Python Scripting for Spatial Data Processing.

Pete Bunting and Daniel Clewley

Teaching notes on the MSc's in Remote Sensing and GIS.

October 27, 2014
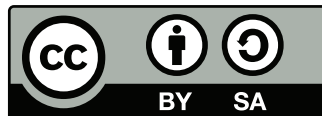
Aberystwyth University

Institute of Geography and Earth Sciences.

Acknowledgements

Authors

Peter Bunting

Dr Pete Bunting joined the Institute of Geography and Earth Sciences (IGES), Aberystwyth University, in September 2004 for his Ph.D. where upon completion in the summer of 2007 he received a lectureship in remote sensing and GIS. Prior to joining the department, Peter received a BEng(Hons) in software engineering from the department of Computer Science at Aberystwyth University. Pete also spent a year working for Landcare Research in New Zealand before rejoining IGES in 2012 as a senior lecturer in remote sensing.

Contact Details

EMail: pfb@aber.ac.uk

Senior Lecturer in Remote Sensing
Institute of Geography and Earth Sciences
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
United Kingdom

Daniel Clewley

Dr Dan Clewley joined IGES in 2006 undertaking an MSc in Remote Sensing and GIS, following his MSc Dan undertook a Ph.D. entitled "Retrieval of Forest Biomass and Structure from Radar Data using Backscatter Modelling and Inversion" under the supervision of Prof. Lucas, Dr. Bunting and Prof. Mahta Moghaddam. Prior to joining the department Dan completed his BSc(Hons) in Physics within Aberystwyth University. Dan is currently an Airborne Remote Sensing Data Analyst at Plymouth Marine Laboratory. He writes a blog on open source software in GIS and Remote Sensing (`http://spectraldifferences.wordpress.com/`)

Contact Details

Email: daniel.clewley@gmail.com

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.1.1 What is Python?

Python is a high level scripting language which is interpreted, interactive and object-oriented. A key attribute of python is its clear and understandable syntax which should allow you to quickly get up to speed and develop useful application, while the syntax is similar enough to lower level languages, for example C/C++ and Java, to provide a background from which you can grow your expertise. Python is also a so called memory managed language, meaning that you the developer are not directly in control of the memory usage within your application, making development much simpler. That is not saying that memory usage does not need to be considered and you, the developer, cannot influence the memory footprint of your scripts but these details are out of the scope of this course. Python is cross-platform with support for Windows, Linux, Mac OS X and most other UNIX platforms. In addition, many libraries (e.g., purpose built and external C++ libraries) are available to python and it has become a very popular language for many applications, including on the internet and within remote sensing and GIS.

1.1.2 What can it be used for?

Python can be used for almost any task from simple file operations and text manipulation to image processing. It may also be used to extend the functionality of other, larger applications.

1.1.3 A word of warning

There are number of different versions of python and these are not always compatible. For these worksheets we will be using version 3.X (at the time of writing the latest version is 3.3.0). With the exception of the quiz in Chapter 2, where `raw_input` must be used instead of `input`, the examples will also work python 2.7. One of the most noticeable differences between python 2 and python 3 is that the print statement is now a function. So whilst:

```
print "Hello World"
```

will work under python 2, scripts using it won't run under python 3 and must use:

```
print("Hello World")
```

instead. As the second is backwards compatible with python 2 it is good practice to use this, even if you are working with python 2.

It is possible to have both python 2 and python 3 installed, in this case python 3 will be called using `python3`, if you're system is set up like this remember to use `python3` wherever you see `python` in this tutorial.

1.2 Example of Python in use

1.2.1 Software in Python

Many applications have been built in python and a quick search of the web will reveal the extent of this range. Commonly, applications solely developed in python are web applications, run from within a web server (e.g., Apache; `http:`

`//httpd.apache.org` with `http://www.modpython.org`) but Desktop applications and data processing software such as TuiView (`https://bitbucket.org/chchrsc/tuiview`) and RIOS (`https://bitbucket.org/chchrsc/rios`) have also been developed.

In large standalone applications Python is often used to facilitate the development of plugins or extensions to application. Examples of python used in this form include ArcMap and SPSS.

For a list of applications supporting or written in python refer to the following website `http://en.wikipedia.org/wiki/Python_software`.

1.3 Python Libraries

Many libraries are available to python. Libraries are collections of functions which can be called from your script(s). Python provides extensive libraries (`http://docs.python.org/lib/lib.html`) but third parties have also developed additional libraries to provide specific functionality (e.g., plotting). A list of available libraries is available from `http://wiki.python.org/moin/UsefulModules` and by following the links provides on the page.

The following sites provide links to libraries and packages specific to remote sensing and GIS, many of which are open source with freely available software packages and libraries for use with python.

- `http://freegis.org`

- `http://opensourcegis.org`

- `http://www.osgeo.org`

1.4 Installing Python

For this tutorial Python alongside the libraries GDAL (`http://www.gdal.org`), numpy (`http://www.numpy.org`), scipy (`http://www.scipy.org`), RIOS (`https:`

//bitbucket.org/chchrsc/rios) and matplotlib (http://matplotlib.sourceforge.
net) are required, built against the version of Python you are using. There are
a number of ways of installing these packages, the recomended option is through the
Anaconda Python Distribution (http://docs.continuum.io/anaconda/install.
html), versions are available for Linux, OS X and Windows. Once Anaconda has
been installed, the additional packages required can be installed using:

```
conda install -c osgeo \
    gdal rios
```

1.5   Text Editors

To write your Python scripts a text editor is required. A simple text editor such
as Microsoft's Notepad will do but it is recommended that you use a syntax aware
editor that will colour, and in some cases format, your code automatically. There
are many text editors available for each operating system and it is up to you
to choose one to use. The recommend editor for this course is Spyder which is
installed with Anaconda. From within Spyder you can directly run your Python
scripts (using the run button), additionally it will alert you to errors within your
scripts before you run them.

1.5.1   Windows

Under Windows, the notepad++ (http://notepad-plus.sourceforge.net) text
editor can also be used. Notepad++ is a free to use open source text editor and
can therefore be downloaded and installed onto any Windows PC. If you use this
editor it is recommended you change the settings for Python to use spaces instead
of tabs using the following steps:

1. Go to Setting – Preferences

2. Select 'Language Menu / Tab Settings'

3. Under 'Tab Settings' for python tick 'Replace by space'

1.5.2 Linux

Under Linux either the command line editor ne (nice editor), vim or its graphic interface equivalent gvim is recommend but kdeveloper, gedit and many others are also good choices.

1.5.3 Mac OS X

Under Mac OS X either BBEdit, SubEthaEdit or TextMate are recommended, while the freely available TextWrangler is also a good choice. The command line editors ne and vi are also available under OS X.

1.5.4 Going between Windows and UNIX

If you are writing your scripts on Windows and transferring them to a UNIX/Linux machine to be executed (e.g., a High Performance Computing (HPC) environment) then you need to be careful with the line ending (the invisible symbol defining the end of a line within a file) as these are different between the various operating systems. Using notepad++ line ending can be defined as UNIX and this is recommended where scripts are being composed under Windows.

Alternatively, if RSGISLib is installed then the command flip can be used to convert the line ending, the example below converts to UNIX line endings.

```
flip -u InputFile.py
```

1.6 Starting Python

Python may be started by opening a command window and typing:

```
python
```

(Alternatively select python(x,y) – Command Prompts – Python interpreter from the windows start menu).

This opens python in interactive mode. It is possible to perform some basic maths try:

```
>>> 1 + 1
2
```

To exit type:

```
>>>exit()
```

To perform more complex tasks in python often a large number of commands are required, it is therefore more convenient to create a text file containing the commands, referred to as a 'script'

1.6.1 Indentation

There are several basic rules and syntax which you need to know to develop scripts within Python. The first of which is code layout. To provide the structure of the script Python uses indentation. Indentation can be in the form of tabs or spaces but which ever is used needs to be consistent throughout the script. The most common and recommend is to use 4 spaces for each indentation. The example given below shows an if-else statement where you can see that after the if part the statement which is executed if the if-statement is true is indented from rest of the script as with the corresponding else part of the statement. You will see this indentation as you go through the examples and it is important that you follow the indentation shown in the examples or your scripts will not execute.

```
1  if␣x␣==␣1:
2  ␣␣␣␣x␣=␣x␣+␣1
3  else:
4  ␣␣␣␣x␣=␣x␣-␣1
```

1.6.2 Keywords

As with all scripting and programming languages python has a set of keywords, which have special meanings to the compiler or interpreter when the code is executed. As with all python code, these keywords are case sensitive i.e., 'else' is a

keyword but 'Else' is not. A list of pythons keywords is given below:

Table 1.1: Keywords within the Python language

and	as	assert	break
class	continue	def	del
elif	else	exec	except
finally	for	from	global
if	import	in	is
lambda	not	or	pass
print	raise	return	try
while	with	yield	

1.6.3   File Naming

It is important that you use sensible and identifiable names for all the files you generate throughout these tutorial worksheets otherwise you will not be able to identify the script at a later date. Additionally, it is highly recommended that you do not included spaces in file names or in the directory path you use to store the files generated during this tutorial.

1.6.4   Case Sensitivity

Something else to remember when using python, is that the language is case sensitivity therefore if a name is in lowercase then it needs to remain in lowercase everywhere it is used.

For example:

```
VariableName is not the same as variablename
```

1.6.5   File paths in examples

In the examples provided (in the text) file paths are given as './PythonCourse/TutorialX/File.xxx'. When writing these scripts out for yourself you will need to update these paths to the location on your machine where the files are located (e.g., /home/pete.bunting

or C:\). Please note that it is recommended that you do not have any spaces within your file paths. In the example (answer) scripts provided no file path has been written and you will therefore need to either save input and output files in the same directory as the script or provide the path to the file. Please note that under Windows you need to insert a double slash (i.e., \\) within the file path as a single slash is an escape character (e.g., \n for new line) within strings.

1.6.6 Independent Development of Scripts

There is a significant step to be made from working your way through notes and examples, such as those provided in this tutorial, and independently developing your own scripts from scratch. Our recommendation for this, and when undertaking the exercises from this tutorial, is to take it slowly and think through the steps you need to undertake to perform the operation(s) you need.

I would commonly first 'write' the script using comments or on paper breaking the process down into the major steps required. For example, if I were asked to write a script to uncompress a directory of files into another directory I might write the following outline, where I use indentation to indicate where a process is part of the parent:

```
1   # Get input directory (containing the compressed files)
2
3   # Get output directory (where the files, once uncompressed, will be placed).
4
5   # Retrieve list of all files (to be uncompressed) in the input directory.
6
7   # Iterator through input files, uncompressing each in turn.
8       # Get single file from list
9       # create command line command for the current file
10      # execute command
```

By writing the process out in this form it makes translating this into python much simpler as you only need to think of how to do small individual elements in python and not how to do the whole process in one step.

1.6.7 Getting Help

Python provides a very useful help system through the command line. To get access to the help run python from the terminal

```
> python
```

Then import the library want to get help on

```
>>> import math
```

and then run the help tool on the whole module

```
>>> import math
>>> help(math)
```

or on individual classes or functions within the module

```
>>> import osgeo.gdal
>>> help(math.cos)
```

To exit the help system just press the 'q' key on the keyboard.

1.7 Further Reading

- An Introduction to Python, G. van Rossum, F.L. Drake, Jr. Network Theory ISBN 0-95-416176-9 (Also available online - `http://docs.python.org/3/tutorial/`). Chapters $1 - 3$

- Python FAQ – `http://docs.python.org/faq/general.html`

- Python on Windows – `http://docs.python.org/faq/windows`

- How to think Like a Computer Scientist: Python Edition – `http://www.greenteapress.com/thinkpython/`

Chapter 2

The Basics

2.1 Hello World Script

To create your first python script, create a new text file using your preferred text
editor and enter the text below:

```python
1   #! /usr/bin/env python
2
3   #######################################
4   # A simple Hello World Script
5   # Author: <YOUR NAME>
6   # Emai: <YOUR EMAIL>
7   # Date: DD/MM/YYYY
8   # Version: 1.0
9   #######################################
10
11  print('Hello World')
```

Save your script to file (e.g., helloworld.py) and then run it either using a command
prompt (Windows) or Terminal (UNIX), using the following command:

```
> python helloworld.py
Hello World
```

To get a command prompt under Windows type 'cmd' from the run dialog box in the start menu (Start – run), further hints for using the command prompt are given below. Under OS X, terminal is located in within the 'Utilities' folder in 'Applications'. If you are using Spyder to create your Python scripts you can run by clicking the run button.

Hints for using the Windows command line
'cd' allows you to change directory, e.g.,

```
cd directory1\directory2
```

'dir' allows you to list the contents of a directory, e.g.,

```
dir
```

To change drives, type the drive letter followed by a colon, e.g.,

```
D:
```

If a file path has spaces, you need to use quote, e.g, to change directory:

```
cd "Directory with spaces in name\another directory\"
```

2.2 Comments

In the above script there is a heading detailing the script function, author, and version. These lines are preceded by a hash (#), this tells the interpreter they are comments and are not part of the code. Any line starting with a hash is a comment. Comments are used to annotate the code, all examples in this tutorial use comments to describe the code. It is recommended you use comments in your own code.

2.3 Variables

The key building blocks within all programming languages are variables. Variables allow data to be stored either temperately for use in a single operation or throughout the whole program (global variables). Within python the variable data type does not need to be specified and will be defined by the first assignment. Therefore, if the first assignment to a variable is an integer (i.e., whole number) then that variable will be an integer for the remained of the program. Examples defining variables are provided below:

```python
name = 'Pete' # String
age = 25 # Integer
height = 6.2 # Float
```

2.3.1 Numbers

There are three types of numbers within python:

Integers are the most basic form of number, contain only whole numbers where calculation are automatically rounded to provide whole number answers.

Decimal or floating point numbers provide support for storing all those number which do not form a whole number.

Complex provide support for complex numbers and are defined as $a + bj$ where a is the real part and b the imaginary part, e.g., $4.5 + 2.5j$ or $4.5 - 2.5j$ or $-4.5 + 2.5j$

The syntax for defining variables to store these data types is always the same as python resolves the suitable type for the variable. Python allows a mathematical operations to be applied to numbers, listed in Table reftab:maths

2.3.2 Boolean

The boolean data type is the simplest and just stores a true or false value, an example of the syntax is given below:

Table 2.1: The mathematical functions available within python.

Function	Operation
x + y	x plus y
x - y	x minus y
x * y	x multiplied by y
x / y	x divided by y
x ** y	x to the power of y
int(obj)	convert string to int
long(obj)	convert string to long
float(obj)	convert string to float
complex(obj)	convert string to complex
complex(real, imag)	create complex from real and imaginary components
abs(num)	returns absolute value
pow(num1, num2)	raises num1 to num2 power
round(float, ndig=0)	rounds float to ndig places

```
moveForwards = True
moveBackwards = False
```

2.3.3 Text (Strings)

To store text the string data type is used. Although not a base data type like a float or int a string can be used in the same way. The difference lies in the functions available to manipulate a string are similar to those of an object. A comprehensive list of functions is available for a string is given in the python documentation `http://docs.python.org/lib/string-methods.html`.

To access these functions the string modules needs to be imported as shown in the example below. Copy this example out and save it as StringExamples.py. When you run this script observe the change in the printed output and using the python documentation to identify what each of the functions lstrip(), rstrip() and strip() do.

```
1   #! /usr/bin/env python
2
3   #####################################
4   # Example with strings
```

```
5   # Author: <YOUR NAME>
6   # Emai: <YOUR EMAIL>
7   # Date: DD/MM/YYYY
8   # Version: 1.0
9   #######################################
10
11  import string
12
13  stringVariable = '       Hello World      '
14
15  print('\'' +  stringVariable + '\'')
16
17  stringVariable_lstrip = stringVariable.lstrip()
18  print('lstrip: \'' +  stringVariable_lstrip + '\'')
19
20  stringVariable_rstrip = stringVariable.rstrip()
21  print('rstrip: \'' +  stringVariable_rstrip + '\'')
22
23  stringVariable_strip = stringVariable.strip()
24  print('strip: \'' +  stringVariable_strip + '\'')
```

2.3.4   Example using Variables

An example script illustrating the use of variables is provided below. It is recommend you copy this script and execute making sure you understand each line. In addition, try making the following changes to the script:

1. Adding your own questions.

2. Including the persons name within the questions.

3. Remove the negative marking.

```
1   #! /usr/bin/env python
2
3   #######################################
4   # A simple script illustrating the use of
5   # variables.
6   # Author: <YOUR NAME>
```

```python
 7   # Emai: <YOUR EMAIL>
 8   # Date: DD/MM/YYYY
 9   # Version: 1.1
10   #
11   # Added helper function for 2/3 compatibility
12   ######################################

13
14   # Compatibility function for 2/3
15   import sys
16   def getInput(question):
17       """ Python 2/3 helper function
18           for getting input.
19       """
20       if sys.version[0] == '3':
21           answer = input(question)
22       else:
23           answer = raw_input(question)
24       return(answer)

25
26   score = 0 # A variable to store the ongoing score

27
28   # print is used to 'print' the text to the command line
29   print('#############################################')
30   print('Sample Python program which asks the user a few ' \
31         'simple questions.')
32   print('#############################################')

33
34   # input is used to retrieve user input from the
35   # command line
36   name = getInput('What is your name?\n')

37
38   print('Hello ' + name + '. You will now be asked a series' \
39   ' of questions please answer \'y\' for YES and \'n\' for ' \
40   'NO unless otherwise stated.')

41
42   print('Question 1:')
43   answer = getInput('ALOS PALSAR is a L band spaceborne SAR.\n')
44   if answer == 'y': # test whether the value returned was equal to y
45       print('Well done')
46       score = score + 1 # Add 1 to the score
47   else: # if not then the anser must be incorrect
```

```
48      print('Bad Luck')
49      score = score - 1 # Remove 1 from the score
50
51  print('Question 2:')
52  answer = getInput('CASI provides hyperspectral data in ' \
53  'the Blue to NIR part of the spectrum.\n')
54  if answer == 'y':
55      print('Well done')
56      score = score + 1
57  else:
58      print('Bad Luck')
59      score = score - 1
60
61  print('Question 3:')
62  answer = getInput('HyMap also only provides data in the ' \
63  'Blue to NIR part of the spectrum.\n')
64  if answer == 'y':
65      print('Bad Luck')
66      score = score - 1
67  else:
68      print('Well done')
69      score = score + 1
70
71  print('Question 4:')
72  answer = getInput('Landsat is a spaceborne sensor.\n')
73  if answer == 'y':
74      print('Well done')
75      score = score + 1
76  else:
77      print('Bad Luck')
78      score = score - 1
79
80  print('Question 5:')
81  answer = getInput('ADS-40 is a high resolution aerial ' \
82  'sensor capturing RGB-NIR wavelengths.\n')
83  if answer == 'y':
84      print('Well done')
85      score = score + 1
86  else:
87      print('Bad Luck')
88      score = score - 1
```

```python
89
90  print('Question 6:')
91  answer = getInput('eCognition is an object oriented ' \
92  'image analysis software package.\n')
93  if answer == 'y':
94      print('Well done')
95      score = score + 1
96  else:
97      print('Bad Luck')
98      score = score - 1
99
100 print('Question 7:')
101 answer = getInput('Adobe Photoshop provides the same ' \
102 'functionality as eCognition.\n')
103 if answer == 'y':
104     print('Bad Luck')
105     score = score - 1
106 else:
107     print('Well done')
108     score = score + 1
109
110 print('Question 8:')
111 answer = getInput('Python can be executed within ' \
112 'the a java virtual machine.\n')
113 if answer == 'y':
114     print('Well done')
115     score = score + 1
116 else:
117     print('Bad Luck')
118     score = score - 1
119
120 print('Question 9:')
121 answer = getInput('Python is a scripting language ' \
122 'not a programming language.\n')
123 if answer == 'y':
124     print('Well done')
125     score = score + 1
126 else:
127     print('Bad Luck')
128     score = score - 1
129
```

```
130  print('Question 10:')
131  answer = getInput('Aberystwyth is within Mid Wales.\n')
132  if answer == 'y':
133      print('Well done')
134      score = score + 1
135  else:
136      print('Bad Luck')
137      score = score - 1
138
139  # Finally print out the users final score.
140  print(name + ' you got a score of ' + str(score))
```

Note, due to differences between Python 2 and Python 3 a function is defined called 'getInput', which determines uses the appropriate function depending on the version of python being used with `input` used for Python 3 and `raw_input` used for Python 2.

2.4 Lists

Each of the data types outlined above only store a single value at anyone time, to store multiple values in a single variable a sequence data type is required. Python offers the List class, which allows any data type to be stored in a sequence and even supports the storage of objects of different types within one list. The string data type is a sequence data type and therefore the same operations are available.

List are very flexible structures and support a number of ways to create, append and remove content from the list, as shown below. Items in the list are numbered consecutively from 0-n, where n is one less than the length of the list.

Additional functions are available for List data types (e.g., len(aList), aList.sort(), aList.reverse()) and these are described in `http://docs.python.org/lib/typesseq.html` and `http://docs.python.org/lib/typesseq-mutable.html`.

2.4.1 List Examples

```python
1   #! /usr/bin/env python
2
3   #######################################
4   # Example with lists
5   # Author: <YOUR NAME>
6   # Emai: <YOUR EMAIL>
7   # Date: DD/MM/YYYY
8   # Version: 1.0
9   #######################################
10
11  # Create List:
12  aList = list()
13  anotherList = [1, 2, 3, 4]
14  emptyList = []
15
16  print(aList)
17  print(anotherList)
18  print(emptyList)
19
20  # Adding data into a List
21  aList.append('Pete')
22  aList.append('Dan')
23  print(aList)
24
25  # Updating data in the List
26  anotherList[2] = 'three'
27  anotherList[0] = 'one'
28  print(anotherList)
29
30  # Accessing data in the List
31  print(aList[0])
32  print(anotherList[0:2])
33  print(anotherList[2:3])
34
35  # Removing data from the List
36  del anotherList[1]
37  print(anotherList)
38
39  aList.remove('Pete')
40  print(aList)
```

2.4.2 n-dimensional list

Additionally, n-dimensional lists can be created by inserting lists into a list, a simple example of a 2-d structure is given below. This type of structure can be used to store images (e.g., the example given below would form a grey scale image) and additions list dimensions could be added for additional image bands.

```python
#! /usr/bin/env python

#######################################
# Example with n-lists
# Author: <YOUR NAME>
# Emai: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

# Create List:
aList = [
[1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[1,1,0,0,1,1,1,1,1,0,0,1,1,1],
[1,1,0,0,1,1,1,1,1,0,0,1,1,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1],
[1,1,1,1,1,1,0,1,1,1,1,1,1,1],
[1,1,1,1,1,1,0,1,1,1,1,1,1,1],
[1,1,1,1,1,0,0,0,1,1,1,1,1,1],
[1,0,1,1,1,1,1,1,1,1,1,1,0,1],
[1,0,1,1,1,1,1,1,1,1,1,1,0,1],
[1,1,0,0,0,0,0,0,0,0,0,0,1,1],
[1,1,1,1,1,1,1,1,1,1,1,1,1,1]
]

print(aList)
```

2.5 IF-ELSE Statements

As already illustrated in the earlier quiz example the ability to make a decision is key to any software. The basic construct for decision making in most programming and scripting languages are if-else statements. Python uses the following syntax for if-else statements.

```
if <logic statement>:
    do this if true
else:
    do this


if <logic statement>:
    do this if true
elif <logic statement>:
    do this if true
elif <logic statement>:
    do this if true
else
    do this
```

Logic statements result in a true or false value being returned where if a value of true is returned the contents of the if statement will be executed and remaining parts of the statement will be ignored. If a false value is returned then the if part of the statement will be ignored and the next logic statement will be analysis until either one returns a true value or an else statement is reached.

2.5.1 Logic Statements

Table 2.2 outlines the main logic statements used within python in addition to these statements functions which return a boolean value can also be used to for decision making, although these will be described in later worksheets.

Table 2.2: Logic statements available within python

Function	Operation	Example
==	equals	expr1 == expr2
>	greater than	expr1 > expr2
<	less than	expr1 < expr2
>=	greater than and equal to	$expr1 \geq expr2$
<=	less than and equal to	$expr1 \leq expr2$
not	logical not	not expr
and	logical and	expr1 and expr2
or	logical or	expr1 or expr2
is	is the same object	expr1 is expr2

2.6 Looping

In addition to the if-else statements for decision making loops provide another key component to writing any program or script. Python offers two forms of loops, while and for. Each can be used interchangeably given the developers preference and available information. Both types are outlined below.

2.6.1 while Loop

The basic syntax of the while loop is very simple (shown below) where a logic statement is used to terminate the loop, when false is returned.

```
while <logic statement> :
    statements
```

Therefore, during the loop a variable in the logic statement needs to be altered allowing the loop to terminate. Below provides an example of a while loop to count from 0 to 10.

```python
1  #! /usr/bin/env python
2
3  ######################################
4  # A simple example of a while loop
5  # Author: <YOUR NAME>
6  # Emai: <YOUR EMAIL>
7  # Date: DD/MM/YYYY
```

```
8   # Version: 1.0
9   ####################################
10
11  count = 0
12  while count <= 10:
13      print(count)
14      count = count + 1
```

2.6.2 for Loop

A for loop provides similar functionality to that of a while loop but it provides the counter for termination. The syntax of the for loop is provided below:

```
1  for <iter_variable> in <iterable>:
2      statements
```

The common application of a for loop is for the iteration of a list and an example if this is given below:

```
1   #! /usr/bin/env python
2
3   ####################################
4   # A simple example of a for loop
5   # Author: <YOUR NAME>
6   # Emai: <YOUR EMAIL>
7   # Date: DD/MM/YYYY
8   # Version: 1.0
9   ####################################
10
11  aList = ['Pete', 'Richard', 'Johanna', 'Philippa', 'Sam', 'Dan', 'Alex']
12
13  for name in aList:
14          print('Current name is: ' + name)
```

A more advance example is given below where two for loops are used to iterate through a list of lists.

```
1   #! /usr/bin/env python
2
```

```
3   ########################################
4   # Example with for loop and n-lists
5   # Author: <YOUR NAME>
6   # Emai: <YOUR EMAIL>
7   # Date: DD/MM/YYYY
8   # Version: 1.0
9   ########################################
10
11  # Create List:
12  aList = [
13  [1,1,1,1,1,1,1,1,1,1,1,1,1,1],
14  [1,1,0,0,1,1,1,1,1,0,0,1,1,1],
15  [1,1,0,0,1,1,1,1,1,0,0,1,1,1],
16  [1,1,1,1,1,1,1,1,1,1,1,1,1,1],
17  [1,1,1,1,1,1,0,1,1,1,1,1,1,1],
18  [1,1,1,1,1,1,0,1,1,1,1,1,1,1],
19  [1,1,1,1,1,0,0,0,1,1,1,1,1,1],
20  [1,0,1,1,1,1,1,1,1,1,1,1,0,1],
21  [1,0,1,1,1,1,1,1,1,1,1,1,0,1],
22  [1,1,0,0,0,0,0,0,0,0,0,0,1,1],
23  [1,1,1,1,1,1,1,1,1,1,1,1,1,1]
24  ]
25
26  for cList in aList:
27          for number in cList:
28              # Print with a space at the end
29              # rather than a new line
30                  print (number,end=" ")
31          print()
```

2.7 Exercises

During this tutorial you should have followed through each of the examples and
experimented with the code to understand each of components outlined. To test
your understanding of all the material, you will now be asked to complete a series
of tasks:

    1. Update the quiz so the questions and answers are stored in lists which are

iterated through as the script is executed.

2. Create a script that loops through the smiling face 2-d list of lists flipping it so the face is up side down.

2.8 Further Reading

- An Introduction to Python, G. van Rossum, F.L. Drake, Jr. Network Theory ISBN 0-95-416176-9 (Also available online - `http://docs.python.org/3/tutorial/`) - Chapters 4 and 5.

- Spyder Documentation – `http://packages.python.org/spyder/`

- Python Documentation – `http://www.python.org/doc/`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

- How to think Like a Computer Scientist: Python Edition – `http://www.greenteapress.com/thinkpython/`

- Learn UNIX in 10 minutes – `http://freeengineer.org/learnUNIXin10minutes.html` (Optional, but recommended if running on OS X / Linux)

Chapter 3

Text Processing

3.1 Read a Text File

An example of a script to read a text file is given below, copy this example out and use the numbers.txt file to test your script. Note, that the numbers.txt file needs to be within the same directory as your python script.

```python
#! /usr/bin/env python

######################################
# A simple example reading in a text file
# two versions of the script are provided
# to illustrate that there is not just one
# correct solution to a problem.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
######################################

import string

# 1) Splits the text file into individual characters
# to identify the commas and parsing the individual
# tokens.
```

```
19   numbers = list()
20   dataFile = open('numbers.txt', 'r')
21
22   for eachLine in dataFile:
23       #print(eachLine)
24       tmpStr = ''
25       for char in eachLine:
26           #print(char)
27           if char.isdigit():
28               tmpStr += char
29           elif char == ',' and tmpStr != '':
30               numbers.append(int(tmpStr))
31               tmpStr = ''
32       if tmpStr.isdigit():
33           numbers.append(int(tmpStr))
34
35   print(numbers)
36   dataFile.close()
37
38   # 2) Uses the string function split to line from the file
39   # into a list of substrings
40   numbers = list()
41   dataFile = open('numbers.txt', 'r')
42
43   for eachLine in dataFile:
44       #print eachLine
45       substrs = eachLine.split(',',eachLine.count(','))
46       #print substrs
47       for strVar in substrs:
48           if strVar.isdigit():
49               numbers.append(int(strVar))
50
51   print(numbers)
52   dataFile.close()
```

As you can see reading a text file from within python is a simple process[1]. The first step is to open the file for reading, option r is used as the file is only going to be read, the other options are available in Table 3.1. If the file is a text file then

---

[1] If your data are in tabular format (e.g., CSV) the csv module in the Python Standard Library and the genfromtxt from NumPy provide even simpler ways of reading data.

the contents can then be read a line at a time, if a binary file (e.g., tiff or doc) then reading is more complicated and not covered in this tutorial.

