
GeogG122 Documentation

Release 1.0

Prof. P. Lewis

September 30, 2014

CONTENTS

Contents:

WELCOME TO GEOGG122: SCIENTIFIC COMPUTING

1.1 Course information

1.1.1 Course Convenor

Prof P. Lewis

1.1.2 Course Staff

Prof P. Lewis Dr. J. Gomez-Dans

1.2 Purpose of this course

This course, GeogG122 Scientific Computing, is a term 1 MSc module worth 15 credits (25% of the term 1 credits) that aims to:

- impart an understanding of scientific computing
- give students a grounding in the basic principles of algorithm development and program construction
- to introduce principles of computer-based image analysis and model development

It is open to students from a number of [MSc courses](#) run by the [Department of Geography](#) UCL, but the material should be of wider value to others wishing to make use of scientific computing.

The module will cover:

- Introduction to programming
- Computing for image analysis
- Computing in Python
- Computing for environmental modelling
- Data visualisation for scientific applications

1.3 Learning Outcomes

At the end of the module, students should:

- have an understanding of algorithm development and be able to use widely used scientific computing software to manipulate datasets and accomplish analytical tasks
- have an understanding of the technical issues specific to image-based analysis, model implementation and scientific visualisation

1.4 Additional Outcomes

If you follow the various ‘advanced’ sections of the course, you will gain a deeper understanding of the concepts and codes, and in addition:

- have a working knowledge of linux / unix operating systems and have

the knowledge and confidence to obtain, compile and install commonly available scientific software packages

1.5 Timetable

The course takes place over 10 weeks in term 1, on Wednesdays usually from 10:00 to 13:00 (09:00-13:00 in the first two sessions) in the Geography Department Unix Computing Lab (PB110) in the [Pearson Building, UCL](#).

Classes take place from the second week of term to the final week of term, other than Reading week. See [UCL term dates](#) for further information.

1.6 Structure of the Course

```
#1 1 Oct 09:00-13:00 4 hrs Python 101
#2 8 Oct 09:00-13:00 4 hrs Plotting and Numerical Python
#3 15 Oct 10:00-13:00 4 hrs Geospatial Data
#4 22 Oct 10:00-13:00 3 hrs Interpolation
#5 29 Oct 10:00-13:00 3 hrs Practical
#5a READING WEEK (ENVI)
#6 12 Nov 10:00-13:00 3 hrs TBD
#7 19 Nov 10:00-13:00 3 hrs TBD
#8 26 Nov 10:00-13:00 3 hrs TBD
#9 3 Dec 10:00-13:00 3 hrs Group practical
#10 10 Dec 10:00-13:00 3 hrs Group practical
```

Total scheduled hours: 32 hours

Prof Philip Lewis

Rooms: Pearson Building Unix Lab, Room 110, 1st floor

In addition to the 10:300 session, I will be available 09:00-10:00 for help and questions (other than the first few weeks when we start at 09:00 anyway).

1.7 Assessment

Assessment is through one piece of coursework that is submitted in both paper form and electronically via Moodle. See the [Moodle page](#) for more details.

Information on the assessment is directly available in a later section of the notes.

1.8 Useful links

Course Moodle page

A useful reading list for basic and advanced Unix.

1.9 Detailed Structure of the Course

0.0 Introduction

A brief overview of the course (these notes):

[Introductory notes]

An introduction to some of the basics of the Unix operating system. The aim is to enable students to understand the directory structure and basic unix operations (e.g. copying and moving files) as well simple operations such as text editing.

Students should read and follow these notes in their own time* as we will not be going through them in class (unless requested).

[Unix notes] [Unix exercises]

1.0 Python 101

The aim of this section is to introduce you to some of the fundamental concepts in Python. Mainly, this is based around fundamental data types in Python (`int`, `float`, `str`, `bool` etc.) and ways to group them (`tuple`, `list` and `dict`). We then learn about how to loop over groups of things, which gives us control to iterate some process. We need to spend a little time on strings, as you will likely to quite a bit of string processing in Scientific Computing (e.g. reading/writing data to/from ASCII text files). Although some of the examples we use are very simple to explain a concept, the more developed ones should be directly applicable to the sort of programming you are likely to need to do. A set of exercises is developed throughout the chapter, with worked answers available to you once you have had a go yourself. In addition, a more advanced section of the chapter is available, that goes into some more detail and complications. This too has a set of exercises with worked examples.

[Course Notes] [Answers] [Advanced] [Advanced Answers]

2.0 Plotting and Numerical Python

In this session, we will introduce and use some packages that you will commonly use in scientific programming.

These are:

`numpy`: NumPy is the fundamental package for scientific computing with Python

`matplotlib`: Python 2D plotting library

We will also introduce some additional programming concepts, and set an exercise that you can do and get feedback on.

[Course Notes] [Answers] [Advanced]

3.0 Geospatial Data

In this session, we will introduced the gdal geospatial module which can read a wide range of scientific data formats. You will find that using it to read data is quite similar to the work we did last week on netCDF datasets.

The main challenges are also much the same: very often, you need to be able to read data from a ‘stack’ of image files and generate a useful 3D (space and time) dataset from these. Once you have the data in such a form, there are many things we can do with it, and very many of these are convenient to do using array-based expressions such as in numpy (consider the simplicity of the expression `absorbed = rad * (1 - albedo)` from last week’s exercise).

That said, it can sometimes be quite an effort to prepare datasets in this form. Last week, we developed a ‘valid data’ mask from the GlobAlbedo dataset, as invalid data were stored as nan. Very often though, scientific datasets have more complex ‘Quality Control’ (QC) information, that gives per-pixel information describing the quality of the product at that location (e.g. it was very cloudy so the results are not so good).

To explore this, we will first consider the MODIS Leaf Area Index (LAI) product taht is mapped at 1 km resolution, every 8 days from the year 2000.

We will learn how to read in these data (in hdf format) using gdal, and how to interpret the QC information in such products to produce valid data masks. As an exercise, you will wrap some code around that to form a 3D masked array of the dataset.

Next, we will consider how to download such data. This should be a reinforcement of material from last week, but it is useful to know how to conveniently access NASA data products. A challenge in the exercise then is to download a different dataset (MODIS snow cover) for the UK, and form a masked 3D dataset from this.

Finally, we will introduce vector datasets and show you python tools that allow you (among many other things) to build a mask in the projection and sampling of your spatial dataset (MODIS LAI in this case).

There are many features and as many complexities to the Python tools we will deal with today, but in this material, we cover some very typical tasks you will want to do. They all revolve around generating masked 3D datasets from NASA MODIS datasets, which is a very useful form of global biophysical information over the last decade+. We also provide much material for further reading and use when you are more confident in your programming.

A final point here is that the material we cover today is very closely related to what you will need to do in the first section of your assessed practical that we will introduce next week, so you really need to get to grips with this now.

There is not as much ‘new’ material as in previous weeks now, but we assume that you have understood, and can make use of, material from those lectures.

[Course Notes] [Answers] [Advanced]

4.0 Interpolation

In today’s session, we will be using some of the LAI datasets we examined last week (masked by national boundaries) and doing some analysis on them. First, we will examine how to improve our data reading function by extracting only the area we are interested in. This involves querying the ‘country’ mask to find its limits and passing this information through to the reader. Then we will look at methods to interpolate and smooth over gaps in datasets using various methods. Finally, we will look at fitting models to datasets, in this case a model describing LAI phenology.

[Course Notes]

4a ENVI

This session is rather apart from the rest and is included to allow students to familiarise themselves with a package image processing environment (ENVI). This is an *unsupervised* session, that takes place during UCL Reading Week in Term 1.

[Course Notes]

5.0 Practical

This section describes the coursework you are to submit for assessment for this course.

[Practical for assessment]

6.0 ENSO

To finish the course, some practical applications. The first of these looks at predicting fire activity from climatic data.

Using monthly fire count data from MODIS Terra, develop and test a predictive model for the number of fires per unit area per year driven by Sea Surface Temperature anomaly data.

[Course Notes]

7.0 P Theory

Another practical application involves a simple radiative transfer model applied to help interpret hyperspectral remote sensing data over an agricultural site.

Using hyperspectral image data over an agricultural area, use photon recollision theory to produce a map of Leaf Area Index.

[Course Notes]

1.10 Using these notes

These notes and all associated files should be directly available for you in the directory `~/DATA/geogg122`.

You should be able to just start firefox (or another browser) and type:

```
berlin% firefox &  
berlin% cd ~/Data/geogg122  
berlin% ipython notebook --pylab=inline
```

Then navigate to the section you want in the browser. This will allow you to run the sessions in the python notebook.

If you only want to view the notes (in html), then navigate to the [pretty course notes link](#) or the [simple course notes link](#).

1.11 Other ways to access the notes

There are several ways you can access this course material.

These notes are created in [ipython notebooks](#).

The course is all stored online in [github](#), so you can just navigate to that site and download the files as you like.

Probably the easiest option is to access the html version of the notes from notebook links in the README on [github](#) or above on this page. You can access *notebook* links, which are guaranteed to be up to date, or *html* links, which should be (but not guaranteed).

Another option is to access individual notebooks online through the [IPython Notebook Viewer](#).

For example, to view the notebook `Chapter0_Introduction/f2_intro.html`, you use a link to the [github](#) file.

From these viewers, you can download the notebook if you like, using the *Download Notebook* button (top right of the page).

Provided you have a relatively up to date version of [ipython](#) and a few other tools such as [pandoc](#) you can convert your own notebooks to other formats using [ipython](#), e.g.:

```
berlin% ipython nbconvert --to html f2_intro.html
```

You can also convert the notebooks to other [formats](#) though you might need some other tools as well for this. If you have a working copy of [LaTeX](#) on your system (e.g. [MacTeX](#) on OS X), you can convert the notebooks to [pdf](#) format:

```
berlin% ipython nbconvert --to latex --post PDF f2_intro.html
```

1.11.1 Obtaining the course material

Alternatively, you can obtain the whole course from [github](#).

To download the whole course, you can:

1. **using git**

use the command

`git, if available: | Create a place on the system that you want to work in (N.B., don't type berlin%: that represents the command line prompt), e.g.:`

```
berlin% cd ~/DATA
berlin% git clone https://github.com/profLewis/geogg122.git
berlin% cd ~/DATA/geogg122
```

This will create a directory `~/Data/msc/geogg122` which has the current versions of the notebooks for the course and associated files.

If the course notes change at all (e.g. are updated), you can update your copy with:

```
berlin% git pull
```

To find out more about using `git`, type `git --help`, get [help online](#) or download and use a [gui](#) tool.

If you set up an account on [github](#), you can fork the [course repository](#) to make your own version of the course notes, and add in your own comments and examples, if that helps you learn or remember things.

2. using a zip file

Download the course as a zip file:

```
berlin% mkdir -p ~/Data/msc
berlin% cd ~/Data/msc
```

```
berlin% wget -O geogg122.zip https://github.com/profLewis/geogg122/archive/master.zip
berlin% unzip geogg122.zip\berlin% cd ~/Data/msc/geogg122-master
```

You can directly use the notebooks, or you can open the `html` files, e.g. opening

`file:///home/plewis/Data/geogg122/Chapter1_Unci/f3_1_unix_intro.html` in a browser (obviously changing the username and path as appropriate).

1.11.2 Using the course material

Once you have copied the course material as described above (and have changed directory to where you have put the course (e.g. `~/DATA/geogg122-master` or `~/DATA/geogg122`) then `cd` to the chapter you want, e.g.:

```
berlin% cd ~/DATA/geogg122/Chapter0_Introduction
```

and you can start the notebooks with:

```
berlin% ipython notebook
```

This should launch a web browser with the address `http://127.0.0.1:8888/` or similar with links to the notebooks you have available.

To load a *specific* notebook, you can type e.g.:

```
berlin% ipython notebook f1_index.html
```

1.11.3 Python

For most users wanting to install a working python environment, Anaconda appears to be far easier and overall quite nice to use: <https://store.continuum.io/cshop/anaconda/>. Comes with Python notebooks, spyder and a wealth of other things not in some other releases.

1.11.4 System access

You should be able to install python on a windows operating system and so could run most of the class material from any windows computer that you have. As we have noted above, you can download all of the class notes as python notebooks or other formats (such as html).

For windows users, it's probably best if you just use <http://mobaxterm.mobatek.net/> to connect to the UCL system (you don't need exceed, it's free, got SFTP, etc).

For linux and OS X machines, it's very straightforward as you already have a unix system. For OS X, you can find the terminal in the Utilities folder under Applications. For X windows on OS X, you may need to [install this](#) if you have a recent version of the operating system.

Another approach is to use the [UCL WTS](#) system, where you have access to some software called exceed to allow you to log on to the system.

Using any of these (or other!) approaches, you want to be able to use the command ssh (or similar) to log on to the gateway machine shankly.geog.ucl.ac.uk:

This will normally be:

```
ssh -Y username@shankly.geog.ucl.ac.uk
```

From there, you should log in (with ssh) to another computer in the lab (or else everyone will be on the same computer).

An alternative gateway, if shankly is down or busy is lyon.geog.ucl.ac.uk.

1. INTRODUCTION TO PYTHON

The aim of this Chapter is to introduce you to some of the fundamental concepts in Python. Mainly, this is based around fundamental data types in Python (`int`, `float`, `str`, `bool` etc.) and ways to group them (`tuple`, `list` and `dict`).

We then learn about how to loop over groups of things, which gives us control to iterate some process.

We need to spend a little time on strings, as you will likely to quite a bit of string processing in Scientific Computing (e.g. reading/writing data to/from ASCII text files).

Although some of the examples we use are very simple to explain a concept, the more developed ones should be directly applicable to the sort of programming you are likely to need to do.

A set of exercises is developed throughout the chapter, with worked answers available to you once you have had a go yourself.

In addition, a more advanced section of the chapter is available, that goes into some more detail and complications. This too has a set of exercises with worked examples.

2.1 1.1 Python

Python is a high level programming language that is freely available, relatively easy to learn and portable across different computing systems. In Python, you can rapidly develop solutions for the sorts of problems you might need to solve in your MSc courses and in the world beyond. Code written in Python is also easy to maintain, is (or should be) self-documented, and can easily be linked to code written in other languages.

Relevant features include:

- it is automatically compiled and executed
- code is portable provided you have the appropriate Python modules.
- for compute intensive tasks, you can easily make calls to methods written in (faster) lower-level languages such as C or FORTRAN
- there is an active user and development community, which means that new capabilities appear over time and there are many existing extensions and enhancements easily available to you.

For further background on Python, look over the material on [Advanced Scientific Programming in Python](#) and/or the [software-carpentry.org](#) and [python.org](#) web sites.

In this session, you will be introduced to some of the basic concepts in Python.

2.2 1.2 Running Python Programs

2.2.1 1.2.1 Requirements

For this course, we suggest you use the [anaconda](#) Python distribution (this is what is installed in the unix lab computers), though you are free to use whichever version of it you like on your own computers.

If you are intending to use these notes on your opwn computer, you will need a relatively comprehensive installation of Python (such as that from [anaconda](#)), and will also need [GDAL](#) installed for some of the work. You may also find it of value to have [git](#) installed.

We are assuming that you are new to computing in this course, but that you are aware of the basic unix material covered in the previous lecture.

2.2.2 2.2.2 Running Python

We will generally use the `ipython` interpreter for running interactive Python programs.

You will probably want to run each session and store scripts in your `Data` (or `DATA`) directory.

If you are taking this course at UCL, the notes should already have been downloaded to your `DATA` directory.

If so, then:

```
berlin% cd ~/DATA/geogg122  
berlin% git reset --hard HEAD  
berlin% git pull
```

will update the notes (for any changes I make over the sessions).

If you need to download the notes and want to run the session directly in the notebook, you will need to download the course material from [github](#) and run the notebook with e.g.:

```
berlin% cd ~/DATA  
berlin% git clone https://github.com/profLewis/geogg122.git
```

to obtain the notes.

You should next check that you are using the version of Python that we intend:

```
berlin% which ipython  
/opt/anaconda/bin/ipython
```

If this isn't the version of Python that you are picking up (note the use of the unix command `which` here), then you can either just type the full path name:

```
berlin% /opt/anaconda/bin/ipython
```

in place of where it says `ipython` in these notes, or modify your shell initialisation file (`~/.bashrc` if you are using `bash` or `~/.cshrc` for `tcsh` or `csh`) to include `/opt/anaconda/bin` early on in the `PATH`.

To go to the directory for this session:

```
berlin% cd ~/Data/geogg122/Chapter2_Python_intro  
berlin% ipython notebook python101.html --pylab=inline
```

You quit an `ipython` notebook session with `^C` (Control C).

To exectute ('run') blocks of Python code in the notebook, use `^<return>` (SHIFT and RETURN keys together).

Alternatively, just run `ipython`:

```
berlin% cd ~/DATA/geogg122/Chapter2_Python_intro  
berlin% ipython --pylab=inline
```

and type your own commands in at the prompt, following the class or the material on the webpages.

2.3 2.3 Getting Started

2.3.1 2.3.1 Variables, Values and Data types

The idea of **variables** is fundamental to any programming. You can think of this as the *name of something*, so it is a way of allowing us to refer to some object in the language.

What the variable *is* set to is called its **value**.

So let's start with a variable we will call (*declare to be*) `x`.

We will give the *value* 1 to this variable:

```
x = 1
```

In a computing language, the *sort of thing* the variable can be set to is called its **data type**.

In the example above, the datatype is an **integer** number (e.g. 1, 2, 3, 4).

In ‘natural language’, we might read the example above as ‘`x` is one’.

This is different to:

```
x = 'one'
```

because here we have set value of the variable `x` to a **string** (i.e. some text).

A string is enclosed in quotes, e.g. "one" or 'one', or even "'one'" or '"one"'.

```
print "one"
print 'one'
print "'one'"
print '"one"'
```



```
one
one
'one'
"one"
```

This is different to:

```
x = 1.0
```

because here we have set value of the variable `x` to a **floating point** number (these are treated and stored differently to integers in computing).

This is different to:

```
x = True
```

where `True` is a **logical** or **boolean** datatype (something is `True` or `False`).

We have so far seen three datatypes:

- `integer (int)`: 32 bits long on most machines
- `(double-precision) floating point (float)`: (64 bits long)
- `Boolean (bool)`
- `string (str)`

but we will come across more (and even create our own!) as we go through the course.

type

In each of these cases above, we have used the variable `x` to contain these different data types. If you want to know what the data type of a variable is, you can use the method `type()`

```
print type(1);
print type(1.0);
print type('one');
print type(True);

<type 'int'>
<type 'float'>
<type 'str'>
<type 'bool'>
```

You can explicitly convert between data types, e.g.:

```
print 'int(1.1) = ',int(1.1)
print 'float(1) = ',float(1)
print 'str(1) = ',str(1)
print 'bool(1) = ',bool(1)

int(1.1) = 1
float(1) = 1.0
str(1) = 1
bool(1) = True
```

but only when it makes sense:

```
print "converting the string '1' to an integer makes sense:",int('1')

converting the string '1' to an integer makes sense: 1

print "converting the string 'one' to an integer doesn't:",int('one')

converting the string 'one' to an integer doesn't:

-----
ValueError                                     Traceback (most recent call last)

<ipython-input-9-20c6d9c1fc49> in <module>()
----> 1 print "converting the string 'one' to an integer doesn't:",int('one')

ValueError: invalid literal for int() with base 10: 'one'
```

When you get an error (such as above), you will need to learn to *read* the error message to work out what you did wrong.

del

You can delete a variable with `del`:

```
x = 100.
print x

100.0

x = 100.
del x

# so now if we try to do anything with the variable
# x, it should fail as x is no longer defined
print x
```

```
-----
NameError                                     Traceback (most recent call last)
```

```
<ipython-input-11-f602427acddb> in <module>()
    4 # so now if we try to do anything with the variable
    5 # x, it should fail as x is no longer defined
--> 6 print x
```

```
NameError: name 'x' is not defined
```

2.3.2 2.3.2 Arithmetic

Often we will want to do some [arithmetic](#) with numbers in a program, and we use the ‘normal’ (derived from C) operators for this.

Note the way this works for integers and floating point representations.

```
'''
Some examples of arithmetic operations in Python

Note how, if we mix float and int, the result is raised to float
(as the more general form)
'''

print 10 + 100      # int addition
print 10. - 100     # float subtraction
print 1./2.          # float division
print 1/2            # int division
print 10.*20.        # float multiplication
print 2 ** 3.        # float exponent
print 8%2            # int remainder

print '====='
# demonstration of floor (//) and remainder (%)
number      = 9.5
base        = 2.0
remainder   = number%base # float remainder
floor       = number//base # 'floor' operation
print number,'is',floor,'times',base,'plus',remainder

110
-90.0
0.5
0
200.0
8.0
0
=====
9.5 is 4.0 times 2.0 plus 1.5
```

2.3.3 Exercise

Change the numbers in the examples above to make sure you understand these basic operations.

Try combining operations and use brackets () to check that that works as expected.

2.3.4 2.3.3 Assignment Operators

```
'''
Assignment operators
```

```
x = 3    assigns the value 3 to the variable x
x += 2  adds 2 onto the value of x
        so is the same as x = x + 2
        similarly /=, *=, -=
x %= 2  is the same as x = x % 2
x **= 2 is the same as x = x ** 2
x // 2  is the same as x = x // 2

A 'magic' trick
=====

http://www.wikihow.com/Read-Someone%27s-Mind-With-Math-Trick

whatever you put as myNumber, the answer is 3

Try this with integers or floating point numbers ...
'''

# pick a number
myNumber = 34.67

# assign this to the variable x
x = myNumber

# multiply it by 2
x *= 2

# multiply this by 5
x *= 5

# divide by the original number
x /= myNumber

# subtract 7
x -= 7

# The answer will always be 3
print x

3.0
```

2.3.5 2.3.4 Logical Operators

```
'''

Logical operators
'''

alive = True
dead = not alive
print 'dead or alive is', dead or alive
print 'dead and alive is', dead and alive

dead or alive is True
dead and alive is False
```

The result of running comparison operators will give a logical (i.e. bool) output.

Most of this should be obvious, but consider carefully how this works for string data types.

2.3.6 2.3.5 Comparison Operators

```
"""
Related, comparison operators:

== : equivalence
!= : not equivalent
> : greater than
>= : greater than or equal to
< : less than
<= : less than or equal to
"""

print "is one plus one equal to two?"
print 1 + 1 == 2

print "is one less than or equal to 0.999?"
print 1 <= 0.999

print "is one plus one not equal to two?"
print 1 + 1 != 2

# note the use of double quotes inside a single quoted string here
print 'is "Hello" not the same as "hello"?
print 'Hello' != 'hello'

# note the use of single quotes inside a double quoted string here
print "is 'more' greater than 'less'?"
print "more" > "less"

print "is '100' less than '2'?"
print '100' < '2'

print "is 100 less than 2?"
print 100 < 2

# a boolean example just to see what happens
print "is True greater than False?"
print True > False

is one plus one equal to two?
True
is one less than or equal to 0.999?
False
is one plus one not equal to two?
False
is "Hello" not the same as "hello"?
True
is 'more' greater than 'less'?
True
is '100' less than '2'?
True
is 100 less than 2?
False
is True greater than False?
True
```

We can combine such logical statements, bracketing the terms as required:

```
print (1 < 2) and (True or False)

True
```

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

"""Exercise in logical statements

P. Lewis p.lewis@ucl.ac.uk

Tue 8 Oct 2013 10:11:03 BST
"""

# hunger threshold in hours
hungerThreshold = 3.0
# sleep threshold in hours
sleepThreshold = 8.0

# time since fed, in hours
timeSinceFed = 4.0
# time since sleep, in hours
timeSinceSleep = 3.0

# Note use of \ as line continuation here
# It is poor style to have code lines > 79 characters
#
# see http://www.python.org/dev/peps/pep-0008/#maximum-line-length
#
print "Tired and hungry?", (timeSinceSleep >= sleepThreshold) and \
      (timeSinceFed >= hungerThreshold)
print "Just tired?", (timeSinceSleep >= sleepThreshold) and \
      (not (timeSinceFed >= hungerThreshold))
print "Just hungry?", (not (timeSinceSleep >= sleepThreshold)) and \
      (timeSinceFed >= hungerThreshold)

Tired and hungry? False
Just tired? False
Just hungry? True
```

2.3.7 Exercise 2.1

The code above works fine, but the large blocks of logical tests are not very clear or readable, and contain repeated items.

Type the code into a file (or download it).

To run the code *either* type at the unix prompt:

berlin% python hungry.py

OR, within ipython:

In [17]: run hungry.py

Modify this block of code to be clearer by assigning the individual logical tests to variables,

e.g.

```
tired = timeSinceSleep >= sleepThreshold
```

2.4 2.4 Groups of things

2.4.1 2.4.1 tuples, lists and dictionaries

Very often, we will want to group items together.

There are several main mechanisms for doing this in Python, known as:

- tuple, e.g. (1, 2, 3)
- list, e.g. [1, 2, 3]
- dict, e.g. {1:'one', 2:'two', 3:'three'}

You will notice that each of these grouping structures uses a different form of bracket.

tuple

A tuple is a group of items separated by commas.

```
t = 1, 2, 'three', False
print t

(1, 2, 'three', False)
```

Note that when you declare the tuple, you don't need to put the braces (brackets) as this is implicit.

Often though, it is a good idea to do so.

