Python Scripting for Spatial Data Processing.

Pete Bunting and Daniel Clewley

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

January 31, 2015

Aberystwyth University

Department of Geography and Earth Sciences.

Acknowledgements

The authors would like to acknowledge to the supports of others but specifically (and in no particular order) Prof. Richard Lucas, Sam Gillingham (developer of RIOS and TuiView) and Neil Flood (developer of RIOS) for their support and time.

Authors

Peter Bunting

Dr Pete Bunting joined the Department of Geography and Earth Sciences (DGES), 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 DGES 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.

```
if␣x␣==␣1:
␣␣␣␣x␣=␣x␣+␣1
else:
␣␣␣␣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:

```
# Get input directory (containing the compressed files)

# Get output directory (where the files, once uncompressed, will be placed).

# Retrieve list of all files (to be uncompressed) in the input directory.

# Iterator through input files, uncompressing each in turn.
        # Get single file from list
        # create command line command for the current file
        # 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 rsgislib.imageutils
>>> help(rsgislib.imageutils.subset)
```

Note, you can us the `as` keyword to shorten long imports, which can also make your code more readable and save typing long module paths a number of times.

```
>>> import rsgislib.imageutils as imgutil
>>> help(imgutil.subset)
```

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:

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

#######################################
# A simple Hello World Script
# Author: <YOUR NAME>
# Emai: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

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:

```
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.

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

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

import string

stringVariable = ’       Hello World       ’

print(’\’’ +  stringVariable + ’\’’)

stringVariable_lstrip = stringVariable.lstrip()
print(’lstrip: \’’ +  stringVariable_lstrip + ’\’’)

stringVariable_rstrip = stringVariable.rstrip()
print(’rstrip: \’’ +  stringVariable_rstrip + ’\’’)

stringVariable_strip = stringVariable.strip()
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.

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

########################################
# A simple script illustrating the use of
# variables.
# Author: <YOUR NAME>
# Emai: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.1
#
# Added helper function for 2/3 compatibility
#######################################

# Compatibility function for 2/3
import sys
def getInput(question):
    """ Python 2/3 helper function
        for getting input.
    """
    if sys.version[0] == '3':
        answer = input(question)
    else:
        answer = raw_input(question)
    return(answer)

score = 0 # A variable to store the ongoing score

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

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

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

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

```
    print('Bad Luck')
    score = score - 1 # Remove 1 from the score

print('Question 2:')
answer = getInput('CASI provides hyperspectral data in ' \
'the Blue to NIR part of the spectrum.\n')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 3:')
answer = getInput('HyMap also only provides data in the ' \
'Blue to NIR part of the spectrum.\n')
if answer == 'y':
    print('Bad Luck')
    score = score - 1
else:
    print('Well done')
    score = score + 1

print('Question 4:')
answer = getInput('Landsat is a spaceborne sensor.\n')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

print('Question 5:')
answer = getInput('ADS-40 is a high resolution aerial ' \
'sensor capturing RGB-NIR wavelengths.\n')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1
```

```

print(’Question 6:’)
answer = getInput(’eCognition is an object oriented ’ \
’image analysis software package.\n’)
if answer == ’y’:
    print(’Well done’)
    score = score + 1
else:
    print(’Bad Luck’)
    score = score - 1

print(’Question 7:’)
answer = getInput(’Adobe Photoshop provides the same ’ \
’functionality as eCognition.\n’)
if answer == ’y’:
    print(’Bad Luck’)
    score = score - 1
else:
    print(’Well done’)
    score = score + 1

print(’Question 8:’)
answer = getInput(’Python can be executed within ’ \
’the a java virtual machine.\n’)
if answer == ’y’:
    print(’Well done’)
    score = score + 1
else:
    print(’Bad Luck’)
    score = score - 1

print(’Question 9:’)
answer = getInput(’Python is a scripting language ’ \
’not a programming language.\n’)
if answer == ’y’:
    print(’Well done’)
    score = score + 1
else:
    print(’Bad Luck’)
    score = score - 1

```

```
print('Question 10:')
answer = getInput('Aberystwyth is within Mid Wales.\n')
if answer == 'y':
    print('Well done')
    score = score + 1
else:
    print('Bad Luck')
    score = score - 1

# Finally print out the users final score.
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
#! /usr/bin/env python

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

# Create List:
aList = list()
anotherList = [1, 2, 3, 4]
emptyList = []

print(aList)
print(anotherList)
print(emptyList)

# Adding data into a List
aList.append('Pete')
aList.append('Dan')
print(aList)

# Updating data in the List
anotherList[2] = 'three'
anotherList[0] = 'one'
print(anotherList)

# Accessing data in the List
print(aList[0])
print(anotherList[0:2])
print(anotherList[2:3])

# Removing data from the List
del anotherList[1]
print(anotherList)

aList.remove('Pete')
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.

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

#######################################
# A simple example of a while loop
# Author: <YOUR NAME>
# Emai: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

count = 0
while count <= 10:
    print(count)
    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:

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

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

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

#######################################
# A simple example of a for loop
# Author: <YOUR NAME>
# Emai: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

aList = ['Pete', 'Richard', 'Johanna', 'Philippa', 'Sam', 'Dan', 'Alex']

for name in aList:
        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.

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

```

```
#######################################
# Example with for loop and 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]
]

for cList in aList:
        for number in cList:
            # Print with a space at the end
            # rather than a new line
                print (number,end=" ")
        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.

```
#! /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.
```

```
numbers = list()
dataFile = open(’numbers.txt’, ’r’)

for eachLine in dataFile:
    #print(eachLine)
    tmpStr = ’’
    for char in eachLine:
        #print(char)
        if char.isdigit():
            tmpStr += char
        elif char == ’,’ and tmpStr != ’’:
            numbers.append(int(tmpStr))
            tmpStr = ’’
    if tmpStr.isdigit():
        numbers.append(int(tmpStr))

print(numbers)
dataFile.close()

# 2) Uses the string function split to line from the file
# into a list of substrings
numbers = list()
dataFile = open(’numbers.txt’, ’r’)

for eachLine in dataFile:
    #print eachLine
    substrs = eachLine.split(’,’,eachLine.count(’,’))
    #print substrs
    for strVar in substrs:
        if strVar.isdigit():
            numbers.append(int(strVar))

print(numbers)
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:

```
#! /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

```

```
    # METHOD TO EXECUTE CLASS
    def run(self):
        self.printName()
        self.setName('Pete')
        self.printName()

# IF EXECUTED FROM THE COMMAND LINE
if __name__ == '__main__':
    obj = Person()
    obj.run()

# 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:

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

#######################################
# An python class to parse a comma
# separates text file to calculate
# the mean and standard deviation
# of the inputted floating point
# numbers.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

# import the squareroot function from python math
from math import sqrt

# Define a new class called CalcMeanStdDev
class CalcMeanStdDev (object):

    # Define a function which parses a comma
    # separated file - you should understand
    # the contents of this script from the
    # previous examples.
    # Note: the file is passed into the
    # function.
    def parseCommaFile(self, file):
        floatingNumbers = list()
        for eachLine in file:
            substrs = eachLine.split(’,’,eachLine.count(’,’))
            for strVar in substrs:
                floatingNumbers.append(float(strVar))
        return floatingNumbers

    # Define a function to calculate the mean
    # value from a list of numbers.
    # Note. The list of numbers is passed into
    # the function.
    def calcMean(self, numbers):
        # A variable to sum all the numbers
        sum = 0.0
        # Iterate through the numbers list
        for number in numbers:
            # add each number to the sum
            sum += number
        # Divide the sum by the number of
        # values within the numbers list
        # (i.e., its length)
        mean = sum/len(numbers)
        # return the mean value calculated
        return mean

    # Define a function which calculates the
    # standard deviation of a list of numbers
    # Note. The list of numbers is passed into
    # the function alongside a previously
    # calculated mean value for the list.
    def calcStdDev(self, numbers, mean):
        # Varible for total deviation
        deviation = 0.0
        # Variable for a single deviation
        singleDev = 0.0
        # Iterate through the list of numbers.
        for number in numbers:
            # Calculate a single Deviation
            singleDev = number-mean
            # Add the squared single deviation to
            # to the on going total.
            deviation += (singleDev**2)
        # Calcate the standard devaition
        stddev = sqrt(deviation/(len(numbers)-1))
        # return the standard deviation
        return stddev

    # The main thread of processing. A function
    # which defines the order of processing.
    # Note. The filename is passed in.
    def run(self, filename):
        # Open the input file
        inFile = open(filename, 'r')
        # Parse the file to retrieve a list of
        # numbers
        numbers = self.parseCommaFile(inFile)

        # Calculate the mean value of the list
        mean = self.calcMean(numbers)
        # Calculate the standard deviation of the
        # list.
        stddev = self.calcStdDev(numbers, mean)

        # Print the results to screen
        print('Mean: ' + str(mean))
        print('Stddev: ' + str(stddev))

        # Close the input file
        inFile.close()

# When python is executed python executes
# the code with the lowest indentation first.
#
# We can identify when python is executed from
# the command line using the following if statment.
#
# When executed we want the run() function to be
# executed therefore we create a CalcMeanStdDev
# object and call run on that object - passing
# in the file name of the file to be processed.
if __name__ == '__main__':
    obj = CalcMeanStdDev()
    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.

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