Table 3.1: Options when opening a file.

File Mode	Operations
r	Open for read
w	Open for write (truncate)
a	Open for write (append)
r+	Open for read/write
w+	Open for read/write (truncate)
a+	Open for read/write (append)
rb	Open for binary read
wb	Open for binary write (truncate)
ab	Open for binary write (append)
rb+	Open for read/write
wb+	Open for read/write (truncate)
ab+	Open for read/write (append)

Now your need to adapt the one of the methods given in the script above to allow numbers and words to be split into separate lists. To do this you will need to use the isalpha() function alongside the isdigit() function. Adapt the numbers.txt file to match the input shown below and then run your script and you should receive the output shown below:

Input:

```
1,
2,pete,
3,
4,dan,5,
6,7,8,richard,10,11,12,13
```

Output:

```
>python simplereadsplit.py
[1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13]
['pete', 'dan', 'richard']
```

3.2 Write to a Text File

Writing to a text file is similar to reading from the file. When opening the file two choices are available either to append or truncate the file. Appending to the file leaves any content already within the file untouched while truncating the file removes any content already within the file. An example of writing a list to a file with each list item on a new line is given below.

```python
#! /usr/bin/env python

######################################
# A simple script parsing numbers of
# words from a comma seperated text file
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
######################################

aList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
'one', 'two', 'three', 'four', 'five',
'six', 'seven', 'eight', 'nine', 'ten']

dataFile = open('writetest.txt', 'w')

for eachitem in aList:
    dataFile.write(str(eachitem)+'\n')

dataFile.close()
```

3.3 Programming Styles

There are two main programming styles, both of which are supported by python, and these are procedural and object oriented programming. Procedural programming preceded object oriented programming and procedural scripts provide lists of commands which are run through sequentially.

Object oriented programming differs from procedural programming in that the program is split into a series of objects, usually representing really world objects or functionality, generally referred to as a 'class'. Objects support the concepts of inheritance where functionality can be used in many sub-objects. For example, a Person class maybe written with functions such as eat, drink, beat heart etc. and specialist sub-objects may then be created with Person as a super-object, for example child, adult, male and female. These objects all require the functionality of Person but it is inefficient to duplicate the functionality they share individual rather then group this functionality into the Person class.

This course will concentrate on basic object oriented programming but below are the basic python file outlines for both procedural and object oriented scripts.

3.3.1 Procedural Programming – File Outline

When creating a procedural python script each of your files will have the same basic format outlined below:

```python
#! /usr/bin/env python

########################################
# Comment explaining scripts purpose
# Author: <Author Name>
# Email: <Author's Email>
# Date: <Date Last Editor>
# Version: <Version Number>
########################################

# IMPORTS
# e.g., import os

# SCRIPT
print("Hello World")

# End of File
```

3.3.2   Object Orientated Programming – File Outline

When creating an object oriented script each python file you create will have the
same basic format outlined below:

```python
#! /usr/bin/env python

######################################
# Comment explaining scripts purpose
# Author: <Author Name>
# Emai: <Author's Email>
# Date: <Date Last Editor>
# Version: <Version Number>
######################################

# IMPORTS
import os

# CLASS EXPRESSION - In this case class name is Person
class Person (object): # Object is the superclass

    # CLASS ATTRIBUTES
    name = ''

    # INITIALISE THE CLASS (OFTEN EMPTY)
    def __init__(self):
        self.name = 'Dan'

    # METHOD TO PRINT PERSON NAME
    def printName(self):
        print('Name: ' + self.name)

    # METHOD TO SET PERSON NAME
    def setName(self, inputName):
        self.name = inputName

    # METHOD TO GET PERSON NAME
    def getName(self):
        return self.name

```

```
36      # METHOD TO EXECUTE CLASS
37      def run(self):
38          self.printName()
39          self.setName('Pete')
40          self.printName()
41
42  # IF EXECUTED FROM THE COMMAND LINE
43  if __name__ == '__main__':
44      obj = Person()
45      obj.run()
46
47  # End of File
```

3.4 Object Oriented Script

For simple scripts like those demonstrated so far simple procedural scripts are all that have been required. When creating more complex scripts the introduction of more structured and reusable designs are preferable. To support this design Python supports object oriented program design.

3.4.1 Object Oriented Script for Text File Processing

To illustrate the difference in implementation an example is given and explained below. The example reads a comma separated text file (randfloats.txt) of random floating point numbers from which the mean and standard deviation is calculated. Create a new python script and copy the script below:

```
1   #! /usr/bin/env python
2
3   #######################################
4   # An python class to parse a comma
5   # separates text file to calculate
6   # the mean and standard deviation
7   # of the inputted floating point
8   # numbers.
```

```python
9   # Author: <YOUR NAME>
10  # Email: <YOUR EMAIL>
11  # Date: DD/MM/YYYY
12  # Version: 1.0
13  ######################################
14
15  # import the squareroot function from python math
16  from math import sqrt
17
18  # Define a new class called CalcMeanStdDev
19  class CalcMeanStdDev (object):
20
21      # Define a function which parses a comma
22      # separated file - you should understand
23      # the contents of this script from the
24      # previous examples.
25      # Note: the file is passed into the
26      # function.
27      def parseCommaFile(self, file):
28          floatingNumbers = list()
29          for eachLine in file:
30              substrs = eachLine.split(',',eachLine.count(','))
31              for strVar in substrs:
32                  floatingNumbers.append(float(strVar))
33          return floatingNumbers
34
35      # Define a function to calculate the mean
36      # value from a list of numbers.
37      # Note. The list of numbers is passed into
38      # the function.
39      def calcMean(self, numbers):
40          # A variable to sum all the numbers
41          sum = 0.0
42          # Iterate through the numbers list
43          for number in numbers:
44              # add each number to the sum
45              sum += number
46          # Divide the sum by the number of
47          # values within the numbers list
48          # (i.e., its length)
49          mean = sum/len(numbers)
```

```python
50        # return the mean value calculated
51        return mean
52
53    # Define a function which calculates the
54    # standard deviation of a list of numbers
55    # Note. The list of numbers is passed into
56    # the function alongside a previously
57    # calculated mean value for the list.
58    def calcStdDev(self, numbers, mean):
59        # Varible for total deviation
60        deviation = 0.0
61        # Variable for a single deviation
62        singleDev = 0.0
63        # Iterate through the list of numbers.
64        for number in numbers:
65            # Calculate a single Deviation
66            singleDev = number-mean
67            # Add the squared single deviation to
68            # to the on going total.
69            deviation += (singleDev**2)
70        # Calcate the standard devaition
71        stddev = sqrt(deviation/(len(numbers)-1))
72        # return the standard deviation
73        return stddev
74
75    # The main thread of processing. A function
76    # which defines the order of processing.
77    # Note. The filename is passed in.
78    def run(self, filename):
79        # Open the input file
80        inFile = open(filename, 'r')
81        # Parse the file to retrieve a list of
82        # numbers
83        numbers = self.parseCommaFile(inFile)
84
85        # Calculate the mean value of the list
86        mean = self.calcMean(numbers)
87        # Calculate the standard deviation of the
88        # list.
89        stddev = self.calcStdDev(numbers, mean)
90
```

```
91          # Print the results to screen
92          print('Mean: ' + str(mean))
93          print('Stddev: ' + str(stddev))
94
95          # Close the input file
96          inFile.close()
97
98   # When python is executed python executes
99   # the code with the lowest indentation first.
100  #
101  # We can identify when python is executed from
102  # the command line using the following if statment.
103  #
104  # When executed we want the run() function to be
105  # executed therefore we create a CalcMeanStdDev
106  # object and call run on that object - passing
107  # in the file name of the file to be processed.
108  if __name__ == '__main__':
109      obj = CalcMeanStdDev()
110      obj.run('randfloats.txt') # Update with full file path.
```

NOTE:

`__name__`

and

`__main__`

each have TWO underscores either side (i.e., ⎯ ⎯).

Although, an object oriented design has been introduced making the above code, potentially, more reusable the design does not separate more general functionality from the application. To do this the code will be split into two files the first, named MyMaths.py, will contain the mathematical operations calcMean and calcStdDev while the second, named FileSummary, contains the functions run, which controls the flow of the script, and parseCommaFile(). The code for these files is given below but first try and split the code into the two files yourself.

```
1    #! /usr/bin/env python
2
```

```python
 3    ########################################
 4    # An python class to hold maths operations
 5    # Author: <YOUR NAME>
 6    # Email: <YOUR EMAIL>
 7    # Date: DD/MM/YYYY
 8    # Version: 1.0
 9    ########################################
10
11    from math import sqrt
12
13    class MyMathsClass (object):
14
15        def calcMean(self, numbers):
16            sum = 0.0
17            for number in numbers:
18                sum += number
19            mean = sum/len(numbers)
20            return mean
21
22        def calcStdDev(self, numbers, mean):
23            deviation = 0.0
24            singleDev = 0.0
25            for number in numbers:
26                singleDev = number-mean
27                deviation += (singleDev**2)
28            stddev = sqrt(deviation/(len(numbers)-1))
29            return stddev
30
31
```

```python
 1    #! /usr/bin/env python
 2
 3    ########################################
 4    # An python class to parse a comma
 5    # separates text file to calculate
 6    # the mean and standard deviation
 7    # of the inputted floating point
 8    # numbers.
 9    # Author: <YOUR NAME>
10    # Email: <YOUR EMAIL>
```

```python
11  # Date: DD/MM/YYYY
12  # Version: 1.0
13  #######################################
14
15  # To import the class you have created
16  # you need to define the file within
17  # which the class is held. In this
18  # case MyMaths.py and the name of the
19  # class to be imported (i.e., MyMathsClass)
20  from MyMaths import MyMathsClass
21
22  class FileSummary (object):
23
24      def parseCommaFile(self, file):
25          floatingNumbers = list()
26          for eachLine in file:
27              substrs = eachLine.split(',',eachLine.count(','))
28              for strVar in substrs:
29                  floatingNumbers.append(float(strVar))
30          return floatingNumbers
31
32      def run(self, filename):
33          inFile = open(filename, 'r')
34          numbers = self.parseCommaFile(inFile)
35
36          mathsObj = MyMathsClass()
37          mean = mathsObj.calcMean(numbers)
38          stddev = mathsObj.calcStdDev(numbers, mean)
39
40          print('Mean: ' + str(mean))
41          print('Stddev: ' + str(stddev))
42
43
44  if __name__ == '__main__':
45      obj = FileSummary()
46      obj.run('randfloats.txt')
47
```

To allow the script to be used as a command line tool the path to the file needs be passed into the script at runtime therefore the following changes are made to

the FileSummary script:

```python
1   #! /usr/bin/env python
2
3   #######################################
4   # An python class to parse a comma
5   # separates text file to calculate
6   # the mean and standard deviation
7   # of the inputted floating point
8   # numbers.
9   # Author: <YOUR NAME>
10  # Email: <YOUR EMAIL>
11  # Date: DD/MM/YYYY
12  # Version: 1.0
13  #######################################
14
15  from MyMaths import MyMathsClass
16  # To allow command line options to be
17  # retrieved the sys python library needs
18  # to be imported
19  import sys
20
21  class FileSummary (object):
22
23      def parseCommaFile(self, file):
24          floatingNumbers = list()
25          for eachLine in file:
26              substrs = eachLine.split(',',eachLine.count(','))
27              for strVar in substrs:
28                  floatingNumbers.append(float(strVar))
29          return floatingNumbers
30
31      def run(self):
32          # To retrieve the command line arguments
33          # the sys.argv[X] is used where X refers to
34          # the argument. The argument number starts
35          # at 1 and is the index of a list.
36          filename = sys.argv[1]
37          inFile = open(filename, 'r')
38          numbers = self.parseCommaFile(inFile)
39
```

```
40          mathsObj = MyMathsClass()
41          mean = mathsObj.calcMean(numbers)
42          stddev = mathsObj.calcStdDev(numbers, mean)
43
44          print('Mean: ' + str(mean))
45          print('Stddev: ' + str(stddev))
46
47  if __name__ == '__main__':
48      obj = FileSummary()
49      obj.run()
50
```

To read the new script the following command needs to be run from the command prompt:

```
python fileSummary_commandline.py randfloats.txt
```

3.5 Exercise

Calculate the mean and standard deviation from only the first column of data

Hint:

You will need to replace:

```
substrs = eachLine.split(',',eachLine.count(','))
for strVar in substrs:
    floatingNumbers.append(float(strVar))
```

With:

```
substrs = eachLine.split(',',eachLine.count(','))
# Select the column the data is stored in
column1 = substrs[0]
floatingNumbers.append(float(column1))
```

3.6   Further Reading

- An Introduction to Python, G. van Rossum, F.L. Drake, Jr. Network Theory ISBN 0-95-416176-9 (Also available online - `http://docs.python.org/3/tutorial/`) - Chapter 7.

- Python Documentation – `http://www.python.org/doc/`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

Chapter 4

File System – Finding files

4.1 Introduction

A common task for which python is used is to batch process a task or series of tasks. To do this the files to be processed need to be identified from within the file system. Therefore, in this tutorial you will learn to implement code to undertake this operation.

To start this type out the code below into a new file (save it as IterateFiles.py).

```python
#! /usr/bin/env python

######################################
# A class that iterates through a directory
# or directory structure and prints out theatre
# identified files.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
######################################

import os.path
import sys

```

```python
16  class IterateFiles (object):

17

18      # A function which iterates through the directory
19      def findFiles(self, directory):
20          # check whether the current directory exits
21          if os.path.exists(directory):
22              # check whether the given directory is a directory
23              if os.path.isdir(directory):
24                  # list all the files within the directory
25                  dirFileList = os.listdir(directory)
26                  # Loop through the individual files within the directory
27                  for filename in dirFileList:
28                      # Check whether file is directory or file
29                      if(os.path.isdir(os.path.join(directory,filename))):
30                          print(os.path.join(directory,filename) + \
31                          ' is a directory and therefore ignored!')
32                      elif(os.path.isfile(os.path.join(directory,filename))):
33                          print(os.path.join(directory,filename))
34                      else:
35                          print(filename + ' is NOT a file or directory!')
36              else:
37                  print(directory + ' is not a directory!')
38          else:
39              print(directory + ' does not exist!')

40

41

42

43      def run(self):
44          # Set the folder to search
45          searchFolder = './PythonCourse' # Update path...
46          self.findFiles(searchFolder)

47

48

49  if __name__ == '__main__':
50      obj = IterateFiles()
51      obj.run()
```

Using the online python documentation read through the section on the file system:

`http://docs.python.org/library/filesys.html`

`http://docs.python.org/library/os.path.html`

This documentation will allow you to understand the functionality which is available for manipulating the file system.

4.2 Recursion

The next stage is to add allow the function recursively go through the directory structure. To do this add the function below to your script above:

```python
#! /usr/bin/env python

#######################################
# A class that iterates through a directory
# or directory structure and prints out theatre
# identified files.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

import os.path
import sys

class IterateFiles (object):

    # A function which iterates through the directory
    def findFilesRecurse(self, directory):
        # check whether the current directory exits
        if os.path.exists(directory):
            # check whether the given directory is a directory
            if os.path.isdir(directory):
                # list all the files within the directory
                dirFileList = os.listdir(directory)
                # Loop through the individual files within the directory
                for filename in dirFileList:
                    # Check whether file is directory or file
```

```python
29                    if(os.path.isdir(os.path.join(directory,filename))):
30                        # If a directory is found recall this function.
31                        self.findFilesRecurse(os.path.join(directory,filename))
32                    elif(os.path.isfile(os.path.join(directory,filename))):
33                        print(os.path.join(directory,filename))
34                    else:
35                        print(filename + ' is NOT a file or directory!')
36                else:
37                    print(directory + ' is not a directory!')
38            else:
39                print(directory + ' does not exist!')
40
41        def run(self):
42            # Set the folder to search
43            searchFolder = './PythonCourse' # Update path...
44            self.findFilesRecurse(searchFolder)
45
46    if __name__ == '__main__':
47        obj = IterateFiles()
48        obj.run()
```

Now call this function instead of the findFiles. Think and observe what effect a function which calls itself will have on the order in which the file are found.

4.3 Checking file Extension

The next step is to include the function checkFileExtension to your class and create two new functions which only print out the files with the file extension of interest. This should be done for both the recursive and non-recursive functions above.

```python
1   #! /usr/bin/env python
2
3   ######################################
4   # A class that iterates through a directory
5   # or directory structure and prints out theatre
6   # identified files.
```

```python
7    # Author: <YOUR NAME>
8    # Email: <YOUR EMAIL>
9    # Date: DD/MM/YYYY
10   # Version: 1.0
11   ######################################
12
13   import os.path
14   import sys
15
16   class IterateFiles (object):
17
18       # A function which checks a file extension and returns
19       def checkFileExtension(self, filename, extension):
20           # Boolean variable to be returned by the function
21           foundExtension = False;
22           # Split the filename into two parts (name + ext)
23           filenamesplit = os.path.splitext(filename)
24           # Get the file extension into a varaiable
25           fileExtension = filenamesplit[1].strip()
26           # Decide whether extensions are equal
27           if(fileExtension == extension):
28               foundExtension = True
29           # Return result
30           return foundExtension
31
32       # A function which iterates through the directory and checks file extensions
33       def findFilesExtRecurse(self, directory, extension):
34           # check whether the current directory exits
35           if os.path.exists(directory):
36               # check whether the given directory is a directory
37               if os.path.isdir(directory):
38                   # list all the files within the directory
39                   dirFileList = os.listdir(directory)
40                   # Loop through the individual files within the directory
41                   for filename in dirFileList:
42                       # Check whether file is directory or file
43                       if(os.path.isdir(os.path.join(directory,filename))):
44                           # If a directory is found recall this function.
45                           self.findFilesRecurse(os.path.join(directory,filename))
46                       elif(os.path.isfile(os.path.join(directory,filename))):
47                           if(self.checkFileExtension(filename, extension)):
```

```python
48                        print(os.path.join(directory,filename))
49                    else:
50                        print(filename + ' is NOT a file or directory!')
51                else:
52                    print(directory + ' is not a directory!')
53            else:
54                print(directory + ' does not exist!')
55


57        # A function which iterates through the directory and checks file extensions
58        def findFilesExt(self, directory, extension):
59            # check whether the current directory exits
60            if os.path.exists(directory):
61                # check whether the given directory is a directory
62                if os.path.isdir(directory):
63                    # list all the files within the directory
64                    dirFileList = os.listdir(directory)
65                    # Loop through the individual files within the directory
66                    for filename in dirFileList:
67                        # Check whether file is directory or file
68                        if(os.path.isdir(os.path.join(directory,filename))):
69                            print(os.path.join(directory,filename) + \
70                            ' is a directory and therefore ignored!')
71                        elif(os.path.isfile(os.path.join(directory,filename))):
72                            if(self.checkFileExtension(filename, extension)):
73                                print(os.path.join(directory,filename))
74                        else:
75                            print(filename + ' is NOT a file or directory!')
76                else:
77                    print(directory + ' is not a directory!')
78            else:
79                print(directory + ' does not exist!')
80
81        def run(self):
82            # Set the folder to search
83            searchFolder = './PythonCourse' # Update path...
84            self.findFilesExt(searchFolder, '.txt')
85
86    if __name__ == '__main__':
87        obj = IterateFiles()
88        obj.run()
```

4.4 Exercises

1. Rather than print the file paths to screen add them to a list and return them from the function. This would be useful for applications where the files to be process need to be known up front and creates a more generic piece of python which can be called from other scripts.

2. Using the return list add code to loop through the returned list and print out the file information in the following comma separated format.

```
[FILE NAME], [EXTENSION], [PATH], [DRIVE LETTER (On Windows)], [MODIFICATION TIME]
```

4.5 Further Reading

- Python Documentation – `http://www.python.org/doc/`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

Chapter 5

Plotting - Matplotlib

5.1 Introduction

Many open source libraries are available from within python. These significantly
increase the available functionality, decreasing your development time. One such
library is matplotlib (`http://matplotlib.sourceforge.net`), which provides a
plotting library with a similar interface to those available within Matlab. The mat-
plotlib website provides a detailed tutorial and documentation for all the different
options available within the library but this worksheet provides some examples of
the common plot types and a more complex example continuing on from previous
examples.

5.2 Simple Script

Below is your first script using the matplotlib library. The script demonstrates
the plotting of a mathematical function, in this case a sine function. The plot
function requires two lists of numbers to be provided, which provides the x and
y locations of the points which go to create the displayed function. The axis can
be labelled using the xlabel() and ylabel() functions while the title is set using
the title() function. Finally, the show() function is used to reveal the interface

displaying the plot.

```python
#! /usr/bin/env python


#########################################
# A simple python script to display a
# sine function
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#########################################


# import the matplotlib libraries
from matplotlib import pylab as plt


# Create a list with values from
# 0 to 3 with 0.01 intervals
t = arange(0.0, 3, 0.01)
# Calculate the sin curve for
# the values within t
s = sin(pi*t)


# Plot the values in s and t
plt.plot(t, s)
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Simple Plot')
# save plot to disk.
plt.savefig('simpleplot.pdf', dpi=200, format='PDF')
```

5.3 Bar Chart

The creation of a bar chart is equally simply where two lists are provided, the first contains the locations on the X axis at which the bars start and the second the heights of the bars. The width of the bars can also be specified and their colour. More options are available in the documentation (`http://matplotlib.`
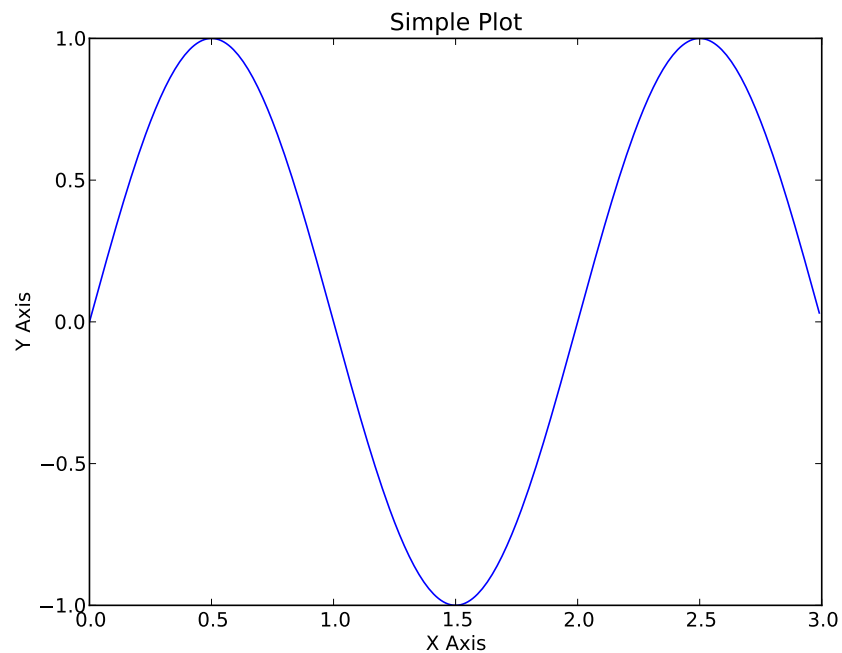
Figure 5.1: A simple plot using matplotlib.

sourceforge.net/matplotlib.pylab.html#-bar)

```python
#! /usr/bin/env python

#######################################
# A simple python script to display a
# bar chart.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

from matplotlib import pylab as plt

# Values for the Y axis (i.e., height of bars)
height = [5, 6, 7, 8, 12, 13, 9, 5, 7, 4, 3, 1]
# Values for the x axis
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
18
19  # create plot with colour grey
20  plt.bar(x, height, width=1, color='gray')
21  # save plot to disk.
22  plt.savefig('simplebar.pdf', dpi=200, format='PDF')
```
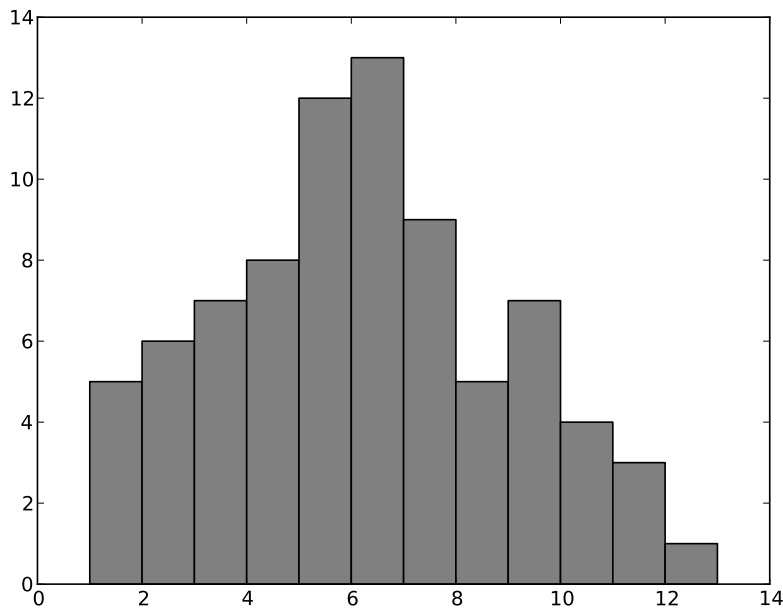

Figure 5.2: A simple bar chart using matplotlib.

5.4 Pie Chart

A pie chart is similar to the previous scripts where a list of the fractions making up the pie chart is given alongside a list of labels and if required a list of fractions to explode the pie chart. Other options including colour and shadow are available and outlined in the documentation (`http://matplotlib.sourceforge.net/matplotlib.pylab.html#-pie`) This script also demonstrates the use of the savefig() function allowing the plot to be saved to file rather than simply displayed on screen.

```python
1   #! /usr/bin/env python
2
3   ######################################
4   # A simple python script to display a
5   # pie chart.
6   # Author: <YOUR NAME>
7   # Email: <YOUR EMAIL>
8   # Date: DD/MM/YYYY
9   # Version: 1.0
10  ######################################
11
12  from matplotlib import pylab as plt
13
14  frac = [25, 33, 17, 10, 15]
15  labels = ['25', '33', '17', '10', '15']
16  explode = [0, 0.25, 0, 0, 0]
17
18  # Create pie chart
19  plt.pie(frac, explode, labels, shadow=True)
20  # Give it a title
21  plt.title('A Sample Pie Chart')
22  # save the plot to a PDF file
23  plt.savefig('pichart.pdf', dpi=200, format='PDF')
```

5.5   Scatter Plot

The following script demonstrates the production of a scatter plot (`http://matplotlib.sourceforge.net/matplotlib.pylab.html#-scatter`) where the lists x and y provide the locations of the points in the X and Y axis and Z provides the third dimension used to colour the points.

```python
1   #! /usr/bin/env python
2
3   ######################################
4   # A simple python script to display a
5   # scatter plot.
6   # Author: <YOUR NAME>
```
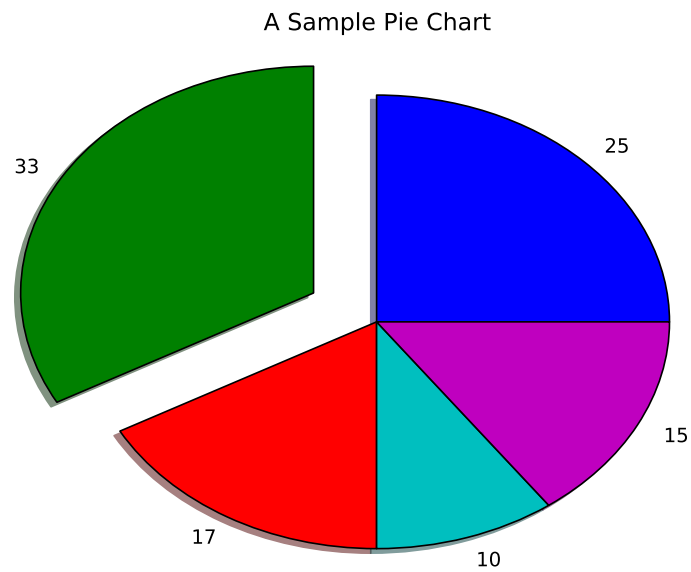
A Sample Pie Chart

Figure 5.3: A simple pie chart using matplotlib.

```
7   # Email: <YOUR EMAIL>
8   # Date: DD/MM/YYYY
9   # Version: 1.0
10  #####################################
11
12  from matplotlib import pylab as plt
13  # Import a random number generator
14  from random import random
15
16  x = []
17  y = []
18  z = []
19
20  # Create data values for X, Y, Z axis'
21  for i in range(5000):
22      x.append(random() * 100)
23      y.append(random() * 100)
24      z.append(x[i]-y[i])
25
```

```
26   # Create figure
27   plt.figure()
28   # Create scatter plot where the plots are coloured using the
29   # Z values.
30   plt.scatter(x, y, c=z, marker='o', cmap=cm.jet, vmin=-100, vmax=100)
31   # Display colour bar
32   colorbar()
33   # Make axis' tight to the data
34   plt.axis('tight')
35   plt.xlabel('X Axis')
36   plt.ylabel('Y Axis')
37   plt.title('Simple Scatter Plot')
38   # save plot to disk.
39   plt.savefig('simplescatter.pdf', dpi=200, format='PDF')
```
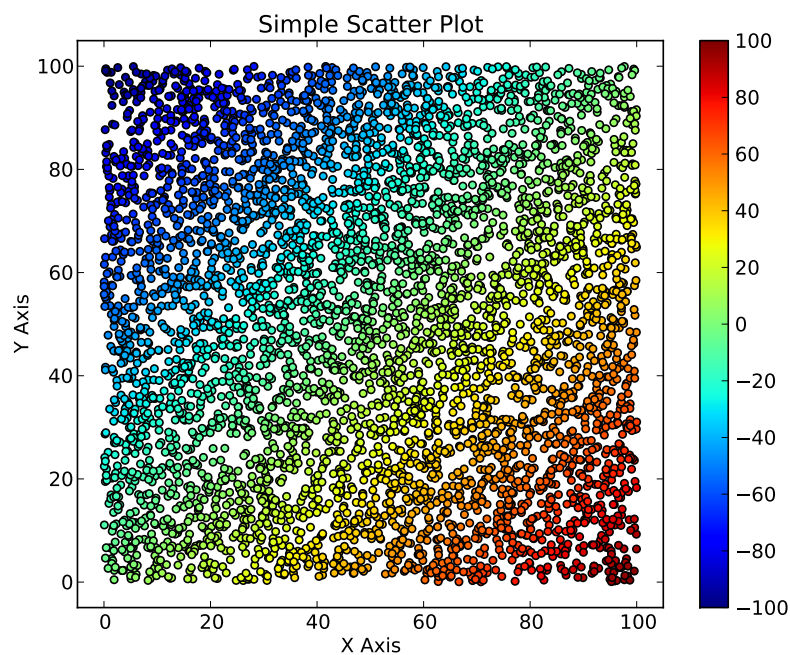

Figure 5.4: A simple scatter plot using matplotlib.