```
t = (1, 2, 'three', False)
print t

(1, 2, 'three', False)
```

If there is only one element in a tuple, you must put a comma , at the end, otherwise it is *not* interpreted as a tuple:

```
t = (1)
print t, type(t)

1 <type 'int'>

t = (1,)
print t, type(t)

(1,) <type 'tuple'>
```

You can have an *empty* tuple though:

```
t = ()
print t, type(t)

() <type 'tuple'>
```

Notice that the tuple can contain data of different types.

It can also be nested (i.e. a tuple can contain a tuple):

```
t = ('one', 2), 3, ((4, 5), 6)
print t

((('one', 2), 3, ((4, 5), 6)))
```

Some operations we can perform on tuples include:

- length : `len()`
- selection ('slice') : []

len

```
# set up a simple tuple
t = (1,2,3)
print "The length of the tuple (1,2,3) is", len(t)

# a nested example
t = (1,('2a','2b'),3)
print "The length of the tuple (1,(1,('2a','2b'),3),3) is", len(t)
```

```
The length of the tuple (1,2,3) is 3
The length of the tuple (1,(1,('2a','2b'),3),3) is 3
```

slice

```
# select an item with []
# note the first item is 0, i.e. we start counting at 0
# Python uses a 0-based indexing system

t = ('it','is','a','truth','universally','acknowledged')

print 'item 0',t[0]
print 'item 4',t[4]

item 0 it
item 4 universally

# using negative to count from the end
# so -1 is the last item, -2 the second to last etc

t = ('it','is','a','truth','universally','acknowledged')

print 'item -1',t[-1]
print 'item -3',t[-3]

item -1 acknowledged
item -3 truth

# select a range (a 'slice') of items with [start:end:step]

t = ('it','is','a','truth','universally','acknowledged')

print 'items 0:1\t',t[0:1]           # 0 to 1, so only item 0
print 'items 0:2\t',t[0:2]           # 0 to 2, so items 0 and 1
print 'items :4:2\t',t[:4:2]         # :4:2 so items 0 (implicit) to 4, in steps of 2
                                         # so items 0 and 3
print 'items ::-1\t',t[::-1]          # 0 to end in steps of -1, so reverse order
print 'items 1:-1:2\t',t[1:-1:2]      # 1 to -1 in steps of 2 so items 1 and 3

# Note the use of \t in the strings above e.g. 'items 0:1\t'
# where \t is a tab character (for prettier formatting)

items 0:1    ('it',)
items 0:2    ('it', 'is')
items :4:2   ('it', 'a')
items ::-1   ('acknowledged', 'universally', 'truth', 'a', 'is', 'it')
items 1:-1:2  ('is', 'truth')
```

In effect, when we set up a tuple, we are *packing* some group of items together:

```
t = ('the', 'past', ('is', 'a', 'foreign', 'country'))
```

And we can similarly *unpack*:

```
a, b, c = t

print 'a:',a
print 'b:',b
print 'c:',c

a: the
b: past
c: ('is', 'a', 'foreign', 'country')

t = ('As', 'Gregor', 'Samsa', 'awoke', 'one', 'morning')

a,b = t[1:4:2]
print 'a:',a
print 'b:',b

a: Gregor
b: awoke
```

The set of operations we can perform on tuples includes:

```
<tr><td><b>Name</b></td><td>Example</b></td><b>Result</b></td><td>Meaning</b></td></tr>
<tr><td><pre>len()</pre></td><td><pre>len((1,2,(3,4))</pre></td><td><pre>3</pre></td><td>Length</td></tr>
<tr><td><pre>+</pre></td><td><pre>(1,2) + (3,4,5)</pre></td><td><pre>(1,2,3,4,5)</pre></td><td>Concatenation</td></tr>
<tr><td><pre>\*</pre></td><td><pre>(1,2) * 3</pre></td><td><pre>(1, 2, 1, 2, 1, 2)</pre></td><td>Multiplication</td></tr>
<tr><td><pre>in</pre></td><td><pre>2 in (1,2,3,4)</pre></td><td><pre>True</pre></td><td>Membership</td></tr>
<tr><td><pre>index</pre></td><td><pre>('a','b','c','d').index('c')</pre></td><td><pre>2</pre></td><td>Indexing</td></tr>
<tr><td><pre>count</pre></td><td><pre>('a','b','c','d','c').count('c')</pre></td><td><pre>3</pre></td><td>Counting</td></tr>
<tr><td><pre>min(), max()</pre></td><td><pre>min(('a','b','c'))</pre></td><td><pre>max((3,4,1))</pre></td><td>Minimum and Maximum</td></tr>
<tr><td><pre>tuple()</pre></td><td><pre>tuple([1,2,3])</pre></td><td><pre>tuple('hello')</pre></td><td>Creating Tuples</td></tr>
```

You will find that these basic operators work with any of the group types, so you should make sure you are aware of them.

```
# some examples

print "\ntuple('hello world'):\n\t",tuple('hello world')
print "\ntuple('hello world').count('o'):\n\t",tuple('hello world').count('o')
print "\n('1',)*5 + (1,)*5:\n\t",('1',)*5 + (1,)*5

# Note use of \t in string for tab and \n for newline

tuple('hello world'):
    ('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')

tuple('hello world').count('o'):
    2

('1',)*5 + (1,)*5:
    ('1', '1', '1', '1', '1', 1, 1, 1, 1, 1)
```

You **cannot** directly replace an element in a tuple:

```
t = (1,2,3,4)
t[2] = 'three'

-----
TypeError                                 Traceback (most recent call last)

<ipython-input-32-0dc2fa4129fe> in <module>()
```

```
1 t = (1,2,3,4)
----> 2 t[2] = 'three'
```

TypeError: 'tuple' object does not support item assignment

so you would need to find another way to do this, e.g.

```
t = (1,2,3,4)

t = t[:2] + ('three',) + t[3:]
print t

(1, 2, 'three', 4)
```

Neither can you delete an item in a tuple:

```
t = (1,2,3,4)

del t[2]
print t

-----
TypeError                                         Traceback (most recent call last)

<ipython-input-34-b33678abdb80> in <module>()
      1 t = (1,2,3,4)
      2
----> 3 del t[2]
      4 print t
```

TypeError: 'tuple' object doesn't support item deletion

```
# again, find another way around

t = (1,2,3,4)

t = t[:2] + t[3:]
print t

(1, 2, 4)
```

string as a group

You might have noticed that a string data type `str` acts as a collection of individual characters, e.g.:

```
word = 'hello world'

print "word =\t", word, "\n"

print "tuple(word) =\t", tuple(word)
# slice
print "word[2:5] =\t", word[2:5]
# len
print "len(word) =\t", len(word)
# max (similarly min)
print "max(word) =\t", max(word)
# in (membership)
print "'w' in word =\t", 'w' in word
# count
print "word.count('l')=\t", word.count('l')
```

```

# index
print "word.index('l')=\t",word.index('l')
# + (concatenation)
print "word + ' again'=\t",word + ' again'
# * (repetition)
print "word * 2=\t",word*2

word =      hello world

tuple(word) =      ('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')
word[2:5] = llo
len(word) = 11
max(word) = w
'w' in word =     True
word.count('l')=   3
word.index('l')=   2
word + ' again' = hello world again
word * 2=    hello worldhello world

```

This sort of consistency or operation is one of the things that makes Python a good language to program in.

list

lists or *sequences* are contained within square brackets []:

The operators available for tuple work in much the same way as for lists (more formally, sequences):

```

t = ['It', 'was', 'the', 'best', 'of', 'times']
print t

"""slicing is the same as for tuples"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----slice----"
print 't[:4:2]:\n\t',t[:4:2]          # 0th item to 4th in steps of 2, so 0,2
print 't[::-1]:\n\t',t[::-1]          # reversal

"""index"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----index----"
print 't.index("best"):\n\t',t.index('best')

"""plus"""
t = ['It', 'was'] + ['the', 'best', 'of', 'times']

print "\n----plus----"
print "t = ['It', 'was'] + ['the', 'best', 'of', 'times']\n\t",t

"""multiply"""
t = ['It', 'was', 'the'] + ['best'] * 3 + ['of', 'times']

print "\n----multiply----"
print "t = ['It', 'was', 'the'] + ['best'] * 3 + ['of', 'times']\n\t",t

['It', 'was', 'the', 'best', 'of', 'times']

----slice----
t[:4:2]:

```

```
['It', 'the']
t[::-1]:
    ['times', 'of', 'best', 'the', 'was', 'It']

----index----
t.index("best"):
    3

----plus----
t = ['It', 'was'] + ['the', 'best', 'of', 'times']
    ['It', 'was', 'the', 'best', 'of', 'times']

----multiply----
t = ['It', 'was', 'the'] + ['best'] * 3 + ['of', 'times']
    ['It', 'was', 'the', 'best', 'best', 'best', 'of', 'times']
```

But there are many other things one can do with a list, e.g.:

```
"""replace"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----replace----"
t[3:5] = ['New', 'York']
print "t[3:5] = 'New York':\n\t", t

"""index and replace"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----index and replace----"
t[t.index('best')] = 'worst'
print 't[t.index("best")] = "worst":\n\t', t

"""can delete one or more items"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----del item----"
del t[2:4]                      # delete items 2 to 4, i.e. 2, 3
                                    # i.e. 'the', 'best'
print 'del t[2:4]:\n\t', t

"""can sort"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----sort----"
t.sort()                         # sort inplace

print 't.sort():\n\t', t

"""can insert"""
t = ['It', 'was', 'the', 'best', 'of', 'times']

print "\n----insert----"
t.insert(1,'really')             # insert inplace

print "t.insert(1,'really'):\n\t", t

----replace----
t[3:5] = 'New York':
    ['It', 'was', 'the', 'New', 'York', 'times']
```

```

----index and replace----
t[t.index("best")] = "worst":
    ['It', 'was', 'the', 'worst', 'of', 'times']

----del item----
del t[2:4]:
    ['It', 'was', 'of', 'times']

----sort----
t.sort():
    ['It', 'best', 'of', 'the', 'times', 'was']

----insert----
t.insert(1,'really'):
    ['It', 'really', 'was', 'the', 'best', 'of', 'times']

```

range

Many functions that return multiple items will make use of a list to do so.

An example of this that we will use below is `'range(start,stop,step) <http://docs.python.org/2/library/functions.html#range>'` that returns a list of integers from start to (but not including) stop in steps of step.

```

print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print range(1,3)
[1, 2]

print range(-10,10,2)
[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]

# set a value 3 to the variable x
x = 3

# range(10) produces the lis
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
x in range(10)

True

```

2.5 2.5 Loops and Conditional Statements: if, for, while

So far, we have come across the ideas of variables, data types, and two ways of grouping objects together (tuples and lists).

Another fundamental aspect of any programming language is conditional statements. The simplest form of this is an `if ... else ...` statement:

```

pockets = ['phone','keys','wallet','frog']

this_item = 'nothing'

# test if something is in the list
if this_item in pockets:

```

```
    print "You do have",this_item,'in your pocket'
else:
    print "You don't have",this_item,'in your pocket'
```

You don't have nothing in your pocket

Here,

```
this_item in pockets
```

is a membership test as we have seen above (it returns True or False).

If it's True, the code block

```
print "You do have",this_item,'in your pocket'
```

is executed. If False, then the next condition is checked.

Note the use of indentation here (using tab or spaces) to represent the structure of the conditional statements.

Note also the use of a colon (:) to mark the end of the conditional test.

```
'''An if example
```

```
    Threshold the value of x at zero
'''

x = 3

# threshold at zero
# and print some information about what we did

if x < 0:
    print 'x less than 0'
    x = 0
elif x == 0:
    print 'x is zero'
else:
    print 'x is more than zero'

print 'thresholded x = ',x

x is more than zero
thresholded x =  3
```

The syntax of this is:

```
if condition1 is True:
    ...
elif condition2 is True:
    ...
else:
    ...
```

where condition1 and condition2 are logical tests (e.g. this_item in pockets, today == "Wednesday", x > 10 etc.) and the is True part of the syntax is implicit (i.e. you don't need to type is True).

The word elif means else if. The tests are considered in the order they are given so that if condition1 is not True, we examine condition2. If that is not True, we fall through to the final else block.

```
# nested conditional statements: If
```

```
what_you_keep = 'your head'
when_you_do_it = 'all about you are losing theirs'
```

```

whom_you_trust_when_all_men_doubt_you = 'yourself'

# first tests
if ( what_you_keep == 'your head' ) and \
(when_you_do_it == 'all about you are losing theirs'):

    # second level tests
    if whom_you_trust_when_all_men_doubt_you == 'yourself':
        print "Yours is the Earth and everything that's in it ..."
    else:
        print "Nearly there ..."

else:
    print "Have a look at http://www.poetryfoundation.org/poem/175772"

```

Yours is the Earth and everything that's in it ...

2.5.1 Exercise 2.2

Exercise 2.2 A.

A small piece of Python code that will set the variable `today` to be a string with the day of the week today is:

```

# This imports a module that we can use to access dates
from datetime import datetime

# set up a list of days of the week, starting Monday
# Note the line continuation here with \
week = ['Monday','Tuesday','Wednesday','Thursday',\
        'Friday','Saturday','Sunday']

# This part gives the day of the week
# as an integer, 0 -> Monday, 1 -> Tuesday etc.
day_number = datetime.now().weekday()

today = week[day_number]

# print item day_number in the list week
print "today is",today

```

`today` is Tuesday

Based on the example below, set up a diary for yourself for the week to print out what you should be doing today, using the conditional structure “`if .. elif .. else`“.

```

if day_number == 2:
    print "Remember to wake up early to get to the Python class at UCL"
elif day_number == 4:
    print "Remember to wake up early to go to classes at Imperial College"
else:
    print "get some sleep"

get some sleep

```

Exercise 2.2 B.

You could set up the basic calendar for the week in a list, with the first entry representing Monday, the second Tuesday etc.

```
my_diary = ['Spend the day practicing Python', \
            'Do some reading in the library at UCL', \
            'Remember to wake up early to get to the Python class at UCL', \
            'Spend the day practicing Python', \
            'Remember to wake up early to go to classes at Imperial College', \
            'Work at Python exercises from home', \
            'Work at Python exercises from home']
```

Using a list of this sort, **print the diary entry for today *without* using conditional statements.**

Criticise the code you develop and make suggestions for improvement.

for

Very commonly, we need to iterate or ‘loop’ over some set of items.

The basic stucture for doing this (in Python, and many other languages) is `for ... in ...:`

```
count_list = range(1, 4)

# for loop
for count in count_list:
    '''print counter in loop'''
    print count

print 'blast off'

1
2
3
blast off
```

which has the syntax:

```
for var in list:
    ...
```

where the variable `var` is set to each of the items in `list`, in the order in which they appear in `list`.

xrange

When we have a loop, there must be something that defines what it is we loop over. In the example above, this was a list, `count_list`, which here is `[1, 2, 3]`.

In Python, we can use either use some explicit list, tuple etc. to define what we loop over, or, we can use a generator expression, which is something that returns one of its members at a time.

Normally then, instead of using `range` above, which involves an *explicit* calculation and storage of all elements of the list, we use a generator function, ‘`xrange <http://docs.python.org/2/library/functions.html#xrange>`’, which, in essence, returns the elements ‘on demand’ as needed in the loop (and so uses less memory, though there is little real difference except for very large loops).

```
# for loop
for count in xrange(1, 4):
    '''print counter in loop'''
    print count

print 'blast off'

1
2
3
blast off
```

If you need to force an `iterable <<http://docs.python.org/2/glossary.html#term-iterable>>`_ to e.g. return a list, you can convert the data type to list:

```
print xrange(1, 4)

xrange(1, 4)

print list(xrange(1, 4))

[1, 2, 3]
```

enumerate

Commonly, when iterating over a set of items, we also need access to a counter, telling us which item in the list we are currently on.

This is done using the function `enumerate () <<http://docs.python.org/2/library/functions.html#enumerate>>`_. This returns the tuple (count, item) where count is the index of item in list.

```
word_list = ['Call', 'me', 'Ishmael']

for i,w in enumerate(word_list):
    print 'The',i,'th','word is',w

The 0 th word is Call
The 1 th word is me
The 2 th word is Ishmael
```

Here the syntax:

```
for count,var in enumerate(list):
    ...
```

2.6 2.6 Strings and things

We have seen the data type `str` above and noted some of the operations we can use of strings.

You can look over some more [detailed notes](#) on strings at some point, but here we will now go through some other typical operations you will use in scientific computing:

2.6.1 2.6.1 Some basic string operations

As a recap, with some slightly more complicated examples:

```
word = 'hello world'

print "word \n\t=",word, "\n"

print "list(word) \n\t=",list(word)

# slice
print "word[::-1] \n\t=",word[::-1]
# len
print "len(word) \n\t=",len(word)
# min (similarly max)
print "min(word) \n\t=",min(word), '\n\t... what was printed there?'
# in (membership)
print "'n' in word \n\t=", 'n' in word
# count
```

```
print "word.count('p')\n\t=", word.count('p')
# + (concatenation)
print "'hey!' + word[len('hello')::2]\n\t='hey!' + word[len('hello')::2]
# * (repetition)
print "word[:6]* 3\n\t=", word[:6]*3

word
= hello world

list(word)
= ['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
word[::-1]
= dlrow olleh
len(word)
= 11
min(word)
=
... what was printed there?
'n' in word
= False
word.count('p')
= 0
'hey!' + word[len('hello')::2]
= hey! ol
word[:6]* 3
= hello hello hello

# look what happens if we call index
# for something that doesn't exist in the string
print "word.index('x')=\t", word.index('x')

word.index('x')=
-----
ValueError                                     Traceback (most recent call last)

<ipython-input-55-515f07310ca9> in <module>()
      1 # look what happens if we call index
      2 # for something that doesn't exist in the string
----> 3 print "word.index('x')=\t", word.index('x')

ValueError: substring not found

# Sometimes, we might wish to use
# the string operator find instead
print "word.find('x')=\t", word.find('x')

word.find('x')=      -1
```

2.6.2 2.6.2 split

Suppose we have some data that are presented to us as a string with white space separating each data element, e.g.:

```
data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"
```

These data are total fossil fuel emissions for Zimbabwe (thousand metric tons of C) for selected years (dataset doi 10.3334/CDIAC/00001_V2011).

The even elements are the year (1964, 1974 etc.) and the odd elements (1220, 2470) the data for that year.

We can use the string operator `split()` to separate this into a list of strings:

```

data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"
sdata = data.split()
print sdata

['1964', '1220', '1974', '2470', '1984', '2706', '1994', '4812', '2004', '2707']

```

We could convert these to integer:

```

data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"
sdata = data.split()

# how many items are there?
# use len(), and divide by 2 in this case
n_items = len(sdata)

# create an empty list: years
years = []

# create an empty list: emissions
emissions = []

# loop over sdata in steps of 2
# and append years and emissions
# data as int

# xrange(0,n_items,2) because
# we want to step every 2 in this case
for i in xrange(0,n_items,2):
    years.append(int(sdata[i]))
    emissions.append(int(sdata[i+1]))
print years
print emissions

[1964, 1974, 1984, 1994, 2004]
[1220, 2470, 2706, 4812, 2707]

```

2.6.3 join

The ‘opposite’ of split is join.

This returns an iterable of the form S.join(list), where S is the separator and list is a **list of strings** e.g.:

```

str1 = 'hello'
str2 = 'world'

# joint with space
print ' '.join([str1,str2])
# joint with no space
print ''.join([str1,str2])
# join with tab
print '\t'.join([str1,str1,str2])
# join with colon :
# note what happens we pass a
# string, rather than a list
print ':'.join(str1)

hello world
helloworld
hello      hello      world
h:e:l:l:o

```

```
# remember that it has to be a list
# of strings: years here is a list
# of integers
print ' '.join(years)

-----
TypeError                                         Traceback (most recent call last)

<ipython-input-61-795783cbd46a> in <module>()
      2 # of strings: years here is a list
      3 # of integers
----> 4 print ' '.join(years)

TypeError: sequence item 0: expected string, int found

data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"
sdata = data.split()

for i in xrange(0, len(sdata), 2):
    print ' '.join(sdata[i:i+2])

1964 1220
1974 2470
1984 2706
1994 4812
2004 2707
```

2.6.4 2.6.4 listcomp

That is a perfectly fine way to pull these data out of a string, but it's not very 'Pythonic' (it doesn't make best use of some of the elegant features of this language).

Better in this sense is what are known as [listcomps](#) (list comprehensions).

With a listcomp, you define a list (enclosed in [,]), with two or three terms. The first term is some function `fn(item)`. The second is a `for` statement. The third, if present, is a conditional (`if`) statement.

For example:

```
fdata = [int(s) for s in data.split()]

print fdata

[1964, 1220, 1974, 2470, 1984, 2706, 1994, 4812, 2004, 2707]
```

This generates a list. Within this list, we iterate over the loop `for s in data.split()`, and enter the result of the function `int(s)`.

So this example is directly equivalent to:

```
data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"

fdata = []
for s in data.split():
    fdata.append(int(s))

print fdata

[1964, 1220, 1974, 2470, 1984, 2706, 1994, 4812, 2004, 2707]
```

You will very commonly use listcomps of this nature when performing some function over elements in a list where you have to perform the function on each element at a time.

```

data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"
fdata = [int(s) for s in data.split()]

# use slicing to separate the odd
# and even data
years      = fdata[0::2]
emissions = fdata[1::2]

print years
print emissions

[1964, 1974, 1984, 1994, 2004]
[1220, 2470, 2706, 4812, 2707]

```

Listcomps can be very convenient, as they are a compact way of specifying a loop (with a conditional statement if required).

Don't use listcomps if they obscure the meaning of what you are doing though.

2.6.5 2.6.5 generator expressions

A listcomp generates everything in the list, then returns the list.

Sometimes, you only need one element at a time (e.g. within a loop). In such cases, it is better to use generator expressions.

These look much like listcomps but use () rather than []

```

fdata = (int(s) for s in data.split())
print fdata

for i in fdata:
    print i

<generator object <genexpr> at 0x106540140>
1964
1220
1974
2470
1984
2706
1994
4812
2004
2707

```

but only return one item at a time, on demand in the loop.

2.6.6 2.6.6 replace

Another useful string operator is `replace`, e.g.:

```

# change white space separation to comma separation
data = "1964 1220 1974 2470 1984 2706 1994 4812 2004 2707"
print data.replace(' ', ',')

1964,1220,1974,2470,1984,2706,1994,4812,2004,2707

```

2.6.7 2.6.7 format

The most common way you are likely to be formatting strings is using expressions such as:

```
how_many = 10
how_much = "hours"

print "There are only %d things to learn.\n\
      \nBut\tit will take you %s to do so."%(how_many,how_much)
```

There are only 10 things to learn.
But it will take you hours to do so.

Using this style of string formatting, you put control characters into the string, e.g.:

```
"%d: hello %s"
```

and put a tuple of variables after the string, separated by % which are inserted into the string in the order in which they appear:

```
"%d: hello %s"%(10,'ten')
```

Note that you must get the data types correct, or you will generate an error.

The most common formatting codes are:

- %d represents an integer
- %s represents a string
- %f represents a float
- %e represents a float in exponential form

e.g.:

```
print "\\\
      integer      %d\n\
      float       %f\n\
      string      %s\n\
      exponential %e"%(3,3.1415926536,"pies are squared",3.1415926536)

integer      3
float       3.141593
string      pies are squared
exponential 3.141593e+00
```

2.6.8 2.6.8 strip

A useful method is `strip()` that strips off any unnecessary white space and newlines e.g.:

```
string = '  hello world      \t'
print '|%s|'%string
print '|%s|'%string.strip()

|  hello world      |
|hello world|
```

2.6.9 2.6.9 Getting and splitting filenames

glob

```
# example, with directory names

# glob unix style pattern matching for files and directories
import glob
```

```

# returns a list (or [] if empty)
# to match the pattern given
file_list = glob.glob("files/data/*.txt")
print "file_list:\n\t",file_list

# e.g. the first string in the list
this_file = file_list[0]

file_list:
['files/data/HadSEEP_monthly_qc.txt', 'files/data/heathrowdata.txt', 'files/data/modis_files.txt']

this_file = 'files/data/HadSEEP_monthly_qc.txt'

# split the filename on the field '/'
print "\nthis_file.split('/'):\\n\t",this_file.split('/')

# so the filename is just the last element in this list
print "\nthis_file.split('/')[-1]:\\n\t",this_file.split('/')[-1]

this_file.split('/'):
['files', 'data', 'HadSEEP_monthly_qc.txt']

this_file.split('/')[-1]:
HadSEEP_monthly_qc.txt

# another example, with directory names

# glob unix style pattern matching for files and directories
import glob

# returns a list
file_list = glob.glob("files/data/*.txt")
print "file_list:\\n\t",file_list

print "\nfile names:"
# loop over the list of file namnes
for this_file in file_list:
    # for each of these
    # split the filename on the field '/'
    print "\t",this_file.split('/')[-1]

file_list:
['files/data/HadSEEP_monthly_qc.txt', 'files/data/heathrowdata.txt', 'files/data/modis_files.txt']

file names:
HadSEEP_monthly_qc.txt
heathrowdata.txt
modis_files.txt
modis_files2a.txt
modis_files2b.txt
some_modis_files.txt

```

2.7 Exercise 2.3

The data below are fields of:

0 year
1 month

- 2 tmax (degC)
- 3 tmin (degC)
- 4 air frost (days)
- 5 rain (mm)
- 6 sun (hours)

for Lowestoft in the UK for the year 2012, taken from Met Office data.