```

```
########################################
# An python class to hold maths operations
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
########################################

from math import sqrt

class MyMathsClass (object):

    def calcMean(self, numbers):
        sum = 0.0
        for number in numbers:
            sum += number
        mean = sum/len(numbers)
        return mean

    def calcStdDev(self, numbers, mean):
        deviation = 0.0
        singleDev = 0.0
        for number in numbers:
            singleDev = number-mean
            deviation += (singleDev**2)
        stddev = sqrt(deviation/(len(numbers)-1))
        return stddev


```

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

########################################
# An python class to parse a comma
# separates text file to calculate
# the mean and standard deviation
# of the inputted floating point
# numbers.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

# To import the class you have created
# you need to define the file within
# which the class is held. In this
# case MyMaths.py and the name of the
# class to be imported (i.e., MyMathsClass)
from MyMaths import MyMathsClass

class FileSummary (object):

    def parseCommaFile(self, file):
        floatingNumbers = list()
        for eachLine in file:
            substrs = eachLine.split(’,’,eachLine.count(’,’))
            for strVar in substrs:
                floatingNumbers.append(float(strVar))
        return floatingNumbers

    def run(self, filename):
        inFile = open(filename, ’r’)
        numbers = self.parseCommaFile(inFile)

        mathsObj = MyMathsClass()
        mean = mathsObj.calcMean(numbers)
        stddev = mathsObj.calcStdDev(numbers, mean)

        print(’Mean: ’ + str(mean))
        print(’Stddev: ’ + str(stddev))


if __name__ == ’__main__’:
    obj = FileSummary()
    obj.run(’randfloats.txt’)
```

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:

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

########################################
# An python class to parse a comma
# separates text file to calculate
# the mean and standard deviation
# of the inputted floating point
# numbers.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
########################################

from MyMaths import MyMathsClass
# To allow command line options to be
# retrieved the sys python library needs
# to be imported
import sys

class FileSummary (object):

    def parseCommaFile(self, file):
        floatingNumbers = list()
        for eachLine in file:
            substrs = eachLine.split(',',eachLine.count(','))
            for strVar in substrs:
                floatingNumbers.append(float(strVar))
        return floatingNumbers

    def run(self):
        # To retrieve the command line arguments
        # the sys.argv[X] is used where X refers to
        # the argument. The argument number starts
        # at 1 and is the index of a list.
        filename = sys.argv[1]
        inFile = open(filename, 'r')
        numbers = self.parseCommaFile(inFile)

        mathsObj = MyMathsClass()
        mean = mathsObj.calcMean(numbers)
        stddev = mathsObj.calcStdDev(numbers, mean)

        print('Mean: ' + str(mean))
        print('Stddev: ' + str(stddev))

if __name__ == '__main__':
    obj = FileSummary()
    obj.run()

```

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

Plotting - Matplotlib

4.1 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 matplotlib 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.

4.2 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.

```
#! /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')
```

4.3 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.`

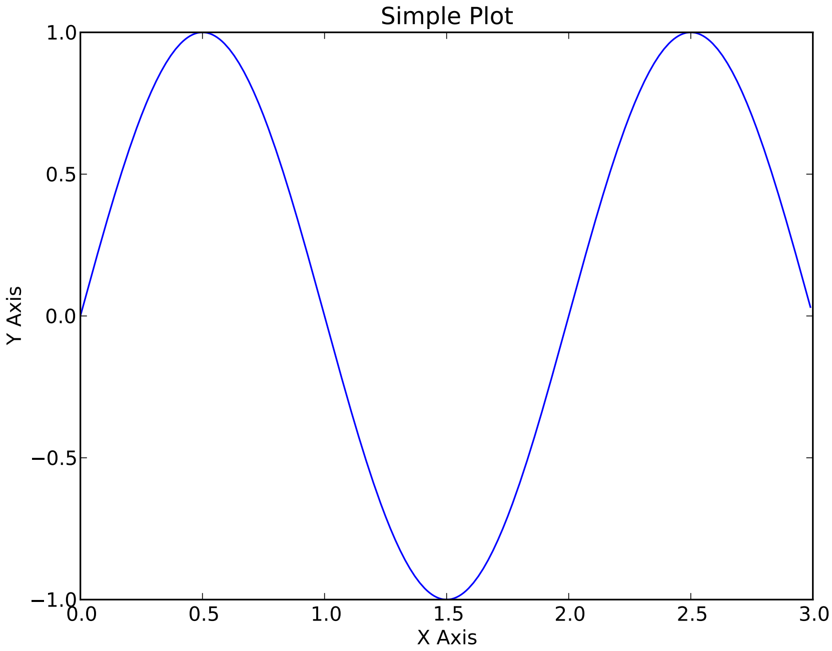

Figure 4.1: A simple plot using matplotlib.

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

```
#! /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]
```

```

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

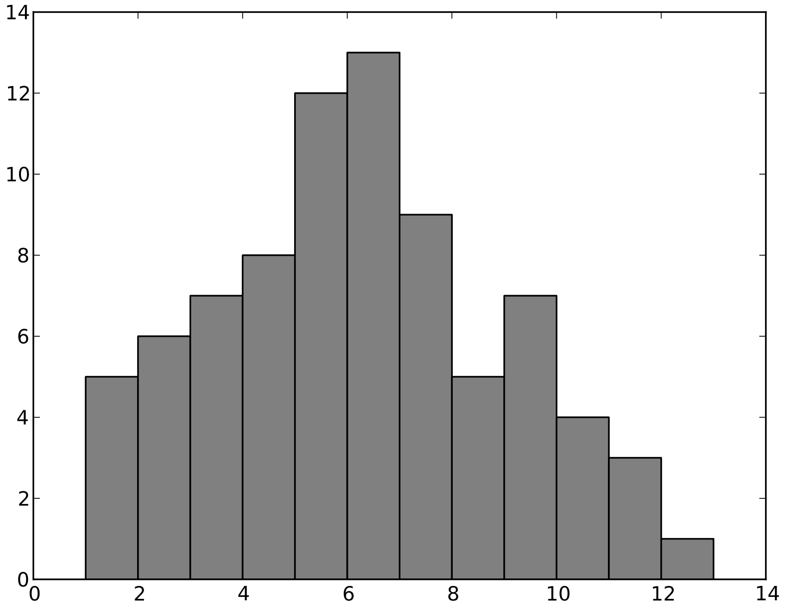

Figure 4.2: A simple bar chart using matplotlib.

4.4 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.

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

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

from matplotlib import pylab as plt

frac = [25, 33, 17, 10, 15]
labels = ['25', '33', '17', '10', '15']
explode = [0, 0.25, 0, 0, 0]

# Create pie chart
plt.pie(frac, explode, labels, shadow=True)
# Give it a title
plt.title('A Sample Pie Chart')
# save the plot to a PDF file
plt.savefig('pichart.pdf', dpi=200, format='PDF')
```

4.5 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.