5.6   Line Plot

A more complicated example is now given building on the previous tutorial where the data is read in from a text file before being plotted. In this case data was downloaded from the Environment Agency and converted from columns to rows. The dataset provides the five year average rainfall for the summer (June - August) and winter (December - February) from 1766 to 2006. Two examples of plotting this data are given where the first plots the two datasets onto the same axis (Figure 5.5) while the second plots them onto individual axis (Figure 5.6). Information on the use of the subplot() function can be found in the matplotlib documentation (`http://matplotlib.sourceforge.net/matplotlib.pylab.html#-subplot`).

```python
1   #######################################
2   # A python script to read in a text file
3   # of rainfall data for summer and winter
4   # within the UK and display as a plot.
5   # Author: <YOUR NAME>
6   # Email: <YOUR EMAIL>
7   # Date: DD/MM/YYYY
8   # Version: 1.0
9   #######################################
10
11  from matplotlib import pylab as plt
12  import os.path
13  import sys
14
15  class PlotRainfall (object):
16
17      # Parse the input file - Three columns year, summer, winter
18      def parseDataFile(self, dataFile, year, summer, winter):
19          line = 0
20          for eachLine in dataFile:
21              commaSplit = eachLine.split(',', eachLine.count(','))
22              first = True
23              for token in commaSplit:
24                  if first:
25                      first = False
26                  else:
```

```python
27                     if line == 0:
28                         year.append(int(token))
29                     elif line == 1:
30                         summer.append(float(token))
31                     elif line == 2:
32                         winter.append(float(token))
33                 line += 1
34
35     # Plot data onto the same axis'
36     def plotData(self, year, summer, winter, outFile):
37         plt.figure()
38         plt.plot(year, summer)
39         plt.plot(year, winter)
40         plt.legend(['Summer', 'Winter'])
41         plt.xlabel('Year')
42         plt.ylabel('Rainfall (5 Year Mean)')
43         plt.title('Summer and Winter rainfall across the UK')
44         # save plot to disk.
45         plt.savefig(outFile, dpi=200, format='PDF')
46
47     # Plot the data onto separate axis, using subplot.
48     def plotDataSeparate(self, year, summer, winter, outFile):
49         plt.figure()
50         plt.subplot(2,1,1)
51         plt.plot(year, summer)
52         plt.ylabel('Rainfall (5 Year Mean)')
53         plt.title('Summer rainfall across the UK')
54         plt.axis('tight')
55
56         plt.subplot(2,1,2)
57         plt.plot(year, winter)
58         plt.xlabel('Year')
59         plt.ylabel('Rainfall (5 Year Mean)')
60         plt.title('Winter rainfall across the UK')
61         plt.axis('tight')
62         # save plot to disk.
63         plt.savefig(outFile, dpi=200, format='PDF')
64
65     def run(self):
66         filename = 'ukweatheraverage.csv'
67         if os.path.exists(filename):
```

```
68            year = list()
69            summer = list()
70            winter = list()
71            try:
72                dataFile = open(filename, 'r')
73            except IOError as e:
74                print('\nCould not open file:\n', e)
75                return
76            self.parseDataFile(dataFile, year, summer, winter)
77            dataFile.close()
78            self.plotData(year, summer, winter, "Rainfall_SinglePlot.pdf")
79            self.plotDataSeparate(year, summer, winter, "Rainfall_MultiplePlots.pdf")
80        else:
81            print('File \'' + filename + '\' does not exist.')
82
83 if __name__ == '__main__':
84     obj = PlotRainfall()
85     obj.run()
```
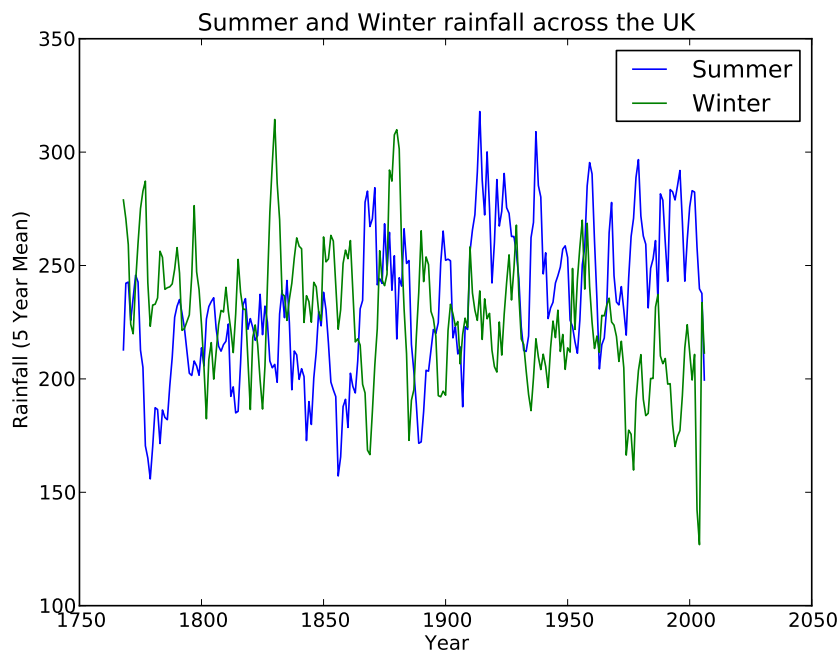

Figure 5.5: Rainfall data for summer and winter on the same axis'.
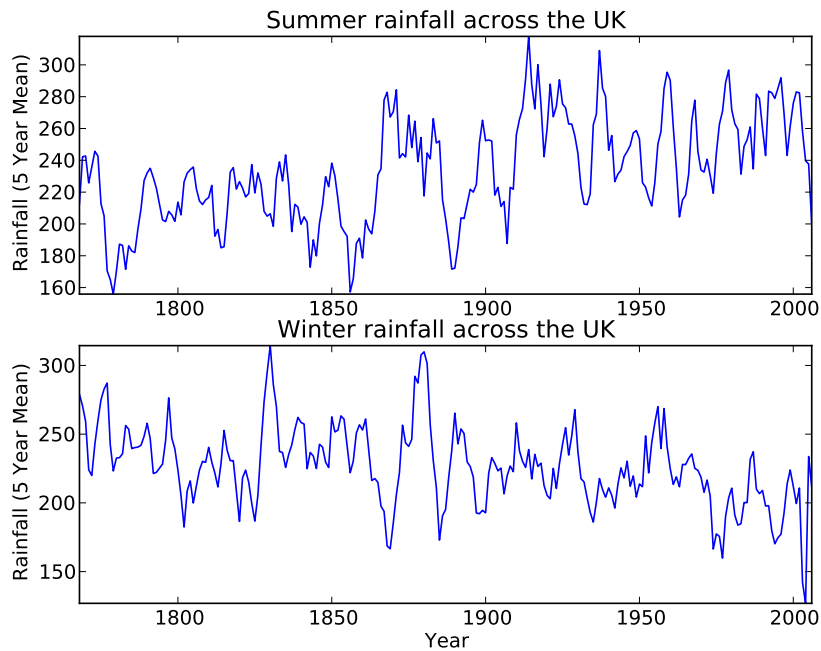
Figure 5.6: Rainfall data for summer and winter on different axis'.

5.7 Exercise:

Based on the available data is there a correlation between summer and winter rainfall? Use the lists read in of summer and winter rainfall and produce a scatterplot to answer this question.

5.8 Further Reading

- Matplotlib – `http://matplotlib.sourceforge.net`

- Python Documentation – `http://www.python.org/doc/`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

Chapter 6

Statistics (SciPy / NumPy)

6.1 Introduction

NumPy is a library for storing and manipulating multi-dimensional arrays. NumPy arrays are similar to lists, however they have a lot more functionality and allow faster operations. SciPy is a library for maths and science using NumPy arrays and includes routines for statistics, optimisation and numerical integration. A comprehensive list is available from the SciPy website (`http://docs.scipy.org/doc/scipy/reference`). The combination of NumPy, SciPy and MatPlotLib provides similar functionality to that available in packages such as MatLab and Mathematica and allows for complex numerical analysis.

This tutorial will introduce some of the statistical functionality of NumPy / SciPy by calculating statistics from forest inventory data, read in from a text file. Linear regression will also be used to calculate derive relationships between parameters.

There are a number of ways to create NumPy arrays, one of the easiest (and the method that will be used in this tutorial) is to convert a python list to an array:

```
import numpy
pythonList = [1 , 4 , 2 , 5, 3]
numpyArray = numpy.array(pythonList)
```

6.2 Simple Statistics

Forest inventory data have been collected for a number of plots within Penglais woods (Aberystwyth, Wales). For each tree, the diameter, species height, crown size and position have been recorded. An example script is provided to read the diameters into a separate list for each species. The lists are then converted to NumPy arrays, from which statistics are calculated and written out to a text file.

```python
#! /usr/bin/env python

#######################################
# A script to calculate statistics from
# a text file using NumPy
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

import numpy
import scipy
# Import scipy stats functions we need
import scipy.stats as spstats

class CalculateStatistics (object):

    def run(self):
        # Set up lists to hold input diameters
        # A seperate list is used for each species
        beechDiameter = list()
        ashDiameter = list()
        birchDiameter = list()
        oakDiameter = list()
        sycamoreDiameter = list()
        otherDiameter = list()

        # Open input and output files
        inFileName = 'PenglaisWoodsData.csv'
```

```
31            outFileName = 'PenglaisWoodsStats.csv'
32            inFile = open(inFileName, 'r')
33            outFile = open(outFileName,'w')
34
35            # Iterate through the input file and save diameter into
36            # lists, based on species
37            header = True
38            for eachLine in inFile:
39                if header: # Skip header row
40                    print('Skipping header row')
41                    header = False
42                else:
43                    substrs = eachLine.split(',',eachLine.count(','))
44
45                    species = substrs[3]
46                    if substrs[4].isdigit: # Check diameter is a number
47                        diameter = float(substrs[4])
48
49                        if species == 'BEECH':
50                            beechDiameter.append(diameter)
51                        elif species == 'ASH':
52                            ashDiameter.append(diameter)
53                        elif species == 'BIRCH':
54                            birchDiameter.append(diameter)
55                        elif species == 'OAK':
56                            oakDiameter.append(diameter)
57                        elif species == 'SYC':
58                            sycamoreDiameter.append(diameter)
59                        else:
60                            otherDiameter.append(diameter)
61
62            # Convert input lists to NumPy arrays
63            beechDiameter = numpy.array(beechDiameter)
64            ashDiameter = numpy.array(ashDiameter)
65            birchDiameter = numpy.array(birchDiameter)
66            oakDiameter = numpy.array(oakDiameter)
67            sycamoreDiameter = numpy.array(sycamoreDiameter)
68            otherDiameter = numpy.array(otherDiameter)
69
70            # Write header line to output file
71            headerLine = 'species, meanDiameter, medianDiameter, stDevDiameter\n'
```

```
72          outFile.write(headerLine)
73
74          # Calculate statistics for each species and write to file
75          outLine = 'Beech,' + self.createStatsLine(beechDiameter) + '\n'
76          outFile.write(outLine)
77          outLine = 'Ash,' + self.createStatsLine(ashDiameter) + '\n'
78          outFile.write(outLine)
79          outLine = 'Birch,' + self.createStatsLine(birchDiameter) + '\n'
80          outFile.write(outLine)
81          outLine = 'Oak,' + self.createStatsLine(oakDiameter) + '\n'
82          outFile.write(outLine)
83          outLine = 'Sycamore,' + self.createStatsLine(sycamoreDiameter) + '\n'
84          outFile.write(outLine)
85          outLine = 'Other,' + self.createStatsLine(otherDiameter) + '\n'
86          outFile.write(outLine)
87
88          print('Statistics written to: ' + outFileName)
89
90
91      def createStatsLine(self, inArray):
92          # Calculate statsistics for NumPy array and return output line.
93          meanArray = numpy.mean(inArray)
94          medianArray = numpy.median(inArray)
95          stDevArray = numpy.std(inArray)
96          skewArray = spstats.skew(inArray)
97
98          # Create output line with stats
99          statsLine = str(meanArray) + ',' + str(medianArray) + ',' + str(stDevArray)
100         return statsLine
101
102 if __name__ == '__main__':
103     obj = CalculateStatistics()
104     obj.run()
```

Note in tutorial three, functions were written to calculate the mean and standard deviation a list, in this tutorial the same result is accomplished using the built in functionality of NumPy.

6.2.1 Exercises

1. Based on the example script also calculate mean, median and standard deviation for tree heights and add to the output file.

2. Look at other statistics functions available in SciPy and calculate for height and density.

6.3 Calculate Biomass

One of the features of NumPy arrays is the ability to perform mathematical operation on all elements of an array.

For example, for NumPy array a:

```
a = numpy.array([1,2,3,4])
```

Performing

```
b = 2 * a
```

Gives

```
b = array([2,4,6,8])
```

Some special versions of functions are available to work on arrays. To calculate the natural log of a single number log may be used, to perform the natural log of an array np.log may be used (where NumPy has been imported as np).

Tree volume may be calculated from height and stem diameter using:

$$\text{Volume} = a + bD^2h^{0.75} \tag{6.1}$$

Where D is diameter and h is height. The coefficients a and b vary according to species (see Table 6.1). From volume, it is possible to calculate biomass by multiplying by the specific gravity.

$$\text{Biomass} = \text{Volume} \times \text{SpecificGravity} \tag{6.2}$$

The specific gravity also varies by species, values for each species are given in Table 6.1.

Table 6.1: Coefficients for estimating volume and the specific gravity required for estimating the biomass by species.

Species	a-coefficient	b-coefficient	Specific gravity
Beech	0.014306	0.0000748	0.56
Ash	0.012107	0.0000777	0.54
Beech	0.009184	0.0000673	0.53
Oak	0.011724	0.0000765	0.56
Sycamore	0.012668	0.0000737	0.54

The following function takes two arrays containing height and density, and a string for species. From these biomass is calculated.

```python
def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
        if inSpecies == 'BEECH':
            a = 0.014306
            b = 0.0000748
            specificGravity = 0.56

        # Calculate Volume
        volume = a + ((b*(inDiameterArray / 100)**2) * (inHeightArray**0.75))
        # Calculate biomass
        biomass = volume * specificGravity
        # Return biomass
        return biomass
```

Note only the coefficients for 'BEECH' have been included therefore, if a different species is passed in, the program will produce an error (try to think about what the error would be). A neater way of dealing with the error would be to throw an exception if the species was not recognised. Exceptions form the basis of controlling errors in a number of programming languages (including C++ and Java) the simple concept is that as a program is running, if an error occurs an exception is thrown, at which point processing stops until the exception is caught and dealt with. If the

exception is never caught, then the software crashes and stops. Python provides the following syntax for exception programming,

```
try:
    < Perform operations during which
     an error is likely to occur >
except <ExceptionName>:
   < If error occurs do something
     appropriate >
```

where the code you wish to run is written inside the 'try' statement and the 'except' statement is executed only when a named exception (within the except statement) is produced within the 'try' block. It is good practise you use exceptions where possible as when used properly they provide more robust code which can provide more feedback to the user.

The function to calculate biomass may be rewritten to throw an exception if the species is not recognised.

```python
1  def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
2          if inSpecies == 'BEECH':
3              a = 0.014306
4              b = 0.0000748
5              specificGravity = 0.56
6          else: # Raise exception if species is not recognised
7              raise Exception('Species not recognised')
8
9          # Calculate Volume
10         volume = a + ((b*(inDiameterArray / 100)**2) * (inHeightArray**0.75))
11         # Calculate biomass
12         biomass = volume * specificGravity
13         # Return biomass
14         return biomass
```

The function below, calls 'calcBiomass' to calculate biomass for an array. From this mean, median and standard deviation are calculated and an output array is returned. By calling the function from within a 'try and except' block if the species is not recognised, it will not try to calculate stats and will return the string 'na' (not available) for all values in the output line.

```
1  def calcBiomassStatsLine(self, inDiameterArray, inHeightArray, inSpecies):
2          # Calculates biomass, calculates stats from biomass and returns output line
3          biomassStatsLine = ''
4          try:
5              # Calculate biomass
6              biomass = self.calcBiomass(inDiameterArray, inHeightArray, inSpecies)
7              # Calculate stats from biomass
8              meanBiomass = numpy.mean(biomass)
9              medianBiomass = numpy.median(biomass)
10             stDevBiomass = numpy.std(biomass)
11
12             # Create output line
13             biomassStatsLine = str(meanBiomass) + ',' + str(medianBiomass) + ',' + \
14  str(stDevBiomass)
15
16         except Exception:
17             # Catch exception and write 'na' for all values
18             biomassStatsLine = 'na,na,na'
19
20         return biomassStatsLine
```

Therefore, the final script should result in the following:

```
1  #! /usr/bin/env python
2
3  ######################################
4  # A script to calculate statistics from
5  # a text file using NumPy
6  # Author: <YOUR NAME>
7  # Email: <YOUR EMAIL>
8  # Date: DD/MM/YYYY
9  # Version: 1.0
10 ######################################
11
12 import numpy
13 import scipy
14 # Import scipy stats functions we need
15 import scipy.stats as spstats
16
17 class CalculateStatistics (object):
18
```

```python
19        def run(self):
20            # Set up lists to hold input diameters and heights
21            # A seperate list is used for each species
22            beechDiameter = list()
23            beechHeight = list()
24            ashDiameter = list()
25            ashHeight = list()
26            birchDiameter = list()
27            birchHeight = list()
28            oakDiameter = list()
29            oakHeight = list()
30            sycamoreDiameter = list()
31            sycamoreHeight = list()
32            otherDiameter = list()
33            otherHeight = list()
34
35            # Open input and output files
36            inFileName = 'PenglaisWoodsData.csv'
37            outFileName = 'PenglaisWoodsStats.csv'
38            inFile = open(inFileName, 'r')
39            outFile = open(outFileName,'w')
40
41            # Iterate through the input file and save diameter and height
42            # into lists, based on species
43            header = True
44            for eachLine in inFile:
45                if header: # Skip header row
46                    print('Skipping header row')
47                    header = False
48                else:
49                    substrs = eachLine.split(',',eachLine.count(','))
50
51                    species = substrs[3]
52                    if substrs[4].isdigit: # Check diameter is a number
53                        diameter = float(substrs[4])
54                        height = float(substrs[10])
55
56                        if species == 'BEECH':
57                            beechDiameter.append(diameter)
58                            beechHeight.append(height)
59                        elif species == 'ASH':
```

```
60                         ashDiameter.append(diameter)
61                         ashHeight.append(height)
62                     elif species == 'BIRCH':
63                         birchDiameter.append(diameter)
64                         birchHeight.append(height)
65                     elif species == 'OAK':
66                         oakDiameter.append(diameter)
67                         oakHeight.append(height)
68                     elif species == 'SYC':
69                         sycamoreDiameter.append(diameter)
70                         sycamoreHeight.append(height)
71                     else:
72                         otherDiameter.append(diameter)
73                         otherHeight.append(height)
74
75         # Convert to NumPy arrays
76         beechDiameter = numpy.array(beechDiameter)
77         ashDiameter = numpy.array(ashDiameter)
78         birchDiameter = numpy.array(birchDiameter)
79         oakDiameter = numpy.array(oakDiameter)
80         sycamoreDiameter = numpy.array(sycamoreDiameter)
81         otherDiameter = numpy.array(otherDiameter)
82
83         beechHeight = numpy.array(beechHeight)
84         ashHeight = numpy.array(ashHeight)
85         birchHeight = numpy.array(birchHeight)
86         oakHeight = numpy.array(oakHeight)
87         sycamoreHeight = numpy.array(sycamoreHeight)
88         otherHeight = numpy.array(otherHeight)
89
90         # Write header line to output file
91         headerLine = 'species,meanDiameter,medianDiameter,stDevDiameter,\
92 meanHeight,medianHeight,stDevHeight,\
93 meanBiomass,medianBiomass,stDevBiomass\n'
94         outFile.write(headerLine)
95
96         # Calculate statistics and biomass for each species and write to file
97         outLine = 'Beech,' + self.createStatsLine(beechDiameter) + ',' + \
98 self.createStatsLine(beechHeight) + ',' + \
99 self.calcBiomassStatsLine(beechDiameter, beechHeight, 'BEECH') + '\n'
100         outFile.write(outLine)
```

```
101         outLine = 'Ash,' + self.createStatsLine(ashDiameter)  + ',' +  \
102  self.createStatsLine(ashHeight) + ',' + \
103  self.calcBiomassStatsLine(ashDiameter, ashHeight, 'ASH') + '\n'
104         outFile.write(outLine)
105         outLine = 'Birch,' + self.createStatsLine(birchDiameter) + ',' + \
106  self.createStatsLine(birchHeight) + ',' +  \
107  self.calcBiomassStatsLine(birchDiameter, birchHeight, 'BIRCH') + '\n'
108         outFile.write(outLine)
109         outLine = 'Oak,' + self.createStatsLine(oakDiameter)  + ',' + \
110  self.createStatsLine(oakHeight) + ',' +  \
111  self.calcBiomassStatsLine(oakDiameter, oakHeight, 'OAK') + '\n'
112         outFile.write(outLine)
113         outLine = 'Sycamore,' + self.createStatsLine(sycamoreDiameter)  + ',' + \
114  self.createStatsLine(sycamoreHeight) + ',' +  \
115  self.calcBiomassStatsLine(sycamoreDiameter, sycamoreHeight, 'SYC') + '\n'
116         outFile.write(outLine)
117         outLine = 'Other,' + self.createStatsLine(otherDiameter)  + ',' + \
118  self.createStatsLine(otherHeight) + ',' +  \
119  self.calcBiomassStatsLine(otherDiameter, otherHeight, 'Other') + '\n'
120         outFile.write(outLine)
121
122         print('Statistics written to: ' + outFileName)
123
124     def createStatsLine(self, inArray):
125         # Calculate statsistics for array and return output line.
126         meanArray = numpy.mean(inArray)
127         medianArray = numpy.median(inArray)
128         stDevArray = numpy.std(inArray)
129
130         # Create output line with stats
131         statsLine = str(meanArray) + ',' + str(medianArray) + ',' + str(stDevArray)
132         return statsLine
133
134     def calcBiomassStatsLine(self, inDiameterArray, inHeightArray, inSpecies):
135         # Calculates biomass, calculates stats from biomass and returns output line
136         biomassStatsLine = ''
137         try:
138             # Calculate biomass
139             biomass = self.calcBiomass(inDiameterArray, inHeightArray, inSpecies)
140             # Calculate stats from biomass
141             meanBiomass = numpy.mean(biomass)
```

```
142            medianBiomass = numpy.median(biomass)
143            stDevBiomass = numpy.std(biomass)
144
145            # Create output line
146            biomassStatsLine = str(meanBiomass) + ',' + str(medianBiomass) + ',' + \
147    str(stDevBiomass)
148
149        except Exception:
150            # Catch exception and write 'na' for all values
151            biomassStatsLine = 'na,na,na'
152
153        return biomassStatsLine
154
155    def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
156        if inSpecies == 'BEECH':
157            a = 0.014306
158            b = 0.0000748
159            specificGravity = 0.56
160        else: # Raise exception is species is not recognised
161            raise Exception('Species not recognised')
162
163        # Calcualte volume
164        volume = a + ((b*(inDiameterArray)**2) * (inHeightArray**0.75))
165        # Calculate biomass
166        biomass = volume * specificGravity
167        # Return biomass
168        return biomass
169
170 if __name__ == '__main__':
171    obj = CalculateStatistics()
172    obj.run()
```

6.3.1 Exercise

1. Add in the coefficients to calculate biomass for the other species

2. Write the statistics for biomass out to the text file. Remember to change the header line.

6.4 Linear Fitting

One of the built in feature of SciPy is the ability to perform fits. Using the linear regression function (linregress) it is possible to fit equations of the form:

$$y = ax + b \qquad (6.3)$$

to two NumPy arrays (x and y) using:

```
(aCoeff,bCoeff,rVal,pVal,stdError) = linregress(x, y)
```

Where aCoeff and bCoeff are the coefficients rVal is the r value (r**2 gives R^2), pVal is the p value and stdError is the standard error.

It is possible to fit the following equation to the collected data expressing height as a function of diameter.

$$\text{height} = a \log\left(\text{diameter}\right) + b \qquad (6.4)$$

To fit an equation of this form an array must be created containing log diameter . Linear regression may then be performed using:

```
linregress(np.log(inDiameterArray), inHeightArray)
```

To test the fit it may be plotted against the original data using MatPlotLib. The following code first performs the linear regression then creates a plot showing the fit against the original data.

```python
def plotLinearRegression(self, inDiameterArray, inHeightArray, outPlotName):
        # Perform fit
        (aCoeff,bCoeff,rVal,pVal,stdError) = linregress(np.log(inDiameterArray), \
                                             inHeightArray)

        # Use fits to predict height for a range of diameters
        testDiameter = arange(min(inDiameterArray), max(inDiameterArray), 1)
        predictHeight = (aCoeff * np.log(testDiameter)) + bCoeff

        # Create a string, showing the form of the equation (with fitted coefficients)
```

```
11          # and r squared value
12          # Coefficients are rounded to two decimal places.
13          equation = str(round(aCoeff,2)) + 'log(D) + ' + str(round(bCoeff,2)) + \
14                  ' (r$^2$ = ' + str(round(rVal**2,2)) + ')'
15
16          # Plot fit against origional data
17          plt.plot(inDiameterArray, inHeightArray,'.')
18          plt.plot(testDiameter, predictHeight)
19          plt.xlabel('Diameter (cm)')
20          plt.ylabel('Height (m)')
21          plt.legend(['measured data',equation])
22
23          # Save plot
24          plt.savefig(outPlotName, dpi=200, format='PDF')
```

The coefficients and r^2 of the fit are displayed in the legend. To display the superscript '2' in the data it is possible to use LaTeX syntax. So r^2 is written as: r$\wedge$2$.

The function may be called using:

```
# Set output directory for plots
outDIR = './output/directory/'


self.plotLinearRegression(beechDiameter, beechHeight, outDIR + 'beech.pdf')
```

Produce a plot similar to the one shown in Figure 6.1 and save as a PDF.