```
data = """    2012    1    8.7    3.1      5    33.1    53.9
        2    7.1    1.6    13    13.8    86.6
        3   11.3    3.7     2    64.2   141.3
        4   10.9    4.3     3   108.9   151.1
        5   15.1    8.6     0    46.6   171.3
        6   17.9   10.9     0    74.4   189.0
        7   20.3   12.8     0    93.6   206.9
        8   22.0   14.0     0    59.6   217.3
        9   18.9    9.5     0    38.8   200.8
       10   13.6    7.9     0    92.7   94.7
       11   10.5    4.4     2    62.1   79.6
       12    7.9    2.4     8    95.6   41.9"""
print data

2012    1    8.7    3.1      5    33.1    53.9
2012    2    7.1    1.6    13    13.8    86.6
2012    3   11.3    3.7     2    64.2   141.3
2012    4   10.9    4.3     3   108.9   151.1
2012    5   15.1    8.6     0    46.6   171.3
2012    6   17.9   10.9     0    74.4   189.0
2012    7   20.3   12.8     0    93.6   206.9
2012    8   22.0   14.0     0    59.6   217.3
2012    9   18.9    9.5     0    38.8   200.8
2012   10   13.6    7.9     0    92.7   94.7
2012   11   10.5    4.4     2    62.1   79.6
2012   12    7.9    2.4     8    95.6   41.9
```

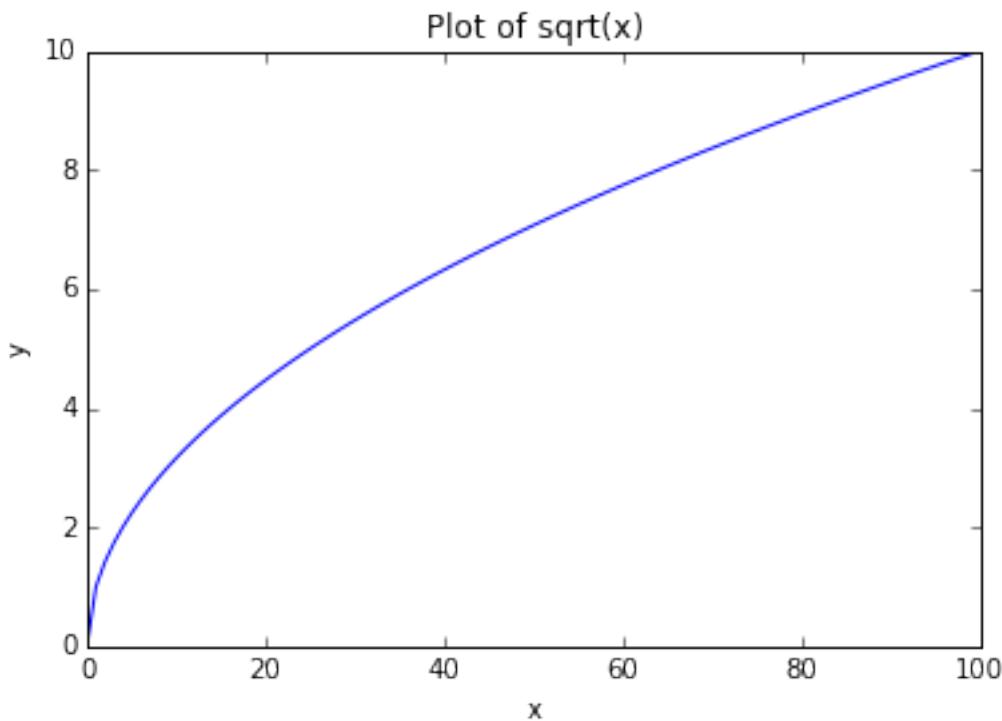
You can use the Python package `pylab` to simply plot data on a graph:

```
# import the pylab module
import pylab as plt

# some e.g. x and y data
x = range(100)
y = [i**0.5 for i in x]

plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plot of sqrt(x)')

<matplotlib.text.Text at 0x1065d6590>
```



Produce a plot of the number of sunshine hours for Lowestoft for the year 2012 using the data given above.

Hint: the data have newline characters `\n` at the end of each line of data, and are separated by white space within each line.

2.8 2.7 Files

To open a file that already exists for *reading*, we use:

```
fp = open(filename, 'r')
```

where `filename` here is the name of a file and the '`r`' argument tells us that we want to open in 'read' mode.

This returns a file object, `fp` here that we can use to read data from the file etc.

When we have finished doing what we want to do, we should close the file:

```
fp.close()
```

To read ASCII data from a file as a list of strings for each line, use:

```
fp.readlines()
```

To write ASCII text to the file, use:

```
fp.write("some text")
```

or to write a list of strings all at once:

```
fp.writelines(["some text\n", "some more\n"])
```

As a first example, let's open a file for *reading* from a file 'files/data/elevation.dat' that contains a list of dates, times and solar elevation angles (degrees).

```
!head -10 < files/data/elevation.dat
```

```
2013/10/8 00:00:00 -44.2719952943
2013/10/8 00:30:00 -43.5276412785
2013/10/8 00:59:59 -41.9842746582
2013/10/8 01:30:00 -39.7226999863
2013/10/8 02:00:00 -36.8452198361
2013/10/8 02:30:00 -33.459799008
2013/10/8 03:00:00 -29.6691191507
2013/10/8 03:30:00 -25.5652187709
2013/10/8 03:59:59 -21.2281801291
2013/10/8 04:30:00 -16.7272357302
```

What we are going to want to do is to create a new file which has the time, specified in decimal hours, and the solar zenith angle (i.e. 90 degrees minus the elevation) into a new file `files/data/zenith.dat`, but only when the Sun is above the horizon.

Let's first concentrate on reading the data in:

```
filename = 'files/data/elevation.dat'
fp = open(filename, "r")
```

Now we will use `readlines` to return a list of strings:

```
sdata = fp.readlines()
print sdata
```

```
[`2013/10/8 00:00:00 -44.2719952943n', `2013/10/8 00:30:00 -43.5276412785n', `2013/10/8
```

We can see that each line of the file contains three fields e.g.:

```
print sdata[0]
```

```
2013/10/8 00:00:00 -44.2719952943
```

The first field is the date (year, day, month), the second is the time (hour, minute, second), and the third is the solar elevation at UCL at that time/date.

To decode each line then we can use `split()` e.g.

```
print sdata[0].split()
['2013/10/8', '00:00:00', '-44.2719952943']
```

what we want is the time and elevation fields, so we will make a loop to get this, but `break` from the loop after the first entry at the moment:

```
filename = 'files/data/elevation.dat'
fp = open(filename, "r")

for i in fp.readlines():
    data = i.split()
    time = data[1]
    elevation = float(data[2])
    print time, elevation
    break
fp.close()
```

```
00:00:00 -44.2719952943
```

We need to convert the time field to minutes. We can start this by splitting the string on `:`:

```
time = data[1].split(':')
print time
['00', '00', '00']
```

then convert these to float and add up the minutes:

```
time = [float(i) for i in data[1].split(':')]
hours = time[0] + time[1]/60. + time[2]/(60.*60)
print hours

0.0
```

we can easily convert elevation to zenith angle.

```
zenith = 90. - elevation
print zenith

134.271995294
```

Putting this together, only printing when the zenith is less than or equal to 90.:

```
filename = 'files/data/elevation.dat'
fp = open(filename,"r")

for i in fp.readlines():
    data = i.split()
    time = [float(i) for i in data[1].split(':')]
    hours = time[0] + time[1]/60. + time[2]/(60.*60)
    zenith = 90. - float(data[2])
    if zenith <= 90.:
        print hours, zenith
fp.close()

6.5 87.9244267471
6.99972222222 83.5831764399
7.5 79.281204843
8.0 75.1590678582
8.5 71.2958940806
9.0 67.7639076381
9.5 64.6374192814
9.99972222222 61.9937023556
10.5 59.9088502907
11.0 58.45158162
11.5 57.6753886774
12.0 57.610853635
12.5 58.2606880779
12.99972222222 59.5992548911
13.5 61.5768473728
14.0 64.1270231504
14.5 67.1746142845
15.0 70.6423218166
15.5 74.4547489962
15.9997222222 78.5412204408
16.5 82.8236457438
17.0 87.1843234569
```

Now, writing to an output file files/data/zenith.dat:

```
ifilename = 'files/data/elevation.dat'
ofilename = 'files/data/zenith.dat'

ifp = open(ifilename,"r")
ofp = open(ofilename,"w")

for i in ifp.readlines():
    data = i.split()
    time = [float(i) for i in data[1].split(':')]
    hours = time[0] + time[1]/60. + time[2]/(60.*60)
    zenith = 90. - float(data[2])
```

```
if zenith <= 90.:
    ofp.write("%.1f %.3f\n"%(hours, zenith))
ifp.close()
ofp.close()
```

We could check the output file from unix:

```
!head -10 < files/data/zenith.dat

6.5 87.924
7.0 83.583
7.5 79.281
8.0 75.159
8.5 71.296
9.0 67.764
9.5 64.637
10.0 61.994
10.5 59.909
11.0 58.452

!ls -l files/data/zenith.dat

-rw-r--r-- 1 plewis staff 257 30 Sep 11:04 files/data/zenith.dat
```

2.9 Exercise 2.4

The text file files/data/modis_files.txt contains a listing of hdf format files that are in the directory /data/geospatial_19/ucfajlg/fire/Angola/MOD09 on the UCL Geography system. The contents of the file looks like (first 10 lines):

```
!head -10 < files/data/modis_files.txt
```

```
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004001.h19v10.005.2008109063923.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004002.h19v10.005.2008108084250.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004003.h19v10.005.2008108054126.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004004.h19v10.005.2008108112322.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004005.h19v10.005.2008108173219.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004006.h19v10.005.2008108214033.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004007.h19v10.005.2008109081257.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004008.h19v10.005.2008109111447.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004009.h19v10.005.2008109211421.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004010.h19v10.005.2008110031925.hdf
```

Your task is to create a new file 'files/data/some_modis_files.txt' that contains *only* the file names for the month of August.

You will notice that the file names have a field in them such as A2004006. This is the one you will need to concentrate on, as it specifies the year (2004 here) and the day of year (doy), (006 in this example).

There are various ways to find the day of year for a particular month / year, e,g, look on a [website](#).

2.10 2.8 Doing Some Science

2.10.1 Maximum Precipitation

The Problem

We want to calculate the **maximum** monthly precipitation for regions of the UK for all years in the 20th Century.

The Data

We have access to monthly average precipitation data as regional totals from the [UK Met Office](#).

These data are in ASCII format, available over the internet.

e.g. http://www.metoffice.gov.uk/hadobs/hadukp/data/monthly/HadSEEP_monthly_qc.txt

or locally as `files/data/HadSEEP_monthly_qc.txt <files/data/HadSEEP_monthly_qc.txt>'__.

for South East England.

Solving the Problem

With just the Python skills you have learned so far, you should be able to solve a problem of this nature.

Before diving into this though, you need to think through **what steps** you need to go through to achieve your aim?

At a ‘high’ level, this could be:

1. Examine the data
2. Read the data into the computer program
3. Select which years you want
4. Find the maximum value for each year over all months
5. Print the results

So now we need to think about how to implement these steps.

Examine the data

For the first step, there are many ways you could do this.

You will probably want to look at the data set in a browser.

You should see that the data are ‘white space’ separated.

The first 4 lines are ‘header’ text, giving contextual information on the data.

Subsequent lines have the **year** in the first column, then 12 columns of monthly precipitation, then an annual total.

Read the data into the computer program

1. You could simply save the file using ‘Save As ...’ from the browser.
2. You could, if you really wanted, just copy and paste the data into a file on the local system.
3. You could use the unix command wget:

```
berlin% mkdir -p ~/Data/python/Chapter2_Python_intro/files/data
berlin% cd ~/Data/python/Chapter2_Python_intro
berlin% wget -O files/data/HadSEEP_monthly_qc.txt \
http://www.metoffice.gov.uk/hadobs/hadukp/data/monthly/HadSEEP_monthly_qc.txt
```

4. You could download and read the file directly from a URL within Python

Let us suppose that you had downloaded the file and saved it as `files/data/HadSEEP_monthly_qc.txt`.

In this case, we would use:

```
filename = 'files/data/HadSEEP_monthly_qc.txt'

fp = open(filename, 'r')
raw_data = fp.readlines()
fp.close()

print raw_data[:10]

['Monthly Southeast England precipitation (mm). Daily automated values used after 1996.r
So we have read the data in well enough, but it's not really in a convenient format.
```

First, it has 4 lines at the top that we don't want.

Second, although the data are in a list for each line, each line is

stored as a string.

Since we know about lists, we might suppose that it would be better to have each line as a list, with each ‘white space’ separated item being an element of the list.

If we have a string such as:

```
' 1873 87.1 50.4 52.9 19.9 41.1 63.6 53.2 56.4 62.0 86.0 59.4 15.7 647.7\n'
```

one way to achieve this would be to use `split()`:

```
line_data = ' 1873 87.1 50.4 52.9 19.9 41.1 63.6 53.2 56.4 62.0 86.0 59.4 15.7 647.7\n'
year_data = line_data.split()
print year_data

['1873', '87.1', '50.4', '52.9', '19.9', '41.1', '63.6', '53.2', '56.4', '62.0', '86.0', '59.4',
```

That's useful, and we could loop over each line and perform this to get line lists with the first element as the year, second as precipitation in January, etc.

But each element is still a string, and really, we want these as `float`.

We can convert `str` to `float` using `float()`, but we have to do this for each string individually.

We can do this in a loop:

```
line_data = ' 1873 87.1 50.4 52.9 19.9 41.1 63.6 53.2 56.4 62.0 86.0 59.4 15.7 647.7\n'
year_data = line_data.split()

for column,this_element in enumerate(year_data):
    year_data[column] = float(this_element)

print 'format now',type(year_data[0])
print year_data

format now <type 'float'>
[1873.0, 87.1, 50.4, 52.9, 19.9, 41.1, 63.6, 53.2, 56.4, 62.0, 86.0, 59.4, 15.7, 647.7]
```

Now we know how to convert each line of data into a list of floating point numbers.

In practice, we will see later in the course that there are simpler ways of achieving (using `'numpy <http://www.numpy.org/>'`).

We should first chop off the first 4 lines of data:

```
required_data = raw_data[4:]

# lets check what the first line is now
print required_data[0]
```

```
1873 87.1 50.4 52.9 19.9 41.1 63.6 53.2 56.4 62.0 86.0 59.4 15.7 647.7
```

Putting all of this together:

```
# specify filename
filename = 'files/data/HadSEEP_monthly_qc.txt'

# read the data, chop off first 4 lines
# and store in required_data
fp = open(filename,'r')
raw_data = fp.readlines()
fp.close()
required_data = raw_data[4:]

# set up list to store data in
data = []

# loop over each line
for line_data in required_data:
    # split on white space
    year_data = line_data.split()

    # convert data to float
    for column,this_element in enumerate(year_data):
        year_data[column] = float(this_element)
    data.append(year_data)
```

Now we have the data read in, as floating point values, in the 2-D list called data:

```
print data[0]
print data[1]

[1873.0, 87.1, 50.4, 52.9, 19.9, 41.1, 63.6, 53.2, 56.4, 62.0, 86.0, 59.4, 15.7, 647.7]
[1874.0, 46.8, 44.9, 15.8, 48.4, 24.1, 49.9, 28.3, 43.6, 79.4, 96.1, 63.9, 52.3, 593.5]
```

and we can compare this with the original data we saw on the web or the file we downloaded to check it's been read in correctly.

Select which years you want

We want years 1900 to 1999.

The year is stored in the first column, e.g. data[10][0], and we want columns 1 to -1 (i.e. skip the first and last column):

```
print data[10][0]
print data[10][1:-1]

1883.0
[60.2, 97.4, 23.6, 36.5, 48.1, 53.3, 69.7, 21.7, 100.6, 58.1, 97.8, 22.5]

c20_data = []
for line in data:
    if (line[0] >= 1900) and (line[0] < 2000):
        c20_data.append(line[1:-1])
```

Find the maximum value for each year over all months

We want to find the maximum value in each row of c20_data now.

If we consider row 0, we can use e.g.:

```
print c20_data[0]
print max(c20_data[0])

[92.8, 125.9, 23.3, 30.7, 30.0, 74.6, 31.0, 73.3, 22.3, 56.1, 72.1, 88.2]
125.9

i.e. in the year 1900 (row 0), for S.E. England, the maximum rainfall was in February.

# Aside: show which month that was
month_names = [ "Jan", "Feb", "Mar", "Apr", \
    "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ]

print "In South East England"
for row in xrange(0,100,10):
    year = 1900 + row
    max_precip = max(c20_data[row])
    month = c20_data[row].index(max_precip)

    print "In the year",year,"the rainiest month was",month_names[month],"with",max_precip,"mm"

In South East England
In the year 1900 the rainiest month was Feb with 125.9 mm
In the year 1910 the rainiest month was Dec with 119.4 mm
In the year 1920 the rainiest month was Jul with 116.9 mm
In the year 1930 the rainiest month was Nov with 121.0 mm
In the year 1940 the rainiest month was Nov with 203.0 mm
In the year 1950 the rainiest month was Nov with 137.9 mm
In the year 1960 the rainiest month was Oct with 167.0 mm
In the year 1970 the rainiest month was Nov with 186.4 mm
In the year 1980 the rainiest month was Oct with 110.1 mm
In the year 1990 the rainiest month was Feb with 122.3 mm

# max precip for all years

result = []
for row in xrange(100):
    result.append(max(c20_data[row]))
```

Print the results

```
print "In South East England, the maximum value for the years 1900 to 1999 is"  
print result  
print "The highest rainfall in a month was",max(result),"mm"  
print "It occurred in",result.index(max(result))+1900
```

In South East England, the maximum value for the years 1900 to 1999 is
[125.9, 98.7, 97.2, 188.4, 96.0, 104.5, 127.3, 129.5, 83.8, 143.9, 119.4, 153.6, 151.6, 106.5, 19
The highest rainfall in a month was 203.0 mm
It occurred in 1940

2.11 Exercise 2.5

That is quite an achievement, given the limited amount of programming you know so far.

If you go through this though, you will (should) see that it is really not very efficient.

For example:

- we read all the data in and then filter out the years we want (what if the dataset were **huge**?)
 - we loop over the 100 years multiple times

- we store intermediate results

For this exercise, you should look through the code we developed and try to make it more efficient.

Efficiency should not override clarity and understanding though, so make sure you can understand what is going on at each stage.

2.12 Exercise 2.6

2.12.1 Average Temperature

We want to calculate the long-term average temperature (tmax degC) using observational data at one or more meteorological stations. Such data are [relevant to understanding climate and its dynamics](#).

We choose the period 1960 to 1990 (30 years average to even out natural variability).

We can obtain monthly average data for a number of UK stations from the [UK Met. Office](#).

Not all station records are complete enough for this calculation, so we select, for example [Heathrow](#).

Just with the Python skills you have learned so far, you should be able to solve a problem of this nature.

Before diving into this though, you need to think through **what steps** you need to go through to achieve your aim?

At a ‘high’ level, this could be:

1. Get hold of the data
2. Read the data into the computer program
3. Select which years you want
4. Average the data for each month over all selected years
5. Print the results

So now we need to think about how to implement these steps.

For the first step, there are many ways you could do this.

2.13 2.9 Dictionaries

Dictionaries (type `dict`) are another way of grouping objects.

These are defined within curly brackets `{ }` and are distinguished by having a ‘key’ and `value` for each item.

e.g.:

```
a = {'one': 1, 'two': 2, 'three': 3}

# we then refer to the keys and values in the dict as:

print 'a:\n\t', a
print 'a.keys():\n\t', a.keys()      # the keys
print 'a.values():\n\t', a.values()  # returns the values
print 'a.items():\n\t', a.items()    # returns a list of tuples

a:
{'three': 3, 'two': 2, 'one': 1}
a.keys():
['three', 'two', 'one']
a.values():
[3, 2, 1]
a.items():
[('three', 3), ('two', 2), ('one', 1)]
```

Notice that the order they appear in is not necessarily the same as when we generated the dictionary.

We refer to specific items as e.g.:

```
print a['one']
```

```
1
```

We can loop over dictionaries in various interesting ways:

```
for k in a.keys():
    print a[k]
```

```
3
2
1
```

```
for k,v in a.items():
    print k,v
```

```
three 3
two 2
one 1
```

If you really need to process in some order, you need to sort the keys in some way:

```
for k in sorted(a.keys()):
    print a[k]
```

```
1
3
2
```

though in this case this still might not be what you want, so be careful.

You can add to a list with update:

```
a.update({'four':4,'five':5})
print a
```

```
{'four': 4, 'three': 3, 'five': 5, 'two': 2, 'one': 1}
```

or for a single item:

```
a['six'] = 6
print a
```

```
{'six': 6, 'three': 3, 'two': 2, 'four': 4, 'five': 5, 'one': 1}
```

or delete items:

```
del a['three']
print a
```

```
{'six': 6, 'two': 2, 'four': 4, 'five': 5, 'one': 1}
```

These trivial examples are useful for understanding some basic operations on dictionaries, but don't show their real power.

A good example is to consider a configuration file.

When we have to run complicated processing jobs (e.g. on stacks of satellite data), we will often control the processing of these jobs with some text that we may put in a file.

This would describe the particular conditions that one subset of the jobs would process, for example.

Following from exercise 2.4 above, we might create a configuration file with:

```
[TERRA]
dir = /data/geospatial_19/ucfajlg/fire/Angola/MOD09
name = MODIS TERRA data
year = 2004
doy_start = 214
doy_end = 245
file_list = files/data/modis_files2a.txt

[AQUA]
dir = /data/geospatial_19/ucfajlg/fire/Angola/MYD09
name = MODIS AQUA data
year = 2004
doy_start = 214
doy_end = 245
file_list = files/data/modis_files2b.txt
```

This information is in the file `files/data/modis.cfg`.

We could read and parse this file ourselves:

```
fp = open('files/data/modis.cfg')

# empty dict
modis = {}
this_section = modis

# loop over each line
for line in fp.readlines():
    # strip any extra white space
    line = line.strip()
    # check that there is some text
    # and it starts with [ and ends with ]
    if len(line) and line[0] == '[' and line[-1] == ']':
        section = line[1:-1]
        modis[section] = this_section = {}
    elif len(line) and line.find("=") != -1:
        key,value = line.split(">")
        this_section[key.strip()] = value.strip()

fp.close()
print modis
```

though in practice, we might choose to use the `ConfigParser` module:

```
import ConfigParser

config = ConfigParser.ConfigParser()
config.read('files/data/modis.cfg')

# we can convert this to a normal dictionary
modis = {}
for k in config.sections():
    modis[k] = dict(config.items(k))
print modis

{'AQUA': {'doy_end': '245', 'doy_start': '214', 'name': 'MODIS AQUA data', 'year': '2004', 'file_...}
```

The text file `files/data/modis_files2a.txt` contains a listing of hdf format files that are in the directory `/data/geospatial_19/ucfajlg/fire/Angola/MOD09` and `files/data/modis_files2b.txt` those in `/data/geospatial_19/ucfajlg/fire/Angola/MYD09` on the UCL Geography system. The contents of the files looks like (first 10 lines):

```
!head -10 < files/data/modis_files2b.txt

MYD09GA.A2004001.h19v10.005.2008035021539.hdf
MYD09GA.A2004002.h19v10.005.2008035115941.hdf
MYD09GA.A2004003.h19v10.005.2008035223215.hdf
MYD09GA.A2004004.h19v10.005.2008036154947.hdf
MYD09GA.A2004005.h19v10.005.2008036025835.hdf
MYD09GA.A2004006.h19v10.005.2008037030304.hdf
MYD09GA.A2004007.h19v10.005.2008037072048.hdf
MYD09GA.A2004008.h19v10.005.2008037155636.hdf
MYD09GA.A2004009.h19v10.005.2008037162301.hdf
MYD09GA.A2004010.h19v10.005.2008038034819.hdf
```

Let's try to use the information in the configuration dictionary `modis` to generate a list of the files we want to process:

```
# first, work out how to loop over config sections
# and get the sub-dictionary

for k,v in modis.items():
    print "\nexamining section",k
    sub_dict = v
    print v

examining section AQUA
{'doy_end': '245', 'doy_start': '214', 'name': 'MODIS AQUA data', 'year': '2004', 'file_list': 'files/data/modis_files2a.txt'}

examining section TERRA
{'doy_end': '245', 'doy_start': '214', 'name': 'MODIS TERRA data', 'year': '2004', 'file_list': 'files/data/modis_files2b.txt'}
```

Next, make sure we can read the `file_list`:

```
print 'reading',sub_dict['file_list']
fp = open(sub_dict['file_list'],'r')
file_data = fp.readlines()
fp.close()

# print the first one, just to see what it looks like
count = 0
print file_data[count]

reading files/data/modis_files2a.txt
MOD09GA.A2004001.h19v10.005.2008109063923.hdf
```

Make sure we can get the day of year from this:

```
print file_data[count].split('.')