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

#######################################
# A simple python script to display a
# scatter plot.
# Author: <YOUR NAME>
```

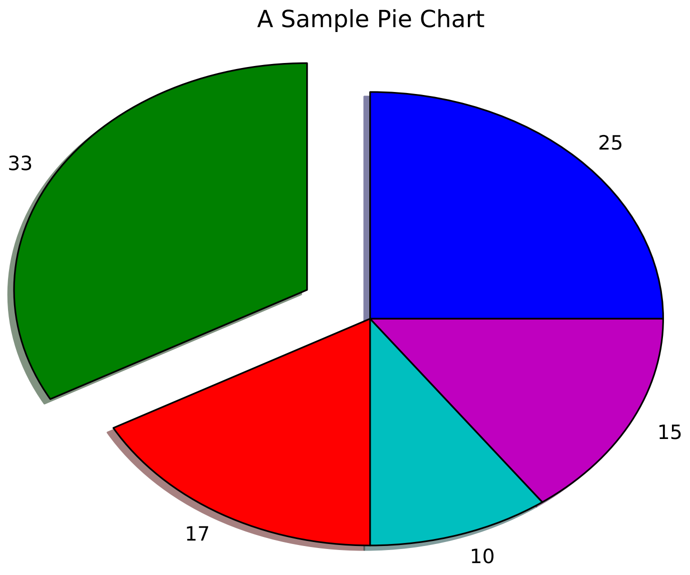

Figure 4.3: A simple pie chart using matplotlib.

```
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

from matplotlib import pylab as plt
# Import a random number generator
from random import random

x = []
y = []
z = []

# Create data values for X, Y, Z axis'
for i in range(5000):
    x.append(random() * 100)
    y.append(random() * 100)
    z.append(x[i]-y[i])

```

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

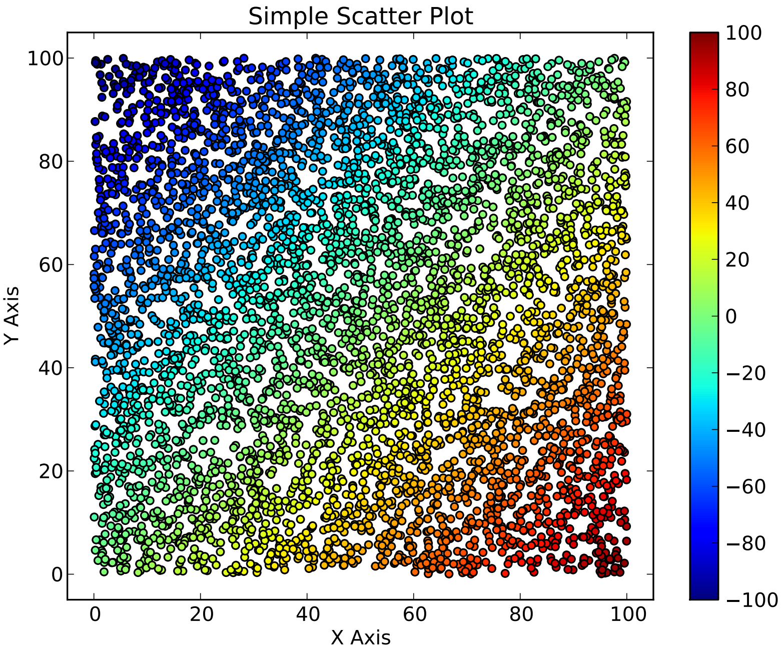

Figure 4.4: A simple scatter plot using matplotlib.

4.6 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 4.5) while the second plots them onto individual axis (Figure 4.6). Information on the use of the subplot() function can be found in the matplotlib documentation (`http://matplotlib.sourceforge.net/matplotlib.pylab.html#-subplot`).

```
#######################################
# A python script to read in a text file
# of rainfall data for summer and winter
# within the UK and display as a plot.
# Author: <YOUR NAME>
# Email: <YOUR EMAIL>
# Date: DD/MM/YYYY
# Version: 1.0
#######################################

from matplotlib import pylab as plt
import os.path
import sys

class PlotRainfall (object):

    # Parse the input file - Three columns year, summer, winter
    def parseDataFile(self, dataFile, year, summer, winter):
        line = 0
        for eachLine in dataFile:
            commaSplit = eachLine.split(',', eachLine.count(','))
            first = True
            for token in commaSplit:
                if first:
                    first = False
                else:
                    if line == 0:
                        year.append(int(token))
                    elif line == 1:
                        summer.append(float(token))
                    elif line == 2:
                        winter.append(float(token))
            line += 1

    # Plot data onto the same axis'
    def plotData(self, year, summer, winter, outFile):
        plt.figure()
        plt.plot(year, summer)
        plt.plot(year, winter)
        plt.legend(['Summer', 'Winter'])
        plt.xlabel('Year')
        plt.ylabel('Rainfall (5 Year Mean)')
        plt.title('Summer and Winter rainfall across the UK')
        # save plot to disk.
        plt.savefig(outFile, dpi=200, format='PDF')

    # Plot the data onto separate axis, using subplot.
    def plotDataSeparate(self, year, summer, winter, outFile):
        plt.figure()
        plt.subplot(2,1,1)
        plt.plot(year, summer)
        plt.ylabel('Rainfall (5 Year Mean)')
        plt.title('Summer rainfall across the UK')
        plt.axis('tight')

        plt.subplot(2,1,2)
        plt.plot(year, winter)
        plt.xlabel('Year')
        plt.ylabel('Rainfall (5 Year Mean)')
        plt.title('Winter rainfall across the UK')
        plt.axis('tight')
        # save plot to disk.
        plt.savefig(outFile, dpi=200, format='PDF')

    def run(self):
        filename = 'ukweatheraverage.csv'
        if os.path.exists(filename):
            year = list()
            summer = list()
            winter = list()
            try:
                dataFile = open(filename, 'r')
            except IOError as e:
                print('\nCould not open file:\n', e)
                return
            self.parseDataFile(dataFile, year, summer, winter)
            dataFile.close()
            self.plotData(year, summer, winter, "Rainfall_SinglePlot.pdf")
            self.plotDataSeparate(year, summer, winter, "Rainfall_MultiplePlots.pdf")
        else:
            print('File \'' + filename + '\' does not exist.')