The final script should result in the following:

```
1   #! /usr/bin/env python
2
3   ####################################
4   # A script to calculate statistics from
5   # a text file using NumPy
6   # Author: <YOUR NAME>
7   # Email: <YOUR EMAIL>
8   # Date: DD/MM/YYYY
9   # Version: 1.0
10  ####################################
11
```
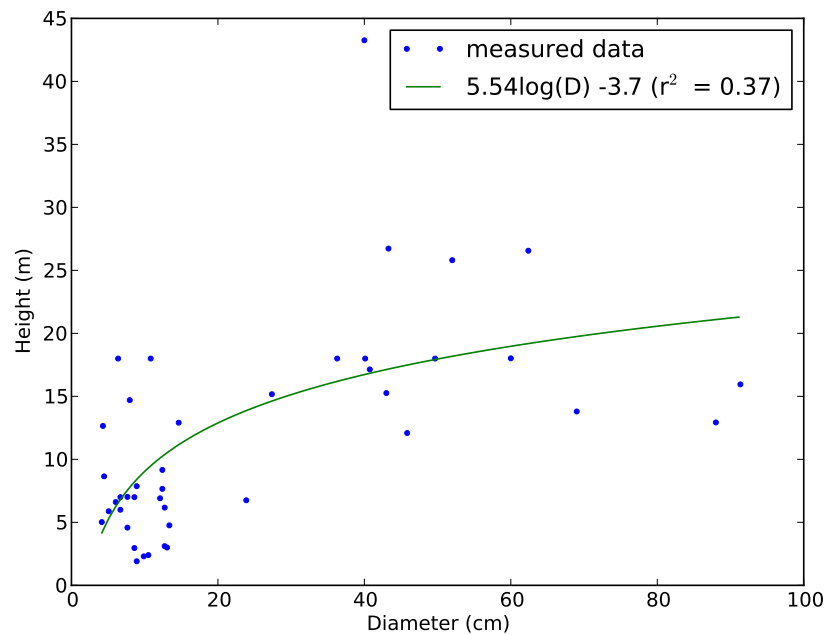
Figure 6.1: A simple plot using matplotlib.

```
12  import numpy
13  import scipy
14  # Import scipy stats functions we need
15  import scipy.stats as spstats
16  # Import plotting library as plt
17  import matplotlib.pyplot as plt
18
19  class CalculateStatistics (object):
20
21      def run(self):
22          # Set up lists to hold input diameters and heights
23          # A seperate list is used for each species
24          beechDiameter = list()
25          beechHeight = list()
26          ashDiameter = list()
27          ashHeight = list()
28          birchDiameter = list()
29          birchHeight = list()
30          oakDiameter = list()
```

```
31          oakHeight = list()
32          sycamoreDiameter = list()
33          sycamoreHeight = list()
34          otherDiameter = list()
35          otherHeight = list()
36
37          # Open input and output files
38          inFileName = 'PenglaisWoodsData.csv'
39          outFileName = 'PenglaisWoodsStats.csv'
40          inFile = open(inFileName, 'r')
41          outFile = open(outFileName,'w')
42
43          # Iterate through the input file and save diameter and height
44          # into lists, based on species
45          header = True
46          for eachLine in inFile:
47              if header: # Skip header row
48                  print('Skipping header row')
49                  header = False
50              else:
51                  substrs = eachLine.split(',',eachLine.count(','))
52
53                  species = substrs[3]
54                  if substrs[4].isdigit: # Check diameter is a number
55                      diameter = float(substrs[4])
56                      height = float(substrs[10])
57
58                      if species == 'BEECH':
59                          beechDiameter.append(diameter)
60                          beechHeight.append(height)
61                      elif species == 'ASH':
62                          ashDiameter.append(diameter)
63                          ashHeight.append(height)
64                      elif species == 'BIRCH':
65                          birchDiameter.append(diameter)
66                          birchHeight.append(height)
67                      elif species == 'OAK':
68                          oakDiameter.append(diameter)
69                          oakHeight.append(height)
70                      elif species == 'SYC':
71                          sycamoreDiameter.append(diameter)
```

```
72                    sycamoreHeight.append(height)
73                else:
74                    otherDiameter.append(diameter)
75                    otherHeight.append(height)
76
77        # Convert to NumPy arrays
78        beechDiameter = numpy.array(beechDiameter)
79        ashDiameter = numpy.array(ashDiameter)
80        birchDiameter = numpy.array(birchDiameter)
81        oakDiameter = numpy.array(oakDiameter)
82        sycamoreDiameter = numpy.array(sycamoreDiameter)
83        otherDiameter = numpy.array(otherDiameter)
84
85        beechHeight = numpy.array(beechHeight)
86        ashHeight = numpy.array(ashHeight)
87        birchHeight = numpy.array(birchHeight)
88        oakHeight = numpy.array(oakHeight)
89        sycamoreHeight = numpy.array(sycamoreHeight)
90        otherHeight = numpy.array(otherHeight)
91
92        # Write header line to output file
93        headerLine = 'species, meanDiameter, medianDiameter, stDevDiameter\n'
94        outFile.write(headerLine)
95
96        # Calculate statistics for each species and write to file
97        outLine = 'Beech,' + self.createStatsLine(beechDiameter) + '\n'
98        outFile.write(outLine)
99        outLine = 'Ash,' + self.createStatsLine(ashDiameter) + '\n'
100       outFile.write(outLine)
101       outLine = 'Birch,' + self.createStatsLine(birchDiameter) + '\n'
102       outFile.write(outLine)
103       outLine = 'Oak,' + self.createStatsLine(oakDiameter) + '\n'
104       outFile.write(outLine)
105       outLine = 'Sycamore,' + self.createStatsLine(sycamoreDiameter) + '\n'
106       outFile.write(outLine)
107       outLine = 'Other,' + self.createStatsLine(otherDiameter) + '\n'
108       outFile.write(outLine)
109
110       print('Statistics written to: ' + outFileName)
111
112       # Fit line to each file and save out plot
```

```
113
114            # Set output directory for plots
115            outDIR = './'
116
117            # Plot linear regression for Beech
118            print('Generating plot:')
119            self.plotLinearRegression(beechDiameter, beechHeight, outDIR + 'beech.pdf')
120
121        def plotLinearRegression(self, inDiameterArray, inHeightArray, outPlotName):
122            # Perform fit
123            (aCoeff,bCoeff,rVal,pVal,stdError) = spstats.linregress(numpy.log(inDiameterArray), inHei
124
125            # Use fits to predict height for a range of diameters
126            testDiameter = numpy.arange(min(inDiameterArray), max(inDiameterArray), 1)
127            predictHeight = (aCoeff * numpy.log(testDiameter)) + bCoeff
128
129            # Create a string, showing the form of the equation (with fitted coefficients)
130            # and r squared value
131            # Coefficients are rounded to two decimal places.
132            equation = str(round(aCoeff,2)) + 'log(D) ' + str(round(bCoeff,2)) + \
133                ' (r$^2$ = ' + str(round(rVal**2,2)) + ')'
134
135            # Plot fit against origional data
136            plt.plot(inDiameterArray, inHeightArray,'.')
137            plt.plot(testDiameter, predictHeight)
138            plt.xlabel('Diameter (cm)')
139            plt.ylabel('Height (m)')
140            plt.legend(['measured data',equation])
141
142            # Save plot
143            plt.savefig(outPlotName, dpi=200, format='PDF')
144
145
146        def createStatsLine(self, inArray):
147            # Calculate statsistics for array and return output line.
148            meanArray = numpy.mean(inArray)
149            medianArray = numpy.median(inArray)
150            stDevArray = numpy.std(inArray)
151
152            # Create output line with stats
153            statsLine = str(meanArray) + ',' + str(medianArray) + ',' + str(stDevArray)
```

```
154            return statsLine
155
156    def calcBiomassStatsLine(self, inDiameterArray, inHeightArray, inSpecies):
157        # Calculates biomass, calculates stats from biomass and returns output line
158        biomassStatsLine = ''
159        try:
160            # Calculate biomass
161            biomass = self.calcBiomass(inDiameterArray, inHeightArray, inSpecies)
162            # Calculate stats from biomass
163            meanBiomass = numpy.mean(biomass)
164            medianBiomass = numpy.median(biomass)
165            stDevBiomass = numpy.std(biomass)
166
167            # Create output line
168            biomassStatsLine = str(meanBiomass) + ',' + str(medianBiomass) + ','\
169                + str(stDevBiomass)
170
171        except Exception:
172            # Catch exception and write 'na' for all values
173            biomassStatsLine = 'na,na,na'
174
175        return biomassStatsLine
176
177    def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
178        if inSpecies == 'BEECH':
179            a = 0.014306
180            b = 0.0000748
181            specificGravity = 0.56
182        else: # Raise exception is species is not recognised
183            raise Exception('Species not recognised')
184
185        # Calcualte volume
186        volume = a + ((b*(inDiameterArray)**2) * (inHeightArray**0.75))
187        # Calculate biomass
188        biomass = volume * specificGravity
189        # Return biomass
190        return biomass
191
192 if __name__ == '__main__':
193     obj = CalculateStatistics()
194     obj.run()
```

6.4.1 Exercise

Produce plots, showing linear regression fits, for the other species.

6.5 Further Reading

- SciPy – `http://www.scipy.org/SciPy`

- NumPy – `http://numpy.scipy.org`

- An Introduction to Python, G. van Rossum, F.L. Drake, Jr. Network Theory ISBN 0-95-416176-9 (Also available online – `http://docs.python.org/3/tutorial/`) - Chapter 8.

- Python Documentation – `http://www.python.org/doc/`

- Matplotlib – `http://matplotlib.sourceforge.net`

Chapter 7

Batch Processing Command Line Tools

7.1   Introduction

There are many command line tools and utilities available for all platforms (e.g., Windows, Linux, Mac OSX), these tools are extremely useful and range from simple tasks such as renaming a file to more complex tasks such as merging ESRI shapefiles. One problem with these tools is that if you have a large number of files, which need to be processed in the same way, it is time consuming and error prone to manual run the command for each file. Therefore, if we can write scripts to do this work for us then processing large number of individual files becomes a much simpler and quicker task.

For this worksheet you will need to have the command line tools which come with the GDAL/OGR (`http://www.gdal.org`) open source software library installed and available with your path. With the installation of python(x,y) the python libraries for GDAL/OGR have been installed but not the command line utilities which go along with these libraries. If you do not already have them installed therefore details on the GDAL website for your respective platform.

7.2   Merging ESRI Shapefiles

The first example illustrates how the 'ogr2ogr' command can be used to merge shapefiles and a how a python script can be used to turn this command into a batch process where a whole directory of shapefiles can be merged.

To perform this operation two commands are required. The first makes a copy of the first shapefile within the list of files into a new file, shown below:

```
> ogr2ogr <inputfile> <outputfile>
```

While the second command appends the contents of the inputted shapefile onto the end of an existing shapefile (i.e., the one just copied).

```
> ogr2ogr -update -append <inputfile> <outputfile> -nln <outputfilename>
```

For both these commands the shapefiles all need to be of the same type (point, polyline or polygon) and contain the same attributes. Therefore, your first exercise is to understand the use of the ogr2ogr command and try them from the command line with the data provided. *Hint*, running ogr2ogr without any options the help file will be displayed.

The second stage is to develop a python script to call the appropriate commands to perform the required operation, where the following processes will be required:

1. Get the user inputs.

2. List the contents of the input directory.

3. Iterate through the directory and run the required commands.

But the first step is to create the class structure in which the code will fit, this will be something similar to that shown below:

```
1   #! /usr/bin/env python
2
3   ######################################
4   # MergeSHPfiles.py
5   # A python script to merge shapefiles
6   # Author: <YOUR NAME>
```

```
7   # Email: <YOUR EMAIL>
8   # Date: DD/MM/YYYY
9   # Version: 1.0
10  ######################################
11
12  import os
13
14  class MergeSHPfiles (object):
15
16      # A function which controls the rest of the script
17      def run(self):
18          # Define the input directory
19          filePath = './TreeCrowns/'
20          # Define the output file
21          newSHPfile = 'Merged_shapefile.shp'
22
23  # The start of the code
24  if __name__ == '__main__':
25      # Make an instance of the class
26      obj = MergeSHPfiles()
27      # Call the function run()
28      obj.run()
```

The script will have the input directory and output file hard coded (as shown) within the run function. Therefore, you need to edit these file paths to the location you have the files saved. Please note that under Windows you need to insert a double slash (i.e., \\) within the file path as a single slash is an escape character (e.g., \n for new line) within strings.

The next step is to check that the input directory exists and is a directory, to do this edit your run function as below.

```
1   # A function which controls the rest of the script
2   def run(self):
3       # Define the input directory
4       filePath = './TreeCrowns/'
5       # Define the output file
6       newSHPfile = 'Merged_shapefile.shp'
7
8       # Check input file path exists and is a directory
```

```
9      if not os.path.exists(filePath):
10          print 'Filepath does not exist'
11      elif not os.path.isdir(filePath):
12          print 'Filepath is not a directory!'
13      else:
14          # Merge the shapefiles within the filePath
15          self.mergeSHPfiles(filePath, newSHPfile)
```

Additionally, you need to add the function mergeSHPFiles, which is where the shapefiles will be merged.

```
# A function to control the merging of shapefiles
def mergeSHPfiles(self, filePath, newSHPfile):
```

To merge the shapefiles the first task is to get a list of all the shapefiles within a directory. To do this, use the code you developed in Tutorial 4 to list files within a directory and edit it such that the files are outputted to a list rather than printed to screen, as shown below.

```
1   # A function to test the file extension of a file
2   def checkFileExtension(self, filename, extension):
3       # Boolean variable to be returned by the function
4       foundExtension = False;
5       # Split the filename into two parts (name + ext)
6       filenamesplit = os.path.splitext(filename)
7       # Get the file extension into a varaiable
8       fileExtension = filenamesplit[1].strip()
9       # Decide whether extensions are equal
10      if(fileExtension == extension):
11          foundExtension = True
12      # Return result
13      return foundExtension
14
15
16  # A function which iterates through the directory and checks file extensions
17  def findFilesExt(self, directory, extension):
18      # Define a list to store output list of files
19      fileList = list()
20      # check whether the current directory exits
21      if os.path.exists(directory):
22          # check whether the given directory is a directory
```

```
23        if os.path.isdir(directory):
24            # list all the files within the directory
25            dirFileList = os.listdir(directory)
26            # Loop through the individual files within the directory
27            for filename in dirFileList:
28                # Check whether file is directory or file
29                if(os.path.isdir(os.path.join(directory,filename))):
30                    print os.path.join(directory,filename) + \
31                    ' is a directory and therefore ignored!'
32                elif(os.path.isfile(os.path.join(directory,filename))):
33                    if(self.checkFileExtension(filename, extension)):
34                        fileList.append(os.path.join(directory,filename))
35                else:
36                    print filename + ' is NOT a file or directory!'
37        else:
38            print directory + ' is not a directory!'
39    else:
40        print directory + ' does not exist!'
41    # Return the list of files
42    return fileList
```

Note, that you also need the function to check the file extension.

This can then be added to the mergeSHPfiles function with a list to iterate through the identified files.

```
1  # A function to control the merging of shapefiles
2  def mergeSHPfiles(self, filePath, newSHPfile):
3      # Get the list of files within the directory
4      # provided with the extension .shp
5      fileList = self.findFilesExt(filePath, '.shp')
6      # Iterate through the files.
7      for file in fileList:
8          print(file)
```

When iterating through the files the ogr2ogr commands to be executed to merge the shapefiles need to be built and executed therefore the following code needs to be added to your script. Once the command is built it is called using the subprocess module's 'call' command.

```python
1   # A function to control the merging of shapefiles
2   def mergeSHPfiles(self, filePath, newSHPfile):
3       # Get the list of files within the directory
4       # provided with the extension .shp
5       fileList = self.findFilesExt(filePath, '.shp')
6       # Variable used to identify the first file
7       first = True
8       # A string for the command to be built
9       command = ''
10      # Iterate through the files.
11      for file in fileList:
12          if first:
13              # If the first file make a copy to create the output file
14              command = 'ogr2ogr ' + newSHPfile + ' ' + file
15              first = False
16          else:
17              # Otherwise append the current shapefile to the output file
18              command = 'ogr2ogr -update -append ' + newSHPfile + ' '  + \
19              file + ' -nln ' + \
20              self.removeSHPExtension(self.removeFilePathUNIX(newSHPfile))
21          # Execute the current command
22          subprocess.call(command,shell=True)
```

Note, the use of the option `shell=True` within the subprocess function call, this means to treat the command as a string and execute as is. While this is fine for simple scripts for your own use if poses a security risk when using in production code as it leaves the program open to shell injection attempts. For example, if the program tool a user provided file name and used this when building a command a user could potentially pass in a malicious command (e.g., `rm -fr *;`) rather than a file name and this would be executed. To avoid these problems, by default subprocess requires a list of command line arguments and shell is set to 'False'.

You also require the additional functions to remove the shapefile extension (.shp) and the windows file path, creating the layer name which are given below.

```python
1   # A function to remove a .shp extension from a file name
2   def removeSHPExtension(self, name):
3       # The output file name
```

```
4        outName = name
5        # Find how many '.shp' strings are in the current file
6        # name
7        count = name.find('.shp', 0, len(name))
8        # If there are no instances of .shp then -1 will be returned
9        if not count == -1:
10            # Replace all instances of .shp with empty string.
11            outName = name.replace('.shp', '', name.count('.shp'))
12        # Return output file name without .shp
13        return outName
14
15   # A function to remove the file path a file
16   # (in this case a windows file path)
17   def removeFilePathWINS(self, name):
18        # Remove white space (i.e., spaces, tabs)
19        name = name.strip()
20        # Count the number of slashs
21        # A double slash is required because \ is a
22        # string escape charater.
23        count = name.count('\\')
24        # Split string into a list where slashs occurs
25        nameSegments = name.split('\\', count)
26        # Return the last item in the list
27        return nameSegments[count]
28
29   # A function to remove the file path a file
30   def removeFilePathUNIX(self, name):
31        # Remove white space (i.e., spaces, tabs)
32        name = name.strip()
33        # Count the number of slashs
34        count = name.count('/')
35        # Split string into a list where slashs occurs
36        nameSegments = name.split('/', count)
37        # Return the last item in the list
38        return nameSegments[count]
```

If you wanted to use this script on UNIX (i.e., Linux or Mac OS X) you would
need to use the removeFilePathUNIX as shown while for windows change the code
to use the removeFilePathWINS function such that the double escaped slashes are
used.

You script should now be complete so execute it on the data provided, within the TreeCrowns directory. Take time to understand the lines of code which have been provided and make sure your script works.

```python
#! /usr/bin/env python

#######################################
# MergeSHPfiles.py
# A python script to merge shapefiles
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

import os
import subprocess

class MergeSHPfiles (object):

    # A function to remove a .shp extension from a file name
    def removeSHPExtension(self, name):
        # The output file name
        outName = name
        # Find how many '.shp' strings are in the current file
        # name
        count = name.find('.shp', 0, len(name))
        # If there are no instances of .shp then -1 will be returned
        if not count == -1:
            # Replace all instances of .shp with empty string.
            outName = name.replace('.shp', '', name.count('.shp'))
        # Return output file name without .shp
        return outName

    # A function to remove the file path a file
    # (in this case a windows file path)
    def removeFilePathWINS(self, name):
        # Remove white space (i.e., spaces, tabs)
        name = name.strip()
        # Count the number of slashs
```

```
37              # A double slash is required because \ is a
38              # string escape charater.
39              count = name.count('\\')
40              # Split string into a list where slashs occurs
41              nameSegments = name.split('\\', count)
42              # Return the last item in the list
43              return nameSegments[count]
44
45          # A function to remove the file path a file
46          def removeFilePathUNIX(self, name):
47              # Remove white space (i.e., spaces, tabs)
48              name = name.strip()
49              # Count the number of slashs
50              count = name.count('/')
51              # Split string into a list where slashs occurs
52              nameSegments = name.split('/', count)
53              # Return the last item in the list
54              return nameSegments[count]
55
56          # A function to test the file extension of a file
57          def checkFileExtension(self, filename, extension):
58              # Boolean variable to be returned by the function
59              foundExtension = False;
60              # Split the filename into two parts (name + ext)
61              filenamesplit = os.path.splitext(filename)
62              # Get the file extension into a varaiable
63              fileExtension = filenamesplit[1].strip()
64              # Decide whether extensions are equal
65              if(fileExtension == extension):
66                  foundExtension = True
67              # Return result
68              return foundExtension
69
70
71          # A function which iterates through the directory and checks file extensions
72          def findFilesExt(self, directory, extension):
73              # Define a list to store output list of files
74              fileList = list()
75              # check whether the current directory exits
76              if os.path.exists(directory):
77                  # check whether the given directory is a directory
```

```python
78              if os.path.isdir(directory):
79                  # list all the files within the directory
80                  dirFileList = os.listdir(directory)
81                  # Loop through the individual files within the directory
82                  for filename in dirFileList:
83                      # Check whether file is directory or file
84                      if(os.path.isdir(os.path.join(directory,filename))):
85                          print(os.path.join(directory,filename) + \
86                          ' is a directory and therefore ignored!')
87                      elif(os.path.isfile(os.path.join(directory,filename))):
88                          if(self.checkFileExtension(filename, extension)):
89                              fileList.append(os.path.join(directory,filename))
90                      else:
91                          print(filename + ' is NOT a file or directory!')
92              else:
93                  print(directory + ' is not a directory!')
94          else:
95              print(directory + ' does not exist!')
96          # Return the list of files
97          return fileList
98
99      # A function to control the merging of shapefiles
100     def mergeSHPfiles(self, filePath, newSHPfile):
101         # Get the list of files within the directory
102         # provided with the extension .shp
103         fileList = self.findFilesExt(filePath, '.shp')
104         # Variable used to identify the first file
105         first = True
106         # A string for the command to be built
107         command = ''
108         # Iterate through the files.
109         for file in fileList:
110             if first:
111                 # If the first file make a copy to create the output file
112                 command = 'ogr2ogr ' + newSHPfile + ' ' + file
113                 first = False
114             else:
115                 # Otherwise append the current shapefile to the output file
116                 command = 'ogr2ogr -update -append ' + newSHPfile + ' '  + \
117                 file + ' -nln ' + \
118                 self.removeSHPExtension(self.removeFilePathUNIX(newSHPfile))
```

```
119              # Execute the current command
120              subprocess.call(command,shell=True)
121
122      # A function which controls the rest of the script
123      def run(self):
124          # Define the input directory
125          filePath = './TreeCrowns/'
126          # Define the output file
127          newSHPfile = 'Merged_TreeCrowns.shp'
128
129          # Check input file path exists and is a directory
130          if not os.path.exists(filePath):
131              print('Filepath does not exist')
132          elif not os.path.isdir(filePath):
133              print('Filepath is not a directory!')
134          else:
135              # Merge the shapefiles within the filePath
136              self.mergeSHPfiles(filePath, newSHPfile)
137
138  # The start of the code
139  if __name__ == '__main__':
140      # Make an instance of the class
141      obj = MergeSHPfiles()
142      # Call the function run()
143      obj.run()
```

7.3   Convert Images to GeoTIFF using GDAL.

The next example will require you to use the script developed above as the basis
for a new script using the command below to convert a directory of images to
GeoTIFF using the command given:

```
gdal_translate -of <OutputFormat> <InputFile> <OutputFile>
```

A useful step is to first run the command from the command line manually to
make sure you understand how this command is working.

The two main things you need to think about are:

1. What file extension will the input files have? This should be user selectable alongside the file paths.

2. What output file name should be provided? The script should generate this.

Four test images have been provided in ENVI format within the directory ENVI_Images, you can use these for testing your script. If you are struggling then an example script with a solution to this task has been provided within the code directory.

7.3.1 Passing Inputs from the Command Line into your script

It is often convenient to provide the inputs the scripts requires (e.g., input and output file locations) as arguments to the script rather than needing to the edit the script each time a different set of parameters are required (i.e., changing the files paths in the scripts above). This is easy within python and just requires the following changes to your run function (in this case for the merge shapefiles script).

```python
# A function which controls the rest of the script
def run(self):
    # Get the number of arguments
    numArgs = len(sys.argv)
    # Check there are only 2 input argument (i.e., the input file
    # and output base).
    # Note that argument 0 (i.e., sys.argv[0]) is the name
    # of the script currently running.
    if numArgs == 3:
        # Retrieve the input directory
        filePath = sys.argv[1]
        # Retrieve the output file
        newSHPfile = sys.argv[2]

        # Check input file path exists and is a directory
        if not os.path.exists(filePath):
            print("Filepath does not exist")
```

```
18          elif not os.path.isdir(filePath):
19              print("Filepath is not a directory!")
20          else:
21              # Merge the shapefiles within the filePath
22              self.mergeSHPfiles(filePath, newSHPfile)
23      else:
24          print("ERROR. Command should have the form:")
25          print("python MergeSHPfiles_cmd.py <Input File Path> <Output File>")
```

In addition, to these changes you need to import the system library into your script to access these arguments.

```
# Import the sys package from within the
# standard library
import sys
```

Please note that the list of user provided inputs starts at index 1 and not 0. If you call sys.argv[0] then the name of the script being executed will be returned. When retrieving values from the user in this form it is highly advisable to check whether the inputs provided are valid and that all required inputs have been provided.

Create a copy of the script you created earlier and edit the run function to be as shown above, making note of the lines which require editing.

7.4 Exercises

1. Using ogr2ogr develop a script that will convert the attribute table of a shapefile to a CSV file which can be opened within Microsoft Excel. Note, that the outputted CSV will be put into a separate directory.

2. Create a script which calls the gdal_translate command and converts all the images within a directory to a byte data type (i.e., with a range of 0 to 255).

7.5 Further Reading

- GDAL - `http://www.gdal.org`

- OGR - `http://www.gdal.org/ogr`

- Python Documentation - `http://www.python.org/doc`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

- Learn UNIX in 10 minutes - `http://freeengineer.org/learnUNIXin10minutes.html`

- The Linux Command Line. W. E. Shotts. No Starch Press. ISBN 978-1-59327-389-7 (Available to download from `http://linuxcommand.org/tlcl.php`)

Chapter 8

Image Processing using GDAL and RIOS

8.1 Reading and Updating Header Information

Image files used within spatial data processing (i.e., remote sensing and GIS) require the addition of a spatial header to the files which provides the origin (usually from the top left corner of the image), the pixel resolution of the image and a definition of the coordinate system and projection of the dataset. Additionally, most formats also allow a rotation to be defined. Using these fields the geographic position on the Earth's surface can be defined for each pixel within the scene.

Images can also contain other information in the header of the file including no data values, image statistics and band names/descriptions.

8.1.1 Reading Image Headers

The GDAL software library provides a python interface to the C++ library, such that when the python functions are called is it the C++ implementation which is executed. These model has significant advantages for operations such as reading and writing to and from image files as in pure python these operations would be slow but they as very fast within C++. Although, python is an easier language

for people to learn and use, therefore allows software to be more quickly developed so combing C++ and python in this way is a very productive way for software to be developed.

Argparser

Up until this point we have read parameters from the system by just using the sys.argv list where the user is required to enter the values in a given pre-defined order. The problem with this is that it is not very helpful to the user as no help is provided or error messages given if the wrong parameters are entered. For command line tools it is generally accepted that when providing command line options they will use switches such as -i or –input where the user specifies with a switch what the input they are providing is.