['MOD09GA', `A2004001', `h19v10', `005', `2008109063923', `hdfn']

print file_data[count].split('.')[1]

A2004001
```

```
doy = int(file_data[count].split('.')[1][-3:])
print doy
```

1

whats the range of days we want?

```
doy_range = range(int(sub_dict['doy_start']),int(sub_dict['doy_end']))
print doy_range
```

```
[214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233]
```

Is doy in the range?

```
print doy in doy_range
```

```
False
```

So now we have all of the parts we need to create this code:

```
# 1. Read the configuration file
# into the dict modis
import ConfigParser

config = ConfigParser.ConfigParser()
config.read('files/data/modis.cfg')

# we can convert this to a normal dictionary
modis = {}
for k in config.sections():
    modis[k] = dict(config.items(k))

# 2. Now, loop over config sections
# and get the sub-dictionary which we call sub_dict

# 3. set up an empty list to contain the
# files we want to process
wanted_files = []

for k,v in modis.items():

    sub_dict = v

    # 3a. Read the file list
    fp = open(sub_dict['file_list'], 'r')
    file_data = fp.readlines()
    fp.close()

    # 3b. find the doy range
    doy_range = range(int(sub_dict['doy_start']), \
                      int(sub_dict['doy_end']))

    # 3c. loop over each file read from
    #      sub_dict['file_list']
    for count in xrange(len(file_data)):
        # 3d. extract doy from the file name
        this_file = file_data[count]

        doy = int(this_file.split('.')[1][-3:])

        # 3e. see if doy is in the range we want?
        if doy in doy_range:

            # 3f. put the directory on the front
            full_name = sub_dict['dir'] + \
                        '/' + this_file
            wanted_files.append(full_name)

print "I found %d files to process"%len(wanted_files)

# I won't print the whole list as its too long
# just the first 10
for f in wanted_files[:10]:
```

```
print f

I found 62 files to process
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004214.h19v10.005.2007299212915.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004215.h19v10.005.2007300042347.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004216.h19v10.005.2007300091257.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004217.h19v10.005.2007300153436.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004218.h19v10.005.2007300215826.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004219.h19v10.005.2007302194509.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004220.h19v10.005.2007302093547.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004221.h19v10.005.2007302222054.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004222.h19v10.005.2007303011606.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004223.h19v10.005.2007303073538.hdf
```

2.14 Exercise 2.7

You should modify the example above to make it simpler, if you can spot any places for that (don't make it more complicated!).

You should then modify it so that the list of files that we want to process is printed to a file.

2.15 2.10 Where we have reached

That is plenty of Python for one day. We would not expect you to be able to remember all of this the first time you go through it: you will have to work at it and go through it multiple times. Make sure that the time you spend going through codes and examples is profitable though: it is all too easy to sit and stare at a section of code for hours without *learning* anything ... you need to be engaged with the learning for this to happen, so take frequent breaks, write your own summary notes and do whatever you need to get to grips with the basics here.

Although we have rather crammed a lot into this session, that is for timetabling reasons more than for effectiveness. When you go through it in your own time, break it into sections and try to get to grips with each part.

We would not expect you to be able to develop codes of the sort we have developed above from scratch to start with, but as you go through these notes and do the exercises, you should start to understand something of how we build a piece of code.

Our experience is that you should be able to get to grips with these examples (or at least most of them). Don't start by looking at some finished piece of code ... start by going through the simple examples to learn the commands and syntax, then try to put the pieces together.

Once you have learned the basic tools, you will be in a much better position to think about *algorithms*, i.e. how to break a problem down into smaller parts that you can solve with the Python that you know.

2.16 2.11 Answers

Answers to the exercises in this session are made available to you, though you should obviously only consult these when you are finished (or if you are very stuck).

2.17 2.12 Advanced

Once you have got to grips with the basics in this session, you might consider stretching yourself and going through the advanced section.

Consult the answers to the exercises once you have completed them, and come along to office hours if you want to go through anything.

3. PLOTTING AND NUMERICAL PYTHON

3.1 3.1 In this session

In this session, we will introduce and use some packages that you will commonly use in scientific programming.

These are:

- `numpy`: NumPy is the fundamental package for scientific computing with Python
- `matplotlib`: Python 2D plotting library

We will also introduce some additional programming concepts, and set an exercise that you can do and get feedback on.

3.1.1 3.2.1 Getting Started

To get started for this session, you should go to your `Data` area and download or update your material for this course.

If you haven't previously downloaded the course, type the following at the unix prompt (illustrated by `berlin%` here):

```
berlin% cd Data
berlin% git clone https://github.com/profLewis/geogg122.git
berlin% cd geogg122/Chapter3_Scientific_Numerical_Python
```

If you already have a clone of the course repository (which you should have from last week):

```
berlin% cd Data/geogg122
berlin% git reset --hard HEAD
berlin% git pull
berlin% cd Chapter3_Scientific_Numerical_Python
```

This will update your local copy with any new notes or files. Any additional files that you yourself have created should be still where they were.

3.2 E3.1 Exercise: listing

Using Python, produce a listing of the files in the subdirectory `data` of `geogg122/Chapter3_Scientific_Numerical_Python` that end with `.nc` and put this listing in a file called `files/data/data.dat` with each entry on a different line

3.3 3.2 Getting and Plotting Some Data: netCDF format

3.3.1 3.2.1 GlobAlbedo data

Before we start to use these new packages, we will start by getting some data for you to use and show you how to read it in and do some basic plotting.

Today, we will be using the [ESA GlobAlbedo data](#).

These data come in different spatial resolutions, but here we will use the global product at 0.5 degrees.

The files are accessible via `http` in the directory:

```
http://www.globalbedo.org/GlobAlbedoXX/mosaics/YYYY/0.5/monthly
```

The filenames are of the pattern:

```
GlobAlbedo.YYYYMM.mosaic.5.nc.gz
```

where `YYYY` is the year (e.g. 2009) and `MM` is the month (01 is January, 12 is December).

`XX` is a code which is given below.

The data are in [netCDF format](#) which is a common binary data format.

```
# versioning (?) codes. At present these are:  
XX = {1998:95,1999:95,2000:97,2001:97,2002:26,2003:66,2004:54,2005:54,  
      2006:29,2007:25,2008:53,2009:56,2010:56,2011:78}  
  
# but check on http://www.globalbedo.org by trying to order some data  
  
year = 2009  
  
root = 'http://www.globalbedo.org/GlobAlbedo%d/mosaics/%d/0.5/monthly/'%(XX[year],year)  
  
# example filename : use formatting string:  
# %d%02d  
month = 1  
url = root + '/GlobAlbedo.%d%02d.mosaic.5.nc.gz'%(year,month)  
  
print url  
  
http://www.globalbedo.org/GlobAlbedo56/mosaics/2009/0.5/monthly//GlobAlbedo.200901.mosaic.5.nc.gz
```

3.3.2 3.2.2 Loading from within Python

You *could* download the data yourself (explained in the advanced section), but these files are available to you in this case in the directory `files/data` for the year 2009.

```
!ls files/data/GlobAlbedo*.nc  
  
files/data/GlobAlbedo.200901.mosaic.5.nc  
files/data/GlobAlbedo.200902.mosaic.5.nc  
files/data/GlobAlbedo.200903.mosaic.5.nc  
files/data/GlobAlbedo.200904.mosaic.5.nc  
files/data/GlobAlbedo.200905.mosaic.5.nc  
files/data/GlobAlbedo.200906.mosaic.5.nc  
files/data/GlobAlbedo.200907.mosaic.5.nc  
files/data/GlobAlbedo.200908.mosaic.5.nc  
files/data/GlobAlbedo.200909.mosaic.5.nc  
files/data/GlobAlbedo.200910.mosaic.5.nc  
files/data/GlobAlbedo.200911.mosaic.5.nc  
files/data/GlobAlbedo.200912.mosaic.5.nc  
files/data/GlobAlbedo.201001.mosaic.5.nc
```

```
files/data/GlobAlbedo.201002.mosaic.5.nc
files/data/GlobAlbedo.201003.mosaic.5.nc
files/data/GlobAlbedo.201004.mosaic.5.nc
files/data/GlobAlbedo.201005.mosaic.5.nc
files/data/GlobAlbedo.201006.mosaic.5.nc
files/data/GlobAlbedo.201007.mosaic.5.nc
files/data/GlobAlbedo.201008.mosaic.5.nc
files/data/GlobAlbedo.201009.mosaic.5.nc
files/data/GlobAlbedo.201010.mosaic.5.nc
files/data/GlobAlbedo.201011.mosaic.5.nc
files/data/GlobAlbedo.201012.mosaic.5.nc
```

We use the netCDF4 module to read netCDF data:

```
from netCDF4 import Dataset

root = 'files/data/'

# example filename : use formatting string:
# %d%02d
year = 2009
month = 1
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)
print local_file

# load the netCDF data from the file local_file
nc = Dataset(local_file,'r')

# the netCDF file has dimensions
# stored as a dictionary
print nc.dimensions.keys()

# and variables also as a dictionary
print nc.variables.keys()

-----
ImportError                                     Traceback (most recent call last)

<ipython-input-3-635e408ccbf9> in <module>()
----> 1 from netCDF4 import Dataset
      2
      3 root = 'files/data/'
      4
      5 # example filename : use formatting string:

ImportError: No module named netCDF4

Here, we will want to access 'DHR_VIS', 'DHR_NIR' and 'DHR_SW', which are the bihemispherical reflectance (DHR: 'black sky albedo' – the albedo under directional illumination conditions) for visible, near infrared and total shortwave wavebands:

# explicitly as:
albedo = [nc.variables['DHR_VIS'],nc.variables['DHR_NIR'],nc.variables['DHR_SW']]

# or more neatly as:

data_fields = ['DHR_VIS','DHR_NIR','DHR_SW']

albedo = [nc.variables[f] for f in data_fields]
```

3.3.3 3.2.3 Plotting an image

We can plot image data using the package `pylab` (from `matplotlib`).

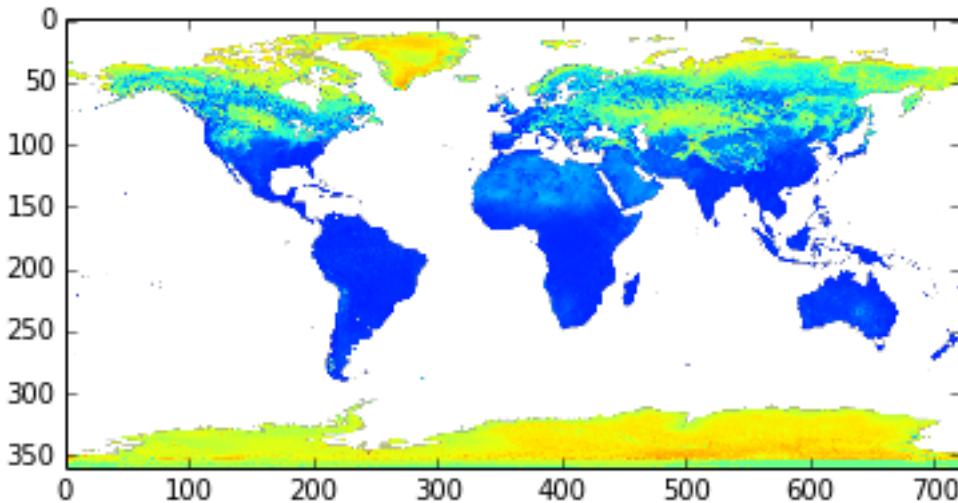
If you are running this as a notebook, make sure you use:

```
ipython notebook --pylab=inline if you want the images to appear in the notebook.
```

First we have to import `pylab`, which we will refer to as `plt` (see `import pylab as plt`).

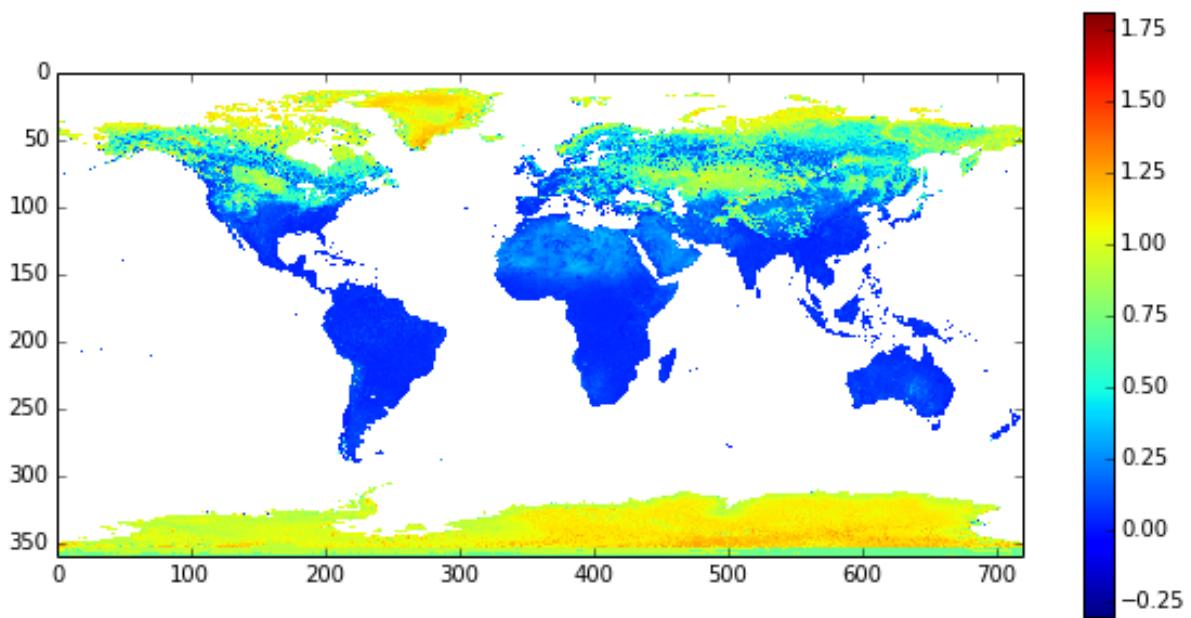
The basic function for plotting an image is `plt.imshow`:

```
import pylab as plt  
  
plt.imshow(albedo[0])  
  
<matplotlib.image.AxesImage at 0x2b05430059d0>
```



That's fine, but we might apply a few modifications to make a better plot:

```
import pylab as plt  
  
# change the figure size  
plt.figure(figsize=(10, 5))  
  
# use nearest neighbour interpolation  
plt.imshow(albedo[0], interpolation='nearest')  
  
# show a colour bar  
plt.colorbar()  
  
<matplotlib.colorbar.Colorbar instance at 0x10bccb90>
```



Albedo should lie between 0 and 1, so there are clearly a few ‘funnies’ that we might flag later.

For now, we might like to plot *only* values between 0 and 1 (thresholding at those values).

Also if we don’t like this colour map, we can try another:

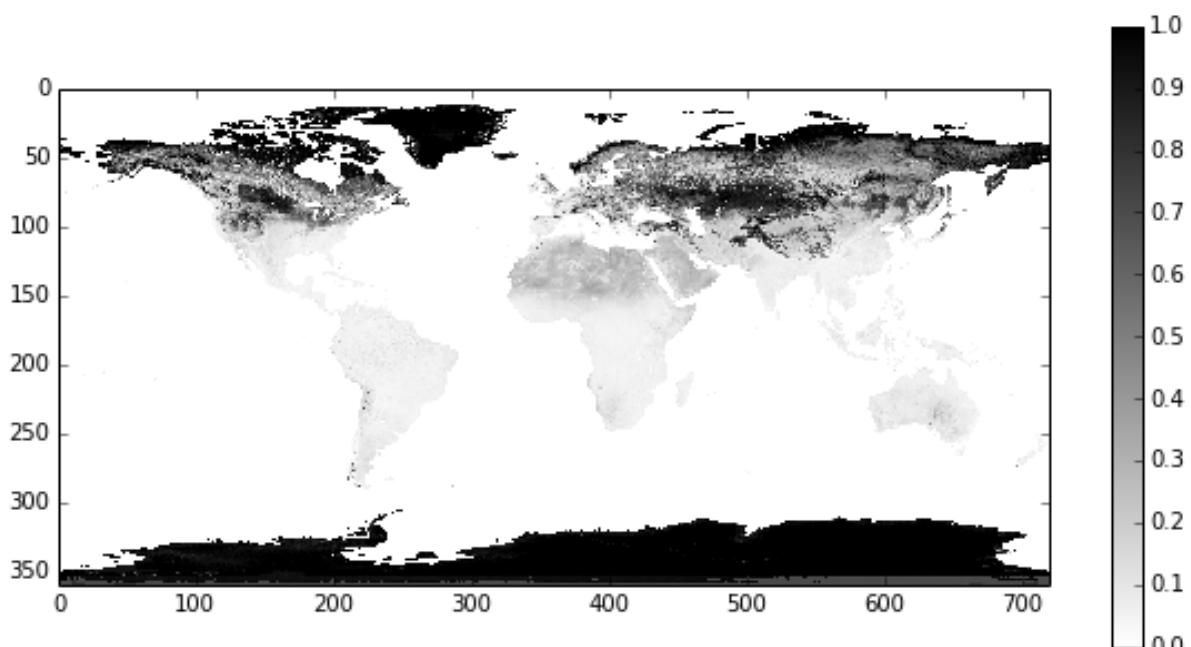
```
import pylab as plt

# change the figure size
plt.figure(figsize=(10,5))

# use nearest neighbour interpolation
plt.imshow(albedo[0],interpolation='nearest',cmap=plt.get_cmap('binary'),vmin=0.0,vmax=1.0)

# show a colour bar
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x1172e5a8>
```



If we wanted to save the plot and put a title on:

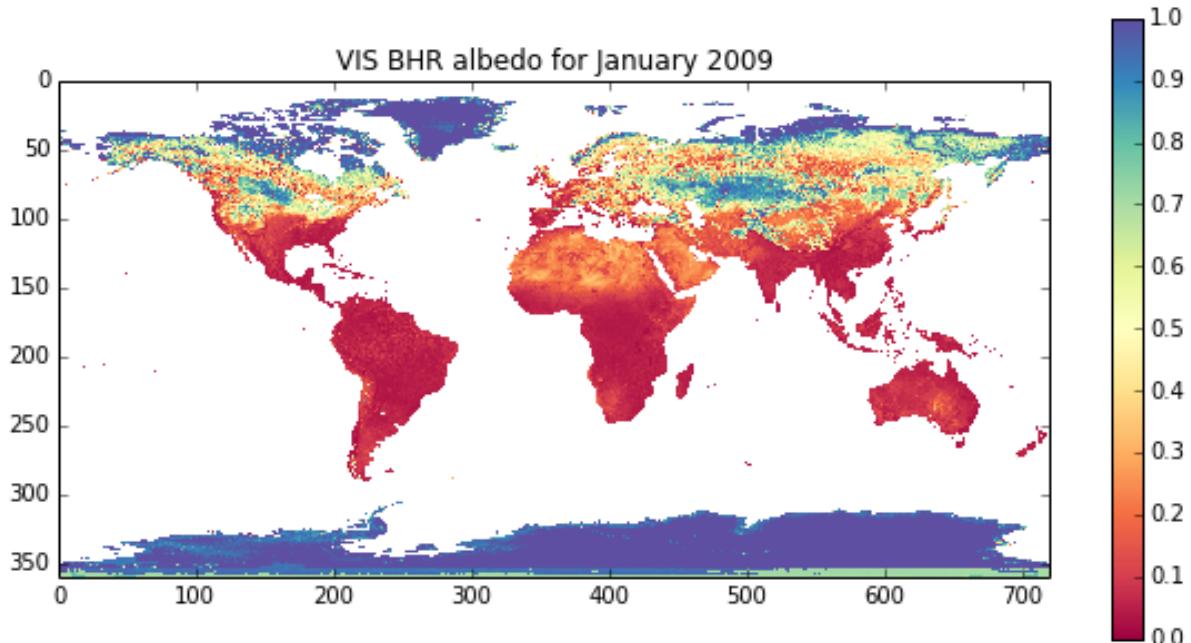
```
import pylab as plt

# set up a month dictionary
month_list = ['January', 'February', 'March', 'April', 'May', 'June', \
              'July', 'August', 'September', 'October', 'November', 'December']
# make a dictionary from 2 lists
month_dict = dict(zip(range(1,13),month_list))

# change the figure size
plt.figure(figsize=(10,5))
# plt.clf(): clear the figure in case anything in it before
plt.clf()

plt.title('VIS BHR albedo for %s %d'%(month_dict[month],year))
# use nearest neighbour interpolation
plt.imshow(albedo[0],interpolation='nearest',cmap=plt.get_cmap('Spectral'),vmin=0.0,vmax=1.0)

# show a colour bar
plt.colorbar()
plt.savefig('files/data/albedo.jpg')
```



3.3.4 E3.2 Exercise: Making Movies

3.3.5 E3.2.1 Software

You can *sort of* make movies in `pylab`, but you generally have to make a system call to unix at some point, so it's probably easier to do this all in unix with the utility '`convert` <http://www.imagemagick.org/script/convert.php>'__.

At the unix prompt, check that you have access to `convert`:

```
berlin% which convert
/usr/bin/convert
```

If this doesn't come up with anything useful, there is probably a version in `/usr/bin/convert` or `/usr/local/bin/convert` (If you don't have it on your local machine, install 'ImageMagick' <http://www.imagemagick.org/script/index.php>'__ which contains the command line tool `convert`).

To use this, e.g.:

from the unix command line:

```
berlin% cd ~/Data/geogg122/Chapter3_Scientific_Numerical_Python
berlin% convert files/data/albedo.jpg files/data/albedo.gif
```

or from within a notebook:

```
!convert files/data/albedo.jpg files/data/albedo.gif
```

Or, more practically here, you can run a unix command directly from Python:

```
import os
cmd = 'convert files/data/albedo.jpg files/data/albedo.gif'
os.system(cmd)
```

```
0
```

This will convert the file `files/data/albedo.jpg` (in jpeg format) to `files/data/albedo.gif` (in gif format).

Figure 3.1: albedo

We can also use `convert` to make animated gifs, which is one way of making a movie.

3.3.6 E3.2.2 Looping over a set of images

You have all of the code you need above to be able to read a GlobAlbedo file for a given month and waveband in Python and save a picture in jpeg format, but to recap for BHR_VIS:

```
from netCDF4 import Dataset
import pylab as plt
import os

root = 'files/data/'

month_list = ['January', 'February', 'March', 'April', 'May', 'June', \
              'July', 'August', 'September', 'October', 'November', 'December']
# make a dictionary from 2 lists
month_dict = dict(zip(range(1,13),month_list))

# example filename : use formatting string:
# %d%02d
year = 2009

# set the month
month = 1

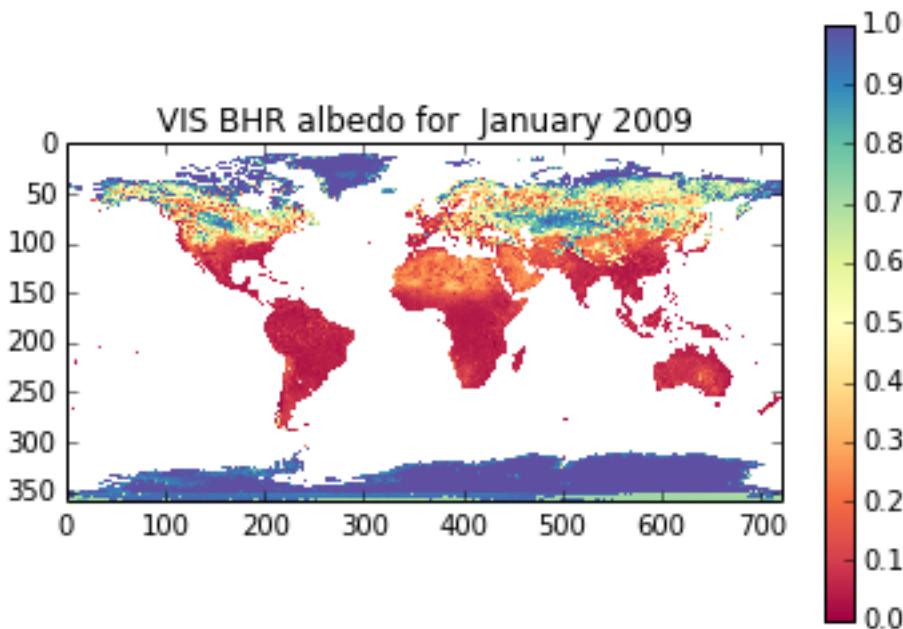
''' Read the data '''
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)
# load the netCDF data from the file f.filename
nc = Dataset(local_file,'r')
band = nc.variables['DHR_VIS']

''' Plot the data and save as picture jpeg format '''
# make a string with the output file name
out_file = root + 'GlobAlbedo.%d%02d.jpg'%(year,month)
# plot
plt.figure()
plt.clf()
```

```
# %9s forces the string to be 8 characters long
plt.title('VIS BHR albedo for %s %d'%(month_dict[month],year))
# use nearest neighbour interpolation
plt.imshow(band,interpolation='nearest',cmap=plt.get_cmap('Spectral'),vmin=0.0,vmax=1.0)
# show a colour bar
plt.colorbar()
plt.savefig(out_file)

''' Convert the file to gif '''
# set up the unix command which is of the form
# convert input output
# Here input will be out_file
# and output we can get with out_file.replace('.jpg','.gif')
# i.e. replacing where it says .jpg with .gif
cmd = 'convert %s %s'%(out_file,out_file.replace('.jpg','.gif'))
os.system(cmd)

0
```



Modify the code above to loop over each month, so that it generates a set of gif format files for the TOTAL SHORTWAVE ALBEDO

You should confirm that these exist, and that the file modification time is when you ran it (not when I generated the files for these notes, which is Oct 10 2013).