if __name__ == '__main__':
    obj = PlotRainfall()
    obj.run()
```

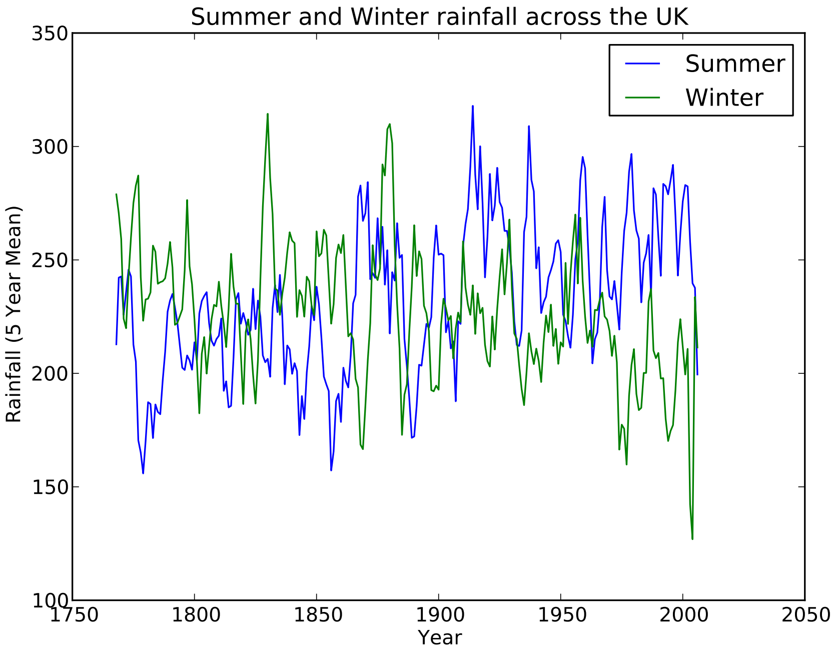

Figure 4.5: Rainfall data for summer and winter on the same axis'.

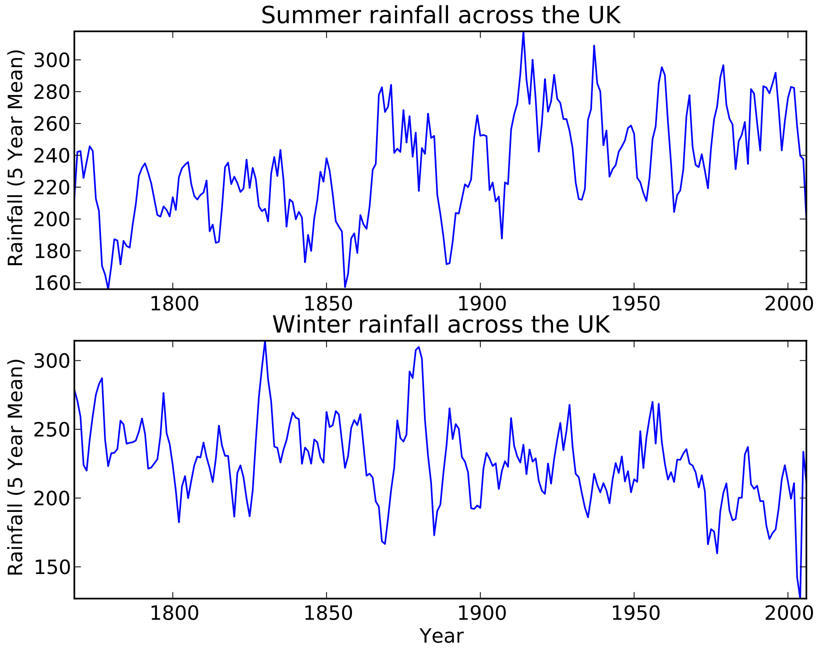

Figure 4.6: Rainfall data for summer and winter on different axis'.

4.7 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.

4.8 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 5

Statistics (SciPy / NumPy)

5.1 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)
```

5.2 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'
        outFileName = 'PenglaisWoodsStats.csv'
        inFile = open(inFileName, 'r')
        outFile = open(outFileName,'w')

        # Iterate through the input file and save diameter into
        # lists, based on species
        header = True
        for eachLine in inFile:
            if header: # Skip header row
                print('Skipping header row')
                header = False
            else:
                substrs = eachLine.split(',',eachLine.count(','))

                species = substrs[3]
                if substrs[4].isdigit: # Check diameter is a number
                    diameter = float(substrs[4])

                    if species == 'BEECH':
                        beechDiameter.append(diameter)
                    elif species == 'ASH':
                        ashDiameter.append(diameter)
                    elif species == 'BIRCH':
                        birchDiameter.append(diameter)
                    elif species == 'OAK':
                        oakDiameter.append(diameter)
                    elif species == 'SYC':
                        sycamoreDiameter.append(diameter)
                    else:
                        otherDiameter.append(diameter)

        # Convert input lists to NumPy arrays
        beechDiameter = numpy.array(beechDiameter)
        ashDiameter = numpy.array(ashDiameter)
        birchDiameter = numpy.array(birchDiameter)
        oakDiameter = numpy.array(oakDiameter)
        sycamoreDiameter = numpy.array(sycamoreDiameter)
        otherDiameter = numpy.array(otherDiameter)

        # Write header line to output file
        headerLine = 'species, meanDiameter, medianDiameter, stDevDiameter\n'
```

```python
        outFile.write(headerLine)

        # Calculate statistics for each species and write to file
        outLine = 'Beech,' + self.createStatsLine(beechDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Ash,' + self.createStatsLine(ashDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Birch,' + self.createStatsLine(birchDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Oak,' + self.createStatsLine(oakDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Sycamore,' + self.createStatsLine(sycamoreDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Other,' + self.createStatsLine(otherDiameter) + '\n'
        outFile.write(outLine)

        print('Statistics written to: ' + outFileName)


    def createStatsLine(self, inArray):
        # Calculate statsistics for NumPy array and return output line.
        meanArray = numpy.mean(inArray)
        medianArray = numpy.median(inArray)
        stDevArray = numpy.std(inArray)
        skewArray = spstats.skew(inArray)

        # Create output line with stats
        statsLine = str(meanArray) + ',' + str(medianArray) + ',' + str(stDevArray)
        return statsLine

if __name__ == '__main__':
    obj = CalculateStatistics()
    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.

5.2.1 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.

5.3 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:

$$\text{Volume} = a + bD^2h^{0.75} \tag{5.1}$$

Where D is diameter and h is height. The coefficients a and b vary according to species (see Table 5.1). From volume, it is possible to calculate biomass by multiplying by the specific gravity.

$$\text{Biomass} = \text{Volume} \times \text{SpecificGravity} \tag{5.2}$$

The specific gravity also varies by species, values for each species are given in Table 5.1.

Table 5.1: 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.

```
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.

```
def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
        if inSpecies == 'BEECH':
            a = 0.014306
            b = 0.0000748
            specificGravity = 0.56
        else: # Raise exception if species is not recognised
            raise Exception('Species not recognised')

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

```
def calcBiomassStatsLine(self, inDiameterArray, inHeightArray, inSpecies):
        # Calculates biomass, calculates stats from biomass and returns output line
        biomassStatsLine = ''
        try:
            # Calculate biomass
            biomass = self.calcBiomass(inDiameterArray, inHeightArray, inSpecies)
            # Calculate stats from biomass
            meanBiomass = numpy.mean(biomass)
            medianBiomass = numpy.median(biomass)
            stDevBiomass = numpy.std(biomass)

            # Create output line
            biomassStatsLine = str(meanBiomass) + ',' + str(medianBiomass) + ',' + \
str(stDevBiomass)

        except Exception:
            # Catch exception and write 'na' for all values
            biomassStatsLine = 'na,na,na'

        return biomassStatsLine
```