Fortunately, python provides a library to simplify the implementation of this type of interface. An example of this is shown below, where first the argparse library is imported. The parser is then created and the arguments added to the parser so the parser knows what to expect from the user. Finally, the parser is called to parse the arguments. Examples will be shown in all the following scripts.

```python
1  # Import the python Argument parser
2  import argparse
3
4  # Create the parser
5  parser = argparse.ArgumentParser()
6  # Define the argument for specifying the input file.
7  parser.add_argument("-i", "--input", type=str, help="Specify the input image file.")
8  # Define the argument for specifying the output file.
9  parser.add_argument("-o", "--output", type=str, help="Specify the output text file.")
10 # Call the parser to parse the arguments.
11 args = parser.parse_args()
```

8.1.2 Read image header example.

The follow example demonstrates how to import the GDAL library into python and to read the image header information and print it to the console - similar to the functionality within the gdalinfo command. Read the comments within the code and ensure you understand the steps involved.

```python
#!/usr/bin/env python

# Import the GDAL python library
import osgeo.gdal as gdal
# Import the python Argument parser
import argparse
# Import the System library
import sys

# Define a function to read and print the images
# header information.
def printImageHeader(inputFile):
    # Open the dataset in Read Only mode
    dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
    # Check that the dataset has correctly opened
    if not dataset is None:
        # Print out the image file path.
        print(inputFile)
        # Print out the number of image bands.
        print("The image has ", dataset.RasterCount, " bands.")
        # Loop through all the image bands and print out the band name
        for n in range(dataset.RasterCount):
            print("\t", n+1, ":\t", dataset.GetRasterBand(n+1).GetDescription(), "\t")

        # Print out the image size in pixels
        print("Image Size [", dataset.RasterXSize, ",", dataset.RasterYSize, "]")

        # Get the geographic header
        # geotransform[0] = TL X Coordinate
        # geotransform[1] = X Pixel Resolution
        # geotransform[2] = X Rotation
        # geotransform[3] = TL Y Coordinate
        # geotransform[4] = Y Rotation
```

```
34          # geotransform[5] = Y Pixel Resolution
35          geotransform = dataset.GetGeoTransform()
36          # Check that the tranformation has been correctly read.
37          if not geotransform is None:
38              # Print out the Origin, Pixel Size and Rotation.
39              print('Origin = (',geotransform[0], ',',geotransform[3],')')
40              print('Pixel Size = (',geotransform[1], ',',geotransform[5],')')
41              print('Rotation = (',geotransform[2], ',',geotransform[4],')')
42          else:
43              # Provide an error message is the transform has not been
44              # correctly read.
45              print("Could not find a geotransform for image file ", inputFile)
46      else:
47          # Provide an error message if the input image file
48          # could not be opened.
49          print("Could not open the input image file: ", inputFile)
50
51
52  # This is the first part of the script to
53  # be executed.
54  if __name__ == '__main__':
55      # Create the command line options
56      # parser.
57      parser = argparse.ArgumentParser()
58      # Define the argument for specifying the input file.
59      parser.add_argument("-i", "--input", type=str,
60                          help="Specify the input image file.")
61      # Call the parser to parse the arguments.
62      args = parser.parse_args()
63
64      # Check that the input parameter has been specified.
65      if args.input == None:
66          # Print an error message if not and exit.
67          print("Error: No input image file provided.")
68          sys.exit()
69      # Otherwise, run the function to print out the image header information.
70      printImageHeader(args.input)
```

Running the script

Run the script as you have done others within these worksheets and as shown below, you need to provide the full path to the image file or copy the image file into the same directory as your script. This should result in an output like the one shown below:

```
> python ReadImageHeader.py -i LSTOA_Tanz_2000Wet.img
LSTOA_Tanz_2000Wet.img
The image has  6  bands.
        1 :         Band 1
        2 :         Band 2
        3 :         Band 3
        4 :         Band 4
        5 :         Band 5
        6 :         Band 6
Image Size [ 1776 , 1871 ]
Origin = ( 35.2128071515 , -3.05897460167 )
Pixel Size = ( 0.000271352299023 , -0.000271352299023 )
Rotation = ( 0.0 , 0.0 )
```

8.1.3   No Data Values

GDAL also allows us to edit the image header values, therefore the following example provides an example of how to edit the no data value for image band. Note that when opening the image file the gdal.GA_Update option is used rather than gdal.GA_ReadOnly.

A no data value is useful for defining regions of the image which are not valid (i.e., outside of the image boundaries) and can be ignored during processing.

Running the script

For the file provided (LSTOA_Tanz_2000Wet.img) the no data value for all the bands should be 0. Therefore, run the following command:

```
> python setnodata.py -i LSTOA_Tanz_2000Wet.img -n 0.0
Setting No data (0.0) for band 1
Setting No data (0.0) for band 2
Setting No data (0.0) for band 3
Setting No data (0.0) for band 4
Setting No data (0.0) for band 5
Setting No data (0.0) for band 6
```

To check that command successfully edited the input file use the gdalinfo command, as shown below:

```
gdalinfo -norat LSTOA_Tanz_2000Wet.img
```

```python
1  #!/usr/bin/env python
2
3  # Import the GDAL python library
4  import osgeo.gdal as gdal
5  # Import the python Argument parser
6  import argparse
7  # Import the System library
8  import sys
9
10  # A function to set the no data value
11  # for each image band.
12  def setNoData(inputFile, noDataVal):
13      # Open the image file, in update mode
14      # so that the image can be edited.
15      dataset = gdal.Open(inputFile, gdal.GA_Update)
16      # Check that the image  has been opened.
17      if not dataset is None:
18          # Iterate throught he image bands
19          # Note. i starts at 0 while the
20          # band count in GDAL starts at 1.
21          for i in range(dataset.RasterCount):
22              # Print information to the user on what is
23              # being set.
24              print("Setting No data (" + str(noDataVal) + ") for band " + str(i+1))
25              # Get the image band
26              # the i+1 is because GDAL bands
```

```
27                     # start with 1.
28                     band = dataset.GetRasterBand(i+1)
29                     # Set the no data value.
30                     band.SetNoDataValue(noDataVal)
31         else:
32             # Print an error message if the file
33             # could not be opened.
34             print("Could not open the input image file: ", inputFile)
35
36  # This is the first part of the script to
37  # be executed.
38  if __name__ == '__main__':
39      # Create the command line options
40      # parser.
41      parser = argparse.ArgumentParser()
42      # Define the argument for specifying the input file.
43      parser.add_argument("-i", "--input", type=str,
44                          help="Specify the input image file.")
45      # Define the argument for specifying the no data value.
46      parser.add_argument("-n", "--nodata", type=float, default=0,
47                          help="Specify the no data value to be set.")
48      # Call the parser to parse the arguments.
49      args = parser.parse_args()
50
51      # Check that the input parameter has been specified.
52      if args.input == None:
53          # Print an error message if not and exit.
54          print("Error: not input image file provided.")
55          sys.exit()
56      # Otherwise, run the function to set the no
57      # data value.
58      setNoData(args.input, args.nodata)
```

8.1.4   Band Name

Band names are useful for a user to understand a data set more easily. Therefore, naming the image bands, such as Blue, Green, Red, NIR and SWIR, is very useful. The following example illustrates how to edit the band name description

using GDAL.

```python
#!/usr/bin/env python


# Import the GDAL python library
import osgeo.gdal as gdal
# Import the python Argument parser
import argparse
# Import the System library
import sys


# A function to set the no data value
# for each image band.
def setBandName(inputFile, band, name):
    # Open the image file, in update mode
    # so that the image can be edited.
    dataset = gdal.Open(inputFile, gdal.GA_Update)
    # Check that the image  has been opened.
    if not dataset is None:
        # Get the image band
        imgBand = dataset.GetRasterBand(band)
        # Check the image band was available.
        if not imgBand is None:
            # Set the image band name.
            imgBand.SetDescription(name)
        else:
            # Print out an error message.
            print("Could not open the image band: ", band)
    else:
        # Print an error message if the file
        # could not be opened.
        print("Could not open the input image file: ", inputFile)

# This is the first part of the script to
# be executed.
if __name__ == '__main__':
    # Create the command line options
    # parser.
    parser = argparse.ArgumentParser()
    # Define the argument for specifying the input file.
    parser.add_argument("-i", "--input", type=str,
```

```
40                            help="Specify the input image file.")
41          # Define the argument for specifying image band.
42          parser.add_argument("-b", "--band", type=int,
43                            help="Specify image band.")
44          # Define the argument for specifying band name.
45          parser.add_argument("-n", "--name", type=str,
46                            help="Specify the band name.")
47          # Call the parser to parse the arguments.
48          args = parser.parse_args()
49
50          # Check that the input parameter has been specified.
51          if args.input == None:
52              # Print an error message if not and exit.
53              print("Error: No input image file provided.")
54              sys.exit()
55
56          # Check that the band parameter has been specified.
57          if args.band == None:
58              # Print an error message if not and exit.
59              print("Error: the band was not specified.")
60              sys.exit()
61
62          # Check that the name parameter has been specified.
63          if args.name == None:
64              # Print an error message if not and exit.
65              print("Error: the band name was not specified.")
66              sys.exit()
67
68          # Otherwise, run the function to set the band
69          # name.
70          setBandName(args.input, args.band, args.name)
```

Running the script

The file provided (LSTOA_Tanz_2000Wet.img) just has some default band names
defined (i.e., Band 1) but use you script to change them to something more useful.
Therefore, run the following commands:

```
python setbandname.py -i LSTOA_Tanz_2000Wet.img -b 1 -n Blue
python setbandname.py -i LSTOA_Tanz_2000Wet.img -b 2 -n Green
python setbandname.py -i LSTOA_Tanz_2000Wet.img -b 3 -n Red
python setbandname.py -i LSTOA_Tanz_2000Wet.img -b 4 -n NIR
python setbandname.py -i LSTOA_Tanz_2000Wet.img -b 5 -n SWIR1
python setbandname.py -i LSTOA_Tanz_2000Wet.img -b 6 -n SWIR2
```

Use you script for reading the image header values and printing them to the screen to find out whether it worked.

8.1.5 GDAL Meta-Data

GDAL supports the concept of meta-data on both the image bands and the whole image. The meta-data allows any other data to be stored within the image file as a string.

The following example shows how to read the meta-data values and to list all the meta-data variables available on both the image bands and the image.

```python
1   #!/usr/bin/env python
2
3   # Import the GDAL python library
4   import osgeo.gdal as gdal
5   # Import the python Argument parser
6   import argparse
7   # Import the System library
8   import sys
9
10  # A function to read a meta-data item
11  # from a image band
12  def readBandMetaData(inputFile, band, name):
13      # Open the dataset in Read Only mode
14      dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
15      # Check that the image  has been opened.
16      if not dataset is None:
17          # Get the image band
18          imgBand = dataset.GetRasterBand(band)
19          # Check the image band was available.
```

```python
20          if not imgBand is None:
21              # Get the meta-data value specified.
22              metaData = imgBand.GetMetadataItem(name)
23              # Check that it is present
24              if metaData == None:
25                  # If not present, print error.
26                  print("Could not find \'", name, "\' item.")
27              else:
28                  # Else print out the metaData value.
29                  print(name, " = \'", metaData, "\'")
30          else:
31              # Print out an error message.
32              print("Could not open the image band: ", band)
33      else:
34          # Print an error message if the file
35          # could not be opened.
36          print("Could not open the input image file: ", inputFile)
37
38  # A function to read a meta-data item
39  # from a image
40  def readImageMetaData(inputFile, name):
41      # Open the dataset in Read Only mode
42      dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
43      # Check that the image  has been opened.
44      if not dataset is None:
45          # Get the meta-data value specified.
46          metaData = dataset.GetMetadataItem(name)
47          # Check that it is present
48          if metaData == None:
49              # If not present, print error.
50              print("Could not find \'", name, "\' item.")
51          else:
52              # Else print out the metaData value.
53              print(name, " = \'", metaData, "\'")
54      else:
55          # Print an error message if the file
56          # could not be opened.
57          print("Could not open the input image file: ", inputFile)
58
59  # A function to read a meta-data item
60  # from a image band
```

```
61  def listBandMetaData(inputFile, band):
62      # Open the dataset in Read Only mode
63      dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
64      # Check that the image  has been opened.
65      if not dataset is None:
66          # Get the image band
67          imgBand = dataset.GetRasterBand(band)
68          # Check the image band was available.
69          if not imgBand is None:
70              # Get the meta-data dictionary
71              metaData = imgBand.GetMetadata_Dict()
72              # Check it has entries.
73              if len(metaData) == 0:
74                  # If it has no entries return
75                  # error message.
76                  print("There is no image meta-data.")
77              else:
78                  # Otherwise, print out the
79                  # meta-data.
80                  # Loop through each entry.
81                  for metaItem in metaData:
82                      print(metaItem)
83          else:
84              # Print out an error message.
85              print("Could not open the image band: ", band)
86      else:
87          # Print an error message if the file
88          # could not be opened.
89          print("Could not open the input image file: ", inputFile)
90
91  # A function to read a meta-data item
92  # from a image
93  def listImageMetaData(inputFile):
94      # Open the dataset in Read Only mode
95      dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
96      # Check that the image  has been opened.
97      if not dataset is None:
98          # Get the meta-data dictionary
99          metaData = dataset.GetMetadata_Dict()
100         # Check it has entries.
101         if len(metaData) == 0:
```

```
102              # If it has no entries return
103              # error message.
104              print("There is no image meta-data.")
105          else:
106              # Otherwise, print out the
107              # meta-data.
108              # Loop through each entry.
109              for metaItem in metaData:
110                  print(metaItem)
111      else:
112          # Print an error message if the file
113          # could not be opened.
114          print("Could not open the input image file: ", inputFile)
115
116  # This is the first part of the script to
117  # be executed.
118  if __name__ == '__main__':
119      # Create the command line options
120      # parser.
121      parser = argparse.ArgumentParser()
122      # Define the argument for specifying the input file.
123      parser.add_argument("-i", "--input", type=str,
124                      help="Specify the input image file.")
125      # Define the argument for specifying image band.
126      parser.add_argument("-b", "--band", type=int, default=0,
127                      help="Specify image band.")
128      # Define the argument for specifying meta-data name.
129      parser.add_argument("-n", "--name", type=str,
130                      help="Specify the meta-data name.")
131      # Define the argument for specifying whether the
132      # meta-data field should be just listed.
133      parser.add_argument("-l", "--list", action="store_true", default=False,
134                      help="Specify that meta data items should be listed.")
135      # Call the parser to parse the arguments.
136      args = parser.parse_args()
137
138      # Check that the input parameter has been specified.
139      if args.input == None:
140          # Print an error message if not and exit.
141          print("Error: No input image file provided.")
142          sys.exit()
```

```
143
144      # Check that the name parameter has been specified.
145      # If it has been specified then run functions to
146      # read band meta-data.
147      if not args.name == None:
148          # Check whether the image band has been specified.
149          # the default was set at 0 (to indicate that it)
150          # hasn't been specified as GDAL band count starts
151          # at 1. This also means the user cannot type in
152          # a value of 0 and get an error.
153          if args.band == 0:
154              # Run the function to print out the image meta-data value.
155              readImageMetaData(args.input, args.name)
156          else:
157              # Otherwise, run the function to print out the band meta-data value.
158              readBandMetaData(args.input, args.band, args.name)
159      elif args.list:
160          if args.band == 0:
161              # Run the function to list image meta-data.
162              listImageMetaData(args.input)
163          else:
164              # Otherwise, run the function to list band meta-data.
165              listBandMetaData(args.input, args.band)
166      else:
167          # Print an error message if not and exit.
168          print("Error: the meta-data name or list option" + \
169                " need to be specified was not specified.")
170          sys.exit()
```

Running the script

This script has a number of options. Have a play with these options on the image
provided, an example shown below.

```
python ReadGDALMetaData.py -h
python ReadGDALMetaData.py -i LSTOA_Tanz_2000Wet.img  -l
python ReadGDALMetaData.py -i LSTOA_Tanz_2000Wet.img  -b 1 -l
python ReadGDALMetaData.py -i LSTOA_Tanz_2000Wet.img  -b 1 -n LAYER_TYPE
python ReadGDALMetaData.py -i LSTOA_Tanz_2000Wet.img  -b 3 -n STATISTICS_MEAN
```

8.2 Raster Input / Output Simplification (RIOS) Library

The raster input and output (I/O) simplification (RIOS) library is a set of python modules which makes it easier to write raster processing code in Python. Built on top of GDAL, it handles the details of opening and closing files, checking alignment of projections and raster grid, stepping through the raster in small blocks, etc., allowing the programmer to concentrate on implementing the solution to the problem rather than on how to access the raster data and detail with the spatial header.

Also, GDAL provides access to the image data through python RIOS makes it much more user friendly and easier to use. RIOS is available for as a free download from `https://bitbucket.org/chchrsc/rios/overview`

8.2.1 Getting Help – Reminder

Python provides a very useful help system through the command line. To get access to the help run python from the terminal

```
> python
```

Then import the library want to get help on

```
>>> import osgeo.gdal
```

and then run the help tool on the whole module

```
>>> import osgeo.gdal
>>> help(osgeo.gdal)
```

or on individual classes within the module

```
>>> import osgeo.gdal
>>> help(osgeo.gdal.Dataset)
```

To exit the help system just press the 'q' key on the keyboard.

8.2.2 Band Maths

Being able to apply equations to combine image bands, images or scale single bands is a key tool for remote sensing, for example to calibrate Landsat to radiance. The following examples demonstrate how to do this within the RIOS framework.

8.2.3 Multiply by a constant

The first example just multiples all the image bands by a constant (provided by the user). The first part of the code reads the users parameters (input file, output file and scale factor). To use the applier interface within RIOS you need to first setup the input and output file associations and then any other options required, in this case the constant for multiplication. Also, the controls object should be defined to set any other parameters

All processing within RIOS is undertaken on blocks, by default 200×200 pixels in size. To process the block a applier function needs to be defined (e.g., mutliplyByValue) where the inputs and outputs are passed to the function (these are the pixel values) and the other arguments object previously defined. The pixel values are represented as a numpy array, the dimensions are (n, y, x) where n is the number of image bands, y is the number of rows and x the number of columns in the block.

Because numpy will iterate through the array for us to multiply the whole array by a constant (e.g., 2) then we can just need the syntax shown below, which makes it very simple.

```python
#!/usr/bin/env python

import sys
# Import the python Argument parser
import argparse
# Import the RIOS applier interface
```

```
7   from rios import applier
8   from rios import cuiprogress
9
10  # Define the applier function
11  def mutliplyByValue(info, inputs, outputs, otherargs):
12      # Multiple the image1 by the scale factor
13      outputs.outimage = inputs.image1 * otherargs.scale
14
15  # This is the first part of the script to
16  # be executed.
17  if __name__ == '__main__':
18      # Create the command line options
19      # parser.
20      parser = argparse.ArgumentParser()
21      # Define the argument for specifying the input file.
22      parser.add_argument("-i", "--input", type=str,
23                          help="Specify the input image file.")
24      # Define the argument for specifying the output file.
25      parser.add_argument("-o", "--output", type=str,
26                          help="Specify the output image file.")
27      # Define the argument for multiply the image by.
28      parser.add_argument("-m", "--multiply", default=1.0, type=float,
29                          help="Multiple the image by.")
30      # Call the parser to parse the arguments.
31      args = parser.parse_args()
32
33      # Check that the input parameter has been specified.
34      if args.input == None:
35          # Print an error message if not and exit.
36          print("Error: No input image file provided.")
37          sys.exit()
38
39      # Check that the output parameter has been specified.
40      if args.output == None:
41          # Print an error message if not and exit.
42          print("Error: No output image file provided.")
43          sys.exit()
44
45      # Create input files file names associations
46      infiles = applier.FilenameAssociations()
47      # Set image1 to the input image specified
```

```
48      infiles.image1 = args.input
49      # Create output files file names associations
50      outfiles = applier.FilenameAssociations()
51      # Set outImage to the output image specified
52      outfiles.outimage = args.output
53      # Create other arguments object
54      otherargs = applier.OtherInputs()
55      # Define the scale arguments
56      otherargs.scale = args.multiply
57      # Create a controls objects
58      aControls = applier.ApplierControls()
59      # Set the progress object.
60      aControls.progress = cuiprogress.CUIProgressBar()
61
62      # Apply the multiply function.
63      applier.apply(mutliplyByValue,
64                      infiles,
65                      outfiles,
66                      otherargs,
67                      controls=aControls)
68
```

Run the Script

Run the script using the following command, the input image is a Landsat scene and all the pixel values will be multiplied by 2.

```
python MultiplyRIOSExample.py -i LSTOA_Tanz_2000Wet.img \
        -o LSTOA_Tanz_2000Wet_Multiby2.img -m 2
```

8.2.4 Calculate NDVI

To use the image bands independently to calculate a new value, usually indices such as the NDVI

$$\text{NDVI} = \frac{\text{NIR} - \text{RED}}{\text{NIR} + \text{RED}} \tag{8.1}$$

requires that the bands are referenced independently within the input data. Using numpy to calculate the index, as shown below, results in a single output block with the dimensions of the block but does not have the third dimension (i.e., the band) which is required for RIOS to identify how to create the output image. Therefore, as you will see in the example below an extra dimension needs to be added before outputting the data to the file. Within the example given the input pixel values are converted to floating point values (rather than whatever they were inputted as from the input) because the output will be a floating point number (i.e., an NDVI have a range of −1 to 1).

```python
#!/usr/bin/env python

# Import the python Argument parser
import argparse
# Import the RIOS applier interface
from rios import applier
from rios import cuiprogress
#
import numpy

# Define the applier function
def mutliplyByValue(info, inputs, outputs, otherargs):
    # Convert the input data to Float32
    # This is because the output is a float due to the
    # divide within the NDVI calculation.
    inputs.image1 = inputs.image1.astype (numpy.float32)
    # Calculate the NDVI for the block.
    # Note. Numpy will deal with the image iterating
    #       to all the individual pixels values.
    #       within python this is very important
    #       as python loops are slow.
    out = ((inputs.image1[otherargs.nirband]-
            inputs.image1[otherargs.redband])
            /
           (inputs.image1[otherargs.nirband]+
           inputs.image1[otherargs.redband]))
    # Add an extra dimension to the output array.
    # The output array needs to have 3 dimensions
    # (No Bands, Y Pixels(Rows), X Pixels(Cols)
```

```python
30        # In this case an extra dimension representing
31        # the single image band is required.
32        outputs.outimage = numpy.expand_dims(out, axis=0)
33
34  # This is the first part of the script to
35  # be executed.
36  if __name__ == '__main__':
37        # Create the command line options
38        # parser.
39        parser = argparse.ArgumentParser()
40        # Define the argument for specifying the input file.
41        parser.add_argument("-i", "--input", type=str,
42                            help="Specify the input image file.")
43        # Define the argument for specifying the output file.
44        parser.add_argument("-o", "--output", type=str,
45                            help="Specify the output image file.")
46        # Define the argument for specifying the red image band
47        parser.add_argument("-r", "--red", type=int,
48                            help="Specifiy red band.")
49        # Define the argument for specifying the NIR image band
50        parser.add_argument("-n", "--nir", type=int,
51                            help="Specifiy NIR band.")
52        # Call the parser to parse the arguments.
53        args = parser.parse_args()
54
55        # Check that the input parameter has been specified.
56        if args.input == None:
57            # Print an error message if not and exit.
58            print("Error: No input image file provided.")
59            sys.exit()
60
61        # Check that the output parameter has been specified.
62        if args.output == None:
63            # Print an error message if not and exit.
64            print("Error: No output image file provided.")
65            sys.exit()
66
67        # Create input files file names associations
68        infiles = applier.FilenameAssociations()
69        # Set image1 to the input image specified
70        infiles.image1 = args.input
```

```
71      # Create output files file names associations
72      outfiles = applier.FilenameAssociations()
73      # Set outImage to the output image specified
74      outfiles.outimage = args.output
75      # Create other arguments object
76      otherargs = applier.OtherInputs()
77      # Define the red band argument
78      otherargs.redband = args.red-1
79      # Define the NIR band argument
80      otherargs.nirband = args.nir-1
81      # Create a controls objects
82      aControls = applier.ApplierControls()
83      # Set the progress object.
84      aControls.progress = cuiprogress.CUIProgressBar()
85
86      # Apply the multiply function.
87      applier.apply(mutliplyByValue,
88                    infiles,
89                    outfiles,
90                    otherargs,
91                    controls=aControls)
92
```

Run the Script

Run the script using the following command, the input image is a Landsat scene
so the red band is therefore band 3 and then NIR band is band 4.

```
python RIOSExampleNDVI.py -i LSTOA_Tanz_2000Wet.img \
       -o LSTOA_Tanz_2000Wet_NDVI.img -r 3 -n 4
```

8.2.5   Calculate NDVI Using Multiple Images

Where multiple input files are required, in this case the NIR and Red bands are
represented by different image files, the input files need to be specified in the input
files association as image1, image2 etc. and the pixel values within the applier

function are therefore referenced in the same way. Because, in this example the images only have a single image band the input images has the same dimensions as the output so no extra dimensions need to be added.

```python
#!/usr/bin/env python

# Import the system library
import sys
# Import the python Argument parser
import argparse
# Import the RIOS applier interface
from rios import applier
# Import the RIOS progress feedback
from rios import cuiprogress
# Import the numpy library
import numpy

# Define the applier function
def mutliplyByValue(info, inputs, outputs):
    # Convert the input data to Float32
    # This is because the output is a float due to the
    # divide within the NDVI calculation.
    inputs.image1 = inputs.image1.astype (numpy.float32)
    inputs.image2 = inputs.image2.astype (numpy.float32)
    # Calculate the NDVI for the block.
    # Note. Numpy will deal with the image iterating
    #       to all the individual pixels values.
    #       within python this is very important
    #       as python loops are slow.
    outputs.outimage = ((inputs.image2-inputs.image1)
                        /
                        (inputs.image2+inputs.image1))

# This is the first part of the script to
# be executed.
if __name__ == '__main__':
    # Create the command line options
    # parser.
    parser = argparse.ArgumentParser()
    # Define the argument for specifying the output file.
```

```
37    parser.add_argument("-o", "--output", type=str,
38                        help="Specify the output image file.")
39    # Define the argument for specifying the red image band
40    parser.add_argument("-r", "--red", type=str,
41                        help="Specifiy red input image file.")
42    # Define the argument for specifying the NIR image band
43    parser.add_argument("-n", "--nir", type=str,
44                        help="Specifiy NIR input image file.")
45    # Call the parser to parse the arguments.
46    args = parser.parse_args()
47
48    # Check that the red input parameter has been specified.
49    if args.red == None:
50        # Print an error message if not and exit.
51        print("Error: No red input image file provided.")
52        sys.exit()
53
54    # Check that the NIR input parameter has been specified.
55    if args.red == None:
56        # Print an error message if not and exit.
57        print("Error: No NIR input image file provided.")
58        sys.exit()
59
60    # Check that the output parameter has been specified.
61    if args.output == None:
62        # Print an error message if not and exit.
63        print("Error: No output image file provided.")
64        sys.exit()
65
66    # Create input files file names associations
67    infiles = applier.FilenameAssociations()
68    # Set images to the input image specified
69    infiles.image1 = args.red
70    infiles.image2 = args.nir
71    # Create output files file names associations
72    outfiles = applier.FilenameAssociations()
73    # Set outImage to the output image specified
74    outfiles.outimage = args.output
75    # Create a controls objects
76    aControls = applier.ApplierControls()
77    # Set the progress object.
```

```
78      aControls.progress = cuiprogress.CUIProgressBar()

79

80      # Apply the multiply function.
81      applier.apply(mutliplyByValue,
82                  infiles,
83                  outfiles,
84                  controls=aControls)

85
```

Run the Script

Run the script using the following command, the input image is a Landsat scene so the red band is therefore band 3 and then NIR band is band 4.

```
python RIOSExampleMultiFileNDVI.py -o LSTOA_Tanz_2000Wet_MultiIn_NDVI.img \
      -r LSTOA_Tanz_2000Wet_Red.img -n LSTOA_Tanz_2000Wet_NIR.img
```

8.3 Filtering Images

To filtering an image is done through a windowing operation where the windows of pixels, such as a 3×3 or 5×5 (it needs to be an odd number), are selected and a new value for the centre pixel is calculated using all the pixel values within the window. In this example a median filter will be used so the middle pixel value will be replaced with the median value of the window.

Scipy (`http://www.scipy.org`) is another library of python functions, which is paired with numpy, and provides many useful functions we can use when processing the images or other datasets within python. The ndimage module (`http://docs.scipy.org/doc/scipy/reference/tutorial/ndimage.html`) provides many useful functions, which can be applied to images in the same way as the median filter has been used in the example below – I strongly recommend you look through the documentation of scipy to get an idea of the types of functions which are available.

```python
1   #!/usr/bin/env python
2
3   import sys
4   # Import the python Argument parser
5   import argparse
6   # Import the scipy filters.
7   from scipy import ndimage
8   #Import the numpy library
9   import numpy
10  # Import the RIOS image reader
11  from rios.imagereader import ImageReader
12  # Import the RIOS image writer
13  from rios.imagewriter import ImageWriter
14
15  # Define the function to iterate through
16  # the image.
17  def applyMedianFilter(inputFile, outputFile, fSize):
18      # Get half the filter size, overlap between blocks
19      hSize = (fSize-1)/2
20      # Create the image reader for the input file
21      # and set the overlap to be half the image
22      # filter size.
23      reader = ImageReader(inputFile, overlap=hSize)
24      # Define the image writer but cannot create
25      # until within the loop as this need the
26      # information within the info object.
27      writer = None
28      # Loop through all the image blocks within
29      # the reader.
30      for (info, block) in reader:
31          # Create an output block of
32          # the same size as the input
33          out = numpy.zeros_like(block)
34          # Iterate through the image bands
35          for i in range(len(out)):
36              # Use scipy to run a median filter
37              # on the image band data. The image
38              # bands are filtered in turn
39              ndimage.median_filter(block[i],size=fSize,output=out[i])
40          # If it is the first time through the loop
41          # (i.e., writer has a value of None) then
```

```
42              # create the loop.
43              if writer is None:
44                  # Create the writer for output image.
45                  writer = ImageWriter(outputFile,
46                                       info=info,
47                                       firstblock=out,
48                                       drivername='HFA')
49              else:
50                  # If the writer is created write the
51                  # output block to the file.
52                  writer.write(out)
53          # Close the writer and calculate
54          # the image statistics.
55          writer.close(calcStats=True)
56
57      # This is the first part of the script to
58      # be executed.
59      if __name__ == '__main__':
60          # Create the command line options
61          # parser.
62          parser = argparse.ArgumentParser()
63          # Define the argument for specifying the input file.
64          parser.add_argument("-i", "--input", type=str,
65                              help="Specify the input image file.")
66          # Define the argument for specifying the output file.
67          parser.add_argument("-o", "--output", type=str,
68                              help="Specify the output image file.")
69          # Define the argument for the size of the image filter.
70          parser.add_argument("-s", "--size", default=3, type=int,
71                              help="Filter size.")
72          # Call the parser to parse the arguments.
73          args = parser.parse_args()
74
75          # Check that the input parameter has been specified.
76          if args.input == None:
77              # Print an error message if not and exit.
78              print("Error: No input image file provided.")
79              sys.exit()
80
81          # Check that the output parameter has been specified.
82          if args.output == None:
```

```
83          # Print an error message if not and exit.
84          print("Error: No output image file provided.")
85          sys.exit()
86
87      # Call the function to execute a median filter
88      # on the input image.
89      applyMedianFilter(args.input, args.output, args.size)
90
```

Run the Script

```
python MedianFilterRIOSExample.py -i LSTOA_Tanz_2000Wet.img \
                            -o LSTOA_Tanz_2000Wet_median7.img -s 7
```

After you have run this command open the images in TuiView and flick between them to observe the change in the image, what do you notice?

