every point in our main raster and generating a focal function value at that point
- So we are doing two things
  1. Traversing our main raster
  2. At each point in that main raster we traverse rasters surrounding that point
- This means we have two nested traverses in the code
- Could create lots of separate mini-rasters but instead we can do it virtually

41

Focal Functions: Example

- Search box is 5 x 5 cells
- If in the main raster we are at point i,j
- Want to search all points in a box from i-2 to i+2 and
- j-2 to j+2

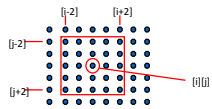


42

How do we engineer this in Python?

- Additional pair of nested loops
- Would be inside the previous pair of nested loops

```
#use a 5x5 window
#for any place in the raster
for ii in range(i-2, i+3):
    for jj in range(j-2,j+3):
        focalSum[i][j]=focalSum[i][j] + data[ii][jj];
##this doesnt work its just an example code set
```



43

How do we engineer this in Python?

- Need one additional thing to check for corners

```
246 #additonal check to cope with corners
247 focalSum=0
248 cellsVisited=0
249 for ii in range(i-2, i+3):
250     for jj in range(j-2, j+3):
251         #check we are still within array bounds
252         if (ii>-1 and ii<cols and jj>-1 and jj<cols):
253             focalSum=focalSum+data[ii,jj]
254             cellsVisited=cellsVisited+1
255 focalMean[i,j]=focalSum/cellsVisited
256 ##this doesnt work its just an example code set
```

Cell is worth evaluating if:
(not off the left) and (not off the right) and (not off the bottom)
and (not off the top)

44

How do we engineer this in Python?

- Which is expressed as:

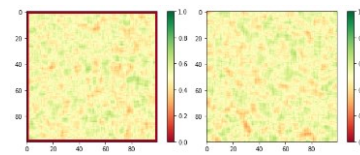
  (ii>-1) and (ii<cols) and (jj>-1) and (jj<cols)

- Corners of the raster break two of these conditions, but that's kind of irrelevant.

- One interesting (but unrelated) point is once FALSE is returned from one of these conditions the rest are not evaluated, as they cannot impact the outcome.
- This feature is often used to stop needless expensive checks, and can be quite elegant and concise code style.

45

Example output



46

Task

Open lecture3_focal.py to see a version of focal points.

1. Comment the code so you are sure you know what is happening – upload the commented code to the learning diary.
2. The first focal array method (slide 44) doesn't quite work Fix it! – upload this to learning diary.
3. Compare the two approaches, are there any differences?
4. Discuss in the learning diary how a focal algorithm could be used in the real world with real raster data

47

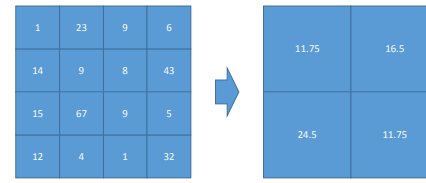
Any Questions?

Break time

48

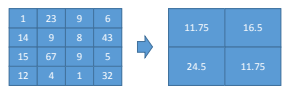
Resampling raster data taking focal points further

Resampling

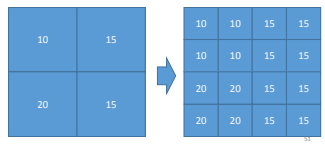


Resampling

- Aggregating or smoothing



- Disaggregating



Resampling: Why?

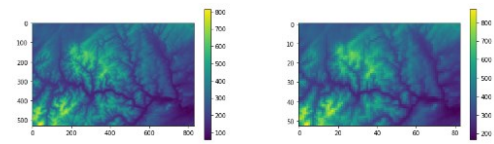
- All arrays have to have same dimensions to be analysed together
- Real data often not in the same format
- Land Use = 500 m resolution
- DEM = 1 km resolution
- Integrating the two together needs resampling before anything else can happen

Python code to aggregate

- Gdal
- We cannot use this in python 3 as it doesn't work properly with anaconda
- But for those of you who want to look into this without conda it is possible.

53

Aggregated Raster



54

Example using numpy

```
39
40          newRowNum = self.getRows() // factor
41          newColNum = self.getCols() // factor
42          newdata = np.zeros([newRowNum, newColNum])
43
44          for i in range(newRowNum):
45              for j in range(newColNum):
46                  sumCellValue = 0.0
47
48                  for k in range(factor):
49                      for l in range(factor):
50                          sumCellValue += self._data[i*factor + k, j*factor + l]
51
52                  newdata[i,j] = sumCellValue / factor / factor + 100
53
54          return Raster(newdata, self._orgs[0], self._orgs[1], self._cellsize*factor)
```

Doesn't quite work… this is for you to work on.