Therefore, the final script should result in the following:

```
#! /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 and heights
        # A seperate list is used for each species
        beechDiameter = list()
        beechHeight = list()
        ashDiameter = list()
        ashHeight = list()
        birchDiameter = list()
        birchHeight = list()
        oakDiameter = list()
        oakHeight = list()
        sycamoreDiameter = list()
        sycamoreHeight = list()
        otherDiameter = list()
        otherHeight = list()

        # Open input and output files
        inFileName = ’PenglaisWoodsData.csv’
        outFileName = ’PenglaisWoodsStats.csv’
        inFile = open(inFileName, ’r’)
        outFile = open(outFileName,’w’)

        # Iterate through the input file and save diameter and height
        # into lists, based on species
        header = True
        for eachLine in inFile:
            if header: # Skip header row
                print(’Skipping header row’)
                header = False
            else:
                substrs = eachLine.split(’,’,eachLine.count(’,’))

                species = substrs[3]
                if substrs[4].isdigit: # Check diameter is a number
                    diameter = float(substrs[4])
                    height = float(substrs[10])

                    if species == ’BEECH’:
                        beechDiameter.append(diameter)
                        beechHeight.append(height)
                    elif species == ’ASH’:
                        ashDiameter.append(diameter)
                        ashHeight.append(height)
                    elif species == 'BIRCH':
                        birchDiameter.append(diameter)
                        birchHeight.append(height)
                    elif species == 'OAK':
                        oakDiameter.append(diameter)
                        oakHeight.append(height)
                    elif species == 'SYC':
                        sycamoreDiameter.append(diameter)
                        sycamoreHeight.append(height)
                    else:
                        otherDiameter.append(diameter)
                        otherHeight.append(height)

        # Convert to NumPy arrays
        beechDiameter = numpy.array(beechDiameter)
        ashDiameter = numpy.array(ashDiameter)
        birchDiameter = numpy.array(birchDiameter)
        oakDiameter = numpy.array(oakDiameter)
        sycamoreDiameter = numpy.array(sycamoreDiameter)
        otherDiameter = numpy.array(otherDiameter)

        beechHeight = numpy.array(beechHeight)
        ashHeight = numpy.array(ashHeight)
        birchHeight = numpy.array(birchHeight)
        oakHeight = numpy.array(oakHeight)
        sycamoreHeight = numpy.array(sycamoreHeight)
        otherHeight = numpy.array(otherHeight)

        # Write header line to output file
        headerLine = 'species,meanDiameter,medianDiameter,stDevDiameter,\
meanHeight,medianHeight,stDevHeight,\
meanBiomass,medianBiomass,stDevBiomass\n'
        outFile.write(headerLine)

        # Calculate statistics and biomass for each species and write to file
        outLine = 'Beech,' + self.createStatsLine(beechDiameter) + ',' + \
self.createStatsLine(beechHeight) + ',' + \
self.calcBiomassStatsLine(beechDiameter, beechHeight, 'BEECH') + '\n'
        outFile.write(outLine)
        outLine = 'Ash,' + self.createStatsLine(ashDiameter)  + ',' +  \
self.createStatsLine(ashHeight) + ',' + \
self.calcBiomassStatsLine(ashDiameter, ashHeight, 'ASH') + '\n'
        outFile.write(outLine)
        outLine = 'Birch,' + self.createStatsLine(birchDiameter) + ',' + \
self.createStatsLine(birchHeight) + ',' +  \
self.calcBiomassStatsLine(birchDiameter, birchHeight, 'BIRCH') + '\n'
        outFile.write(outLine)
        outLine = 'Oak,' + self.createStatsLine(oakDiameter)  + ',' + \
self.createStatsLine(oakHeight) + ',' +  \
self.calcBiomassStatsLine(oakDiameter, oakHeight, 'OAK') + '\n'
        outFile.write(outLine)
        outLine = 'Sycamore,' + self.createStatsLine(sycamoreDiameter)  + ',' + \
self.createStatsLine(sycamoreHeight) + ',' +  \
self.calcBiomassStatsLine(sycamoreDiameter, sycamoreHeight, 'SYC') + '\n'
        outFile.write(outLine)
        outLine = 'Other,' + self.createStatsLine(otherDiameter)  + ',' + \
self.createStatsLine(otherHeight) + ',' +  \
self.calcBiomassStatsLine(otherDiameter, otherHeight, 'Other') + '\n'
        outFile.write(outLine)

        print('Statistics written to: ' + outFileName)

    def createStatsLine(self, inArray):
        # Calculate statsistics for array and return output line.
        meanArray = numpy.mean(inArray)
        medianArray = numpy.median(inArray)
        stDevArray = numpy.std(inArray)

        # Create output line with stats
        statsLine = str(meanArray) + ',' + str(medianArray) + ',' + str(stDevArray)
        return statsLine

    def calcBiomassStatsLine(self, inDiameterArray, inHeightArray, inSpecies):
        # Calculates biomass, calculates stats from biomass and returns output line
        biomassStatsLine = ''
        try:
            # Calculate biomass
            biomass = self.calcBiomass(inDiameterArray, inHeightArray, inSpecies)
            # Calculate stats from biomass
            meanBiomass = numpy.mean(biomass)
            medianBiomass = numpy.median(biomass)
            stDevBiomass = numpy.std(biomass)

            # Create output line
            biomassStatsLine = str(meanBiomass) + ',' + str(medianBiomass) + ',' + \
str(stDevBiomass)

        except Exception:
            # Catch exception and write 'na' for all values
            biomassStatsLine = 'na,na,na'

        return biomassStatsLine

    def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
        if inSpecies == 'BEECH':
            a = 0.014306
            b = 0.0000748
            specificGravity = 0.56
        else: # Raise exception is species is not recognised
            raise Exception('Species not recognised')

        # Calcualte volume
        volume = a + ((b*(inDiameterArray)**2) * (inHeightArray**0.75))
        # Calculate biomass
        biomass = volume * specificGravity
        # Return biomass
        return biomass

if __name__ == '__main__':
    obj = CalculateStatistics()
    obj.run()
```

5.3.1 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.

5.4 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:

$$y = ax + b \tag{5.3}$$

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.

$$\text{height} = a \log\left(\text{diameter}\right) + b \tag{5.4}$$

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.

```
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)
        # and r squared value
        # Coefficients are rounded to two decimal places.
        equation = str(round(aCoeff,2)) + 'log(D) + ' + str(round(bCoeff,2)) + \
                   ' (r$^2$ = ' + str(round(rVal**2,2)) + ')'

        # Plot fit against origional data
        plt.plot(inDiameterArray, inHeightArray,'.')
        plt.plot(testDiameter, predictHeight)
        plt.xlabel('Diameter (cm)')
        plt.ylabel('Height (m)')
        plt.legend(['measured data',equation])

        # Save plot
        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$∧2$.

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:

```
#! /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
#######################################

```

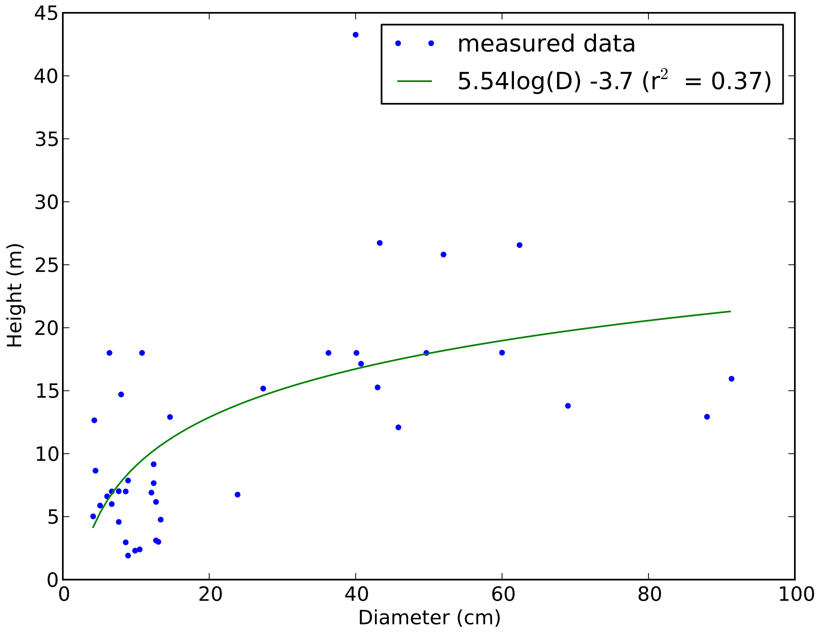

Figure 5.1: A simple plot using matplotlib.

```
import numpy
import scipy
# Import scipy stats functions we need
import scipy.stats as spstats
# Import plotting library as plt
import matplotlib.pyplot as plt

class CalculateStatistics (object):

    def run(self):
        # Set up lists to hold input diameters and heights
        # A seperate list is used for each species
        beechDiameter = list()
        beechHeight = list()
        ashDiameter = list()
        ashHeight = list()
        birchDiameter = list()
        birchHeight = list()
        oakDiameter = list()
        oakHeight = list()
        sycamoreDiameter = list()
        sycamoreHeight = list()
        otherDiameter = list()
        otherHeight = list()

        # Open input and output files
        inFileName = 'PenglaisWoodsData.csv'
        outFileName = 'PenglaisWoodsStats.csv'
        inFile = open(inFileName, 'r')
        outFile = open(outFileName,'w')

        # Iterate through the input file and save diameter and height
        # into lists, based on species
        header = True
        for eachLine in inFile:
            if header: # Skip header row
                print('Skipping header row')
                header = False
            else:
                substrs = eachLine.split(',',eachLine.count(','))

                species = substrs[3]
                if substrs[4].isdigit: # Check diameter is a number
                    diameter = float(substrs[4])
                    height = float(substrs[10])

                    if species == 'BEECH':
                        beechDiameter.append(diameter)
                        beechHeight.append(height)
                    elif species == 'ASH':
                        ashDiameter.append(diameter)
                        ashHeight.append(height)
                    elif species == 'BIRCH':
                        birchDiameter.append(diameter)
                        birchHeight.append(height)
                    elif species == 'OAK':
                        oakDiameter.append(diameter)
                        oakHeight.append(height)
                    elif species == 'SYC':
                        sycamoreDiameter.append(diameter)
                        sycamoreHeight.append(height)
                    else:
                        otherDiameter.append(diameter)
                        otherHeight.append(height)