8.4 Apply a rule based classification

Another option we have is to use the 'where' function within numpy to select pixel corresponding to certain criteria (i.e., pixels with an NDVI < 0.2 is not vegetation) and classify them accordingly where a pixel values are used to indicate the corresponding class (e.g., 1 = Forest, 2 = Water, 3 = Grass, etc). These images where pixel values are not continuous but categories are referred to as thematic images and there is a header value that can be set to indicate this type of image. Therefore, in the script below there is a function for setting the image band metadata field 'LAYER_TYPE' to be 'thematic'. Setting an image as thematic means that the nearest neighbour algorithm will be used when calculating pyramids and histograms needs to be binned with single whole values. It also means that a colour table (See Chapter 9) can also be added.

To build the rule base the output pixel values need to be created, here using the numpy function zeros (`http://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html`). The function zeros creates a numpy array of the requested

shape (in this case the shape is taken from the inputted image) where all the pixels have a value of zero.

Using the 'where' function (`http://docs.scipy.org/doc/numpy/reference/generated/numpy.where.html`) a logic statement can be applied to an array or set of arrays (which must be of the same size) to select the pixels for which the statement is true. The where function returns an array of indexes which can be used to address another array (i.e., the output array) and set a suitable output value (i.e., the classification code).

```python
#!/usr/bin/env python

# Import the system library
import sys
# Import the python Argument parser
import argparse
# Import the RIOS applier interface
from rios import applier
# Import the RIOS progress feedback
from rios import cuiprogress
# Import the numpy library
import numpy
# Import the GDAL library
from osgeo import gdal

# Define the applier function
def rulebaseClassifier(info, inputs, outputs):
    # Create an output array with the same dims
    # as a single band of the input file.
    out = numpy.zeros(inputs.image1[0].shape)
    # Use where statements to select the
    # pixels to be classified. Give them a
    # integer value (i.e., 1, 2, 3, 4) to
    # specify the class.
    out[numpy.where((inputs.image1[0] > 0.4 )&(inputs.image1[0] < 0.7))] = 1
    out[numpy.where(inputs.image1[0] < 0.1 )] = 2
    out[numpy.where((inputs.image1[0] > 0.1 )&(inputs.image1[0] < 0.4))] = 3
    out[numpy.where(inputs.image1[0] > 0.7 )] = 4
    # Expand the output array to include a single
    # image band and set as the output dataset.
```

```
31          outputs.outimage = numpy.expand_dims(out, axis=0)
32
33      # A function to define the image as thematic
34      def setThematic(imageFile):
35          # Use GDAL to open the dataset
36          ds = gdal.Open(imageFile, gdal.GA_Update)
37          # Iterate through the image bands
38          for bandnum in range(ds.RasterCount):
39              # Get the image band
40              band = ds.GetRasterBand(bandnum + 1)
41              # Define the meta-data for the LAYER_TYPE
42              band.SetMetadataItem('LAYER_TYPE', 'thematic')
43
44      # This is the first part of the script to
45      # be executed.
46      if __name__ == '__main__':
47          # Create the command line options
48          # parser.
49          parser = argparse.ArgumentParser()
50          # Define the argument for specifying the input file.
51          parser.add_argument("-i", "--input", type=str,
52                              help="Specify the input image file.")
53          # Define the argument for specifying the output file.
54          parser.add_argument("-o", "--output", type=str,
55                              help="Specify the output image file.")
56          # Call the parser to parse the arguments.
57          args = parser.parse_args()
58
59          # Check that the input parameter has been specified.
60          if args.input == None:
61              # Print an error message if not and exit.
62              print("Error: No input image file provided.")
63              sys.exit()
64
65          # Check that the output parameter has been specified.
66          if args.output == None:
67              # Print an error message if not and exit.
68              print("Error: No output image file provided.")
69              sys.exit()
70
71          # Create input files file names associations
```

```
72      infiles = applier.FilenameAssociations()
73      # Set image1 to the input image specified
74      infiles.image1 = args.input
75      # Create output files file names associations
76      outfiles = applier.FilenameAssociations()
77      # Set outImage to the output image specified
78      outfiles.outimage = args.output
79      # Create a controls objects
80      aControls = applier.ApplierControls()
81      # Specify that stats shouldn't be calc'd
82      aControls.calcStats = False
83      # Set the progress object.
84      aControls.progress = cuiprogress.CUIProgressBar()
85
86      # Apply the classifier function.
87      applier.apply(rulebaseClassifier,
88                    infiles,
89                    outfiles,
90                    controls=aControls)
91
92      # Set the output file to be thematic
93      setThematic(args.output)
```

Run the Script

Run the script with one of the NDVI layers you previously calculated. To see the result then it is recommended that a colour table is added (see next worksheet), the easiest way to do that is to use the gdalcalcstats command, as shown below.

```
python RuleBaseClassification.py -i LSTOA_Tanz_2000Wet_NDVI.img \
                                 -o LSTOA_Tanz_2000Wet_classification.img
# Run gdalcalcstats to add a random colour table
gdalcalcstats LSTOA_Tanz_2000Wet_classification.img
```

8.5 Exercises

1. Create rule based classification using multiple image bands.

2. Create a rule based classification using image bands from different input images.

3. Using the previous work sheet as a basis create a script which calls the gdalwarp command to resample an input image to the same pixel resolution as another image, where the header is read as shown in this work sheet.

8.6 Further Reading

- GDAL - `http://www.gdal.org`

- Python Documentation - `http://www.python.org/doc`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

- Learn UNIX in 10 minutes - `http://freeengineer.org/learnUNIXin10minutes.html`

- SciPy – `http://www.scipy.org/SciPy`

- NumPy – `http://numpy.scipy.org`

- RIOS – `https://bitbucket.org/chchrsc/rios/wiki/Home`

Chapter 9

Raster Attribute Tables (RAT)

The RIOS software also allows raster attribute tables to be read and written through GDAL. Raster attribute tables (RAT) are similar the the attribute tables which are present on a vector (e.g., shapefile). Each row of the attribute table refers to a pixel value within the image (e.g., row 0 refers to all pixels with a value of 0). Therefore, RATs are used within thematic datasets were pixels values are integers and refer to a category, such as a class from a classification, or a spatial region, such as a segment from a segmentation. The columns of the RAT therefore refer to variables, which correspond to information associated with the spatial region cover by the image pixels of the clump(s) relating to the row within the attribute table.

9.1   Reading Columns

To access the RAT using RIOS, you need to import the rat module. The RAT module provides a simple interface for reading and writing columns. When a column is read it is returned as a numpy array where the size is $n \times 1$ (i.e., the number of rows in the attribute table).

As shown in the example below, a reading a column is just a single function call specifying the input image file and the column name.

```python
#!/usr/bin/env python

# Import the system library
import sys
# Import the RIOS rat library.
from rios import rat
# Import the python Argument parser
import argparse

# A function for reading the RAT
def readRatCol(imageFile, colName):
    # Use RIOS to read the column name
    # The contents of the column are
    # printed to the console for the
    # user to see.
    print(rat.readColumn(imageFile, colName))

# This is the first part of the script to
# be executed.
if __name__ == '__main__':
    # Create the command line options
    # parser.
    parser = argparse.ArgumentParser()
    # Define the argument for specifying the input file.
    parser.add_argument("-i", "--input", type=str,
                        help="Specify the input image file.")
    # Define the argument for specifying the column name.
    parser.add_argument("-n", "--name", type=str,
                        help="Specify the column name.")
    # Call the parser to parse the arguments.
    args = parser.parse_args()

    # Check that the input parameter has been specified.
    if args.input == None:
        # Print an error message if not and exit.
        print("Error: No input image file provided.")
        sys.exit()
    # Check that the input parameter has been specified.
    if args.name == None:
        # Print an error message if not and exit.
        print("Error: No RAT column name provided.")
```

```
42        sys.exit()

43

44    # Run the function read and print the
45    # RAT column.
46    readRatCol(args.input, args.name)
```

Run the Script

Run the script as follow, the example below prints the Histogram but use TuiView to see what other columns are within the attribute table.

```
python ReadRATColumn.py -i WV2_525N040W_2m_segments.kea -n Histogram
```

9.2 Writing Columns

Writing a column is also quite straight forward just requiring a $n \times 1$ numpy array with the the data to be written to the output file, the image file path and the name of the column to be written to.

9.2.1 Calculating New Columns

The first example reads a column from the input image and just multiples it by 2 and writes it to the image file as a new column.

```
1   #!/usr/bin/env python

2

3   # Import the system library
4   import sys
5   # Import the RIOS rat library.
6   from rios import rat
7   # Import the python Argument parser
8   import argparse

9

10  # The applier function to multiply the input
```

```
11   # column by 2.
12   def multiplyRATCol(imageFile, inColName, outColName):
13       # Read the input column
14       col = rat.readColumn(imageFile, inColName)
15       # Muliply the column by 2.
16       col = col * 2
17       # Write the output column to the file.
18       rat.writeColumn(imageFile, outColName, col)
19
20   # This is the first part of the script to
21   # be executed.
22   if __name__ == '__main__':
23       # Create the command line options
24       # parser.
25       parser = argparse.ArgumentParser()
26       # Define the argument for specifying the input file.
27       parser.add_argument("-i", "--input", type=str,
28                           help="Specify the input image file.")
29       # Define the argument for specifying the input column name.
30       parser.add_argument("-c", "--inname", type=str,
31                           help="Specify the input column name.")
32       # Define the argument for specifying the output column name.
33       parser.add_argument("-o", "--outname", type=str,
34                           help="Specify the output column name.")
35       # Call the parser to parse the arguments.
36       args = parser.parse_args()
37
38       # Check that the input parameter has been specified.
39       if args.input == None:
40           # Print an error message if not and exit.
41           print("Error: No input image file provided.")
42           sys.exit()
43       # Check that the input parameter has been specified.
44       if args.inname == None:
45           # Print an error message if not and exit.
46           print("Error: No input RAT column name provided.")
47           sys.exit()
48       # Check that the input parameter has been specified.
49       if args.outname == None:
50           # Print an error message if not and exit.
51           print("Error: No output RAT column name provided.")
```

```
52        sys.exit()
53
54    # Otherwise, run the function to multiply the input column by 2.
55    multiplyRATCol(args.input, args.inname, args.outname)
```

Run the Script

Run the script as follows, in this simple case the histogram will be multiplied by 2 and saved as a new column.

```
python MultiplyColumn.py -i WV2_525N040W_2m_segments.kea -c Histogram -o HistoMulti2
```

9.2.2 Add Class Name

A useful column to have within the attribute table, where a classification has been undertaken, is class names. This allows a user to click on the image and rather than having to remember which codes correspond to which class they will be shown a class name.

To add class names to the attribute table a new column needs to be created, where the data type is set to be ASCII (string). To do this a copy of the histogram column is made where the new numpy array is empty, of type string and the same length at the histogram.

The following line using the ... syntax within the array index to specify all elements of the array, such that they are all set to a value of "NA".

Once the new column has been created then the class names can be simply defined through referencing the appropriate array index.

```
1    #!/usr/bin/env python
2
3    # Import the system library
4    import sys
5    # Import the RIOS rat library.
6    from rios import rat
```

```python
7   # Import the python Argument parser
8   import argparse
9   # Import the numpy library
10  import numpy
11
12  # A function to add a colour table.
13  def addClassNames(imageFile):
14      histo = rat.readColumn(imageFile, "Histogram")
15      className = numpy.empty_like(histo, dtype=numpy.dtype('a255'))
16      className[...] = "NA"
17      className[0] = "Other Vegetation"
18      className[1] = "Low Woody Vegetation"
19      className[2] = "Water"
20      className[3] = "Sparse Vegetation"
21      className[4] = "Tall Woody Vegetation"
22      # Write the output column to the file.
23      rat.writeColumn(imageFile, "ClassNames", className)
24
25  # This is the first part of the script to
26  # be executed.
27  if __name__ == '__main__':
28      # Create the command line options
29      # parser.
30      parser = argparse.ArgumentParser()
31      # Define the argument for specifying the input file.
32      parser.add_argument("-i", "--input", type=str,
33                          help="Specify the input image file.")
34      # Call the parser to parse the arguments.
35      args = parser.parse_args()
36
37      # Check that the input parameter has been specified.
38      if args.input == None:
39          # Print an error message if not and exit.
40          print("Error: No input image file provided.")
41          sys.exit()
42
43      # Run the add class names function
44      addClassNames(args.input)
```

Run the Script

Run the script as follows, use the classification you did at the end of worksheet 8.

```
python AddClassNames.py -i LSTOA_Tanz_2000Wet_classification.img
```

9.3 Adding a colour table

Another useful tool is being able to add a colour table to an image, such that classes are displayed in colours appropriate to make interpretation easier. To colour up the per pixel classification undertake at the end of the previous exercise and given class names using the previous scripts the following script is used to add a colour table.

The colour table is represented as an $n \times 5$ dimensional array, where n is the number of colours which are to be present within the colour table.

The 5 values associated with each colour are

1. Image Pixel Value

2. Red $(0 - 255)$

3. Green $(0 - 255)$

4. Blue $(0 - 255)$

5. Opacity $(0 - 255)$

Where an opacity of 0 means completely transparent and 255 means solid with no transparency (opacity is something also referred to as alpha or alpha channel).

```python
#!/usr/bin/env python

# Import the system library
import sys
```

```python
5   # Import the RIOS rat library.
6   from rios import rat
7   # Import the python Argument parser
8   import argparse
9   # Import the numpy library
10  import numpy
11
12  # A function to add a colour table.
13  def addColourTable(imageFile):
14      # Create a colour table (n,5) where
15      # n is the number of classes to be
16      # coloured. The data type must be
17      # of type integer.
18      ct = numpy.zeros([5,5], dtype=numpy.int)
19
20      # Set 0 to be Dark Mustard Yellow.
21      ct[0][0] = 0    # Pixel Val
22      ct[0][1] = 139  # Red
23      ct[0][2] = 139  # Green
24      ct[0][3] = 0    # Blue
25      ct[0][4] = 255  # Opacity
26
27      # Set 1 to be Dark Olive Green.
28      ct[1][0] = 1    # Pixel Val
29      ct[1][1] = 162  # Red
30      ct[1][2] = 205  # Green
31      ct[1][3] = 90   # Blue
32      ct[1][4] = 255  # Opacity
33
34      # Set 2 to be Royal Blue.
35      ct[2][0] = 2    # Pixel Val
36      ct[2][1] = 72   # Red
37      ct[2][2] = 118  # Green
38      ct[2][3] = 255  # Blue
39      ct[2][4] = 255  # Opacity
40
41      # Set 3 to be Dark Sea Green.
42      ct[3][0] = 3    # Pixel Val
43      ct[3][1] = 180  # Red
44      ct[3][2] = 238  # Green
45      ct[3][3] = 180  # Blue
```

```
46        ct[3][4] = 255 # Opacity
47
48        # Set 4 to be Forest Green.
49        ct[4][0] = 4    # Pixel Val
50        ct[4][1] = 34 # Red
51        ct[4][2] = 139 # Green
52        ct[4][3] = 34   # Blue
53        ct[4][4] = 255 # Opacity
54
55        rat.setColorTable(imageFile, ct)
56
57
58
59  # This is the first part of the script to
60  # be executed.
61  if __name__ == '__main__':
62      # Create the command line options
63      # parser.
64      parser = argparse.ArgumentParser()
65      # Define the argument for specifying the input file.
66      parser.add_argument("-i", "--input", type=str,
67                          help="Specify the input image file.")
68      # Call the parser to parse the arguments.
69      args = parser.parse_args()
70
71      # Check that the input parameter has been specified.
72      if args.input == None:
73          # Print an error message if not and exit.
74          print("Error: No input image file provided.")
75          sys.exit()
76
77      # Run the add colour table function
78      addColourTable(args.input)
```

Run the Script

Run the script as follows, use the classification you did at the end of work-sheet 8.

```
python AddClassNames.py -i LSTOA_Tanz_2000Wet_classification.img
```

To find the Red, Green and Blue (RGB) values to use with the colour table there are many websites available only that provide lists of these colours (e.g., `http://cloford.com/resources/colours/500col.htm`).

9.4 Using RATs for rule based classifications.

To use a RAT to undertake a rule based object oriented classification the first step is to create a set of image clumps (e.g., through segmentation see appendix A section A.3), then the rows of the attribute table need populating with information (e.g., see appendix A section A.4). Once these steps have been completed then a rule base using the numpy where statements can be created and executed, resulting in a similar process as the eCognition software.

9.4.1 Developing a rule base

This is a similar process to developing a rule based classification within eCognition, where the clumps are the segments/objects and the columns are the features, such as mean, standard deviation etc.

Using 'where' statements, similar to those used within the rule based classification of the image pixels, the clumps can be classified. The example, shown below, illustrates the classification of the Landcover classification system (LCCS) levels 1 to 3. Where the classes are represented by string names within the class names column within the attribute table.

The classification is undertaken using three dates of WorldView2 imagery captured over Cors Fochno, in Wales UK. A segmentation has been provided and the segments have been populated with mean reflectance values from the three WorldView2 images, the DTM minimum and maximum and the CHM height.

```
1   #!/usr/bin/env python
2
```

```python
3   # Import the system library
4   import sys
5   # Import the rat modele from RIOS
6   from rios import rat
7   # Import the numpy library
8   import numpy
9   # Import the gdal library
10  import osgeo.gdal as gdal
11  # Import the python Argument parser
12  import argparse
13
14  # Thresholds which have been globally defined
15  # so they can easily be changed and therefore
16  # change everywhere they are used.
17  URBAN_MASK_THRES = 0.05
18  CULT_MASK_THRES = 0.05
19  WBI_PRE_THRES = 1
20  WBI_PEAK_THRES = 1
21  WBI_POST_THRES = 1
22  FDI_PRE_THRES = -100
23  FDI_PEAK_THRES = -40
24  FDI_POST_THRES = -115
25  PSRI_PRE_THRES = -0.2
26  REP_PEAK_THRES = -0.005
27  WOODY_PRE_THRES_BG = 0.09
28  WOODY_PEAK_THRES_CG = 0.195
29
30  # A function for classifying the first part of level 1
31  def classifyLevel1FromImg(urbanMask, wbiPeak, fdiPeak, wbiPost,
32                            fdiPost, wbiPre, fdiPre, psriPre, repPeak):
33      # Create Output Array
34      l1P1 = numpy.empty_like(urbanMask, dtype=numpy.dtype('a255'))
35      l1P1[...] = "NA"
36      # Urban
37      l1P1 = numpy.where(numpy.logical_and(l1P1 == "NA", urbanMask > URBAN_MASK_THRES),
38                         "Urban", l1P1)
39      # Water
40      l1P1 = numpy.where(numpy.logical_and(l1P1 == "NA",
41                         numpy.logical_or(wbiPre >= WBI_PRE_THRES,
42                                          wbiPeak >= WBI_PEAK_THRES)),
43                         "Water", l1P1)
```

```
44        # Photosynthetic Vegetation
45        l1P1 = numpy.where(numpy.logical_and(l1P1 == "NA",
46                           numpy.logical_or(fdiPeak > FDI_PEAK_THRES,
47                                            fdiPost > FDI_POST_THRES)),
48                           "Photosynthetic Vegetated", l1P1)
49        # Non PhotoSynthetic Vegetation
50        l1P1 = numpy.where(numpy.logical_and(l1P1 == "NA",
51                                             psriPre >= PSRI_PRE_THRES),
52                       "Non Photosynthetic Vegetated", l1P1)
53        # Non Submerged Aquatic Veg
54        l1P1 = numpy.where(numpy.logical_and(l1P1 == "NA",
55                           numpy.logical_and(repPeak >= REP_PEAK_THRES,
56                                             wbiPost <= WBI_POST_THRES)),
57                           "Non Submerged Aquatic Vegetated", l1P1)
58        return l1P1
59
60    # A function for classifying the second part of level 1
61    def classifyLevel1Assign(classLevel1Img):
62        # Create Output Array
63        level1 = numpy.empty_like(classLevel1Img, dtype=numpy.dtype('a255'))
64        level1[...] = "NA"
65        # Non Vegetated
66        level1 = numpy.where(numpy.logical_or(classLevel1Img == "NA",
67                             numpy.logical_or(classLevel1Img == "Water",
68                                              classLevel1Img == "Urban")),
69                             "Non Vegetated", level1)
70        # Vegetated
71        level1 = numpy.where(numpy.logical_or(
72                             classLevel1Img == "Photosynthetic Vegetated",
73                             classLevel1Img == "Non Photosynthetic Vegetated",
74                             classLevel1Img == "Non Submerged Aquatic Vegetated"),
75                             "Vegetated", level1)
76        return level1
77
78    # A function for classifying level 2
79    def classifyLevel2(wbiPre, wbiPeak, wbiPost, classLevel1Img):
80        # Create Output Array
81        level2 = numpy.empty_like(classLevel1Img, dtype=numpy.dtype('a255'))
82        level2[...] = "NA"
83
84        # Terrestrial Non Vegetated
```

```
85      level2 = numpy.where(numpy.logical_or(classLevel1Img == "NA",
86                                         classLevel1Img == "Urban"),
87                         "Terrestrial Non Vegetated", level2)
88      # Aquatic Non Vegetated
89      level2 = numpy.where(numpy.logical_and(
90                         numpy.logical_not(classLevel1Img == "Urban"),
91                         numpy.logical_or(wbiPre > 1, wbiPeak > 1)),
92                         "Aquatic Non Vegetated", level2)
93      # Terrestrial Vegetated
94      level2 = numpy.where(numpy.logical_or(classLevel1Img == "Photosynthetic Vegetated",
95                         classLevel1Img == "Non Photosynthetic Vegetated"),
96                         "Terrestrial Vegetated", level2)
97      # Aquatic Vegetated
98      level2 = numpy.where(classLevel1Img == "Non Submerged Aquatic Vegetated",
99                         "Aquatic Vegetated", level2)
100     return level2
101
102 # A function for classifying level 3
103 def classifyLevel3(classLevel2, cult, urban):
104     # Create Output Array
105     level3 = numpy.empty_like(classLevel2, dtype=numpy.dtype('a255'))
106     level3[...] = "NA"
107     # Cultivated Terrestrial Vegetated
108     level3 = numpy.where(numpy.logical_and(
109                         classLevel2 == "Terrestrial Vegetated", cult > CULT_MASK_THRES),
110                         "Cultivated Terrestrial Vegetated", level3)
111     # Natural Terrestrial Vegetated
112     level3 = numpy.where(numpy.logical_and(numpy.logical_not
113                         (level3 == "Cultivated Terrestrial Vegetated"),
114                         classLevel2 == "Terrestrial Vegetated"),
115                         "Natural Terrestrial Vegetated", level3)
116     # Cultivated Aquatic Vegetated
117     level3 = numpy.where(numpy.logical_and(classLevel2 == "Aquatic Vegetated",
118                         cult > CULT_MASK_THRES), "Cultivated Aquatic Vegetated", level3)
119     # Natural Aquatic Vegetated
120     level3 = numpy.where(numpy.logical_and(numpy.logical_not
121                         (level3 == "Cultivated Aquatic Vegetated"),
122                         classLevel2 == "Aquatic Vegetated"),
123                         "Natural Aquatic Vegetated", level3)
124     # Artificial Surface
125     level3 = numpy.where(numpy.logical_and(classLevel2 == "Terrestrial Non Vegetated",
```

```
126                                  urban > URBAN_MASK_THRES), "Artificial Surface", level3)
127         # Natural Surface
128         level3 = numpy.where(numpy.logical_and(numpy.logical_not
129                         (level3 == "Artificial Surface"),
130                         classLevel2 == "Terrestrial Non Vegetated"),
131                         "Natural Surface", level3)
132         # Natural Water
133         level3 = numpy.where(classLevel2 == "Aquatic Non Vegetated",
134                         "Natural Water", level3)
135         return level3
136
137     def runClassification(fname):
138         # Open the GDAL Dataset so it is just opened once
139         # and reused rather than each rios call reopening
140         # the image file which will large attribute tables
141         # can be slow.
142         ratDataset = gdal.Open( fname, gdal.GA_Update )
143         # Check the image file was openned correctly.
144         if not ratDataset == None:
145             # Provide feedback to the user.
146             print("Import Columns.")
147             urban = rat.readColumn(ratDataset, "PropUrban")
148             cult = rat.readColumn(ratDataset, "PropCult")
149
150             # Read in the RAT columns for the Pre-Flush image
151             PreCoastal = rat.readColumn(ratDataset, "MarB1")
152             PreBlue = rat.readColumn(ratDataset, "MarB2")
153             PreRed = rat.readColumn(ratDataset, "MarB5")
154             PreRedEdge = rat.readColumn(ratDataset, "MarB6")
155             PreNIR1 = rat.readColumn(ratDataset, "MarB7")
156
157             # Read in the RAT columns for the Peak-flush image.
158             PeakCoastal = rat.readColumn(ratDataset, "JulyB1")
159             PeakBlue = rat.readColumn(ratDataset, "JulyB2")
160             PeakRed = rat.readColumn(ratDataset, "JulyB5")
161             PeakRedEdge = rat.readColumn(ratDataset, "JulyB6")
162             PeakNIR1 = rat.readColumn(ratDataset, "JulyB7")
163             PeakNIR2 = rat.readColumn(ratDataset, "JulyB8")
164
165             # Read in the RAT columns for the Post-flush image.
166             PostCoastal = rat.readColumn(ratDataset, "NovB1")
```

```
167            PostBlue = rat.readColumn(ratDataset, "NovB2")
168            PostRedEdge = rat.readColumn(ratDataset, "NovB6")
169            PostNIR1 = rat.readColumn(ratDataset, "NovB7")
170
171            # Provide more feedback to the user.
172            print("Calculate Indices.")
173            # As all the columns are numpy arrays then
174            # we can do numpy arithmatic between the
175            # arrays to calculate new arrays, such as
176            # indices.
177            wbiPre = PreBlue/PreNIR1
178            fdiPre = PreNIR1 - (PreRedEdge + PreCoastal)
179            psriPre = (PreRed - PreBlue)/PreRedEdge
180
181            wbiPeak = PeakBlue/PeakNIR1
182            fdiPeak = PeakNIR1 - (PeakRedEdge + PeakCoastal)
183            repPeak = PeakRedEdge - (PeakNIR2 - PeakRed)
184
185            wbiPost = PostBlue/PostNIR1
186            fdiPost = PostNIR1 - (PostRedEdge + PostCoastal)
187
188            # Call the function which classifies the first part
189                    # of the level 1 classification
190            print("Classifying Level 1")
191            classLevel1Img = classifyLevel1FromImg(urban,
192                                                    wbiPeak,
193                                                    fdiPeak,
194                                                    wbiPost,
195                                                    fdiPost,
196                                                    wbiPre,
197                                                    fdiPre,
198                                                    psriPre,
199                                                    repPeak)
200            # Write the first part of the level 1 classification
201            # back into the input file.
202            rat.writeColumn(ratDataset, "ClassLevel1Part1", classLevel1Img)
203            # Call function a produce the level 1 classification
204            classLevel1 = classifyLevel1Assign(classLevel1Img)
205            # Write the level 1 classification to the image
206            rat.writeColumn(ratDataset, "ClassLevel1", classLevel1)
207
```

```
208            # Call the function which classifies the level 2 of the classification.
209            print("Classifying Level 2")
210            classLevel2 = classifyLevel2(wbiPre, wbiPeak, wbiPost, classLevel1Img)
211            # Write the level 2 classification to the image.
212            rat.writeColumn(ratDataset, "ClassLevel2", classLevel2)
213
214            # Call the function which classifies level 3 of the classification
215            print("Classifying Level 3")
216            classLevel3 = classifyLevel3(classLevel2, cult, urban)
217            #  Write the level 3 classification to the image.
218            rat.writeColumn(ratDataset, "ClassLevel3", classLevel3)
219        else:
220            print("Image could not be openned")
221
222    # This is the first part of the script to
223    # be executed.
224    if __name__ == '__main__':
225        # Create the command line options
226        # parser.
227        parser = argparse.ArgumentParser()
228        # Define the argument for specifying the input file.
229        parser.add_argument("-i", "--input", type=str,
230                            help="Specify the input image file.")
231        # Call the parser to parse the arguments.
232        args = parser.parse_args()
233
234        # Check that the input parameter has been specified.
235        if args.input == None:
236            # Print an error message if not and exit.
237            print("Error: No input image file provided.")
238            sys.exit()
239
240        # Run the classification
241        runClassification(args.input)
```