55

Modal Values

- Most common value in a data set
- Couple of ways to search for modal values
- If we know how many possible unique values we have (landcover classes) we can tally values in a bin for each class
- If we do not know beforehand how many values we might have to first sort a list of all the values and go through the sorted list keeping a count of when the value changes
- We will look at this method with arrays, then a slightly different method.

56

Binning

- If we have a small raster that had three land use classes (A, B, C)
- We can go through the raster and calculate how many of each class there are
- Code would look something like this:

```
if (raster[i,j] ==‘A’):
        counterA +=1
if (raster[i,j] ==‘B’):
        counterB +=1
if (raster[i,j] ==‘C’):
        counter +=1
```

A	B	C	A	B
C	C	C	B	A
C	B	C	B	A
A	A	A	A	C
A	A	A	C	C

This would give the modal value, but we have to know the classes.

Binning

- If we had a known number of (n) categories we could do something more adaptive
- Our raster only has integer values between 0 and n
- We could have an array to store the bin values themselves

```
binValue = np.zeroes(n)
```

- Then in a loop that traverses the whole array:

```
val = raster[i,j]
binValue[val] +=1
```

- This adds one to the corresponding bin every time we encounter an array [i,j] element that is of a particular value

Binning

- What would happen if we had a set number of values but they were not an ordered set of numbers?
- By storing our bin values in another array:

```
binValue = np.zeroes(n)
```

- Assuming *categories* held a list of all our known unique bin values (A, B,C) within our main loop we could now put:

```
for v in range(n)
        if (raster[i,j] == categories [v]):
                binValue[v]+=1
```


Binning

- Can you create the code yourselves from the Pseudo – code or the English description?
- Try this, and if not, open lecture3_binning_example_modal.py in the start package.
- Comment the code provided

Binning

- The method is slow
- If we had a larger dataset would be a problem
- Why is it slow? The code searches for the right value every time before it adds a value to the bin tally

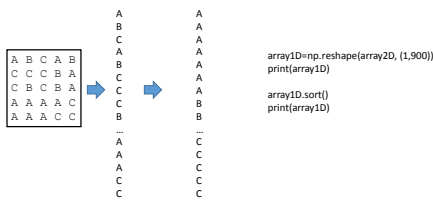
61

Modal values by sorting

- What could we do if we didn't know how many categories we might have in our data?
- We could traverse the array once and find out how many unique categories we have
- Or
- Quicker way is to allocate values in the 2D array into a 1D array with the same number of data slots and sort them (e.g. a 4x4 array has 16 slots)
  - First reshape the array into a 1D array form

62

Reshape array and sort




```
array1D=np.reshape(array2D, (1,900))
print(array1D)

array1D.sort()
print(array1D)
```

63

Modal value algorithm with sort

```
def findModal(data):
    myData.sort()
    i=1
    modalCount=1
    modalVal=data[0]
    last=data[0]
    count=1
    while (i<len(data)):

        if (data[i]==last):
            count=count+1
        else:
            count=1

        if (count>modalCount):
            modalCount=count
            modalVal=data[i]

        last=data[i]
        i=i+1

    print ("i={}, modalCount={}, modalVal={}".format(i, modalCount, modalVal))
myData = [6,6,1,1,2,3,3,3,4,4,4,5,2,4,4]
print (myData)
findModal(myData)
```

Another way of doing this by producing a dictionary is provided in the starter package

Lecture3_modal2.py

64

Summary

- Raster
- Arrays
- Some basic raster manipulation and analysis
- Next week we take this further.

65

Learning Diary Task

- Slide 47 and the following…
- Comment this function and add the code with comments to the learning diary.
- Compare the algorithm to the additional modal algorithm provided called modal_algorithm2.py

- What are the main differences between the two algorithms?
- Consider having a Global Land Use Map with 25 land use classes and 25 million raster cells. Can you see any potential problems for running either of the algorithms? Which would be more efficient?

66

Modal Algorithm 2

```
def findModal(data):
    counts={}

    for item in data:
        if item in counts:
            counts[item] += 1
        else:
            counts[item] = 1


    print ("counts={}".format(counts))

    modalCount=0
    modalVal=None
    for key,count in counts.items():
        if count>modalCount:
            modalVal=key
            modalCount=count

    print ("modalCount={}, modalVal={}".format(modalCount, modalVal))



LandUse = ['Agri', 'Agri', 'Agri', 'Grass', 'Urban', 'Woody', 'Agri', 'Woody', 'Shrub',
           'Agri', 'Grass', 'Barren', 'Woody']

findModal(LandUse)
```

67