        # Convert to NumPy arrays
        beechDiameter = numpy.array(beechDiameter)
        ashDiameter = numpy.array(ashDiameter)
        birchDiameter = numpy.array(birchDiameter)
        oakDiameter = numpy.array(oakDiameter)
        sycamoreDiameter = numpy.array(sycamoreDiameter)
        otherDiameter = numpy.array(otherDiameter)

        beechHeight = numpy.array(beechHeight)
        ashHeight = numpy.array(ashHeight)
        birchHeight = numpy.array(birchHeight)
        oakHeight = numpy.array(oakHeight)
        sycamoreHeight = numpy.array(sycamoreHeight)
        otherHeight = numpy.array(otherHeight)

        # Write header line to output file
        headerLine = 'species, meanDiameter, medianDiameter, stDevDiameter\n'
        outFile.write(headerLine)

        # Calculate statistics for each species and write to file
        outLine = 'Beech,' + self.createStatsLine(beechDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Ash,' + self.createStatsLine(ashDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Birch,' + self.createStatsLine(birchDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Oak,' + self.createStatsLine(oakDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Sycamore,' + self.createStatsLine(sycamoreDiameter) + '\n'
        outFile.write(outLine)
        outLine = 'Other,' + self.createStatsLine(otherDiameter) + '\n'
        outFile.write(outLine)

        print('Statistics written to: ' + outFileName)

        # Fit line to each file and save out plot

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

        # Plot linear regression for Beech
        print('Generating plot:')
        self.plotLinearRegression(beechDiameter, beechHeight, outDIR + 'beech.pdf')

    def plotLinearRegression(self, inDiameterArray, inHeightArray, outPlotName):
        # Perform fit
        (aCoeff,bCoeff,rVal,pVal,stdError) = spstats.linregress(numpy.log(inDiameterArray), inHei

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

        # Create a string, showing the form of the equation (with fitted coefficients)
        # and r squared value
        # Coefficients are rounded to two decimal places.
        equation = str(round(aCoeff,2)) + 'log(D) ' + str(round(bCoeff,2)) + \
        ' (r$^2$ = ' + str(round(rVal**2,2)) + ')'

        # Plot fit against origional data
        plt.plot(inDiameterArray, inHeightArray,'.')
        plt.plot(testDiameter, predictHeight)
        plt.xlabel('Diameter (cm)')
        plt.ylabel('Height (m)')
        plt.legend(['measured data',equation])

        # Save plot
        plt.savefig(outPlotName, dpi=200, format='PDF')


    def createStatsLine(self, inArray):
        # Calculate statsistics for array and return output line.
        meanArray = numpy.mean(inArray)
        medianArray = numpy.median(inArray)
        stDevArray = numpy.std(inArray)

        # Create output line with stats
        statsLine = str(meanArray) + ',' + str(medianArray) + ',' + str(stDevArray)
        return statsLine

    def calcBiomassStatsLine(self, inDiameterArray, inHeightArray, inSpecies):
        # Calculates biomass, calculates stats from biomass and returns output line
        biomassStatsLine = ''
        try:
            # Calculate biomass
            biomass = self.calcBiomass(inDiameterArray, inHeightArray, inSpecies)
            # Calculate stats from biomass
            meanBiomass = numpy.mean(biomass)
            medianBiomass = numpy.median(biomass)
            stDevBiomass = numpy.std(biomass)

            # Create output line
            biomassStatsLine = str(meanBiomass) + ',' + str(medianBiomass) + ','\
            + str(stDevBiomass)

        except Exception:
            # Catch exception and write 'na' for all values
            biomassStatsLine = 'na,na,na'

        return biomassStatsLine

    def calcBiomass(self, inDiameterArray, inHeightArray, inSpecies):
        if inSpecies == 'BEECH':
            a = 0.014306
            b = 0.0000748
            specificGravity = 0.56
        else: # Raise exception is species is not recognised
            raise Exception('Species not recognised')

        # Calcualte volume
        volume = a + ((b*(inDiameterArray)**2) * (inHeightArray**0.75))
        # Calculate biomass
        biomass = volume * specificGravity
        # Return biomass
        return biomass

if __name__ == '__main__':
    obj = CalculateStatistics()
    obj.run()
```

5.4.1 Exercise

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

5.5 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 6

Batch Processing Command Line Tools

6.1 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.

6.2 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. Get the list of input shapefiles.
3. Iterate through the files and run the required commands.

Look through the following script and see how the `glob` module (`https://docs.python.org/3/library/glob.html`) is used to easily retrieve the list of input shapefiles.

```
#! /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.path
import subprocess
import glob

# Define the input directory
inputShpFiles = ’./TreeCrowns/*.shp’
# Define the output file
newSHPFile = ’Merged_TreeCrowns.shp’

# ogr2ogr requires that the layer name is provided, for
# a shapefile this is the file name without the path or
# extention. The following commands use the os.path
# module to remove those if present.

# Remove the directory path from the output file.
newSHPFileBaseName = os.path.basename(newSHPFile)
# Remove the file extention from the output file.
newSHPFileBaseName = os.path.splitext(newSHPFileBaseName)[0]

# Get list of input shapefiles using glob module.
fileList = glob.glob(inputShpFiles)

# Variable used to identify the first file
first = True
# A string for the command to be built
command = ’’
# Iterate through the files.
for file in fileList:
   if first:
           # If the first file make a copy to create the output file
           command = ’ogr2ogr ’ + newSHPFile + ’ ’ + file
           first = False
   else:
           # Otherwise append the current shapefile to the output file
```

```
            command = 'ogr2ogr -update -append ' + newSHPFile + ' '  + \
            file + ' -nln ' + newSHPFileBaseName
    # Execute the current command
    print(command)
    subprocess.call(command,shell=True)
```

6.3 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.

6.4 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).

```
if numArgs == 3:
        # Define the input file directory
        inputShpFiles = sys.argv[1] + "*.shp"
        # Define the output file
        newSHPFile = sys.argv[2]

        #
        # Rest of script is in here...
        #
else:
        print('ERROR. Command should have the form:')
        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.

6.5 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).

6.6 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 7

Image Processing using GDAL and RIOS

7.1 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.

7.1.1 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
# Import the python Argument parser
import argparse

# Create the 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 output file.
parser.add_argument("-o", "--output", type=str, help="Specify the output text file.")
# Call the parser to parse the arguments.
args = parser.parse_args()
```

7.1.2 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.

```
#!/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
        # geotransform[5] = Y Pixel Resolution
        geotransform = dataset.GetGeoTransform()
        # Check that the tranformation has been correctly read.
        if not geotransform is None:
            # Print out the Origin, Pixel Size and Rotation.
            print(’Origin = (’,geotransform[0], ’,’,geotransform[3],’)’)
            print(’Pixel Size = (’,geotransform[1], ’,’,geotransform[5],’)’)
            print(’Rotation = (’,geotransform[2], ’,’,geotransform[4],’)’)
        else:
            # Provide an error message is the transform has not been
            # correctly read.
            print("Could not find a geotransform for image file ", inputFile)
    else:
        # Provide an error message if the input image 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,
                        help="Specify the input image file.")
    # 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()
    # Otherwise, run the function to print out the image header information.
    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 )
```

7.1.3 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
#!/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 setNoData(inputFile, noDataVal):
    # 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:
        # Iterate throught he image bands
        # Note. i starts at 0 while the
        # band count in GDAL starts at 1.
        for i in range(dataset.RasterCount):
            # Print information to the user on what is
            # being set.
            print("Setting No data (" + str(noDataVal) + ") for band " + str(i+1))
            # Get the image band
            # the i+1 is because GDAL bands
            # start with 1.
            band = dataset.GetRasterBand(i+1)
            # Set the no data value.
            band.SetNoDataValue(noDataVal)
    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,
                        help="Specify the input image file.")
    # Define the argument for specifying the no data value.
    parser.add_argument("-n", "--nodata", type=float, default=0,
                        help="Specify the no data value to be set.")
    # 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: not input image file provided.")
        sys.exit()
    # Otherwise, run the function to set the no
    # data value.
    setNoData(args.input, args.nodata)
```

7.1.4 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.

```
#!/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,
                        help="Specify the input image file.")
    # Define the argument for specifying image band.
    parser.add_argument("-b", "--band", type=int,
                        help="Specify image band.")
    # Define the argument for specifying band name.
    parser.add_argument("-n", "--name", type=str,
                        help="Specify the band 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 band parameter has been specified.
    if args.band == None:
        # Print an error message if not and exit.
        print("Error: the band was not specified.")
        sys.exit()

    # Check that the name parameter has been specified.
    if args.name == None:
        # Print an error message if not and exit.
        print("Error: the band name was not specified.")
        sys.exit()

    # Otherwise, run the function to set the band
    # name.
    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.

7.1.5 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.