Run the Classification

To run the classification use the following command:

```
python LCCS_L13_Classification.py -i WV2_525N040W_2m_segments.kea
```

Colour the classification

Following the classification, the clusters need to be coloured and the script for this shown below. The previous example of adding a colour table is not suited to this case as colours are being applied to the individual segments based on their class allocation.

```python
1   #!/usr/bin/env python
2
3   # Import the system library
4   import sys
5   # import the rat module from rios
6   from rios import rat
7   # import numpy
8   import numpy
9   # import gdal
10  import osgeo.gdal as gdal
11  # Import the python Argument parser
12  import argparse
13
14  # A function for
15  def colourLevel3(classLevel3):
16      # Create the empty output arrays and set them
17      # so they all have a value of 0 other than
18      # opacity which is 255 to create solid colours
19      level3red = numpy.empty_like(classLevel3, dtype=numpy.int)
20      level3red[...] = 0
21      level3green = numpy.empty_like(classLevel3, dtype=numpy.int)
22      level3green[...] = 0
23      level3blue = numpy.empty_like(classLevel3, dtype=numpy.int)
24      level3blue[...] = 0
25      level3alpha = numpy.empty_like(classLevel3, dtype=numpy.int)
26      level3alpha[...] = 255
27
28      # For segmentation of class NA set them to be black
29      level3red = numpy.where(classLevel3 == "NA", 0, level3red)
```

```python
30    level3green = numpy.where(classLevel3 == "NA", 0, level3green)
31    level3blue = numpy.where(classLevel3 == "NA", 0, level3blue)
32    level3alpha = numpy.where(classLevel3 == "NA", 255, level3alpha)
33
34    # Colour Cultivated Terrestrial Vegetated
35    level3red = numpy.where(classLevel3 == "Cultivated Terrestrial Vegetated",
36                            192, level3red)
37    level3green = numpy.where(classLevel3 == "Cultivated Terrestrial Vegetated",
38                            255, level3green)
39    level3blue = numpy.where(classLevel3 == "Cultivated Terrestrial Vegetated",
40                            0, level3blue)
41    level3alpha = numpy.where(classLevel3 == "Cultivated Terrestrial Vegetated",
42                            255, level3alpha)
43
44    # Colour Natural Terrestrial Vegetated
45    level3red = numpy.where(classLevel3 == "Natural Terrestrial Vegetated",
46                            0, level3red)
47    level3green = numpy.where(classLevel3 == "Natural Terrestrial Vegetated",
48                            128, level3green)
49    level3blue = numpy.where(classLevel3 == "Natural Terrestrial Vegetated",
50                            0, level3blue)
51    level3alpha = numpy.where(classLevel3 == "Natural Terrestrial Vegetated",
52                            255, level3alpha)
53
54    # Colour Cultivated Aquatic Vegetated
55    level3red = numpy.where(classLevel3 == "Cultivated Aquatic Vegetated",
56                            0, level3red)
57    level3green = numpy.where(classLevel3 == "Cultivated Aquatic Vegetated",
58                            255, level3green)
59    level3blue = numpy.where(classLevel3 == "Cultivated Aquatic Vegetated",
60                            255, level3blue)
61    level3alpha = numpy.where(classLevel3 == "Cultivated Aquatic Vegetated",
62                            255, level3alpha)
63
64    # Colour Natural Aquatic Vegetated
65    level3red = numpy.where(classLevel3 == "Natural Aquatic Vegetated",
66                            0, level3red)
67    level3green = numpy.where(classLevel3 == "Natural Aquatic Vegetated",
68                            192, level3green)
69    level3blue = numpy.where(classLevel3 == "Natural Aquatic Vegetated",
70                            122, level3blue)
```

```python
71    level3alpha = numpy.where(classLevel3 == "Natural Aquatic Vegetated",
72                               255, level3alpha)
73
74    # Colour Artificial Surface
75    level3red = numpy.where(classLevel3 == "Artificial Surface",
76                             255, level3red)
77    level3green = numpy.where(classLevel3 == "Artificial Surface",
78                               0, level3green)
79    level3blue = numpy.where(classLevel3 == "Artificial Surface",
80                              255, level3blue)
81    level3alpha = numpy.where(classLevel3 == "Artificial Surface",
82                               255, level3alpha)
83
84    # Colour Natural Surface
85    level3red = numpy.where(classLevel3 == "Natural Surface",
86                             255, level3red)
87    level3green = numpy.where(classLevel3 == "Natural Surface",
88                               192, level3green)
89    level3blue = numpy.where(classLevel3 == "Natural Surface",
90                              160, level3blue)
91    level3alpha = numpy.where(classLevel3 == "Natural Surface",
92                               255, level3alpha)
93
94    # Colour Artificial Water
95    level3red = numpy.where(classLevel3 == "Artificial Water",
96                             0, level3red)
97    level3green = numpy.where(classLevel3 == "Artificial Water",
98                               0, level3green)
99    level3blue = numpy.where(classLevel3 == "Artificial Water",
100                             255, level3blue)
101   level3alpha = numpy.where(classLevel3 == "Artificial Water",
102                              255, level3alpha)
103
104   # Colour Natural Water
105   level3red = numpy.where(classLevel3 == "Natural Water",
106                            0, level3red)
107   level3green = numpy.where(classLevel3 == "Natural Water",
108                              0, level3green)
109   level3blue = numpy.where(classLevel3 == "Natural Water",
110                             255, level3blue)
111   level3alpha = numpy.where(classLevel3 == "Natural Water",
```

```python
112                               255, level3alpha)
113
114        return level3red, level3green, level3blue, level3alpha
115
116    # This is the first part of the script to
117    # be executed.
118    if __name__ == '__main__':
119        # Create the command line options parser.
120        parser = argparse.ArgumentParser()
121        # Define the argument for specifying the input file.
122        parser.add_argument("-i", "--input", type=str,
123                            help="Specify the input image file.")
124        # Call the parser to parse the arguments.
125        args = parser.parse_args()
126
127        # Check that the input parameter has been specified.
128        if args.input == None:
129            # Print an error message if not and exit.
130            print("Error: No input image file provided.")
131            sys.exit()
132
133        # Open the input file using GDAL
134        ratDataset = gdal.Open(args.input, gdal.GA_Update)
135
136        # Check that is openned correctly
137        if not ratDataset == None:
138            # Print some user feedback
139            print("Import Columns.")
140            # Read the classification column
141            level3 = rat.readColumn(ratDataset, "ClassLevel3")
142
143            # Print some user feedback
144            print("Classifying Level 3")
145            # Call function to assign colours to arrays
146            level3red, level3green, level3blue, level3alpha = colourLevel3(level3)
147            # Write the values to the Output Columns
148            rat.writeColumn(ratDataset, "Red", level3red)
149            rat.writeColumn(ratDataset, "Green", level3green)
150            rat.writeColumn(ratDataset, "Blue", level3blue)
151            rat.writeColumn(ratDataset, "Alpha", level3alpha)
152        else:
```

```
153        # Print an error message to the user if the image
154        # file could not be openned.
155        print("Input Image could not be opened")
```

Run the script using the following command.

```
python LCCS_L13_ColourClassification.py -i WV2_525N040W_2m_segments.kea
```

9.5   Exercises

9.6   Further Reading

- GDAL - `http://www.gdal.org`

- Python Documentation - `http://www.python.org/doc`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

- Learn UNIX in 10 minutes - `http://freeengineer.org/learnUNIXin10minutes.html`

- SciPy – `http://www.scipy.org/SciPy`

- NumPy – `http://numpy.scipy.org`

- RIOS – `https://bitbucket.org/chchrsc/rios/wiki/Home`

Chapter 10

Golden Plover Population Model

10.1 Introduction

The aim of this work sheet is to develop a populate model for a bird, called the Golden Plover.

10.2 Model Output

The model is required to output the total population of the birds for each year and the number of bird, eggs, fledgling and the number of fledglings which are a year old. Providing an option to export the results as a plot should also be provided.

10.3 Reading Parameters

To allow a user to parameterise the model a parameter card, such as the one shown below, needs to be provided.

```
1   numOfYears=20
2   initalAdultPairPop=15
3   winterSurvivalRate=0.66
4   averageEggsPerPair=3.64
5   averageFledgelingsPerPair=3.2
6   predatorControl=False
7   numOfFledgelings=14
8   numOfFledgelingsYearOld=8
9   fledgelingsSurvivePredatorsCtrl=0.75
10  fledgelingsSurvivePredatorsNoCtrl=0.18
```

```python
1   #!/usr/bin/env python
2
3   # Import the system library
4   import sys
5   # Import the python Argument parser
6   import argparse
7   # Import the maths library
8   import math as math
9
10  # A class for the golden plover population model
11  class GoldenPloverPopModel (object):
12
13      # A function to parse the input parameters file.
14      def parseParameterFile(self, inputFile):
15          # A string to store the input parameters to
16          # be outputted in the output file.
17          paramsStr = "## Input Parameters to the model.\n"
18          # Open the input parameter file.
19          parameterFile = open(inputFile, 'r')
20          # Create a dictionary object to store the
21          # input parameters.
22          params = dict()
23          # Loop through each line of the input
24          # text file.
25          for line in parameterFile:
26              # Strip any white space either
27              # side of the text.
28              line = line.strip()
29              # Add the line to the output
30              # parameters file.
```

```
31              paramsStr += "# " + line + "\n"
32              # Split the line on the '=' symbol
33              paramVals = line.split("=", 1)
34              # Find the known parameters and
35              # convert the input string to the
36              # correct data type (i.e., float or int).
37              if paramVals[0] == "numOfYears":
38                  params[paramVals[0]] = int(paramVals[1])
39              elif paramVals[0] == "initalAdultPairPop":
40                  params[paramVals[0]] = int(paramVals[1])
41              elif paramVals[0] == "winterSurvivalRate":
42                  params[paramVals[0]] = float(paramVals[1])
43              elif paramVals[0] == "averageEggsPerPair":
44                  params[paramVals[0]] = float(paramVals[1])
45              elif paramVals[0] == "averageFledgelingsPerPair":
46                  params[paramVals[0]] = float(paramVals[1])
47              elif paramVals[0] == "predatorControl":
48                  if paramVals[1].lower() == "false":
49                      params[paramVals[0]] = False
50                  elif paramVals[1].lower() == "true":
51                      params[paramVals[0]] = True
52                  else:
53                      print("predatorControl must be either True or False.")
54                      sys.exit()
55              elif paramVals[0] == "numOfFledgelings":
56                  params[paramVals[0]] = int(paramVals[1])
57              elif paramVals[0] == "numOfFledgelingsYearOld":
58                  params[paramVals[0]] = int(paramVals[1])
59              elif paramVals[0] == "fledgelingsSurvivePredatorsCtrl":
60                  params[paramVals[0]] = float(paramVals[1])
61              elif paramVals[0] == "fledgelingsSurvivePredatorsNoCtrl":
62                  params[paramVals[0]] = float(paramVals[1])
63              else:
64                  # If parameter is not known then just store as
65                  # a string.
66                  params[paramVals[0]] = paramVals[1]
67      # Return the parameters and parameters string
68      return params, paramsStr
69
70  # The run function controlling the overall order
71  # of when things run.
```

```
72      def run(self, inputFile):
73          # Provide user feedback to the user.
74          print("Parse Input File.")
75          # Call the function to parse the input file.
76          params, paramsStr = self.parseParameterFile(inputFile)
77          # Print he parameters.
78          print(params)
79
80
81  # This is the first part of the script to
82  # be executed.
83  if __name__ == '__main__':
84      # Create the command line options
85      # parser.
86      parser = argparse.ArgumentParser()
87      # Define the argument for specifying the input file.
88      parser.add_argument("-i", "--input", type=str, help="Specify the input image file.")
89      # Call the parser to parse the arguments.
90      args = parser.parse_args()
91
92      # Check that the input parameter has been specified.
93      if args.input == None:
94          # Print an error message if not and exit.
95          print("Error: No input image file provided.")
96          sys.exit()
97
98      obj = GoldenPloverPopModel()
99      obj.run(args.input)
```

10.4   The Model

```
1  #!/usr/bin/env python
2
3  # Import the system library
4  import sys
5  # Import the python Argument parser
6  import argparse
7  # Import the maths library
```

```
 8   import math as math
 9
10   # A class for the golden plover population model
11   class GoldenPloverPopModel (object):
12
13       # A function to parse the input parameters file.
14       def parseParameterFile(self, inputFile):
15           # A string to store the input parameters to
16           # be outputted in the output file.
17           paramsStr = "## Input Parameters to the model.\n"
18           # Open the input parameter file.
19           parameterFile = open(inputFile, 'r')
20           # Create a dictionary object to store the
21           # input parameters.
22           params = dict()
23           # Loop through each line of the input
24           # text file.
25           for line in parameterFile:
26               # Strip any white space either
27               # side of the text.
28               line = line.strip()
29               # Add the line to the output
30               # parameters file.
31               paramsStr += "# " + line + "\n"
32               # Split the line on the '=' symbol
33               paramVals = line.split("=", 1)
34               # Find the known parameters and
35               # convert the input string to the
36               # correct data type (i.e., float or int).
37               if paramVals[0] == "numOfYears":
38                   params[paramVals[0]] = int(paramVals[1])
39               elif paramVals[0] == "initalAdultPairPop":
40                   params[paramVals[0]] = int(paramVals[1])
41               elif paramVals[0] == "winterSurvivalRate":
42                   params[paramVals[0]] = float(paramVals[1])
43               elif paramVals[0] == "averageEggsPerPair":
44                   params[paramVals[0]] = float(paramVals[1])
45               elif paramVals[0] == "averageFledgelingsPerPair":
46                   params[paramVals[0]] = float(paramVals[1])
47               elif paramVals[0] == "predatorControl":
48                   if paramVals[1].lower() == "false":
```

```
49                         params[paramVals[0]] = False
50                     elif paramVals[1].lower() == "true":
51                         params[paramVals[0]] = True
52                     else:
53                         print("predatorControl must be either True or False.")
54                         sys.exit()
55               elif paramVals[0] == "numOfFledgelings":
56                   params[paramVals[0]] = int(paramVals[1])
57               elif paramVals[0] == "numOfFledgelingsYearOld":
58                   params[paramVals[0]] = int(paramVals[1])
59               elif paramVals[0] == "fledgelingsSurvivePredatorsCtrl":
60                   params[paramVals[0]] = float(paramVals[1])
61               elif paramVals[0] == "fledgelingsSurvivePredatorsNoCtrl":
62                   params[paramVals[0]] = float(paramVals[1])
63               else:
64                   # If parameter is not known then just store as
65                   # a string.
66                   params[paramVals[0]] = paramVals[1]
67           # Return the parameters and parameters string
68           return params, paramsStr
69
70       # A function in which the model is implemented.
71       def runGPModel(self, params):
72           # Set up some local variables - to be editted as
73           # the model runs.
74           numOfAdultsPairs = params['initalAdultPairPop']
75           numOfFledgelingsYearOld = params['numOfFledgelingsYearOld']
76           numOfFledgelings = params['numOfFledgelings']
77           numOfEggs = 0
78
79           # Create lists for the information to be outputted.
80           numOfAdultsPairsOut = list()
81           numYearOldFledgelingsOut = list()
82           numOfEggsOut = list()
83           numOfFledgelingsOut = list()
84           numOfFledgelingsB4PredOut = list()
85
86           # The main model loop - looping through the years.
87           for year in range(params['numOfYears']):
88               # Append the output parameters at the start of the year.
89               numOfAdultsPairsOut.append(numOfAdultsPairs)
```

```python
90                  numYearOldFledgelingsOut.append(numOfFledgelingsYearOld)
91                  numOfFledgelingsOut.append(numOfFledgelings)
92
93                  # Get the number of pairs (assuming all adults
94                  # are paired.
95                  numOfAdultsPairs += (numOfFledgelingsYearOld/2)
96                  # Set the number of year old fledgelings
97                  numOfFledgelingsYearOld = numOfFledgelings
98
99                  # Get the number of adults and fledgelings following winter.
100                 # Based on their winter survival rate.
101                 numOfAdultsPairs=int(numOfAdultsPairs*params['winterSurvivalRate'])
102                 numOfFledgelingsYearOld=int(numOfFledgelingsYearOld*params['winterSurvivalRate'])
103
104                 # Get the numbers of eggs to hatch
105                 numOfEggs = int(numOfAdultsPairs * params['averageEggsPerPair'])
106                 # Append to output list.
107                 numOfEggsOut.append(numOfEggs)
108
109                 # Get the number of new fledgelings.
110                 numOfFledgelings = int(numOfAdultsPairs * params['averageFledgelingsPerPair'])
111                 # Append to output.
112                 numOfFledgelingsB4PredOut.append(numOfFledgelings)
113
114                 # Apply fledgeling survival rate with an option to
115                         # apply predator control (or not)
116             if params['predatorControl']:
117                  # With predator control
118                  numOfFledgelings=int(numOfFledgelings*params['fledgelingsSurvivePredatorsCtrl'])
119             else:
120                  # Without predator control
121                  numOfFledgelings=int(numOfFledgelings*params['fledgelingsSurvivePredatorsNoCtrl']
122
123         # Once the model has completed return the output variables for analysis.
124         return numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsOut, numOfFledgelingsOut,
125
126     # The run function controlling the overall order
127     # of when things run.
128     def run(self, inputFile):
129         # Provide user feedback to the user.
130         print("Parse Input File.")
```

```
131         # Call the function to parse the input file.
132         params, paramsStr = self.parseParameterFile(inputFile)
133         # Print he parameters.
134         print(params)
135         # Provide some progress feedback to the user
136         print("Run the model")
137         # Run the model and get the output parameters.
138         numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsOut, numOfFledgelingsOut, numOfFl
139
140 # This is the first part of the script to
141 # be executed.
142 if __name__ == '__main__':
143     # Create the command line options
144     # parser.
145     parser = argparse.ArgumentParser()
146     # Define the argument for specifying the input file.
147     parser.add_argument("-i", "--input", type=str, help="Specify the input image file.")
148     # Call the parser to parse the arguments.
149     args = parser.parse_args()
150
151     # Check that the input parameter has been specified.
152     if args.input == None:
153         # Print an error message if not and exit.
154         print("Error: No input image file provided.")
155         sys.exit()
156
157     # Create instance of the model class
158     obj = GoldenPloverPopModel()
159     # Call the run function to execute the model.
160     obj.run(args.input)
```

10.5 Exporting Data

```
1 #!/usr/bin/env python
2
3 # Import the system library
4 import sys
5 # Import the python Argument parser
```

```python
6   import argparse
7   # Import the maths library
8   import math as math
9
10  # A class for the golden plover population model
11  class GoldenPloverPopModel (object):
12
13      # A function to parse the input parameters file.
14      def parseParameterFile(self, inputFile):
15          # A string to store the input parameters to
16          # be outputted in the output file.
17          paramsStr = "## Input Parameters to the model.\n"
18          # Open the input parameter file.
19          parameterFile = open(inputFile, 'r')
20          # Create a dictionary object to store the
21          # input parameters.
22          params = dict()
23          # Loop through each line of the input
24          # text file.
25          for line in parameterFile:
26              # Strip any white space either
27              # side of the text.
28              line = line.strip()
29              # Add the line to the output
30              # parameters file.
31              paramsStr += "# " + line + "\n"
32              # Split the line on the '=' symbol
33              paramVals = line.split("=", 1)
34              # Find the known parameters and
35              # convert the input string to the
36              # correct data type (i.e., float or int).
37              if paramVals[0] == "numOfYears":
38                  params[paramVals[0]] = int(paramVals[1])
39              elif paramVals[0] == "initalAdultPairPop":
40                  params[paramVals[0]] = int(paramVals[1])
41              elif paramVals[0] == "winterSurvivalRate":
42                  params[paramVals[0]] = float(paramVals[1])
43              elif paramVals[0] == "averageEggsPerPair":
44                  params[paramVals[0]] = float(paramVals[1])
45              elif paramVals[0] == "averageFledgelingsPerPair":
46                  params[paramVals[0]] = float(paramVals[1])
```

```python
47            elif paramVals[0] == "predatorControl":
48                if paramVals[1].lower() == "false":
49                    params[paramVals[0]] = False
50                elif paramVals[1].lower() == "true":
51                    params[paramVals[0]] = True
52                else:
53                    print("predatorControl must be either True or False.")
54                    sys.exit()
55            elif paramVals[0] == "numOfFledgelings":
56                params[paramVals[0]] = int(paramVals[1])
57            elif paramVals[0] == "numOfFledgelingsYearOld":
58                params[paramVals[0]] = int(paramVals[1])
59            elif paramVals[0] == "fledgelingsSurvivePredatorsCtrl":
60                params[paramVals[0]] = float(paramVals[1])
61            elif paramVals[0] == "fledgelingsSurvivePredatorsNoCtrl":
62                params[paramVals[0]] = float(paramVals[1])
63            else:
64                # If parameter is not known then just store as
65                # a string.
66                params[paramVals[0]] = paramVals[1]
67        # Return the parameters and parameters string
68        return params, paramsStr
69
70    # A function in which the model is implemented.
71    def runGPModel(self, params):
72        # Set up some local variables - to be editted as
73        # the model runs.
74        numOfAdultsPairs = params['initalAdultPairPop']
75        numOfFledgelingsYearOld = params['numOfFledgelingsYearOld']
76        numOfFledgelings = params['numOfFledgelings']
77        numOfEggs = 0
78
79        # Create lists for the information to be outputted.
80        numOfAdultsPairsOut = list()
81        numYearOldFledgelingsOut = list()
82        numOfEggsOut = list()
83        numOfFledgelingsOut = list()
84        numOfFledgelingsB4PredOut = list()
85
86        # The main model loop - looping through the years.
87        for year in range(params['numOfYears']):
```

```python
88              # Append the output parameters at the start of the year.
89              numOfAdultsPairsOut.append(numOfAdultsPairs)
90              numYearOldFledgelingsOut.append(numOfFledgelingsYearOld)
91              numOfFledgelingsOut.append(numOfFledgelings)
92
93              # Get the number of pairs (assuming all adults
94              # are paired.
95              numOfAdultsPairs += (numOfFledgelingsYearOld/2)
96              # Set the number of year old fledgelings
97              numOfFledgelingsYearOld = numOfFledgelings
98
99              # Get the number of adults and fledgelings following winter.
100             # Based on their winter survival rate.
101             numOfAdultsPairs=int(numOfAdultsPairs*params['winterSurvivalRate'])
102             numOfFledgelingsYearOld=int(numOfFledgelingsYearOld*params['winterSurvivalRate'])
103
104             # Get the numbers of eggs to hatch
105             numOfEggs = int(numOfAdultsPairs * params['averageEggsPerPair'])
106             # Append to output list.
107             numOfEggsOut.append(numOfEggs)
108
109             # Get the number of new fledgelings.
110             numOfFledgelings = int(numOfAdultsPairs * params['averageFledgelingsPerPair'])
111             # Append to output.
112             numOfFledgelingsB4PredOut.append(numOfFledgelings)
113
114             # Apply fledgeling survival rate with an option to
115                  # apply predator control (or not).
116         if params['predatorControl']:
117             # With predator control
118             numOfFledgelings=int(numOfFledgelings*params['fledgelingsSurvivePredatorsCtrl'])
119         else:
120             # Without predator control
121             numOfFledgelings=int(numOfFledgelings*params['fledgelingsSurvivePredatorsNoCtrl']
122
123      # Once the model has completed return the output variables for analysis.
124      return numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsOut, numOfFledgelingsOut,
125
126    # A function to write the results to a text file
127    # for analysis or visualisation within another
128    # package.
```

```
129     def writeResultsFile(self, outputFile, paramStr, params, numOfAdultsPairsOut, numYearOldFledg
130         # Open the output file for writing.
131         outFile = open(outputFile, 'w')
132         # Write the input parameters (the string formed
133         # when we read the input parameters in. This is
134         # useful as it will allow someone to understand
135             # where these outputs came from
136         outFile.write(paramStr)
137         # Write a header indicating the following is
138         # the model outputs.
139         outFile.write("\n\n## Output Results.\n")
140         # Create a string for each row of the output
141         # file. Each row presents a parameter.
142         yearStrs = "Year"
143         numOfAdultsStrs = "NumberOfAdultsPairs"
144         numOfYearOldFledgesStrs = "NumberOfYearOldFledgelings"
145         numOfFledgesStrs = "NumberOfFledgelings"
146         numOfFledgesB4PredStrs = "NumberOfFledgelingsB4Preds"
147         numOfEggsStrs = "NumberOfEggs"
148         # Loop through each year, building the output strings.
149         for year in range(params['numOfYears']):
150             yearStrs += "," + str(year)
151             numOfAdultsStrs += "," + str(numOfAdultsPairsOut[year])
152             numOfYearOldFledgesStrs += "," + str(numYearOldFledgelingsOut[year])
153             numOfFledgesStrs += "," + str(numOfFledgelingsOut[year])
154             numOfFledgesB4PredStrs += "," + str(numOfFledgelingsB4PredOut[year])
155             numOfEggsStrs += "," + str(numOfEggsOut[year])
156
157         # Add a new line character to the end of each row.
158         yearStrs += "\n"
159         numOfAdultsStrs += "\n"
160         numOfYearOldFledgesStrs += "\n"
161         numOfFledgesStrs += "\n"
162         numOfFledgesB4PredStrs += "\n"
163         numOfEggsStrs += "\n"
164
165         # Write the rows to the output file.
166         outFile.write(yearStrs)
167         outFile.write(numOfAdultsStrs)
168         outFile.write(numOfFledgesStrs)
169         outFile.write(numOfFledgesB4PredStrs)
```

```
170            outFile.write(numOfFledgesStrs)
171            outFile.write(numOfEggsStrs)
172
173            # Close the output file.
174            outFile.close()
175
176        # The run function controlling the overall order
177        # of when things run.
178        def run(self, inputFile):
179            # Provide user feedback to the user.
180            print("Parse Input File.")
181            # Call the function to parse the input file.
182            params, paramsStr = self.parseParameterFile(inputFile)
183            # Print he parameters.
184            print(params)
185            # Provide some progress feedback to the user
186            print("Run the model")
187            # Run the model and get the output parameters.
188            numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsOut, numOfFledgelingsOut, numOfFl
189            # Provide some feedback to the user.
190            print("Write the results to an output file")
191            # Call the function to write the outputs
192            # to a text file.
193            self.writeResultsFile(cmdargs.outputFile, paramsStr, params, numOfAdultsPairsOut, numYear
194
195    # This is the first part of the script to
196    # be executed.
197    if __name__ == '__main__':
198        # Create the command line options
199        # parser.
200        parser = argparse.ArgumentParser()
201        # Define the argument for specifying the input file.
202        parser.add_argument("-i", "--input", type=str, help="Specify the input image file.")
203        # Call the parser to parse the arguments.
204        args = parser.parse_args()
205
206        # Check that the input parameter has been specified.
207        if args.input == None:
208            # Print an error message if not and exit.
209            print("Error: No input image file provided.")
210            sys.exit()
```

```
211
212        # Create instance of the model class
213        obj = GoldenPloverPopModel()
214        # Call the run function to execute the model.
215        obj.run(args.input)
```