```
ls -l files/data/GlobAlbedo*.gif

[Om-rw-rw-r-- 1 plewis plewis 340658 Oct 11 20:04 [0mfiles/data/GlobAlbedo.2009.SW.1.gif[0m
-rw-rw-r-- 1 plewis plewis 340658 Oct 11 20:04 [0mfiles/data/GlobAlbedo.2009.SW.gif[0m
-rw-rw-r-- 1 plewis plewis 26593 Oct 14 11:54 [0mfiles/data/GlobAlbedo.200901.gif[0m
-rw-rw-r-- 1 plewis plewis 28139 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200902.gif[0m
-rw-rw-r-- 1 plewis plewis 28259 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200903.gif[0m
-rw-rw-r-- 1 plewis plewis 28249 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200904.gif[0m
-rw-rw-r-- 1 plewis plewis 28468 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200905.gif[0m
-rw-rw-r-- 1 plewis plewis 28672 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200906.gif[0m
-rw-rw-r-- 1 plewis plewis 28656 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200907.gif[0m
-rw-rw-r-- 1 plewis plewis 28275 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200908.gif[0m
-rw-rw-r-- 1 plewis plewis 28952 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200909.gif[0m
-rw-rw-r-- 1 plewis plewis 28450 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200910.gif[0m
-rw-rw-r-- 1 plewis plewis 28570 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200911.gif[0m
```

```
-rw-rw-r-- 1 plewis plewis 28438 Oct 11 20:04 [0mfiles/data/GlobAlbedo.200912.gif[0m
[m
```

3.3.7 E3.2.3 Make the movie

The unix command to convert these files to an animated gif is:

```
convert -delay 100 -loop 0 files/data/GlobAlbedo.2009???.gif files/data/GlobAlbedo.2009.SW.gif
```

Run this (ideally, from within Python) to create the animated gif GlobAlbedo.2009.SW.gif

Again, confirm that *you* created this file (and it is not just a version you downloaded):

```
ls -l files/data/GlobAlbedo.2009.SW.gif
```

```
[0m-rw-rw-r-- 1 plewis plewis 340658 Oct 11 20:04 [0mfiles/data/GlobAlbedo.2009.SW.gif[0m
[m
```

3.4 3.3 Numpy

3.4.1 3.3.1 Numpy Arrays

At the heart of the ‘numpy <<http://docs.scipy.org/doc/numpy/contents.html>>’ module is a a powerful N-dimensional array object.

Among many other things, this allows us to do arithmetic (and other operations) directly on arrays, rather than having to loop over each element (as we did with lists for example).

Two main benefits of this are:

- the code is much easier to read
- running the code is much more efficient (because, ‘under the hood’ as it were of the programming language, we can do fast operations such as vector processing).

As our first example, let’s read the shortwave albedo data we examined above into a numpy array.

We will start with just one month of data:

```
from netCDF4 import Dataset
import numpy as np

root = 'files/data/'
year = 2009

# set the month
month = 1

''' Read the data '''
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)
# load the netCDF data from the file f.filename
nc = Dataset(local_file,'r')
band = np.array(nc.variables['DHR_SW'])

# some interesting things about numpy arrays
print "the array type is now",type(band)
print "the shape of the array is",band.shape
print "the size of the array is",band.size
```

```
print "the number of dimensions is",band.ndim
print "the data type in the array is",band.dtype
```

```
the array type is now <type 'numpy.ndarray'>
the shape of the array is (360, 720)
the size of the array is 259200
the number of dimensions is 2
the data type in the array is float32
```

A few things to notice here.

First, we imported the numpy module using

```
import numpy as np
```

which means that locally, we refer to this module as np.

Second, we converted from the format that was read from the netCDF file to a numpy array with:

```
band = np.array(nc.variables['DHR_SW'])
```

so this is one way to convert from other data types to numpy representation (e.g. np.array([1, 2, 3]) for a conversion from a list).

Then we saw at the end of this code a number of commonly used methods for the numpy class that provide us with information on the `shape` of the array (N.B. this is (rows,cols) as you might notice from the plots we generated), the total size (number of elements), number of dimensions.

We also saw that we could access the array data type (`dtype`). This is very different to a `list` or `tuple` then, because a numpy array can *only* contain data of a single data type (but lists or tuples can have different data types in each element).

astype

We can convert *between* data types using `astype`:

```
a = np.array([1,2,3])
print a.dtype

int64

b = a.astype(float)
print b
print b.dtype

[ 1.  2.  3.]
float64
```

slice

As you would expect, we can slice in a numpy array. All that is different is that we set up a slice of each dimension, e.g.:

```
# see if you can work out why this is
# the size, shape and ndim it is
```

```
print band[-3:-1,10:30:5]
print band[-3:-1,10:30:5].shape
print band[-3:-1,10:30:5].ndim
```

```
[ [ 0.65842855  0.57553321  0.8013677   0.80455124]
[ 0.66181606  0.66181606  0.6632846   0.58462834] ]
(2, 4)
2
```

If you want to specify **all elements in a given dimension** you need to use at least one `:`, e.g.

```
# what does this mean?
```

```
print band[10:20,0]
print band[10:20,0].shape

[ nan  nan  nan  nan  nan  nan  nan  nan  nan  nan]
(10,)
```

nan, inf

Some of the data in the array `band` appear as `nan` ('not a number'), which is how, in this dataset, non-valid pixels are specified.

`nan` is also what you would get for undefined results from arithmetic:

```
a = np.array([0., 1.])
b = np.array([0., 0.])
c = a/b
print c
print 'so 0./0. =',c[0]
print 'so 1./0. =',c[1]

[ nan  inf]
so 0./0. = nan
so 1./0. = inf
```

`inf` here means 'infinity', which is what anything other than zero divided by zero is.

We can check to see if some value is `nan` or `inf` e.g:

```
print 'is c nan?',np.isnan(c)
print 'is c inf?',np.isinf(c)

is c nan? [ True False]
is c inf? [False  True]
```

Generally we would want to avoid processing array values that contained `nan` or `inf` as any arithmetic involving these is unlikely tell us more than we already know:

```
print 'inf + 1\t\t=' ,np.inf + 1
print 'inf - np.inf\t\t=' ,np.inf - np.inf
print 'nan / 2 \t\t=' ,np.nan / 2

inf + 1          = inf
inf - np.inf     = nan
nan / 2          = nan
```

Or, trying to e.g. calculate the sum of values in the array `band`:

```
print band.sum()

nan
```

which is not really what we wanted.

We have seen above that `np.isnan(c)` and `np.isinf(c)` result in a `bool` array of the same shape as the input array.

We can use these to mask out things we don't want, e.g.

```
no_data = np.isnan(band)
print no_data

[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ...,  False False False]
 [False False False ...,  False False False]
 [False False False ...,  False False False]]
```

This will be `True` where the data in `band` were `nan` ... but what we might often want is the opposite of this, i.e. `True` where the data are valid.

We might guess that we could simply type:

masked array

There is a special type of object in numpy called a `masked array`. With this array representation, operations are *only* performed where the data mask is `False`. We need to import the masked array module `ma` from numpy:

```
import numpy.ma as ma

masked_band = ma.array(band)
print masked_band.mask

False
```

When we convert the representation from a normal numpy `array` to a masked array using `ma.array()`, the default mask is `False` (which means no mask).

In this case, we want to set the mask to `True` where the data are `nan`. We would normally do this when setting up the masked array:

```
masked_band = ma.array(band, mask=np.isnan(band))
print "so the array is now\n",masked_band
print "\nwith a mask\n",masked_band.mask

so the array is now
[[-- -- -- ..., -- -- --]
 [-- -- -- ..., -- -- --]
 [-- -- -- ..., -- -- --]
 ...
 [0.5722442269325256 0.5722442269325256 0.5722442269325256 ...,
 0.5790106058120728 0.5790106058120728 0.5790106058120728]
 [0.5722442269325256 0.5722442269325256 0.5722442269325256 ...,
 0.5790106058120728 0.5790106058120728 0.5790106058120728]
 [0.5722442269325256 0.5722442269325256 0.5722442269325256 ...,
 0.5790106058120728 0.5790106058120728 0.5790106058120728]]
with a mask
[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ...,  False False False]
 [False False False ...,  False False False]
 [False False False ...,  False False False]]
```

3.4.2 3.3.2 Reading data from multiple files

`zeros, ones, empty`

Suppose we want to read in all 12 months of SW albedo data following on from the example above.

```
from netCDF4 import Dataset
import numpy as np

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# loop over month
# but right now we'll just pretend that we're
# doing that and consider what happens the first time
# we are in our loop
#
# set i as index counter
# set the variable month to be
# months[0]
i = 0
month = months[i]

# this then is the file we want
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

# load the netCDF data from the file local_file
nc = Dataset(local_file,'r')
band = np.array(nc.variables['DHR_SW'])
```

Looking at what would happen when we read the first file, we can see that we don't actually know how large the whole dataset will be until we have read our first file.

We know that we will want dimensions of (12,nrows,ncols) (or better still, (len(months),nrows,ncols) in case we decide to change months), but until we have read the data into the variable band with:

```
band = np.array(nc.variables['DHR_SW'])
```

we don't know how many rows or columns the dataset has (well, we might do ... but we want to try to design flexible code, where if the size of the datasets changed, our code would still operate correctly!).

But, when we *have* first read some data, *then* we would be in a position to say what the shape of the whole dataset should be. At that point, we could use the method:

```
np.empty(shape)
```

to (in essence) set aside ('allocate') some memory to store these data. There are three main options for this:

- `np.zeros(shape)` : set up an array with shape defined by shape and initialise with 0
- `np.ones(shape)` : set up an array with shape defined by shape and initialise with 1
- `np.empty(shape)` : set up an array with shape defined by shape but don't initialise (so, in theory, a little faster, but can be filled with random junk)

we can control the `dtype` of the array with a `dtype` option. e.g.:

```
print '-'*30;print '1-D arrays';print '-'*30;
print 'np.zeros(1):',np.zeros(1) # 1 : 1D array
print 'np.zeros(2):',np.zeros(2) # 2 : 1D array
```

```
print '\n' + '-'*30;print '2- and 3-D arrays';print '-'*30;
print 'np.zeros((1,2)):\n',np.zeros((1,2)) # default dtype is float
print 'np.zeros((2,2)):\n',np.zeros((2,2)) # 2 X 2
print 'np.zeros((2,2,3)):\n',np.zeros((2,2,3)) # 2 X 2 X 3

print '\n' + '-'*30;print 'empty and ones';print '-'*30;
print 'np.empty((1,2)):\n',np.empty((1,2)) # default dtype is float
print 'np.ones((1,2)):\n',np.ones((1,2)) # default dtype is float

print '\n' + '-'*30;print 'Changing the data type';print '-'*30;
print 'np.ones((1,2),dtype=int):\n',np.ones((1,2),dtype=int) # dtype int
print 'np.empty((1,2),dtype=int):\n',np.empty((1,2),dtype=int) # dtype int
print 'np.ones((1,2),dtype=bool):\n',np.ones((1,2),dtype=bool) # dtype bool
print 'np.zeros((1,2),dtype=bool):\n',np.zeros((1,2),dtype=bool) # dtype bool
print 'np.empty((1,2),dtype=bool):\n',np.empty((1,2),dtype=bool) # dtype bool

print '\n' + '-'*30;print 'Note that we can have a string array';print '-'*30;
print 'np.ones((1,2),dtype=str):\n',np.ones((1,2),dtype=str) # dtype str

print '\n' + '-'*30;print 'Or even e.g. a list array';print '-'*30;
print 'np.ones((2,2),dtype=list):\n',np.ones((2,2),dtype=list) # dtype list
print 'np.empty((2,2),dtype=list):\n',np.empty((2,2),dtype=list) # dtype list

-----
1-D arrays
-----
np.zeros(1): [ 0.]
np.zeros(2): [ 0.  0.]

-----
2- and 3-D arrays
-----
np.zeros((1,2)):
[[ 0.  0.]]
np.zeros((2,2)):
[[ 0.  0.]
 [ 0.  0.]]
np.zeros((2,2,3)):
[[[ 0.  0.  0.]
   [ 0.  0.  0.]]]

-----
empty and ones
-----
np.empty((1,2)):
[[ 0.00000000e+000  1.56900585e-315]]
np.ones((1,2)):
[[ 1.  1.]]

-----
Changing the data type
-----
np.ones((1,2),dtype=int):
[[1 1]]
np.empty((1,2),dtype=int):
[[ 0  225037488]]
np.ones((1,2),dtype=bool):
[[ True  True]]
np.zeros((1,2),dtype=bool):
[[False False]]
```

```

np.empty((1,2),dtype=bool):
[[ True  True]]


-----
Note that we can have a string array
-----

np.ones((1,2),dtype=str):
[['1' '1']]



-----
Or even e.g. a list array
-----

np.ones((2,2),dtype=list):
[[1 1]
 [1 1]]
np.empty((2,2),dtype=list):
[[None None]
 [None None]]

```

So, once we have read one band, we could write:

```

from netCDF4 import Dataset
import numpy as np

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# loop over month
# but right now we'll just pretend that we're
# doing that and consider what happens the first time
# we are in our loop
#
# set i as index counter
# set the variable month to be
# months[0]
i = 0
month = months[i]

# this then is the file we want
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

# load the netCDF data from the file local_file
nc = Dataset(local_file,'r')
band = np.array(nc.variables['DHR_SW'])
if i == 0: # first band read
    # set up a tuple (or list) saying what shape we want the data
    shape = (len(months),) + band.shape
    data = np.zeros(shape)

print "I set up the array data of shape",data.shape,"with",data.ndim,"dimensions"

I set up the array data of shape (12, 360, 720) with 3 dimensions

```

We also use slicing when assigning ('loading', if you like) data into a larger array:

```
data[i,:,:] = band
```

But this will *only* work if what you are trying to load and where you are trying to load it are the same *shape*:

```
data[i,:,:].shape == band.shape
```

which they are in this case.

Putting this all together then:

```
# this then is the file we want
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

# load the netCDF data from the file local_file
nc = Dataset(local_file,'r')
band = np.array(nc.variables['DHR_SW'])
if i == 0: # first band read
    # set up a tuple (or list) saying what shape we want the data
    shape = (len(months),) + band.shape
    data = np.empty(shape)

# load band into the data array
data[i,:,:] = band
```

And setting the loop around this:

```
from netCDF4 import Dataset
import numpy as np

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# loop over month
# use enumerate so we have an index counter
for i,month in enumerate(months):
    # this then is the file we want
    local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

    # load the netCDF data from the file local_file
    nc = Dataset(local_file,'r')
    band = np.array(nc.variables['DHR_SW'])
    if i == 0: # first band read
        # set up a tuple (or list) saying what shape we want the data
        shape = (len(months),) + band.shape
        data = np.empty(shape)

    # load band into the data array
    data[i,:,:] = band
```

so now we have all of the data loaded into the array `data`.

An alternative way of achieving this same effect is to use a list, creating an empty list before going into the loop and appending `band` to the list each time we read a new band.

```
from netCDF4 import Dataset
import numpy as np

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# empty list
data = []
```

```

# loop over month
# use enumerate so we have an index counter
for i,month in enumerate(months):
    # this then is the file we want
    local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

    # load the netCDF data from the file local_file
    nc = Dataset(local_file,'r')
    # append what we read to the list called data
    data.append(np.array(nc.variables['DHR_SW']))

# convert data to a numpy array (its a list of arrays at the moment)
data = np.array(data)

```

In many ways, this is a simpler and ‘cleaner’ way of setting up a 3-D array from reading multiple 2-D arrays. If the array is *very* large, it can be less efficient (you have to convert a huge list to an array at the end), but that is not normally something to worry about and this approach is often preferable.

sum

To sum data in a numpy (or masked) array, we use the function `sum`.

e.g.:

```

print data.sum(),data[0,0,0]
nan nan

```

That's not very useful in this case, because we have `nan` in the dataset ... though if we use a masked array it *will* work as we expect.

`np.sum` here adds all of the values in the array.

We could select *only* values that are ≥ 0 with the logical statement:

```

not_valid = np.isnan(data)
print not_valid

[[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ..., False False False]
 [False False False ..., False False False]
 [False False False ..., False False False]]

 [[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ..., False False False]
 [False False False ..., False False False]
 [False False False ..., False False False]]

 [[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ..., False False False]
 [False False False ..., False False False]
 [False False False ..., False False False]]]

...

```

```
[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ...,  False False False]
 [False False False ...,  False False False]
 [False False False ...,  False False False]]
```



```
[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ...,  False False False]
 [False False False ...,  False False False]
 [False False False ...,  False False False]]
```



```
[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ...,  False False False]
 [False False False ...,  False False False]
 [False False False ...,  False False False]]]
```

so `not_valid` here is an array of the same size and shape as `data`, with `True` where the value is not valid (`nan`) and `False` elsewhere.

We can use this to select *only* data elements in `data` which are `False` in `not_valid` (not `not_valid` is valid ...), or we could change this into a `valid` array:

```
valid = not_valid == False

ndata = data[valid]
print ndata

[ 0.69403636  0.74866891  0.80311614 ...,  0.58878124  0.58878124
 0.58878124]
```

Now, the array is a 1-D array. It has fewer elements than the original `data` array:

```
print 'original data size\t:',data.size
print 'new data size\t\t:',ndata.size
print 'data shape\t\t:',data.shape
print 'new data shape\t\t:',ndata.shape

original data size   : 3110400
new data size        : 1040376
data shape          : (12, 360, 720)
new data shape       : (1040376,)
```

This is however, one way we can use `np.sum` to add up the values in `data`:

```
print data[valid].sum()
```

```
446890.0
```

mean

Similarly, if we wanted the `mean` or standard deviation and we tried to do this on the array that has `nan` in it:

```
print data.mean(),data.std()
```

```
nan nan
```

but selecting only the values that are valid:

```
print data[valid].mean(), data[valid].std()

0.429546835711 0.30692650158
```

we get what we might expect.

Remember that this is for *normal* (i.e. not masked) numpy arrays. Hassles of this nature with `nan` in the dataset is one good reason to use masked arrays.

axis selection

As used above, functions such as `sum`, `mean` and `std` will give you the sum, mean and standard deviation over *the whole array*.

Often though, we only want to apply these functions over a particular dimension.

Recall that the shape of data is:

```
data.shape

(12, 360, 720)
```

The first dimension here is ‘month’ then, the second is ‘latitude’ and the third ‘longitude’.

If we wanted to know the mean albedo for each month then then, we would want to apply `mean` over axis 1 and 2:

In numpy, this is done by specifying `axis` in the function:

```
month_mean = data.mean(axis=(1,2))
print month_mean.shape

(12,)
```

Thats the correct shape, but if we look at the values:

```
print month_mean

[ nan  nan]
```

we see that we get `nan` ...

That’s hardly surprising, as this is the same issue we had earlier with just calling `sum` or `mean` over the whole array.

Rather than end this section with a negative result, lets consider an array without `nan`.

We have previously generated the array `valid`, which is a logical array, but we could e.g. convert this to `float`:

```
gdata = (valid).astype(float)
print gdata.sum(axis=(1,2))
print gdata.std(axis=(1,2))
print gdata.mean(axis=(1,2))

[ 86698.  86698.  86698.  86698.  86698.  86698.  86698.  86698.
 86698.  86698.  86698.]
[ 0.47180942  0.47180942  0.47180942  0.47180942  0.47180942  0.47180942
 0.47180942  0.47180942  0.47180942  0.47180942  0.47180942  0.47180942]
[ 0.33448302  0.33448302  0.33448302  0.33448302  0.33448302  0.33448302
 0.33448302  0.33448302  0.33448302  0.33448302  0.33448302  0.33448302]
```

There are many useful functions in numpy for manipulating array data, from basic arithmetic through to these function here for some basic stats.

They are all very easy to use and make for clear code, *but* if you have masks in your dataset (as we often do with geospatial data), you need to think carefully about how you want to process in the presence of masks.

It is often a good idea when you have masks to use a masked array.

3.5 E3.3 Exercise: 3D Masked Array

Taking the code above as a starting point, generate a masked array of the GlobAlbedo dataset for the year 2009.

3.6 3.4 Average Albedo

If you run the exercise above, data should now be a masked array.

If you haven't (e.g. when going through this in the class), you can run a function in the file 'files/python/masked.py' to do this now:

```
import sys
sys.path.insert(0,'files/python')
from masked import masked

data = masked(dataset='BHR_SW')
print type(data)
print data.shape
print data.ndim

<class 'numpy.ma.core.MaskedArray'>
(12, 360, 720)
3
```

Now we have the albedo dataset for one year stacked in a 3-D masked array array (assuming you sucessfully completed exercise E3.3 or ran the code above), we can do some interesting things with it with numpy (without the nan hassles above):

3.6.1 3.4.1 Annual Mean and Standard Deviation

```
# calculate the mean albedo over the year
# The axis=0 option tells numpy which
# dimension to apply the function mean() over
mean_albedo = data.mean(axis=0)

# just confirm this is the ndim and shape we expect ...
print mean_albedo.ndim
print mean_albedo.shape

2
(360, 720)

# do a plot

# change the figure size
plt.figure()

plt.title('SW BHR mean albedo for %d' %(year))
# use nearest neighbour interpolation
```

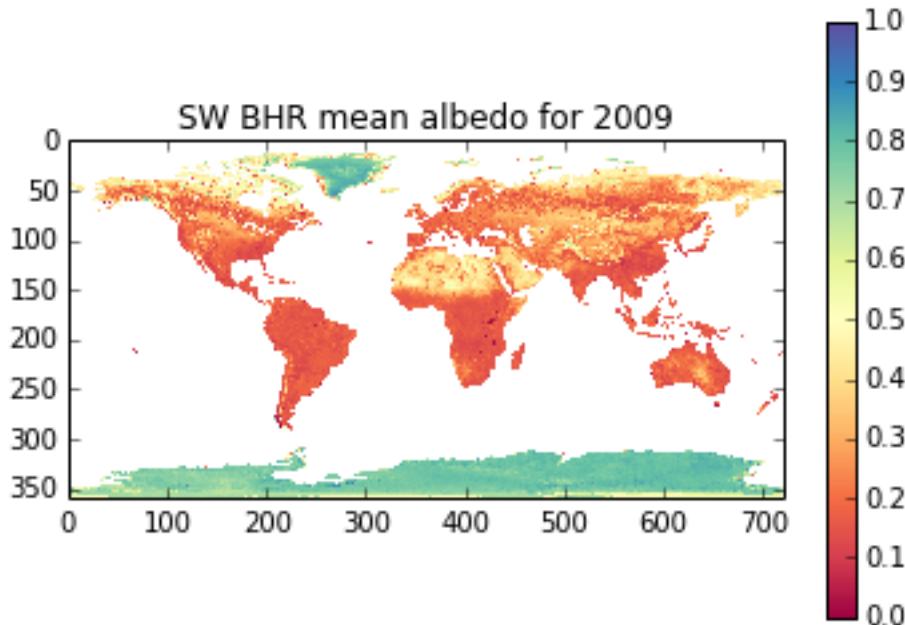
```

plt.imshow(mean_albedo, interpolation='none', \
           cmap=plt.get_cmap('Spectral'), vmin=0.0, vmax=1.0)

# show a colour bar
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x13065ea8>

```



```

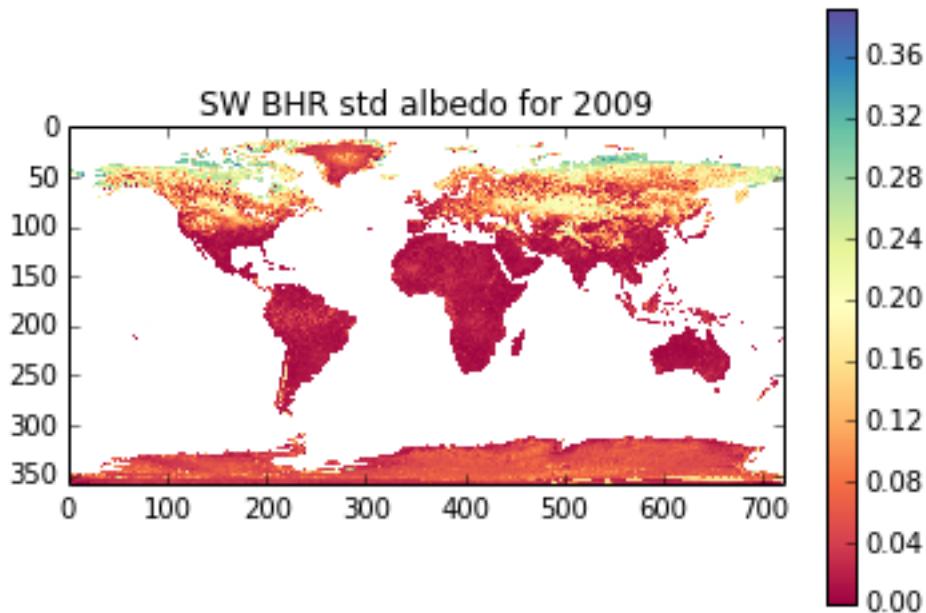
#std
std_albedo = data.std(axis=0)
plt.figure()

plt.title('SW BHR std albedo for %d'%(year))
# use nearest neighbour interpolation
plt.imshow(std_albedo, interpolation='none', \
           cmap=plt.get_cmap('Spectral'), vmin=0.0)