```
#!/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 read a meta-data item
# from a image band
def readBandMetaData(inputFile, band, name):
    # Open the dataset in Read Only mode
    dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
    # 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:
            # Get the meta-data value specified.
            metaData = imgBand.GetMetadataItem(name)
            # Check that it is present
            if metaData == None:
                # If not present, print error.
                print("Could not find \'", name, "\' item.")
            else:
                # Else print out the metaData value.
                print(name, " = \'", metaData, "\'")
        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)

# A function to read a meta-data item
# from a image
def readImageMetaData(inputFile, name):
    # Open the dataset in Read Only mode
    dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
    # Check that the image  has been opened.
    if not dataset is None:
        # Get the meta-data value specified.
        metaData = dataset.GetMetadataItem(name)
        # Check that it is present
        if metaData == None:
            # If not present, print error.
            print("Could not find \'", name, "\' item.")
        else:
            # Else print out the metaData value.
            print(name, " = \'", metaData, "\'")
    else:
        # Print an error message if the file
        # could not be opened.
        print("Could not open the input image file: ", inputFile)

# A function to read a meta-data item
# from a image band
```

```
def listBandMetaData(inputFile, band):
    # Open the dataset in Read Only mode
    dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
    # 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:
            # Get the meta-data dictionary
            metaData = imgBand.GetMetadata_Dict()
            # Check it has entries.
            if len(metaData) == 0:
                # If it has no entries return
                # error message.
                print("There is no image meta-data.")
            else:
                # Otherwise, print out the
                # meta-data.
                # Loop through each entry.
                for metaItem in metaData:
                    print(metaItem)
        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)

# A function to read a meta-data item
# from a image
def listImageMetaData(inputFile):
    # Open the dataset in Read Only mode
    dataset = gdal.Open(inputFile, gdal.GA_ReadOnly)
    # Check that the image  has been opened.
    if not dataset is None:
        # Get the meta-data dictionary
        metaData = dataset.GetMetadata_Dict()
        # Check it has entries.
        if len(metaData) == 0:
            # If it has no entries return
            # error message.
            print("There is no image meta-data.")
        else:
            # Otherwise, print out the
            # meta-data.
            # Loop through each entry.
            for metaItem in metaData:
                print(metaItem)
    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,
                        help="Specify the input image file.")
    # Define the argument for specifying image band.
    parser.add_argument("-b", "--band", type=int, default=0,
                        help="Specify image band.")
    # Define the argument for specifying meta-data name.
    parser.add_argument("-n", "--name", type=str,
                        help="Specify the meta-data name.")
    # Define the argument for specifying whether the
    # meta-data field should be just listed.
    parser.add_argument("-l", "--list", action="store_true", default=False,
                        help="Specify that meta data items should be listed.")
    # 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 name parameter has been specified.
    # If it has been specified then run functions to
    # read band meta-data.
    if not args.name == None:
        # Check whether the image band has been specified.
        # the default was set at 0 (to indicate that it)
        # hasn't been specified as GDAL band count starts
        # at 1. This also means the user cannot type in
        # a value of 0 and get an error.
        if args.band == 0:
            # Run the function to print out the image meta-data value.
            readImageMetaData(args.input, args.name)
        else:
            # Otherwise, run the function to print out the band meta-data value.
            readBandMetaData(args.input, args.band, args.name)
    elif args.list:
        if args.band == 0:
            # Run the function to list image meta-data.
            listImageMetaData(args.input)
        else:
            # Otherwise, run the function to list band meta-data.
            listBandMetaData(args.input, args.band)
    else:
        # Print an error message if not and exit.
        print("Error: the meta-data name or list option" + \
              " need to be specified was not specified.")
        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
```

7.2 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`

7.2.1 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.

7.2.2 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.

7.2.3 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.

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

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

```
from rios import applier
from rios import cuiprogress

# Define the applier function
def mutliplyByValue(info, inputs, outputs, otherargs):
    # Multiple the image1 by the scale factor
    outputs.outimage = inputs.image1 * otherargs.scale

# 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 output file.
    parser.add_argument("-o", "--output", type=str,
                        help="Specify the output image file.")
    # Define the argument for multiply the image by.
    parser.add_argument("-m", "--multiply", default=1.0, type=float,
                        help="Multiple the image by.")
    # 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 output parameter has been specified.
    if args.output == None:
        # Print an error message if not and exit.
        print("Error: No output image file provided.")
        sys.exit()

    # Create input files file names associations
    infiles = applier.FilenameAssociations()
    # Set image1 to the input image specified
    infiles.image1 = args.input
    # Create output files file names associations
    outfiles = applier.FilenameAssociations()
    # Set outImage to the output image specified
    outfiles.outimage = args.output
    # Create other arguments object
    otherargs = applier.OtherInputs()
    # Define the scale arguments
    otherargs.scale = args.multiply
    # Create a controls objects
    aControls = applier.ApplierControls()
    # Set the progress object.
    aControls.progress = cuiprogress.CUIProgressBar()

    # Apply the multiply function.
    applier.apply(mutliplyByValue,
                  infiles,
                  outfiles,
                  otherargs,
                  controls=aControls)

```

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
```

7.2.4 Calculate NDVI

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

$$\text{NDVI} = \frac{\text{NIR} - \text{RED}}{\text{NIR} + \text{RED}} \tag{7.1}$$

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).

```
#!/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)
    # In this case an extra dimension representing
    # the single image band is required.
    outputs.outimage = numpy.expand_dims(out, axis=0)

# 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 output file.
    parser.add_argument("-o", "--output", type=str,
                        help="Specify the output image file.")
    # Define the argument for specifying the red image band
    parser.add_argument("-r", "--red", type=int,
                        help="Specifiy red band.")
    # Define the argument for specifying the NIR image band
    parser.add_argument("-n", "--nir", type=int,
                        help="Specifiy NIR band.")
    # 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 output parameter has been specified.
    if args.output == None:
        # Print an error message if not and exit.
        print("Error: No output image file provided.")
        sys.exit()

    # Create input files file names associations
    infiles = applier.FilenameAssociations()
    # Set image1 to the input image specified
    infiles.image1 = args.input
    # Create output files file names associations
    outfiles = applier.FilenameAssociations()
    # Set outImage to the output image specified
    outfiles.outimage = args.output
    # Create other arguments object
    otherargs = applier.OtherInputs()
    # Define the red band argument
    otherargs.redband = args.red-1
    # Define the NIR band argument
    otherargs.nirband = args.nir-1
    # Create a controls objects
    aControls = applier.ApplierControls()
    # Set the progress object.
    aControls.progress = cuiprogress.CUIProgressBar()

    # Apply the multiply function.
    applier.apply(mutliplyByValue,
                  infiles,
                  outfiles,
                  otherargs,
                  controls=aControls)

```

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
```

7.2.5 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.