10.6   Creating Plots

```
1    #!/usr/bin/env python
2
3    # Import the system library
4    import sys
5    # Import the python Argument parser
6    import argparse
7    # Import the maths library
8    import math as math
9
10   # A function to test whether a module
11   # is present.
12   def module_exists(module_name):
13       # Using a try block will
14       # catch the exception thrown
15       # if the module is not
16       # available
17       try:
18           # Try to import module.
19           __import__(module_name)
20       # Catch the Import error.
21       except ImportError:
22           # Return false because
23           # the module could not
24           # be imported
25           return False
26       else:
27           # The module was successfully
28           # imported so return true.
29           return True
30
```

```python
31  # A class for the golden plover population model
32  class GoldenPloverPopModel (object):
33
34      # A function to parse the input parameters file.
35      def parseParameterFile(self, inputFile):
36          # A string to store the input parameters to
37          # be outputted in the output file.
38          paramsStr = "## Input Parameters to the model.\n"
39          # Open the input parameter file.
40          parameterFile = open(inputFile, 'r')
41          # Create a dictionary object to store the
42          # input parameters.
43          params = dict()
44          # Loop through each line of the input
45          # text file.
46          for line in parameterFile:
47              # Strip any white space either
48              # side of the text.
49              line = line.strip()
50              # Add the line to the output
51              # parameters file.
52              paramsStr += "# " + line + "\n"
53              # Split the line on the '=' symbol
54              paramVals = line.split("=", 1)
55              # Find the known parameters and
56              # convert the input string to the
57              # correct data type (i.e., float or int).
58              if paramVals[0] == "numOfYears":
59                  params[paramVals[0]] = int(paramVals[1])
60              elif paramVals[0] == "initalAdultPairPop":
61                  params[paramVals[0]] = int(paramVals[1])
62              elif paramVals[0] == "winterSurvivalRate":
63                  params[paramVals[0]] = float(paramVals[1])
64              elif paramVals[0] == "averageEggsPerPair":
65                  params[paramVals[0]] = float(paramVals[1])
66              elif paramVals[0] == "averageFledgelingsPerPair":
67                  params[paramVals[0]] = float(paramVals[1])
68              elif paramVals[0] == "predatorControl":
69                  if paramVals[1].lower() == "false":
70                      params[paramVals[0]] = False
71                  elif paramVals[1].lower() == "true":
```

```
72                      params[paramVals[0]] = True
73                  else:
74                      print("predatorControl must be either True or False.")
75                      sys.exit()
76              elif paramVals[0] == "numOfFledgelings":
77                  params[paramVals[0]] = int(paramVals[1])
78              elif paramVals[0] == "numOfFledgelingsYearOld":
79                  params[paramVals[0]] = int(paramVals[1])
80              elif paramVals[0] == "fledgelingsSurvivePredatorsCtrl":
81                  params[paramVals[0]] = float(paramVals[1])
82              elif paramVals[0] == "fledgelingsSurvivePredatorsNoCtrl":
83                  params[paramVals[0]] = float(paramVals[1])
84              else:
85                  # If parameter is not known then just store as
86                  # a string.
87                  params[paramVals[0]] = paramVals[1]
88          # Return the parameters and parameters string
89          return params, paramsStr
90
91      # A function in which the model is implemented.
92      def runGPModel(self, params):
93          # Set up some local variables - to be editted as
94          # the model runs.
95          numOfAdultsPairs = params['initalAdultPairPop']
96          numOfFledgelingsYearOld = params['numOfFledgelingsYearOld']
97          numOfFledgelings = params['numOfFledgelings']
98          numOfEggs = 0
99
100         # Create lists for the information to be outputted.
101         numOfAdultsPairsOut = list()
102         numYearOldFledgelingsOut = list()
103         numOfEggsOut = list()
104         numOfFledgelingsOut = list()
105         numOfFledgelingsB4PredOut = list()
106
107         # The main model loop - looping through the years.
108         for year in range(params['numOfYears']):
109             # Append the output parameters at the start of the year.
110             numOfAdultsPairsOut.append(numOfAdultsPairs)
111             numYearOldFledgelingsOut.append(numOfFledgelingsYearOld)
112             numOfFledgelingsOut.append(numOfFledgelings)
```

```
113
114                # Get the number of pairs (assuming all adults
115                # are paired.
116                numOfAdultsPairs += (numOfFledgelingsYearOld/2)
117                # Set the number of year old fledgelings
118                numOfFledgelingsYearOld = numOfFledgelings
119
120                # Get the number of adults and fledgelings following winter.
121                # Based on their winter survival rate.
122                numOfAdultsPairs=int(numOfAdultsPairs*params['winterSurvivalRate'])
123                numOfFledgelingsYearOld=int(numOfFledgelingsYearOld*params['winterSurvivalRate'])
124
125                # Get the numbers of eggs to hatch
126                numOfEggs = int(numOfAdultsPairs * params['averageEggsPerPair'])
127                # Append to output list.
128                numOfEggsOut.append(numOfEggs)
129
130                # Get the number of new fledgelings.
131                numOfFledgelings = int(numOfAdultsPairs * params['averageFledgelingsPerPair'])
132                # Append to output.
133                numOfFledgelingsB4PredOut.append(numOfFledgelings)
134
135                # Apply fledgeling survival rate with an option to
136                        # apply predator control or not
137            if params['predatorControl']:
138                # With predator control
139                numOfFledgelings=int(numOfFledgelings*params['fledgelingsSurvivePredatorsCtrl'])
140            else:
141                # Without predator control
142                numOfFledgelings=int(numOfFledgelings*params['fledgelingsSurvivePredatorsNoCtrl']
143
144        # Once the model has completed return the output variables for analysis.
145        return numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsOut, numOfFledgelingsOut,
146
147    # A function to write the results to a text file
148    # for analysis or visualisation within another
149    # package.
150    def writeResultsFile(self, outputFile, paramStr, params, numOfAdultsPairsOut, numYearOldFledg
151        # Open the output file for writing.
152        outFile = open(outputFile, 'w')
153        # Write the input parameters (the string formed
```

```
154              # when we read the input parameters in. This is
155              # useful as it will allow someone to understand
156                  # where these outputs came from
157          outFile.write(paramStr)
158          # Write a header indicating the following is
159          # the model outputs.
160          outFile.write("\n\n## Output Results.\n")
161          # Create a string for each row of the output
162          # file. Each row presents a parameter.
163          yearStrs = "Year"
164          numOfAdultsStrs = "NumberOfAdultsPairs"
165          numOfYearOldFledgesStrs = "NumberOfYearOldFledgelings"
166          numOfFledgesStrs = "NumberOfFledgelings"
167          numOfFledgesB4PredStrs = "NumberOfFledgelingsB4Preds"
168          numOfEggsStrs = "NumberOfEggs"
169          # Loop through each year, building the output strings.
170          for year in range(params['numOfYears']):
171              yearStrs += "," + str(year)
172              numOfAdultsStrs += "," + str(numOfAdultsPairsOut[year])
173              numOfYearOldFledgesStrs += "," + str(numYearOldFledgelingsOut[year])
174              numOfFledgesStrs += "," + str(numOfFledgelingsOut[year])
175              numOfFledgesB4PredStrs += "," + str(numOfFledgelingsB4PredOut[year])
176              numOfEggsStrs += "," + str(numOfEggsOut[year])
177
178          # Add a new line character to the end of each row.
179          yearStrs += "\n"
180          numOfAdultsStrs += "\n"
181          numOfYearOldFledgesStrs += "\n"
182          numOfFledgesStrs += "\n"
183          numOfFledgesB4PredStrs += "\n"
184          numOfEggsStrs += "\n"
185
186          # Write the rows to the output file.
187          outFile.write(yearStrs)
188          outFile.write(numOfAdultsStrs)
189          outFile.write(numOfFledgesStrs)
190          outFile.write(numOfFledgesB4PredStrs)
191          outFile.write(numOfFledgesStrs)
192          outFile.write(numOfEggsStrs)
193
194          # Close the output file.
```

```
195            outFile.close()
196
197      def plots(self, outputFile, params, numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsO
198          # Test that the matplotlib library is present
199          # so plots can be created.
200          if module_exists("matplotlib.pyplot"):
201              # The matplotlib library exists so
202              # import it for use in this function.
203              # Importing a library within a function
204                      # like this means it is only
205              # available within this function.
206              import matplotlib.pyplot as plt
207              # Get the number of years as a list
208              # of years.
209              years = range(params['numOfYears'])
210
211              # Create a simple plot for the number of
212              # pairs.
213              fig1 = plt.figure(figsize=(15, 5), dpi=150)
214              plt.plot(years, numOfAdultsPairsOut)
215              plt.title("Number of pairs per year predicted by model")
216              plt.xlabel("Year")
217              plt.ylabel("Number Of Pairs")
218              plt.savefig((outputFile+"_adultpairs.pdf"), format='PDF')
219
220              # Create a simple plot for the number of
221              # year old fledgelings
222              fig2 = plt.figure(figsize=(15, 5), dpi=150)
223              plt.plot(years, numYearOldFledgelingsOut)
224              plt.title("Number of year olf fledgelings predicted by model")
225              plt.xlabel("Year")
226              plt.ylabel("Number Of Fledglings")
227              plt.savefig((outputFile+"_numYearOldFledgelings.pdf"), format='PDF')
228
229              # Create a simple plot for the number of
230              # eggs hatched each year.
231              fig3 = plt.figure(figsize=(15, 5), dpi=150)
232              plt.plot(years, numOfEggsOut)
233              plt.title("Number of eggs per year predicted by model")
234              plt.xlabel("Year")
235              plt.ylabel("Number Of Eggs")
```

```
236                plt.savefig((outputFile+"_numOfEggs.pdf"), format='PDF')
237
238                # Create a simple plot for the number of
239                # new born fledgelings
240                fig4 = plt.figure(figsize=(15, 5), dpi=150)
241                plt.plot(years, numOfFledgelingsOut)
242                plt.title("Number of fledgelings per year predicted by model")
243                plt.xlabel("Year")
244                plt.ylabel("Number Of Fledgelings")
245                plt.savefig((outputFile+"_numOfFledgelings.pdf"), format='PDF')
246
247                # Create a simple plot for the number of
248                # fledgelings before that years breeding
249                fig5 = plt.figure(figsize=(15, 5), dpi=150)
250                plt.plot(years, numOfFledgelingsB4PredOut)
251                plt.title("Number of fledgelings before breeding per year predicted by model")
252                plt.xlabel("Year")
253                plt.ylabel("Number Of Fledgelings")
254                plt.savefig((outputFile+"_numOfFledgelingsB4Pred.pdf"), format='PDF')
255
256            else:
257                # If the matplotlib library is not available
258                # print out a suitable error message.
259                print("Matplotlib is not available and therefore the plots cannot be created.")
260
261        # The run function controlling the overall order
262        # of when things run.
263        def run(self, inputFile, outputFile, plotsPath):
264            # Provide user feedback to the user.
265            print("Parse Input File.")
266            # Call the function to parse the input file.
267            params, paramsStr = self.parseParameterFile(inputFile)
268            # Print he parameters.
269            print(params)
270            # Provide some progress feedback to the user
271            print("Run the model")
272            # Run the model and get the output parameters.
273            numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEggsOut, numOfFledgelingsOut, numOfFl
274            # Provide some feedback to the user.
275            print("Write the results to an output file")
276            # Call the function to write the outputs
```

```
277            # to a text file.
278            self.writeResultsFile(outputFile, paramsStr, params, numOfAdultsPairsOut, numYearOldFledg
279            # Check whether a path has been provided
280            # for the plots. If it has then generate
281            # output plots.
282            if plotsPath is not None:
283                # Give the user feedback of what's happenign.
284                print("Generating plots of the results")
285                # Call the function to generate plots
286                self.plots(plotsPath, params, numOfAdultsPairsOut, numYearOldFledgelingsOut, numOfEgg
287
288  # This is the first part of the script to
289  # be executed.
290  if __name__ == '__main__':
291      # Create the command line options
292      # parser.
293      parser = argparse.ArgumentParser()
294      # Define the argument for specifying the input file.
295      parser.add_argument("-i", "--input", type=str, help="Specify the input image file.")
296      # Define the argument for specifying the output file.
297      parser.add_argument("-o", "--output", type=str, help="Specify the output text file.")
298      # Define the argument for specifying the output file.
299      parser.add_argument("-p", "--plot", type=str, help="Specify the output base path for the plot
300      # Call the parser to parse the arguments.
301      args = parser.parse_args()
302
303      # Check that the input parameter has been specified.
304      if args.input == None:
305          # Print an error message if not and exit.
306          print("Error: No input image file provided.")
307          sys.exit()
308
309      # Check that the input parameter has been specified.
310      if args.output == None:
311          # Print an error message if not and exit.
312          print("Error: No output text file provided.")
313          sys.exit()
314
315      # Create instance of the model class
316      obj = GoldenPloverPopModel()
317      # Call the run function to execute the model.
```

₃₁₈   `obj.run(args.input, args.output, args.plot)`

10.7   Exercises

10.8   Further Reading

- Python Documentation - `http://www.python.org/doc`

- Core Python Programming (Second Edition), W.J. Chun. Prentice Hall ISBN 0-13-226993-7

Appendix A

RSGISLib

A.1 Introduction to RSGISLib

The remote sensing and GIS software library (RSGISLib) was developed at Aberystwyth University by Pete Bunting and Daniel Clewley. Development started in April 2008 and has been actively maintained and added to ever since. For more information see `http://www.rsgislib.org`.

A.2 Using RSGISLib

RSGISLib has an XML interface and Python bindings. The python bindings allows the RSGISLib functions to be called from python scripts, as with other functions.

A.3 Segmentation

The segmentation algorithm (**?**) is based on generating spectrally similar units with a minimum object size.

The algorithm consists of a number of steps

1. Select image bands and stack images

2. Stretch image data

3. Find unique cluster within feature space (KMeans)

4. Assign pixels to clusters

5. Clump the image

6. Eliminate small segments

The KMeans clustering takes just a single image where all the bands are used as input so if multiple images are required to be inputted then they need to be stacked and the bands which are to be used selected. As a Euclidean distance is used within the feature space the image is stretched such that all the pixel values are within the same range (i.e., 0–255).

A clustering algorithm is then used to identify the unique colours within the image, in this case a KMeans clustering is used but other clustering algorithms could also be used instead. The image pixels are then assigned to the clusters (classifying the image) and the image clumped to find the connected regions of the image.

The final step is an iterative elimination of the small segments, starting with the single pixels and going up to the maximum size of the segments specified by the user.

Therefore, there are two key parameters within the algorithm:

1. the number of cluster centres identified by the KMeans clustering (-k)

2. the minimum size of the segments (-n)

There are a number of python functions required to perform the segmentation, an example python script is provided below:

```python
#! /usr/bin/env python

###############################################################################
#  RSGISLibSegmentation.py
#
#  Copyright 2013 RSGISLib.
#
#  RSGISLib: 'The remote sensing and GIS Software Library'
#
#
# Purpose:  Execute the Shepherd et al segmentation algorithm.
#
# Author: Pete Bunting
# Email: pfb@aber.ac.uk
# Date: 24/07/2013
# Version: 1.0
#
# History:
# Version 1.0 - Created.
#
###############################################################################

# Import the python Argument parser
```

```python
24  import argparse
25  # Import the rsgislib module
26  import rsgislib
27  # Import the image utilities module from rsgislib
28  import rsgislib.imageutils
29  # Import the image calc module from rsgislib
30  import rsgislib.imagecalc
31  # Import the image segmentation module from rsgislib
32  import rsgislib.segmentation
33  # Import the image rastergis module from rsgislib
34  import rsgislib.rastergis
35  # Import the module from rsgislib
36  import rsgislib
37  import os.path
38  import os
39  # Import the collections module
40  import collections
41  import fnmatch
42
43  class RSGISLibSegmentation (object):
44
45      def runShepherdSegmentation(self, inputImg, outputClumps, outputMeanImg, tmpath,
46          gdalFormat, noStats, noStretch, noDelete, numClusters, minPxls, distThres):
47          rsgisUtils = rsgislib.RSGISPyUtils()
48
49          basefile = os.path.basename(inputImg)
50          basename = os.path.splitext(basefile)[0]
51
52          outFileExt = rsgisUtils.getFileExtension(gdalFormat)
53
54          createdDIR = False
```

```python
55          if not os.path.isdir(tmpath):
56              os.makedirs(tmpath)
57              createdDIR = True
58
59          # Stretch input data if required.
60          segmentFile = inputImg
61          if not noStretch:
62              segmentFile = os.path.join(tmpath,basename+
63                  str("_stchd")+outFileExt)
64              strchFile = os.path.join(tmpath,basename+
65                  str("_stchdonly")+outFileExt)
66              strchFileOffset = os.path.join(tmpath,basename+
67                  str("_stchdonlyOff")+outFileExt)
68              strchMaskFile = os.path.join(tmpath,basename+
69                  str("_stchdmaskonly")+outFileExt)
70              print("Stretch Input Image")
71              rsgislib.imageutils.stretchImage(inputImg, strchFile,
72                  False, "", True, False, gdalFormat, rsgislib.TYPE_32FLOAT,
73                  rsgislib.imageutils.STRETCH_LINEARSTDDEV, 2)
74
75              print("Add 1 to stretched file to ensure there are no all zeros \
76                          (i.e., no data) regions created.")
77              rsgislib.imagecalc.imageMath(strchFile, strchFileOffset, "b1+1",
78                  gdalFormat, rsgislib.TYPE_32FLOAT)
79
80              print("Create Input Image Mask.")
81              ImgBand = collections.namedtuple('ImgBands',
82                  ['bandName', 'fileName', 'bandIndex'])
83              bandMathBands = list()
84              bandMathBands.append(ImgBand(bandName="b1",
85                  fileName=inputImg, bandIndex=1))
```

```
86              rsgislib.imagecalc.bandMath(strchMaskFile, "b1==0?1:0", gdalFormat,
87                  rsgislib.TYPE_32FLOAT, bandMathBands)
88
89              print("Mask stretched Image.")
90              rsgislib.imageutils.maskImage(strchFileOffset, strchMaskFile,
91                  segmentFile, gdalFormat, rsgislib.TYPE_32FLOAT, 0, 1)
92
93              if not noDelete:
94                  # Deleting extra files
95                  rsgisUtils.deleteFileWithBasename(strchFile)
96                  rsgisUtils.deleteFileWithBasename(strchFileOffset)
97                  rsgisUtils.deleteFileWithBasename(strchMaskFile)
98
99          # Perform KMEANS
100         print("Performing KMeans.")
101         outMatrixFile = os.path.join(tmpath,basename+str("_kmeansclusters"))
102         rsgislib.imagecalc.kMeansClustering(segmentFile, outMatrixFile, numClusters,
103             200, 100, True, 0.0025, rsgislib.imagecalc.INITCLUSTER_DIAGONAL_FULL_ATTACH)
104
105         # Apply KMEANS
106         print("Apply KMeans to image.")
107         kMeansFileZones = os.path.join(tmpath,basename+str("_kmeans")+outFileExt)
108         rsgislib.segmentation.labelPixelsFromClusterCentres(segmentFile,
109             kMeansFileZones, outMatrixFile+str(".gmtxt"), True, gdalFormat)
110
111         # Elimininate Single Pixels
112         print("Eliminate Single Pixels.")
113         kMeansFileZonesNoSgls = os.path.join(tmpath,basename+
114             str("_kmeans_nosgl")+outFileExt)
115         kMeansFileZonesNoSglsTmp = os.path.join(tmpath,basename+
116             str("_kmeans_nosglTMP")+outFileExt)
```

```
117        rsgislib.segmentation.eliminateSinglePixels(segmentFile, kMeansFileZones,
118            kMeansFileZonesNoSgls, kMeansFileZonesNoSglsTmp, gdalFormat, False, True)
119
120        # Clump
121        print("Perform clump.")
122        initClumpsFile = os.path.join(tmpath,basename+str("_clumps")+outFileExt)
123        rsgislib.segmentation.clump(kMeansFileZonesNoSgls, initClumpsFile,
124            gdalFormat, False, 0)
125
126        # Elimininate small clumps
127        print("Eliminate small pixels.")
128        elimClumpsFile = os.path.join(tmpath,basename+str("_clumps_elim")+outFileExt)
129        rsgislib.segmentation.RMSmallClumpsStepwise(segmentFile, initClumpsFile,
130            elimClumpsFile, gdalFormat, False, "", False, False, minPxls, distThres)
131
132        # Relabel clumps
133        print("Relabel clumps.")
134        rsgislib.segmentation.relabelClumps(elimClumpsFile, outputClumps,
135            gdalFormat, False)
136
137        # Populate with stats if required.
138        if not noStats:
139            print("Calculate image statistics and build pyramids.")
140            rsgislib.rastergis.populateStats(outputClumps, True, True)
141
142        # Create mean image if required.
143        if not (outputMeanImg == None):
144            rsgislib.segmentation.meanImage(inputImg, outputClumps, outputMeanImg,
145                gdalFormat, rsgislib.TYPE_32FLOAT)
146            if not noStats:
147                rsgislib.imageutils.popImageStats(outputMeanImg, True, 0, True)
```

```python
148
149          if not noDelete:
150              # Deleting extra files
151              rsgisUtils.deleteFileWithBasename(kMeansFileZones)
152              rsgisUtils.deleteFileWithBasename(kMeansFileZonesNoSgls)
153              rsgisUtils.deleteFileWithBasename(kMeansFileZonesNoSglsTmp)
154              rsgisUtils.deleteFileWithBasename(initClumpsFile)
155              rsgisUtils.deleteFileWithBasename(elimClumpsFile)
156              if not noStretch:
157                  rsgisUtils.deleteFileWithBasename(segmentFile)
158              if createdDIR:
159                  rsgisUtils.deleteDIR(tmpath)
160
161  if __name__ == '__main__':
162      """
163      The command line user interface to the RSGISLib Segmentation tool
164      """
165      parser = argparse.ArgumentParser(prog='rsgislibsegmentation.py',
166          description='''Run the Shepherd et al segmentation algorithm.''')
167
168      # Define the argument for specifying the input image.
169      parser.add_argument("-i", "--input", type=str,
170          help='''Specify the input image file.''', required=True)
171
172      # Define the argument for specifying the output image.
173      parser.add_argument("-o", "--output", type=str,
174          help='''Specify the output clump image file.''', required=True)
175
176      # Define the argument for specifying the output image.
177      parser.add_argument("-m", "--outmeanimg", type=str,
178          help='''Specify the output mean image file.
```

```
179              If not specified a mean image is not created.''')
180
181     # Define the argument for specifying the output image.
182     parser.add_argument("-t", "--tmpath", type=str,
183        help='''Specify the file path for intermediate
184            files.''', required=True)
185
186     # Define the argument for specifying the output image.
187     parser.add_argument("-k", "--numclusters", type=int,
188        help='''Specify the number of clusters within
189            the KMeans clustering.''', required=True)
190
191     # Define the argument for specifying the output image.
192     parser.add_argument("-n", "--minpxls", type=int,
193        help='''Specify the minimum number pixels within
194            a segments.''', required=True)
195
196     # Define the argument for specifying the output image.
197     parser.add_argument("-d", "--distthres", type=float,
198        help='''Specify the distance threshold for joining the segments
199            (default is a very large value which turns off this option.).''',
200            default=1000000, required=False)
201
202     # Define the argument for specifying the image file format.
203     parser.add_argument("-f", "--format", type=str, default="KEA",
204                    help='''Specify the image output format (GDAL name).''')
205
206     # Define the argument for specifying that statistics
207     # and pyramids should be built for all output images.
208     parser.add_argument("--nostats", action='store_true', default=False,
209                    help='''Specifies that no image statistics and
```

```
210                              pyramids should be build for the output images.''')
211
212      # Define the argument for specifying that the input image bands
213      # should not be stretched.
214      parser.add_argument("--nostretch", action='store_true', default=False,
215                          help='''Specifies that the input image
216                              bands should not be stretched.''')
217
218      # Define the argument for specifying that the input image bands
219      # should not be stretched.
220      parser.add_argument("--nodelete", action='store_true', default=False,
221                          help='''Specifies that the temporary images
222                              created during processing should not be deleted
223                              once processing has been completed.''')
224
225      # Call the parser to parse the arguments.
226      args = parser.parse_args()
227
228      segsObj = RSGISLibSegmentation()
229
230      segsObj.runShepherdSegmentation(args.input, args.output, args.outmeanimg,
231          args.tmpath, args.format, args.nostats, args.nostretch, args.nodelete,
232          args.numclusters, args.minpxls, args.distthres)
```

The script can be run as follows:

```
python rsgissegmentation.py -i L7ETM_530N035W.kea \
    -o L7ETM_530N035W_clumps_elim_final.kea \
    -m L7ETM_530N035W_clumps_elim_final_mean.kea \
    -t ./tmp -k 60 -n 100
```

The resulting segmentation will have produced 3 output files

1. `*clusters.gmtxt` – Cluster centres.

2. `*clumps_elim_final.kea` – Segment clumps.

3. `*clumps_elim_mean.kea` – Mean colour image using segments.

A.4 Populating Segments

To populate the segments with statistics (i.e., Mean for each spectral band) there is a function within the rastergis module of RSGISLib. To populate segments with statistics from two images and a DEM the following script is used:

```python
#! /usr/bin/env python

########################################
# A python script to attribute
# segments
#
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
########################################

# Import the argparse module
import argparse
# Import the RSGISLib RasterGIS module
import rsgislib.rastergis

class PopWithStats (object):
```

```
19
20      def __init__(self, inClumps):
21          self.inClumps = inClumps
22
23      def popWithMayStats(self, image):
24          """ Populate with stats from May image """
25          stats2Calc = list()
26          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=1, \
27              meanField="MayBlue", stdDevField="MaySDBlue"))
28          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=2, \
29              meanField="MayGreen", stdDevField="MaySDGreen"))
30          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=3, \
31              meanField="MayRed", stdDevField="MaySDRed"))
32          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=4, \
33              meanField="MayNIR", stdDevField="MaySDNIR"))
34          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=5, \
35              meanField="MaySWIR1", stdDevField="MaySDSWIR1"))
36          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=6, \
37              meanField="MaySWIR2", stdDevField="MaySDSWIR2"))
38
39          rsgislib.rastergis.populateRATWithStats(image, self.inClumps, stats2Calc)
40
41      def popWithJuneStats(self, image):
42          """ Populate with stats from June image """
43          stats2Calc = list()
44          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=1, \
45              meanField="JuneBlue", stdDevField="JuneSDBlue"))
46          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=2, \
47              meanField="JuneGreen", stdDevField="JuneSDGreen"))
48          stats2Calc.append(rsgislib.rastergis.BandAttStats(band=3, \
49              meanField="JuneRed", stdDevField="JuneSDRed"))
```

```
50        stats2Calc.append(rsgislib.rastergis.BandAttStats(band=4, \
51            meanField="JuneNIR", stdDevField="JuneSDNIR"))
52        stats2Calc.append(rsgislib.rastergis.BandAttStats(band=5, \
53            meanField="JuneSWIR1", stdDevField="JuneSDSWIR1"))
54        stats2Calc.append(rsgislib.rastergis.BandAttStats(band=6, \
55            meanField="JuneSWIR2", stdDevField="JuneSDSWIR2"))
56
57        rsgislib.rastergis.populateRATWithStats(image, self.inClumps, stats2Calc)
58
59
60    def popWithDEMStats(self, image):
61        """ Populate with DEM stats """
62        stats2Calc = list()
63        stats2Calc.append(rsgislib.rastergis.BandAttStats(band=1, \
64            minField="MinDEM", maxField="MaxDEM", \
65            stdDevField="StdDevDEM", meanField="MeanDEM"))
66
67        rsgislib.rastergis.populateRATWithStats(image, self.inClumps, stats2Calc)
68
69
70 if __name__ == '__main__':
71
72    parser = argparse.ArgumentParser()
73
74    parser.add_argument("--inclumps", type=str, \
75        help='''Specify the input clumps file.''', required=True)
76    parser.add_argument("--mayimage", type=str, \
77        help='''Specify the input image for May.''', required=True)
78    parser.add_argument("--juneimage", type=str, \
79        help='''Specify the input image for June.''', required=True)
80    parser.add_argument("--dem", type=str, \
```

```
81          help='''Specify the input DEM.''', required=True)

82

83     args = parser.parse_args()

84

85     popStats = PopWithStats(args.inclumps)

86

87     # Pop with may statistics
88     popStats.popWithMayStats(args.mayimage)

89

90     # Pop with june statistics
91     popStats.popWithMayStats(args.juneimage)

92

93     # Pop with DEM statistics
94     popStats.popWithMayStats(args.dem)

95

96

97
```

If you are going to use indices and other derived information within your classification it is quite often a good idea to set up a python script to calculate those indices and write them back to the image rather than over complicating your classification script. An example of this is shown below.

```
1   #!/usr/bin/env python

2

3   import sys
4   from rios import rat
5   import numpy
6   import osgeo.gdal as gdal

7

8
```

```
9   #Input file.

10  fname = "L7ETM_530N035W_Classification.kea"

11  ratDataset = gdal.Open( fname, gdal.GA_Update )

12

13  print("Import Columns.")

14  MayBlue = rat.readColumn(ratDataset, "MayBlue")

15  MayGreen = rat.readColumn(ratDataset, "MayGreen")

16  MayRed = rat.readColumn(ratDataset, "MayRed")

17  MayNIR = rat.readColumn(ratDataset, "MayNIR")

18  MaySWIR1 = rat.readColumn(ratDataset, "MaySWIR1")

19  MaySWIR2 = rat.readColumn(ratDataset, "MaySWIR2")

20

21  JuneBlue = rat.readColumn(ratDataset, "JuneBlue")

22  JuneGreen = rat.readColumn(ratDataset, "JuneGreen")

23  JuneRed = rat.readColumn(ratDataset, "JuneRed")

24  JuneNIR = rat.readColumn(ratDataset, "JuneNIR")

25  JuneSWIR1 = rat.readColumn(ratDataset, "JuneSWIR1")

26  JuneSWIR2 = rat.readColumn(ratDataset, "JuneSWIR2")

27

28  MeanDEM = rat.readColumn(ratDataset, "MeanDEM")

29

30  MayNIR.astype(numpy.float32)

31  MayRed.astype(numpy.float32)

32  JuneNIR.astype(numpy.float32)

33  JuneRed.astype(numpy.float32)

34  MayBlue.astype(numpy.float32)

35  JuneBlue.astype(numpy.float32)

36

37  print("Calculate Indices.")

38  MayNDVI = (MayNIR - MayRed) / (MayNIR + MayRed)

39  JuneNDVI = (JuneNIR - JuneRed) / (JuneNIR + JuneRed)
```

```
40
41   MayWBI = MayBlue/MayNIR
42   JuneWBI = JuneBlue/JuneNIR
43
44   rat.writeColumn(ratDataset, "MayNDVI", MayNDVI)
45   rat.writeColumn(ratDataset, "JuneNDVI", JuneNDVI)
46   rat.writeColumn(ratDataset, "MayWBI", MayWBI)
47   rat.writeColumn(ratDataset, "JuneWBI", JuneWBI)
```