# show a colour bar
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x129f7a28>

```



We can see from this that most of the intra-annual variation in total shortwave albedo is at high latitudes (but *not* Greenland or Antarctica, for instance).

Why do you think that is?

3.6.2 3.4.2 NDVI

It might be interesting to look at NDVI data in this same way and to examine a few more stats using numpy. Even though this is a broadband albedo dataset, we would expect NDVI to give some indication of vegetation activity.

min, max

```
import sys
sys.path.insert(0,'files/python')
from masked import masked

vis = masked(dataset='BHR_VIS')
nir = masked(dataset='BHR_NIR')

ndvi = (nir-vis)/(nir+vis)
# set negative and ndvi > 1 (error) to 0
#ndvi[ndvi>1] = 0
#ndvi[ndvi<0] = 0

mean_ndvi = ndvi.mean(axis=0)
max_ndvi = ndvi.max(axis=0)
min_ndvi = ndvi.min(axis=0)
range_ndvi = max_ndvi - min_ndvi
std_ndvi = ndvi.std(axis=0)

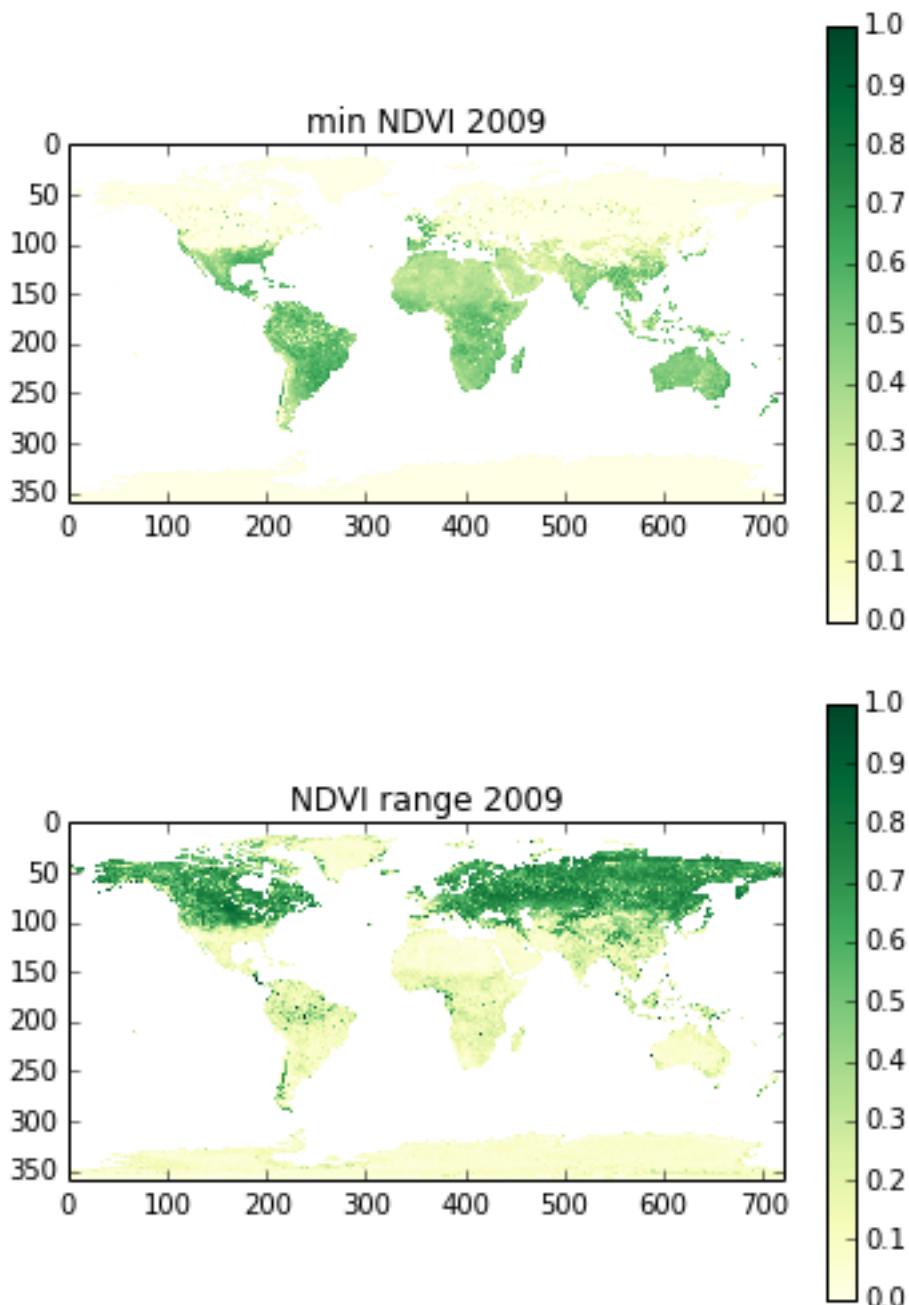
# set up a dictionary
datasets = {"mean NDVI 2009":mean_ndvi,\ 
            "max NDVI 2009":max_ndvi,\ 
            "min NDVI 2009":min_ndvi,\ 
            "NDVI range 2009":range_ndvi,\ 
            "std NDVI 2009":std_ndvi}

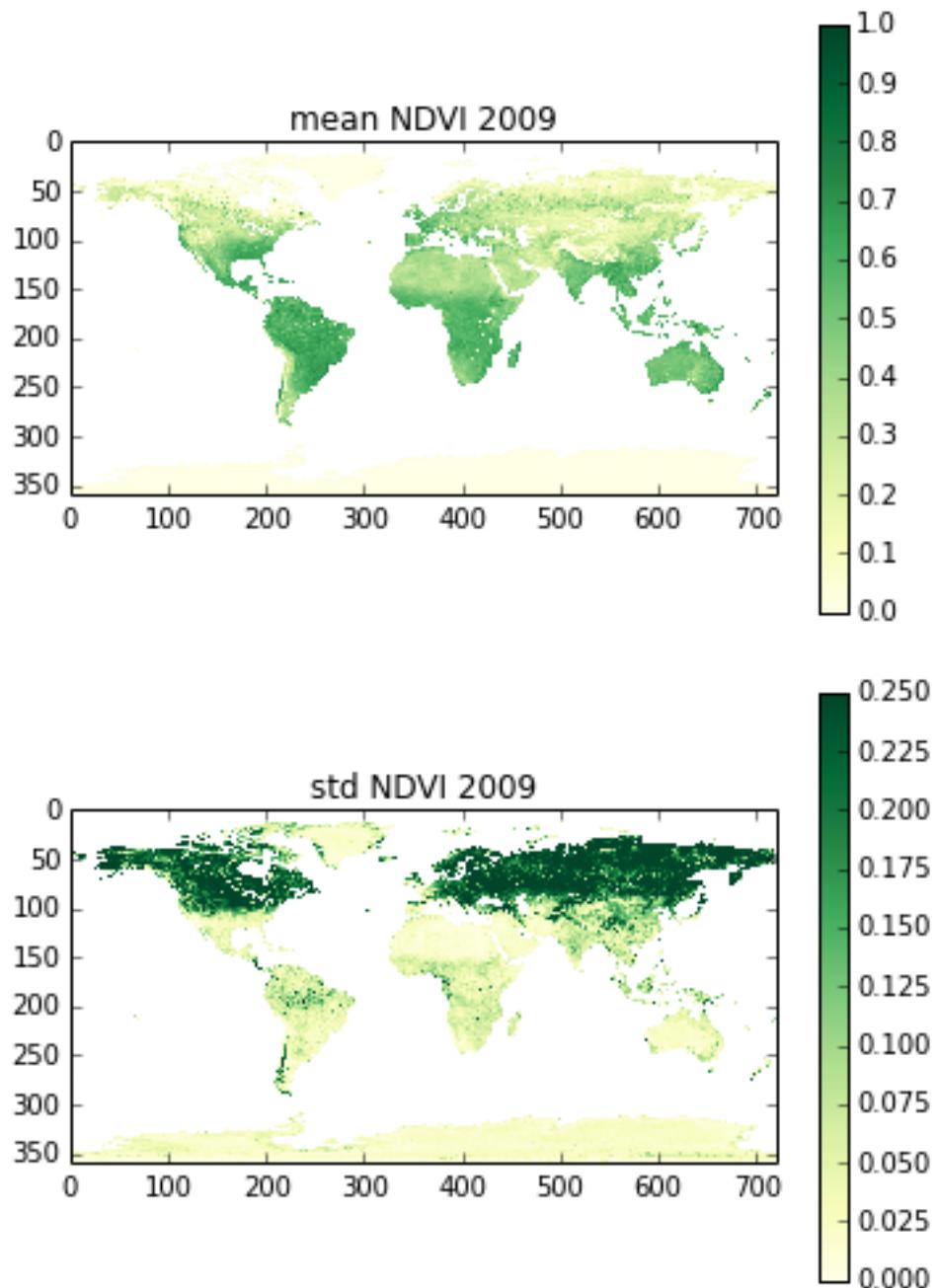
vmax = {"mean NDVI 2009":1.0,\
```

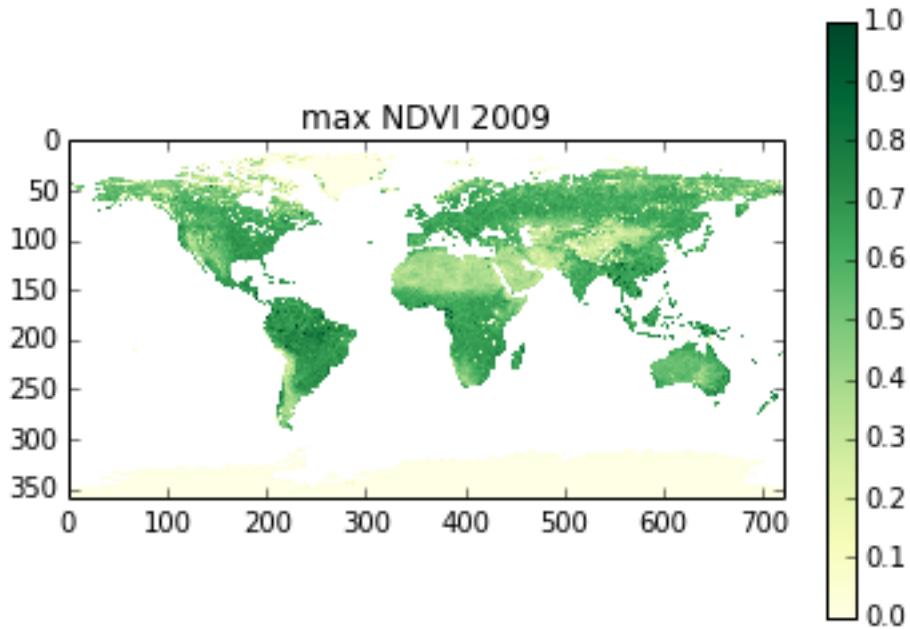
```
"max NDVI 2009":1., \
"min NDVI 2009":1., \
"NDVI range 2009":1., \
"std NDVI 2009":0.25}

# plotting loop

for k in datasets.keys():
    # note the order is not maintained
    # from when we set up the dictionary
    plt.figure()
    plt.title(k)
    plt.imshow(datasets[k], interpolation='none', \
               cmap=plt.get_cmap('YlGn'), vmin=0.0, vmax=vmax[k])
    plt.colorbar()
```







The mean and max NDVI is pretty much what you would expect (MODIS NDVI, Nov-Dec, 2007 shown here):

Figure 3.2: MODIS NDVI, 2007

but again we see the large variations in NDVI in the Northern Hemisphere.

3.7 3.5 Solar Radiation

One common use for albedo data is to calculate the amount of absorbed solar radiation at the surface.

In this section, we will build a model and dataset of downwelling solar radiation.

We can build a *simple* model of solar radiation:

Incoming solar radiation, A , ignoring the effect of the atmosphere, clouds, etc is given by

$$A = E_0 \sin \theta$$

where E_0 is the solar constant (given as $1360 W m^{-2}$), and θ is the solar elevation angle. The solar elevation angle is approximately given by

$$\sin \theta = \cos h \cos \delta \cos \varphi + \sin \delta \sin \varphi$$

where h is the hour angle, δ is the solar declination angle and φ is the latitude. The solar declination can be approximated by

$$\delta = -\arcsin [0.39779 \cos (0.98565 (N + 10) + 1.914 \sin (0.98565 (N - 2)))]$$

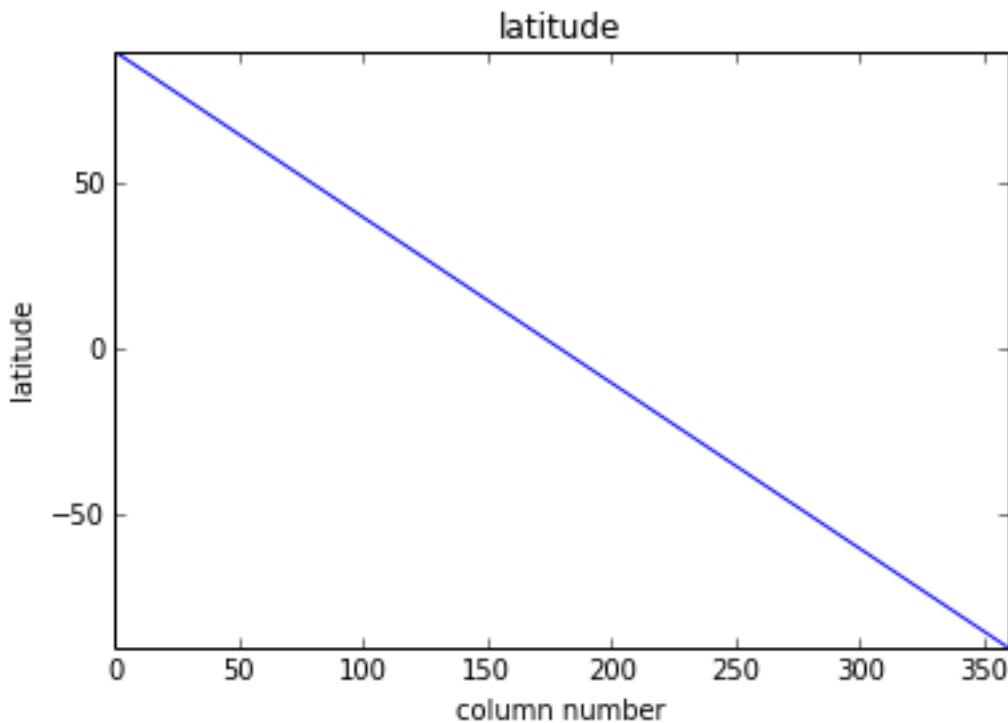
where N is the day of year beginning with $N=0$ at 00:00:00 UTC on January 1

We have the latitude associated with the dataset from:

```
lat = np.array(nc.variables['lat'])

plt.plot(lat)
plt.xlim(0, len(lat))
plt.ylim(-90, 90)
plt.title('latitude')
plt.xlabel('column number')
plt.ylabel('latitude')
```

```
<matplotlib.text.Text at 0x13e9c650>
```



for day of year N then, we have a formula for solar declination.

Approximating N for each month by an average:

```
# mean days per month
av_days = 365.25 / 12.
half = av_days/2.
N = np.arange(half,365.25,av_days)
print N
```



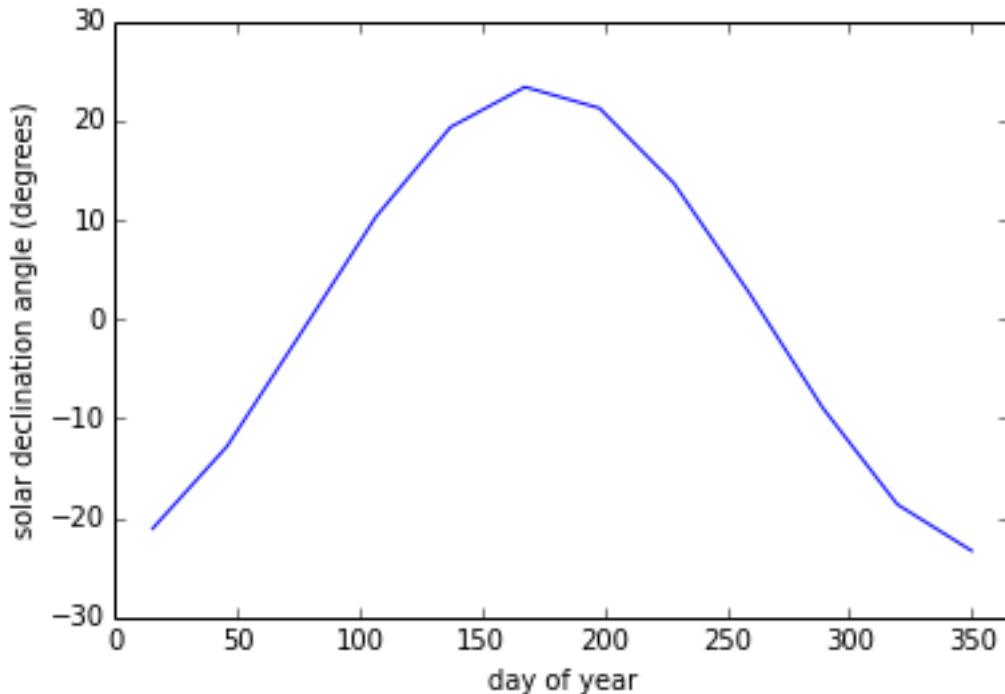
```
[ 15.21875   45.65625   76.09375  106.53125  136.96875  167.40625
 197.84375  228.28125  258.71875  289.15625  319.59375  350.03125]
```

we can approximate the solar declination as:

```
t0 = np.deg2rad (0.98565*(N-2))
t1 = 0.39779*np.cos( np.deg2rad ( 0.98565*(N+10) + 1.914*np.sin ( t0 ) ) )
delta = -np.arcsin ( t1 )

plt.plot(N,np.rad2deg(delta))
plt.xlabel('day of year')
plt.ylabel('solar declination angle (degrees)')
plt.xlim(0,365.25)

(0, 365.25)
```



Although we might think the formulae a little complicated, the translation from the formula to numpy code is really very clear and readable. In particular, by having `N` as an array object, we are able to apply the formulae to the array and not have to obfuscate the code with loops.

Note that the formulae above are given in *degrees*, but trigonometric functions in most programming languages work in *radians*.

In numpy, we can use the functions `deg2rad` and `rad2deg` to convert between degrees and radians and *vice-versa* (alternatively, multiply by `np.pi/180.` and `180./np.pi` respectively).

The trig functions cosine and sine are `cos` and `sin` respectively, and their inverses `arccos` and `arcsin`.

For our solar energy calculations, we will set the ‘hour angle’ to zero (this is solar noon), so:

```
h = 0.0
```

To estimate the solar elevation angle then, we need to include the latitude. At present, we have an array `lat` for latitude and `N` for day of year:

```
print 'lat:', lat.shape
print 'N:', N.shape
```

```
lat: (360,)
N: (12,)
```

what we need is `lat` and `N` in the full 3-D array shape `(12, 360, 720)`.

This involves repeating, which is probably easiest to accomplish with lists:

```
N2 = np.array([[N] * data.shape[1]] * data.shape[2])
print N2.shape
```

```
(720, 360, 12)
```

but the list repetition makes the array the ‘wrong way around’ to what we want, so we apply a transpose operation in numpy:

```
N2 = np.array([[N] * data.shape[1]] * data.shape[2]).T
print N2.shape
```

```
(12, 360, 720)
```

Similarly then with lat, but we have to be a little more careful with the transpose operations:

```
lat2 = np.array([np.array([lat] * data.shape[0]).T] * data.shape[2]).T
print lat2.shape
```

```
(12, 360, 720)
```

At this point, we might consider it useful to write a *function*.

These are of the form:

```
def fn_name(args):
    ...
    return value

def declination(N):
    '''Calculate solar declination (in degrees)'''
    t0 = np.deg2rad(0.98565*(N-2))
    t1 = 0.39779*np.cos(np.deg2rad(0.98565*(N+10) + 1.914*np.sin(t0)))
    delta = -np.arcsin(t1)
    return np.rad2deg(delta)

def solar_elevation(delta,h,lat):
    '''solar elevation in degrees'''
    lat = np.deg2rad(lat)
    delta = np.deg2rad(delta)
    h = np.deg2rad(h)
    sin_theta = np.cos(h)*np.cos(delta)*np.cos(lat) + np.sin(delta)*np.sin(lat)
    return np.rad2deg(np.arcsin(sin_theta))

# create numpy arrays of the right shape

N2 = np.array([[N] * data.shape[1]] * data.shape[2]).T
lat2 = np.array([np.array([lat] * data.shape[0]).T] * data.shape[2]).T

# zeros_like creates array of the same shape as N2 here
# filled with zeros. Similarly ones_like
h2 = np.zeros_like(N2) + h

# now put these things together

# copy the arrays, so we don't overwrite the data

delta = declination(N2.copy())
e0 = 1360.
sea = solar_elevation(delta,h2.copy(),lat2.copy())
sin_theta = np.sin(np.deg2rad(sea))
rad = e0*sin_theta
# threshold at zero
rad[rad < 0] = 0.0

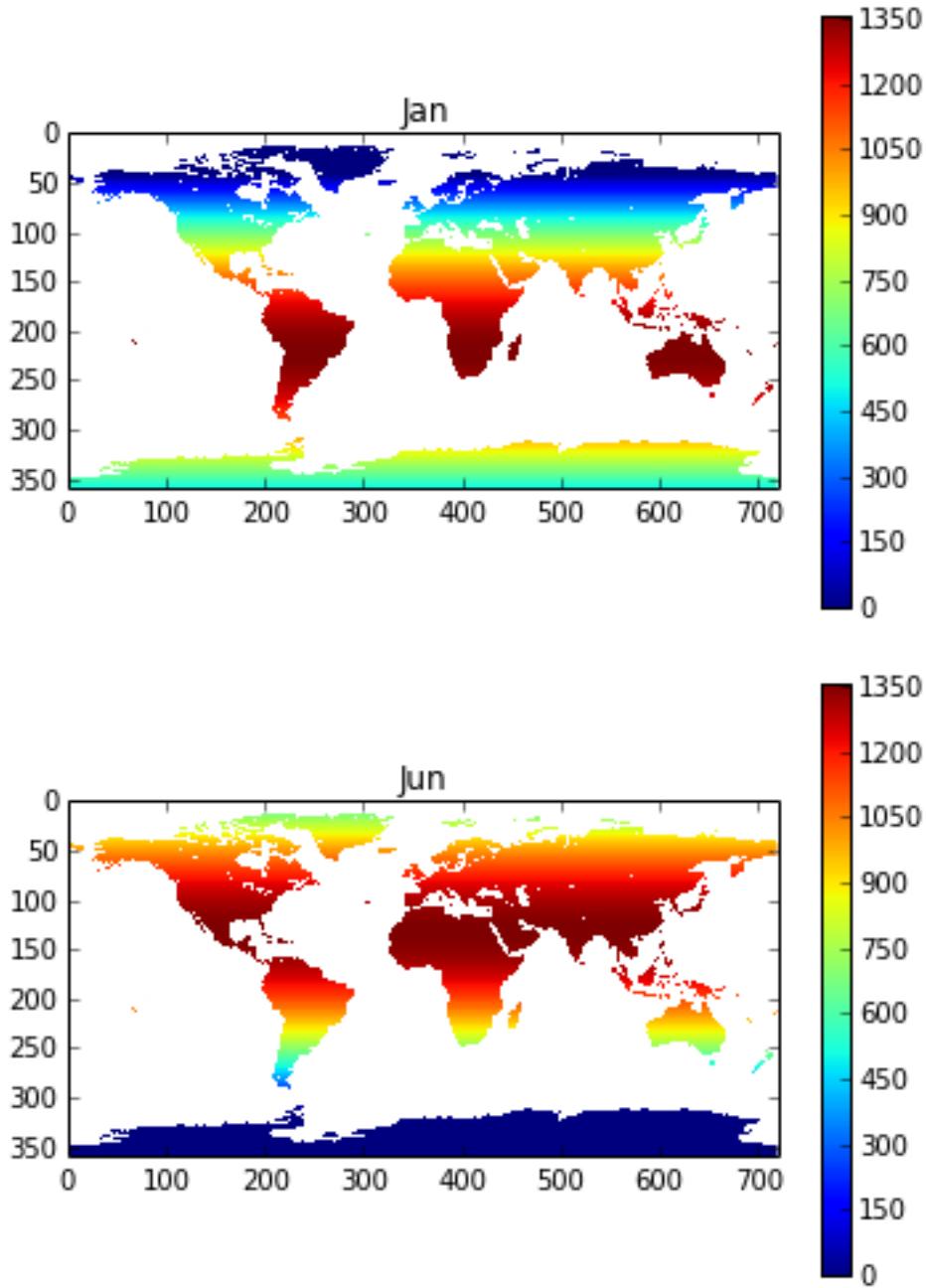
incoming_rad = rad
# now mask this with data mask
rad = ma.array(incoming_rad,mask=data.mask)