```
#!/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.
    parser.add_argument("-o", "--output", type=str,
                        help="Specify the output image file.")
    # Define the argument for specifying the red image band
    parser.add_argument("-r", "--red", type=str,
                        help="Specifiy red input image file.")
    # Define the argument for specifying the NIR image band
    parser.add_argument("-n", "--nir", type=str,
                        help="Specifiy NIR input image file.")
    # Call the parser to parse the arguments.
    args = parser.parse_args()

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

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

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

    # Create input files file names associations
    infiles = applier.FilenameAssociations()
    # Set images to the input image specified
    infiles.image1 = args.red
    infiles.image2 = args.nir
    # Create output files file names associations
    outfiles = applier.FilenameAssociations()
    # Set outImage to the output image specified
    outfiles.outimage = args.output
    # Create a controls objects
    aControls = applier.ApplierControls()
    # Set the progress object.
    aControls.progress = cuiprogress.CUIProgressBar()

    # Apply the multiply function.
    applier.apply(mutliplyByValue,
                  infiles,
                  outfiles,
                  controls=aControls)

```

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
```

7.3 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.

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

import sys
# Import the python Argument parser
import argparse
# Import the scipy filters.
from scipy import ndimage
#Import the numpy library
import numpy
# Import the RIOS image reader
from rios.imagereader import ImageReader
# Import the RIOS image writer
from rios.imagewriter import ImageWriter

# Define the function to iterate through
# the image.
def applyMedianFilter(inputFile, outputFile, fSize):
    # Get half the filter size, overlap between blocks
    hSize = (fSize-1)/2
    # Create the image reader for the input file
    # and set the overlap to be half the image
    # filter size.
    reader = ImageReader(inputFile, overlap=hSize)
    # Define the image writer but cannot create
    # until within the loop as this need the
    # information within the info object.
    writer = None
    # Loop through all the image blocks within
    # the reader.
    for (info, block) in reader:
        # Create an output block of
        # the same size as the input
        out = numpy.zeros_like(block)
        # Iterate through the image bands
        for i in range(len(out)):
            # Use scipy to run a median filter
            # on the image band data. The image
            # bands are filtered in turn
            ndimage.median_filter(block[i],size=fSize,output=out[i])
        # If it is the first time through the loop
        # (i.e., writer has a value of None) then
        # create the loop.
        if writer is None:
            # Create the writer for output image.
            writer = ImageWriter(outputFile,
                                 info=info,
                                 firstblock=out,
                                 drivername='HFA')
        else:
            # If the writer is created write the
            # output block to the file.
            writer.write(out)
    # Close the writer and calculate
    # the image statistics.
    writer.close(calcStats=True)

# 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 output file.
    parser.add_argument("-o", "--output", type=str,
                        help="Specify the output image file.")
    # Define the argument for the size of the image filter.
    parser.add_argument("-s", "--size", default=3, type=int,
                        help="Filter size.")
    # 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 output parameter has been specified.
    if args.output == None:
        # Print an error message if not and exit.
        print("Error: No output image file provided.")
        sys.exit()

    # Call the function to execute a median filter
    # on the input image.
    applyMedianFilter(args.input, args.output, args.size)

```

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?

7.4 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 8) 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.
    outputs.outimage = numpy.expand_dims(out, axis=0)

# A function to define the image as thematic
def setThematic(imageFile):
    # Use GDAL to open the dataset
    ds = gdal.Open(imageFile, gdal.GA_Update)
    # Iterate through the image bands
    for bandnum in range(ds.RasterCount):
        # Get the image band
        band = ds.GetRasterBand(bandnum + 1)
        # Define the meta-data for the LAYER_TYPE
        band.SetMetadataItem('LAYER_TYPE', 'thematic')

# 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 output file.
    parser.add_argument("-o", "--output", type=str,
                        help="Specify the output image file.")
    # 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 output parameter has been specified.
    if args.output == None:
        # Print an error message if not and exit.
        print("Error: No output image file provided.")
        sys.exit()

    # Create input files file names associations
    infiles = applier.FilenameAssociations()
    # Set image1 to the input image specified
    infiles.image1 = args.input
    # Create output files file names associations
    outfiles = applier.FilenameAssociations()
    # Set outImage to the output image specified
    outfiles.outimage = args.output
    # Create a controls objects
    aControls = applier.ApplierControls()
    # Specify that stats shouldn't be calc'd
    aControls.calcStats = False
    # Set the progress object.
    aControls.progress = cuiprogress.CUIProgressBar()

    # Apply the classifier function.
    applier.apply(rulebaseClassifier,
                  infiles,
                  outfiles,
                  controls=aControls)

    # Set the output file to be thematic
    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
```

7.5 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.

7.6 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 8

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.

8.1 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.

```
#!/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.")
        sys.exit()

    # Run the function read and print the
    # RAT column.
    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
```

8.2 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.

8.2.1 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.

```
#!/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

# The applier function to multiply the input
# column by 2.
def multiplyRATCol(imageFile, inColName, outColName):
    # Read the input column
    col = rat.readColumn(imageFile, inColName)
    # Muliply the column by 2.
    col = col * 2
    # Write the output column to the file.
    rat.writeColumn(imageFile, outColName, col)

# 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 input column name.
    parser.add_argument("-c", "--inname", type=str,
                        help="Specify the input column name.")
    # Define the argument for specifying the output column name.
    parser.add_argument("-o", "--outname", type=str,
                        help="Specify the output 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.inname == None:
        # Print an error message if not and exit.
        print("Error: No input RAT column name provided.")
        sys.exit()
    # Check that the input parameter has been specified.
    if args.outname == None:
        # Print an error message if not and exit.
        print("Error: No output RAT column name provided.")
        sys.exit()

    # Otherwise, run the function to multiply the input column by 2.
    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
```

8.2.2 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.

```
#!/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
# Import the numpy library
import numpy

# A function to add a colour table.
def addClassNames(imageFile):
    histo = rat.readColumn(imageFile, "Histogram")
    className = numpy.empty_like(histo, dtype=numpy.dtype('a255'))
    className[...] = "NA"
    className[0] = "Other Vegetation"
    className[1] = "Low Woody Vegetation"
    className[2] = "Water"
    className[3] = "Sparse Vegetation"
    className[4] = "Tall Woody Vegetation"
    # Write the output column to the file.
    rat.writeColumn(imageFile, "ClassNames", className)

# 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.")
    # 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()

    # Run the add class names function
    addClassNames(args.input)
```

Run the Script

Run the script as follows, use the classification you did at the end of worksheet 7.

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

8.3 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).

```
#!/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
# Import the numpy library
import numpy

# A function to add a colour table.
def addColourTable(imageFile):
    # Create a colour table (n,5) where
    # n is the number of classes to be
    # coloured. The data type must be
    # of type integer.
    ct = numpy.zeros([5,5], dtype=numpy.int)

    # Set 0 to be Dark Mustard Yellow.
    ct[0][0] = 0   # Pixel Val
    ct[0][1] = 139 # Red
    ct[0][2] = 139 # Green
    ct[0][3] = 0   # Blue
    ct[0][4] = 255 # Opacity

    # Set 1 to be Dark Olive Green.
    ct[1][0] = 1   # Pixel Val
    ct[1][1] = 162 # Red
    ct[1][2] = 205 # Green
    ct[1][3] = 90  # Blue
    ct[1][4] = 255 # Opacity

    # Set 2 to be Royal Blue.
    ct[2][0] = 2   # Pixel Val
    ct[2][1] = 72  # Red
    ct[2][2] = 118 # Green
    ct[2][3] = 255 # Blue
    ct[2][4] = 255 # Opacity

    # Set 3 to be Dark Sea Green.
    ct[3][0] = 3   # Pixel Val
    ct[3][1] = 180 # Red
    ct[3][2] = 238 # Green
    ct[3][3] = 180 # Blue
    ct[3][4] = 255 # Opacity

    # Set 4 to be Forest Green.
    ct[4][0] = 4   # Pixel Val
    ct[4][1] = 34 # Red
    ct[4][2] = 139 # Green
    ct[4][3] = 34  # Blue
    ct[4][4] = 255 # Opacity

    rat.setColorTable(imageFile, ct)



# 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.")
    # 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()

    # Run the add colour table function
    addColourTable(args.input)
```

Run the Script

Run the script as follows, use the classification you did at the end of worksheet 7.

```
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`).

8.4 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`