# plot to visualise the data and see it looks reasonable

plt.figure()
plt.title('Jan')
plt.imshow(rad[0],interpolation='none',cmap=plt.get_cmap('jet'))
plt.colorbar()
```

```
plt.figure()
plt.title('Jun')
plt.imshow(rad[5], interpolation='none', cmap=plt.get_cmap('jet'))
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x13aad6c8>
```



This sort of application really shows the power of manipulating arrays.

3.8 E3.5 Exercise: Solar Radiation Code: A chance for you to get feedback

In the section above, we have developed some code to estimate solar radiation and we have packaged some of it into functions.

In a file called “files/python/solar.py“, develop some Python code that will

1. Read GlobAlbedo data DHR_SW (shortwave directional-hemispherical reflectance) into a masked array data
2. Generate a spatial Solar Radiation dataset rad of the same shape as data
3. Calculate the amount of solar radiation *absorbed* at the surface (W m⁻²)
4. Calculate the *latitudinal total* absorbed radiation for each month (W m⁻²)
5. Calculate the *global total* absorbed radiation power for each month (W)

Where appropriate, produce images, graphs and/or movies of your results.

Comment on the patterns of absorbed solar radiation that you see.

All of your code should be commented as well as possible.

It should indicate which parts of the code you have simply taken from the material above and which parts you have developed yourself.

If possible, use an ipython notebook to illustrate your code and its operation. As a minimum, submit your code and examples of its output.

If possible, wrap your codes into functions to group things that belong together together.

You will probably want to save the arrays you generated (np.savez) so that you can show your results later.

Hints:

- All of the code you need for parts 1. and 2. can be found in the material above ... you just need to put it together
- Recall that albedo is the proportion of radiation *reflected*.
- Recall that in a lat/long projection the area represented by each pixel varies as a function of latitude.

We will *not* release model answers for this exercise at present.

Instead, we would like you to complete this exercise over the coming week and submit it by the following Monday morning. If you do this, there will be sessions run in (extended) office hours (9-12 Monday, 9-12 Tuesday, 9-10 Wednesday) when you will be given feedback on your submission. To get the most out of these feedback sessions, it would be advisable to request a time slot (You should be getting a doodle poll email to that effect)

We *strongly* recommend that you attempt this exercise on your own. Given the amount of material you are given here, this should be achievable for all of you.

This work does *not* form part of your formal assessment. Instead it is intended to provide you with an opportunity for feedback on how you are getting on in the course at an early stage. So, you are *not* required to do this. If you do not though, we cannot give you feedback.

Come along to the feedback session prepared to *explain* how your code operates and how to run it (one reason why a notebook would be useful if you can manage that).

3.9 3.6 Saving and loading numpy data

When we have gone to lots of effort (well, lots of processing time) to do some calculation, we probably want to save the result.

In numpy, we can save one or more numpy arrays (and similar data types) with np.savez().

So, e.g. to save the arrays data and rad:

```
import numpy as np
# there is a problem saving masked arrays at the moment,
# so convert and save the mask
```

```
np.savez('files/data/solar_rad_data.npz', \
         rad=np.array(rad), data=np.array(data), mask=data.mask)
```

To load this again use `np.load`, which returns a dictionary:

```
f = np.load('files/data/solar_rad_data.npz')
data = ma.array(f['data'], mask=f['mask'])
rad = ma.array(f['rad'], mask=f['mask'])
```

As an aside here, we note that numpy also has convenient functions for loading ASCII text (which is much simpler to use than the methods we learned last session, but you need to know how to do it from scratch if the data format is very awkward).

That said, we can deal with some awkward issues in a dataset:

```
!head -15 < files/data/heathrowdata.txt
```

```
Heathrow (London Airport)
Location 5078E 1767N 25m amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by ---.
Sunshine data taken from an automatic Kipp & Zonen sensor marked with a #, otherwise sunshine data
    yyyy mm tmax tmin af rain sun
          degC degC days mm hours
  1948  1   8.9  3.3 --- 85.0 ---
  1948  2   7.9  2.2 --- 26.0 ---
  1948  3  14.2  3.8 --- 14.0 ---
  1948  4  15.4  5.1 --- 35.0 ---
  1948  5  18.1  6.9 --- 57.0 ---
  1948  6  19.1 10.3 --- 67.0 ---
  1948  7  21.7 12.0 --- 21.0 ---
  1948  8  20.8 11.7 --- 67.0 ---
```

```
# read sun hours from the heathrow data file
```

```
filename = 'files/data/heathrowdata.txt'

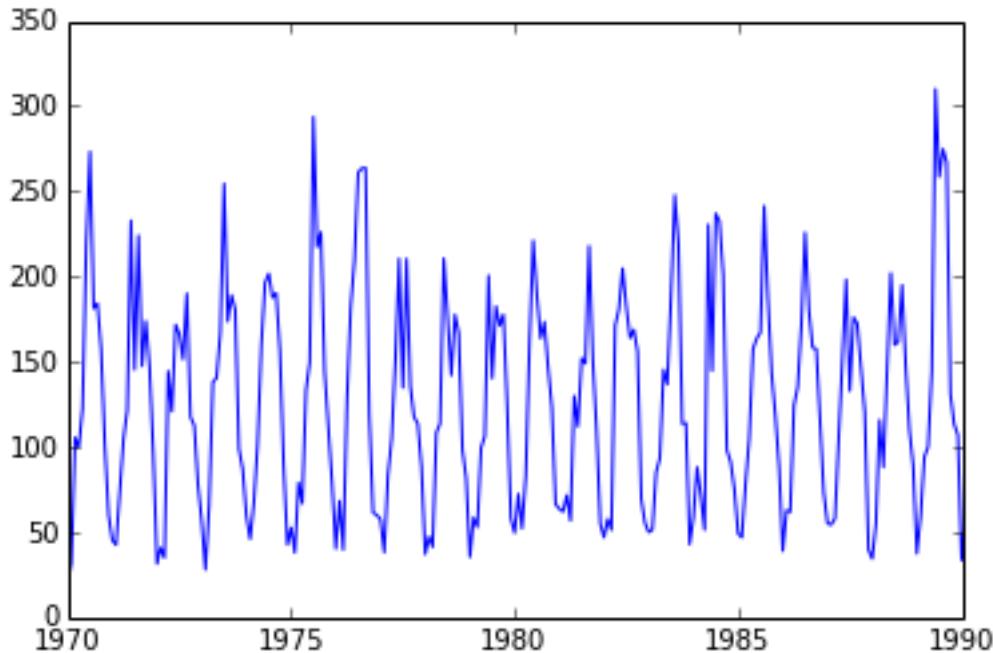
# In this dataset, we need to know what
# to do when we get '---' in the dataset
# so we set up a
# translation dictionary
def awkward(x):
    if x == '---':
        return np.nan
    return x

trans = {6: awkward}

# skip the first 7 rows
# use the awkward translation for column 6
# unpack gets the shape the right way around
sun_hrs = np.loadtxt(filename, skiprows=7, usecols=[0, 1, 6], converters=trans, unpack=True)

print sun_hrs.shape
# plot decimal year and sun hours
plt.plot(sun_hrs[0]+sun_hrs[1]/12., sun_hrs[2])
# zoom in to 1970 to 1990
plt.xlim(1970, 1990)

(3, 785)
(1970, 1990)
```



We can save this to a text file with:

```
header = 'year month sun_hours'
# note, we transpose it if we want the data as columns
np.savetxt('files/data/sunshine.txt', sun_hrs.T, header=header)

!head -10 < files/data/sunshine.txt

# year month sun_hours
1.948000000000000e+03 1.000000000000000e+00 nan
1.948000000000000e+03 2.000000000000000e+00 nan
1.948000000000000e+03 3.000000000000000e+00 nan
1.948000000000000e+03 4.000000000000000e+00 nan
1.948000000000000e+03 5.000000000000000e+00 nan
1.948000000000000e+03 6.000000000000000e+00 nan
1.948000000000000e+03 7.000000000000000e+00 nan
1.948000000000000e+03 8.000000000000000e+00 nan
1.948000000000000e+03 9.000000000000000e+00 nan
```

3.10 E3.6 Exercise: awkward reading

The file `files/data/elevation.dat` look like:

```
!head -10 < files/data/elevation.dat

2013/10/8 00:00:00 -44.2719952943
2013/10/8 00:30:00 -43.5276412785
2013/10/8 00:59:59 -41.9842746582
2013/10/8 01:30:00 -39.7226999863
2013/10/8 02:00:00 -36.8452198361
2013/10/8 02:30:00 -33.459799008
2013/10/8 03:00:00 -29.6691191507
2013/10/8 03:30:00 -25.5652187709
2013/10/8 03:59:59 -21.2281801291
2013/10/8 04:30:00 -16.7272357302
```

Write some code to read these data in using `np.loadtxt`

Hint: You will need to write a function that decodes column 0 and 1, for example:

```
def col0(x):
    # interpret column 1 as a sort of
    # decimal year
    # NB this is very rough ...
    # not all months have 30 days ...
    y,m,d = x.split('/')
    dec_year = y + (m-1)/12. + (d-1)/30.
    return dec_year

# translation ... just for column 1 here
# you'd need to define your own for column 2
trans = {0 : col0}
```

3.11 Advanced and Answers

If you feel comfortable with the contents of this session, or simply would like something a little more challenging, examine the notes in the advanced section.

Answers for the exercises in this session are available, as are the answers for the advanced section.

CHAPTER 4. GEOSPATIAL DATA

In this session, we will introduce the `gdal` geospatial module which can read a wide range of scientific data formats. You will find that using it to read data is quite similar to the work we did last week on netCDF datasets.

The main challenges are also much the same: very often, you need to be able to read data from a ‘stack’ of image files and generate a useful 3D (space and time) dataset from these. Once you have the data in such a form, there are *many* things we can do with it, and very many of these are convenient to do using array-based expressions such as in `numpy` (consider the simplicity of the expression `absorbed = rad * (1 - albedo)` from last week’s exercise).

That said, it can sometimes be quite an effort to prepare datasets in this form. Last week, we developed a ‘valid data’ mask from the GlobAlbedo dataset, as invalid data were stored as `nan`. Very often though, scientific datasets have more complex ‘Quality Control’ (QC) information, that gives per-pixel information describing the quality of the product at that location (e.g. it was very cloudy so the results are not so good).

To explore this, we will first consider the [MODIS Leaf Area Index \(LAI\)](#) product that is mapped at 1 km resolution, every 8 days from the year 2000.

We will learn how to read in these data (in `hdf` format) using `gdal`, and how to interpret the QC information in such products to produce valid data masks. As an exercise, you will wrap some code around that to form a 3D masked array of the dataset.

Next, we will consider how to download such data. This should be a reinforcement of material from last week, but it is useful to know how to conveniently access NASA data products. A challenge in the exercise then is to download a different dataset (MODIS snow cover) for the UK, and form a masked 3D dataset from this.

Finally, we will introduce vector datasets and show you python tools that allow you (among many other things) to build a mask in the projection and sampling of your spatial dataset (MODIS LAI in this case).

There are many features and as many complexities to the Python tools we will deal with today, but in this material, we cover some very typical tasks you will want to do. They all revolve around generating masked 3D datasets from NASA MODIS datasets, which is a very useful form of global biophysical information over the last decade+. We also provide much material for further reading and use when you are more confident in your programming.

A final point here is that the material we cover today is very closely related to what you will need to do in the first section of your assessed practical that we will introduce next week, so you really need to get to grips with this now.

There is not as much ‘new’ material as in previous weeks now, but we assume that you have understood, and can make use of, material from those lectures.

First, we will examine data from a NASA MODIS product on Leaf Area Index (LAI).

- 4.1 MODIS LAI product
- 4.2 Downloading data
- 4.3 Vector masking

4.1 4.1 MODIS LAI product

GDAL covers a much wider set of file formats and methods than the netCDF library that we previously used.

Basic operation involves:

- load gdal
- open a spatial dataset (an hdf format file here)
- specify which subsets you want.

We can explore the subsets in the file with `GetSubDatasets()`:

```
# how to find out which datasets are in the file

import gdal # Import GDAL library bindings

# The file that we shall be using
# Needs to be on current directory
filename = 'files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'

g = gdal.Open(filename)
# g should now be a GDAL dataset, but if the file isn't found
# g will be none. Let's test this:
if g is None:
    print "Problem opening file %s!" % filename
else:
    print "File %s opened fine" % filename

subdatasets = g.GetSubDatasets()
for fname, name in subdatasets:
    print name
    print "\t", fname

File files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf opened fine
[1200x1200] Fpar_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] Lai_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparLai_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparExtra_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] LaiStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
```

In the previous code snippet we have done a number of different things:

1. Import the GDAL library
2. Open a file with GDAL, storing a handler to the file in `g`
3. Test that `g` is not `None` (as this indicates failure opening the file. Try changing `filename` above to something else)
4. We then use the `GetSubDatasets()` method to read out information on the different subdatasets available from this file (compare to the output of `gdalinfo` on the shelf earlier)
5. Loop over the retrieved subdatasets to print the name (human-readable information) and the GDAL filename. This last item is the filename that you need to use to tell GDAL to open a particular data layer of the 6 layers present in this example

Let's say that we want to access the LAI information. By contrasting the output of the above code (or `gdalinfo`) to the contents of the [LAI/fAPAR product information page](#), we find out that we want the layers for `Lai_1km`, `FparLai_QC`, `FparExtra_QC` and `LaiStdDev_1km`.

To read these individual datasets, we need to open each of them individually using GDAL, and the GDAL filenames used above:

```
# How to access specific datasets in gdal

# Let's create a list with the selected layer names
selected_layers = [ "Lai_1km", "FparLai_QC", "LaiStdDev_1km" ]

# We will store the data in a dictionary
# Initialise an empty dictionary
data = {}

# for convenience, we will use string substitution to create a
# template for GDAL filenames, which we'll substitute on the fly:
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
# This has two substitutions (the %s parts) which will refer to:
# - the filename
# - the data layer

for i, layer in enumerate ( selected_layers ):
    this_file = file_template % ( filename, layer )
    print "Opening Layer %d: %s" % (i+1, this_file )
    g = gdal.Open ( this_file )

    if g is None:
        raise IOError
    data[layer] = g.ReadAsArray()
    print "\t>>> Read %s!" % layer

Opening Layer 1: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.2011213154534.hdf":MOD_
    >>> Read Lai_1km!
Opening Layer 2: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.2011213154534.hdf":MOD_
    >>> Read FparLai_QC!
Opening Layer 3: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.2011213154534.hdf":MOD_
    >>> Read LaiStdDev_1km!
```

In the previous code, we have seen a way of neatly creating the filenames required by GDAL to access the independent datasets: a template string that gets substituted with the filename and the layer name. Note that the presence of double quotes in the template requires us to use single quotes around it. The data is now stored in a dictionary, and can be accessed as e.g. `data['Lai_1km']` which is a numpy array:

```
type(data['Lai_1km'])

numpy.ndarray

print data['Lai_1km']

[[ 3  3  2 ...,  6  8 21]
 [ 4  3  6 ...,  8 18 14]
 [ 3 12 11 ..., 12  8  8]
 ...,
 [ 2  3  2 ..., 18 11 17]
 [ 2  3  3 ..., 16 19 15]
 [ 3  2  2 ..., 15 16 15]]
```

Now we have to translate the LAI values into meaningful quantities. According to the [LAI](#) webpage, there is a scale factor of 0.1 involved for LAI and SD LAI:

```
lai = data['Lai_1km'] * 0.1
lai_sd = data['LaiStdDev_1km'] * 0.1

print "LAI"
print lai
print "SD"
print lai_sd

LAI
[[ 0.3  0.3  0.2 ... ,  0.6  0.8  2.1]
 [ 0.4  0.3  0.6 ... ,  0.8  1.8  1.4]
 [ 0.3  1.2  1.1 ... ,  1.2  0.8  0.8]
 ...
 [ 0.2  0.3  0.2 ... ,  1.8  1.1  1.7]
 [ 0.2  0.3  0.3 ... ,  1.6  1.9  1.5]
 [ 0.3  0.2  0.2 ... ,  1.5  1.6  1.5]]
SD
[[ 0.2  0.2  0.1 ... ,  0.2  0.1  0.3]
 [ 0.2  0.2  0.2 ... ,  0.2  0.3  0.2]
 [ 0.  0.1  0.2 ... ,  0.1  0.2  0.2]
 ...
 [ 0.1  0.1  0.1 ... ,  0.3  0.  0.1]
 [ 0.1  0.1  0.1 ... ,  0.2  0.2  0.1]
 [ 0.1  0.1  0.1 ... ,  0.1  0.2  0.1]]

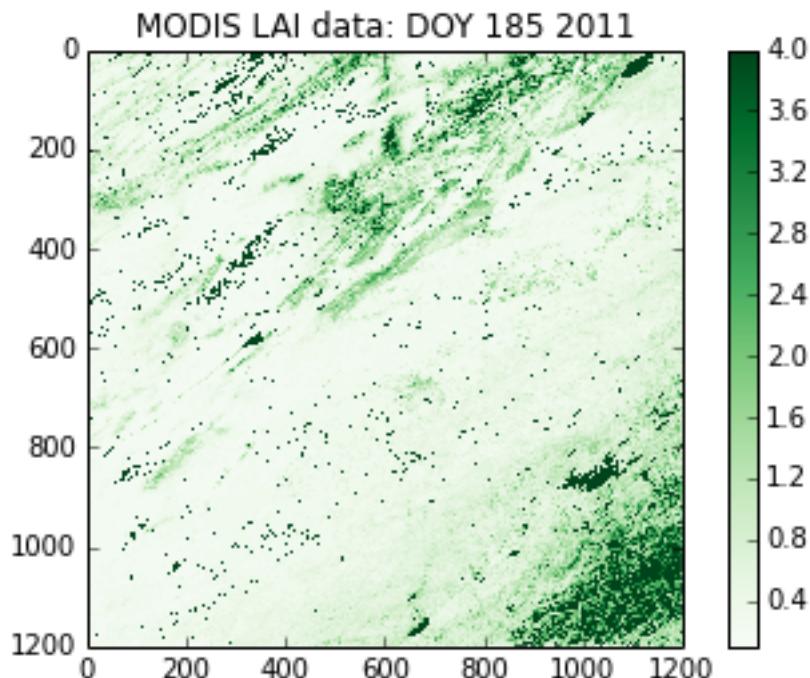
# plot the LAI

import pylab as plt

# colormap
cmap = plt.cm.Greens

plt.imshow(lai,interpolation='none',vmin=0.1,vmax=4.,cmap=cmap)
plt.title('MODIS LAI data: DOY 185 2011')
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc680e1830>
```



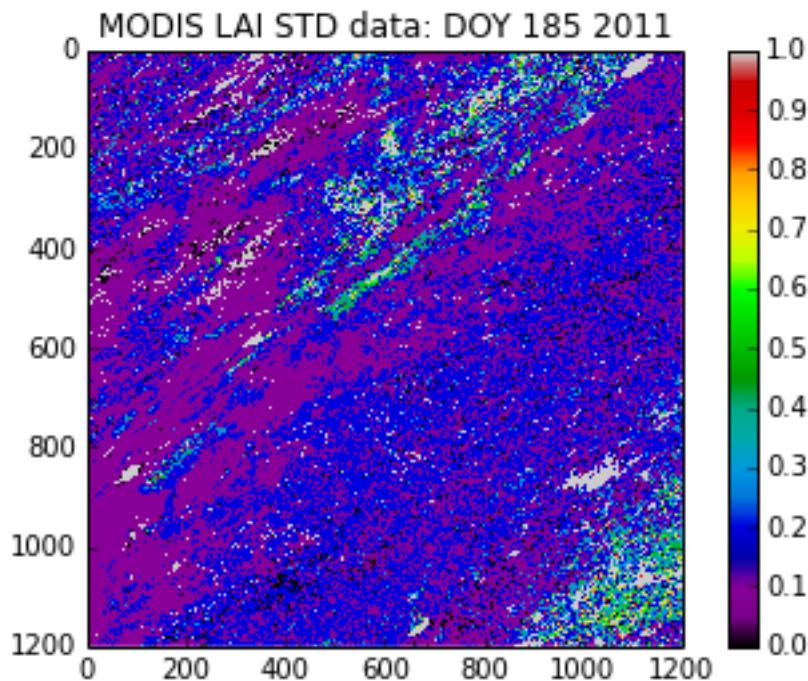
```
# plot the LAI std

import pylab as plt

# colormap
cmap = plt.cm.spectral
# this sets the no data colour. 'k' is black

plt.imshow(lai_sd, interpolation='none', vmax=1., cmap=cmap)
plt.title('MODIS LAI STD data: DOY 185 2011')
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc6954c098>
```



It is not possible to produce LAI estimates if it is persistently cloudy, so the dataset may contain some gaps.

These are identified in the dataset using the QC (Quality Control) information.

We should then examine this.

The codes for this are also given on the LAI product page. They are described as bit combinations:

Bit No.

Parameter Name	Bit Combination	Explanation
----------------	-----------------	-------------

0

MODLAND_QC bits

0

Good quality (main algorithm with or without saturation)

1

Other Quality (back-up algorithm or fill values)

1

Sensor

0</td><td> TERRA</td>

1

AQUA

2

DeadDetector

0

Detectors apparently fine for up to 50% of channels 1 2

1

Dead detectors caused >50% adjacent detector retrieval

3-4

CloudState

00

Significant clouds NOT present (clear)

01

Significant clouds WERE present

10

Mixed clouds present on pixel

11

Cloud state not defined assumed clear

5-7

CF_QC

000

Main (RT) method used best result possible (no saturation) </td>

001

Main (RT) method used with saturation. Good very usable

010

Main (RT) method failed due to bad geometry empirical algorithm used

011

Main (RT) method failed due to problems other than geometry empirical algorithm used

100

Pixel not produced at all value couldn't be retrieved (possible reasons: bad L1B data unusable MODAGAGG data)

In using this information, it is up to the user which data he/she wants to pass through for any further processing. There are clearly trade-offs: if you look for only the highest quality data, then the number of samples is likely to be lower than if you were more tolerant. But if you are too tolerant, you will get spurious results. You may find useful information on how to convert from actual QA flags to diagnostics in [this page](#) (they focus on NDVI/EVI, but the theory is the same).

But let's just say that we want to use only the highest quality data.

This means we want bit 0 to be 0 ...

Let's have a look at the QC data:

```
qc = data['FparLai_QC'] # Get the QC data which is an unsigned 8 bit byte
print qc , qc.dtype

[[2 2 0 ..., 0 2 2]
[2 2 0 ..., 2 0 2]
[0 2 0 ..., 0 0 0]
...
[0 0 2 ..., 0 8 0]
[0 0 0 ..., 0 0 2]
[0 2 0 ..., 2 2 2]] uint8
```

We see various byte values:

```
np.unique(qc)

array([ 0,   2,   8,  10,  16,  18,  32,  34,  40,  42,  48,  50,  97,
       99, 105, 107, 113, 115, 157], dtype=uint8)

# translated into binary using bin()
for i in np.unique(qc):
    print i,bin(i)

0 0b0
2 0b10
8 0b1000
10 0b1010
16 0b10000
18 0b10010
32 0b100000
34 0b100010
40 0b101000
42 0b101010
48 0b110000
50 0b110010
97 0b1100001
99 0b1100011
105 0b1101001
107 0b1101011
113 0b1110001
115 0b1110011
157 0b10011101
```

We could try to come up with an interpretation of each of these ... or we could try to mask the qc bytes to see bit 0 only if that's what we are interested in. This is quite possibly a new concept for most of you, but it is very common that when interpreting QC data in data products, you need to think about bit masking. You will find more details on this in the advanced section of Chapter 1, but we will consider the minimum we need right now.

Byte data are formed of 8 bits, e.g.:

```
105 == (1 * 2**6) + (1 * 2**5) + (0 * 2**4) + (1 * 2**3) + (0 * 2**2) + (0
* 2**1) + (1 * 2**0)
```

So, in binary, we represent the decimal number 105 by 1101001 as we saw above.

The QC values are to be interpreted in this manner.

If we want *only* bit 1, we can perform a *bitwise* operation with the byte data.

In this case, it would be an ‘and’ operation (&) with the value 1:

```
# suppose we consider the value 105
# which from above, we know to have
# bit 0 set as 1
test = 105
```

```
bit_zero = test & 1
print bit_zero

1

# suppose we consider the value 104
# which we could work out has bit 1 as 0
test = 104
bit_zero = test & 1
print bit_zero

0

# other bit fields are a 'little' more complicated
tests = np.unique(qc)

for t in tests:
    # if we want bit field 5-7
    # we form a binary mask
    mask57 = 0b11100000
    # but 0
    mask0 = 0b00000001
    # and use & as before and right shift 5 (>> 5)
    qa57 = (t & mask57) >> 5
    qa0 = (t & mask0) >> 0
    print t,qa57,qa0,bin(t)

0 0 0 0b0
2 0 0 0b10
8 0 0 0b1000
10 0 0 0b1010
16 0 0 0b10000
18 0 0 0b10010
32 1 0 0b100000
34 1 0 0b100010
40 1 0 0b101000
42 1 0 0b101010
48 1 0 0b110000
50 1 0 0b110010
97 3 1 0b1100001
99 3 1 0b1100011
105 3 1 0b1101001
107 3 1 0b1101011
113 3 1 0b1110001
115 3 1 0b1110011
157 4 1 0b10011101
```

So, for example (examining the table above) 105 is interpreted at 0b011 in fields 5 to 7 (which is 3 in decimal). This indicates that ‘Main (RT) method failed due to problems other than geometry empirical algorithm used’. Here, bit zero is set to 1, so this is a ‘bad’ pixel.

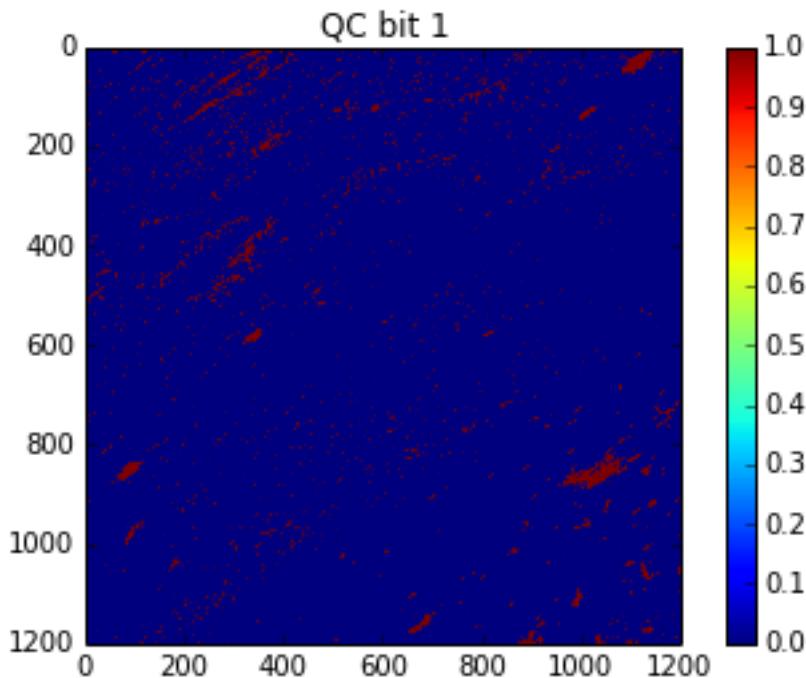
In this case, we are only interested in bit 0, which is an easier task than interpreting all of the bits.

```
# the good data are where qc bit 1 is 0

qc = data['FparLai_QC'] # Get the QC data
# find bit 0
qc = qc & 1

plt.imshow(qc)
plt.title('QC bit 1')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar instance at 0xc6b71b8>
```



We can use this mask to generate a masked array. Masked arrays, as we have seen before, are like normal arrays, but they have an associated mask.

Remember that the mask in a masked array should be `False` for good data, so we can directly use `qc` as defined above.

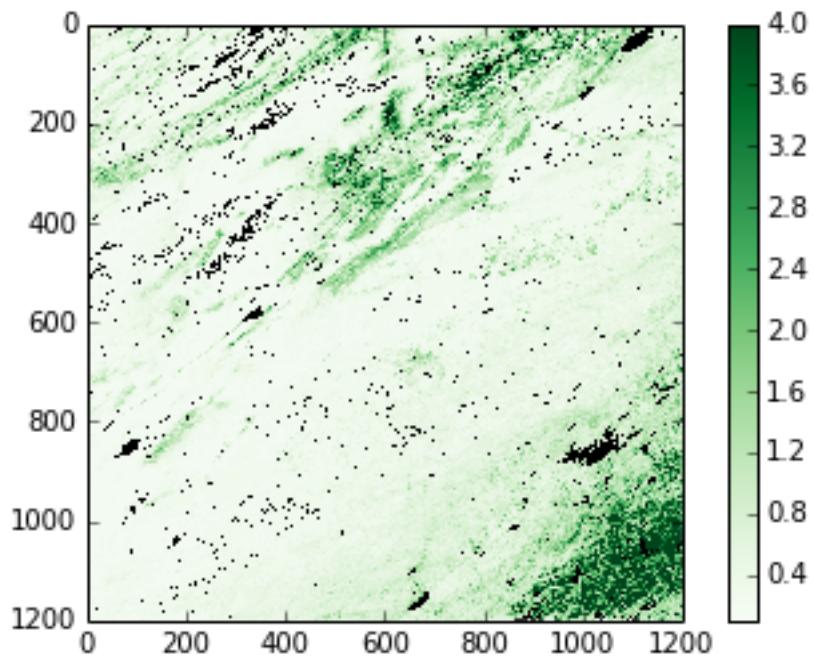
We shall also choose another colormap (there are [lots to choose from](#)), and set values outside the 0.1 and 4 to be shown as black pixels.

```
# colormap
cmap = plt.cm.Greens
cmap.set_bad( 'k' )
# this sets the no data colour. 'k' is black

# generate the masked array
laim = np.ma.array( lai, mask=qc )

# and plot it
plt.imshow( laim, cmap=cmap, interpolation='none', vmin=0.1, vmax=4)
plt.colorbar()
```

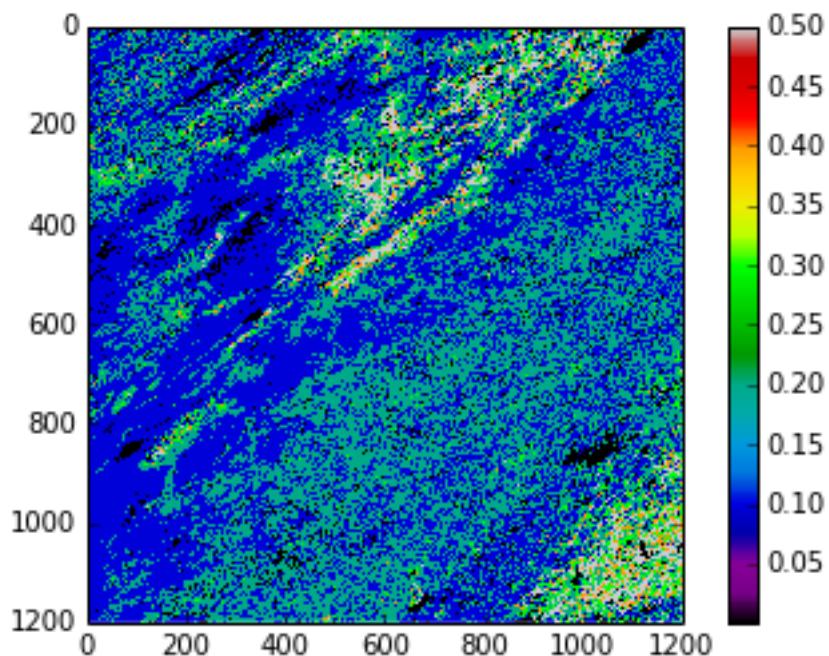
```
<matplotlib.colorbar.Colorbar instance at 0x2adc6b8dd5a8>
```



Similarly, we can do a similar thing for Standard Deviation

```
cmap = plt.cm.spectral
cmap.set_bad( 'k' )
stdm = np.ma.array( lai_sd, mask=qc )
plt.imshow( stdm, cmap=cmap, interpolation='none', vmin=0.001, vmax=0.5)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc70725878>
```



For convenience, we might wrap all of this up into a function:

```
import gdal
import numpy as np
import numpy.ma as ma
```

```

def getLAI(filename, \
    qc_layer = 'FparLai_QC', \
    scale = [0.1, 0.1], \
    selected_layers = ["Lai_1km", "LaiStdDev_1km"]):

    # get the QC layer too
    selected_layers.append(qc_layer)
    scale.append(1)
    # We will store the data in a dictionary
    # Initialise an empty dictionary
    data = {}
    # for convenience, we will use string substitution to create a
    # template for GDAL filenames, which we'll substitute on the fly:
    file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
    # This has two substitutions (the %s parts) which will refer to:
    # - the filename
    # - the data layer
    for i,layer in enumerate(selected_layers):
        this_file = file_template % (filename, layer)
        g = gdal.Open (this_file)

        if g is None:
            raise IOError

        data[layer] = g.ReadAsArray() * scale[i]

    qc = data[qc_layer] # Get the QC data
    # find bit 0
    qc = qc & 1

    odata = {}
    for layer in selected_layers[:-1]:
        odata[layer] = ma.array(data[layer],mask=qc)

    return odata

filename = 'files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'

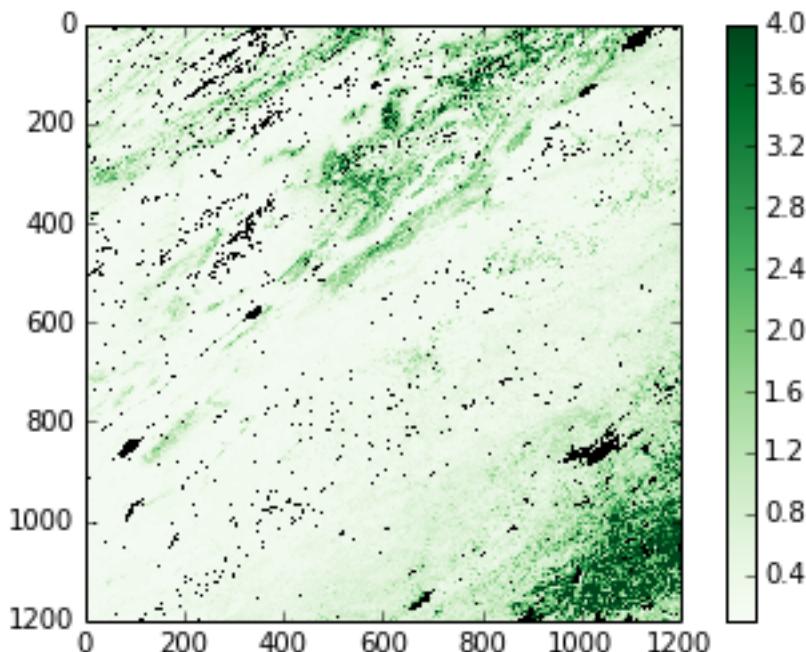
lai_data = getLAI(filename)

# colormap
cmap = plt.cm.Greens
cmap.set_bad ('k')
# this sets the no data colour. 'k' is black

# and plot it
plt.imshow (lai_data['Lai_1km'], cmap=cmap, interpolation='nearest', vmin=0.1, vmax=4)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc709ffe18>

```



4.2 Exercise 4.1

You are given the MODIS LAI data files for the year 2012 in the directory `files/data` for the UK (MODIS tile h17v03).

Read these LAI datasets into a masked array, using QA bit 0 to mask the data (i.e. good quality data only) and generate a movie of LAI.

You should end up with something like:

4.3 4.2 Downloading data

For the exercise and notes above, you were supplied with several datasets that had been previously downloaded.

4.3.1 4.2.1 Reverb

These NASA data can be accessed in several ways (except on Wednesdays when they go down for maintainance (or when there is a Government shutdown ...)). The most direct way is to use [Reverb](#) to explore and access data. If you do this, you can, for example [search for MODIS snow cover MOD10 datasets covering the UK](#) for some given time period. If you follow this through, e.g. selecting [MODIS/Terra Snow Cover Daily L3 Global 500m SIN Grid V005](#) and then search for ‘granules’, you should get access to the datasets you want (select e.g. one of the files for gid h17v03 and save to cart). You then view the items in your cart, click ‘download’ and then ‘save’.

This should give you a text file with some urls in it, e.g.:

```
ftp://n4ft101u.ecs.nasa.gov/DP0/MOST/MOD10A1.005/2013.02.21/MOD10A1.A2013052.h17v03.005.  
ftp://n4ft101u.ecs.nasa.gov/DP0/MOST/MOD10A1.005/2013.02.21/MOD10A1.A2013052.h17v03.005.  
ftp://n4ft101u.ecs.nasa.gov/DP0/BRWS/Browse.001/2013.02.23/BROWSE.MOD10A1.A2013052.h17v03.005.  
http://browse.echo.nasa.gov/NSIDC_ECS/2013/02/23/:BR:Browse.001:46841269:1.BINARY
```

Here, we can note that one of them is a jpeg file:

which is the quicklook (BROWSE file) for that tile / date.

The hdf dataset is, in this case `ftp://n4ftl01u.ecs.nasa.gov/DP0/MOST/MOD10A1.005/2013.02.21/MOD10A1...`

From this, we see that the data server is `n4ftl01u.ecs.nasa.gov` and that the MODIS snow products for the MODIS Terra instrument are in the directory:

`ftp://n4ftl01u.ecs.nasa.gov/DP0/MOST.`

If we explored that, we would find the datasets from the MODIS Aqua platform were in:

`ftp://n4ftl01u.ecs.nasa.gov/DP0/MOSA.`

The directories below that give the date and filename.

4.3.2 4.2.2 FTP access

Now we have discovered something about the directory structure on the server, we could explore this to get the datasets we want (rather than having to go through Reverb).

Some of the datasets on Reverb are accessible only through `http`, but the snow products (at present) are available by `ftp`.

In either case, if we want to run some ‘batch’ process to download many files (e.g. all files for a year for some tile), the first thing we need is the set of urls for the files we want.

We won’t go into detail here about how to get this, but it is covered in the advanced section (at the very least, you could always get the set of urls from Reverb).

So, let’s suppose now that we have a file containing some urls that we want to pull:

```
!head -10 < files/data/robot_snow.2012.txt
```

```
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h00v08.005.2012006235251
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h00v09.005.2012006235244
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h00v10.005.2012006234603
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v08.005.2012006234607
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v09.005.2012006234607
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v10.005.2012006235245
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v11.005.2012006234656
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h02v06.005.2012006235323
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h02v08.005.2012006234419
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h02v09.005.2012006234419
```

This is a file containing the filenames of *all* MOD10A1 (daily snow cover) files for a given year, pulled from the `ftp` server. It is possible to do this in Python (see advanced section), but actually must easier and faster with an `ftp` script ‘`files/python/zat-snow`’. You don’t need to run this, as it has already been run for you and the results put in the files:

```
ls -l files/data/robot_snow*.txt
```

```
[0m-rw-rw-r-- 1 plewis plewis 9034752 Oct 22 09:25 [0mfiles/data/robot_snow.2000.txt[0m
-rw-rw-r-- 1 plewis plewis 10820568 Oct 22 09:25 [0mfiles/data/robot_snow.2001.txt[0m
-rw-rw-r-- 1 plewis plewis 16321938 Oct 22 09:25 [0mfiles/data/robot_snow.2002.txt[0m
-rw-rw-r-- 1 plewis plewis 22423068 Oct 22 09:25 [0mfiles/data/robot_snow.2003.txt[0m
-rw-rw-r-- 1 plewis plewis 7633374 Oct 22 09:25 [0mfiles/data/robot_snow.2004.txt[0m
-rw-rw-r-- 1 plewis plewis 18872448 Oct 22 09:25 [0mfiles/data/robot_snow.2005.txt[0m
-rw-rw-r-- 1 plewis plewis 11433078 Oct 22 09:25 [0mfiles/data/robot_snow.2006.txt[0m
-rw-rw-r-- 1 plewis plewis 22663686 Oct 22 09:25 [0mfiles/data/robot_snow.2007.txt[0m
-rw-rw-r-- 1 plewis plewis 22668990 Oct 22 09:25 [0mfiles/data/robot_snow.2008.txt[0m
-rw-rw-r-- 1 plewis plewis 22705317 Oct 22 09:25 [0mfiles/data/robot_snow.2009.txt[0m
-rw-rw-r-- 1 plewis plewis 22712370 Oct 22 09:25 [0mfiles/data/robot_snow.2010.txt[0m
-rw-rw-r-- 1 plewis plewis 17129166 Oct 22 09:25 [0mfiles/data/robot_snow.2011.txt[0m
-rw-rw-r-- 1 plewis plewis 22756824 Oct 22 09:25 [0mfiles/data/robot_snow.2012.txt[0m
```

```
-rw-rw-r-- 1 plewis plewis 18104898 Oct 22 09:25 [0mfiles/data/robot_snow.2013.txt [0m  
[m
```

We can do similar things for http, but that is a little more complicated (again, see advanced notes or 'files/zat > files/data/robot.txt <files/zat> __ for the MODIS LAI product.

```
!head -10 < files/data/robot.txt
```

```
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h00v08.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h00v09.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h00v10.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v07.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v08.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v09.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v10.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v11.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h02v06.00  
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h02v08.00
```

The file files/data/robot_snow.2012.txt contains the names for all tiles and all files.

So if we want just e.g. tile h17v03 and sensor (MOST), we can most easily filter this in unix:

```
tile=h17v03  
year=2012  
type=MOST  
  
file=files/data/robot_snow.${year}_${type}_${tile}.txt  
  
grep $tile < files/data/robot_snow.$year.txt | grep $type > $file  
  
# how many files?  
wc -l < $file  
  
# look at the first 10 ...  
head -10 < $file  
  
366  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.01/MOD10A1.A2012001.h17v03.005.2012003054416  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.02/MOD10A1.A2012002.h17v03.005.2012004061011  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.03/MOD10A1.A2012003.h17v03.005.2012005061244  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.04/MOD10A1.A2012004.h17v03.005.2012006054639  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.05/MOD10A1.A2012005.h17v03.005.2012007052708  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.06/MOD10A1.A2012006.h17v03.005.2012008070328  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.07/MOD10A1.A2012007.h17v03.005.2012011144012  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.08/MOD10A1.A2012008.h17v03.005.2012011154609  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.09/MOD10A1.A2012009.h17v03.005.2012011222125  
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.10/MOD10A1.A2012010.h17v03.005.2012012062821
```

With a more sensible set of urls now (only a few hundred), we can consider how to download them.

We can (of course) do this in Python (see advanced notes), but we can also use unix tools such as curl or wget, which may be easier.

For example, in bash:

```
tile=h17v03  
year=2012  
type=MOST  
  
file=snow_list_${tile}_${year}_${type}.txt  
  
# cd temporarily to the local directory  
pushd files/data  
# -nc : no clobber : dont download if its there already
```

```
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
popd

or in tcsh:

set tile=h17v03
set year=2012
set type=MOST

file=snow_list_${tile}_${year}_${type}.txt

# cd temporarily to the local directory
pushd files/data
# -nc : no clobber : dont download if its there already
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
popd
```

For the moment, let's just pull only some of these by filtering the month as well:

```
tile=h17v03
year=2012
type=MOST
month=01

file=files/data/robot_snow.${year}_${type}_${tile}_${month}.txt

# the dot in the year / month grep need to be escaped
# because dot means something special to grep
grep $tile < files/data/robot_snow.$year.txt | grep $type | grep "${year}\.${month}" > $file

# how many files?
wc -l < $file

# look at the first 10 ...
head -10 < $file

31
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.01/MOD10A1.A2012001.h17v03.005.2012003054416
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.02/MOD10A1.A2012002.h17v03.005.2012004061011
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.03/MOD10A1.A2012003.h17v03.005.2012005061244
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.04/MOD10A1.A2012004.h17v03.005.2012006054639
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.05/MOD10A1.A2012005.h17v03.005.2012007052708
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.06/MOD10A1.A2012006.h17v03.005.2012008070328
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.07/MOD10A1.A2012007.h17v03.005.2012011144012
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.08/MOD10A1.A2012008.h17v03.005.2012011154609
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.09/MOD10A1.A2012009.h17v03.005.2012011222125
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.10/MOD10A1.A2012010.h17v03.005.2012012062821

tile=h17v03
year=2012
type=MOST
month=01

file=robot_snow.${year}_${type}_${tile}_${month}.txt

# cd temporarily to the local directory
pushd files/data
# -nc : no clobber : dont download if its there already
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
# cd back again
```

```
popd  
Process is terminated.
```

4.4 Exercise 4.2 A Different Dataset

We have now dowloaded a different dataset, the [MOD10A product](#), which is the 500 m MODIS daily snow cover product, over the UK.

This is a good opportunity to see if you can apply what was learned above about interpreting QC information and using `gdal` to examine a dataset.

If you examine the [data description page](#), you will see that the data are in HDF EOS format (the same as the LAI product).

4.4.1 E4.2.1 Download

Download the MODIS Terra daily snow product for the UK for the year 2012 for the month of February using the urls in `files/data/robot_snow.2012.txt` and put them in the directory `files/data`.

4.4.2 E4.2.1 Explore

Show all of the subset data layers in this dataset.

4.4.3 E4.3.3 Read a dataset

Suppose we are interested in the dataset `Fractional_Snow_Cover` over the land surface.

Read this dataset for one of the files into a numpy array and show a plot of the dataset.

4.4.4 E4.3.4 Water mask

The [data description page](#) tells us that values of 239 will indicate whether the data is ocean. You can use this information to build the water mask.

Demonstrate how to build a water mask from one of these files, setting the mask `False` for land and `True` for water.

Produce a plot of this.

4.4.5 E3.4.5 Valid pixel mask

As well as having a land/water mask, we should generate a mask for valid pixels. For the snow dataset, values between 0 and 100 (inclusive) represent valid snow cover data values. Other values are not valid for some reason. Set the mask to `False` for valid pixels and `True` for others. Produce a plot of the mask.

4.4.6 E4.3.6 3D dataset

Generate a 3D masked numpy array using the valid pixel mask for masking, of `Fractional_Snow_Cover` for each day of February 2012.

You might like to produce a movie of the result.

Hint: you will need a list of filenames for this. You can either use `glob` as in previous exercises, or you might notice that you have the file `files/data/robot_snow.2012_MOST_h17v03_02.txt` with the urls, from which you should be able to derive the file names. However you get your list of filenames, you should probably apply a `sort()` to the result to make sure they are in the correct order.

4.5 4.3 Vector masking

In this section, we will use a pre-defined function to generate a mask from some vector boundary data.

In this case, we will generate a mask for Ireland, projected into the coordinate system of the MODIS LAI dataset, and use that to generate a new LAI data only for Ireland.

Sometimes, geospatial data is acquired and recorded for particular geometric objects such as polygons or lines. An example is a road layout, where each road is represented as a geometric object (a line, with points given in a geographical projection), with a number of added *features* associated with it, such as the road name, whether it is a toll road, or whether it is dual-carriageway, etc. This data is quite different to a raster, where the entire scene is tessellated into pixels, and each pixel holds a value (or an array of value in the case of multiband rasterfiles).

If you are familiar with databases, vector files are effectively a database, where one of the fields is a geometry object (a line in our previous road example, or a polygon if you consider a cadastral system). We can thus select different records by writing queries on the features. Some of these queries might be spatial (e.g. check whether a point is inside a particular country polygon).

The most common format for vector data is the **ESRI Shapfile**, which is a multifile format (i.e., several files are needed in order to access the data). We'll start by getting hold of a shapefile that contains the countries of the world as polygons, together with information on country name, capital name, population, etc. The file is available here.

Figure 4.1: World

We will download the file with `wget` (or `curl` if you want to), and uncompress it using `unzip` in the shell:

```
# Downloads the data using wget
!wget -nc http://aprsworld.net/gisdata/world/world.zip -O files/data/world.zip
# or if you want to use curl...
#! curl http://aprsworld.net/gisdata/world/world.zip -o world.zip
!pushd files/data;unzip -o -x world.zip;popd

--2013-10-22 15:32:06-- http://aprsworld.net/gisdata/world/world.zip
Resolving aprsworld.net... 72.251.203.219
Connecting to aprsworld.net|72.251.203.219|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3436277 (3.3M) [application/zip]
Saving to: `files/data/world.zip'

100%[=====] 3,436,277 3.38M/s in 1.0s

2013-10-22 15:32:07 (3.38 MB/s) - `files/data/world.zip' saved [3436277/3436277]

/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter4_GDAL/files/data /archive/
Archive: world.zip
  inflating: world.dbf
  inflating: world.shp
  inflating: world.shx
/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter4_GDAL
```

We need to import `ogr`, and then open the file. As with GDAL, we get a handler to the file, (`g` in this case). OGR files can have different layers, although Shapefiles only have one. We need to select the layer using `GetLayer(0)` (selecting the first layer).

```
from osgeo import ogr

g = ogr.Open( "files/data/world.shp" )
layer = g.GetLayer( 0 )
```

In order to see a field (the field NAME) we can loop over the features in the layer, and use the GetField('NAME') method. We'll only do ten features here:

```
n_feat = 0
for feat in layer:

    print feat.GetField('NAME')

    n_feat += 1
    if n_feat == 10:
        break

GUATEMALA
BOLIVIA
PARAGUAY
URUGUAY
SURINAME
FRENCH GUIANA
WESTERN SAHARA
GAMBIA
MOROCCO
MALI
```

If you wanted to see the different layers, we could do this using:

```
layerDefinition = layer.GetLayerDefn()

for i in range(layerDefinition.GetFieldCount()):
    print "Field %d: %s" % ( i+1, layerDefinition.GetFieldDefn(i).GetName() )

Field 1: NAME
Field 2: CAPITAL
Field 3: APPROX
Field 4: AREA
Field 5: SOURCETHM
```

There is much more information on using `ogr` on the associated notebook OGR_Python that you should explore at some point.

One thing we may often wish to do with such vector datasets is produce a mask, e.g. for national boundaries. One of the complexities of this is changing the projection that the vector data come in to that of the raster dataset.

This is too involved to go over in this session, so we will simply present you with a function to achieve this.

This is available as `files/python/raster_mask.py`.

Most of the code below should be familiar from above (we make use of the `getLAI()` function we developed).

```
import sys
sys.path.insert(0,'files/python')
from raster_mask import raster_mask, getLAI

# test this on an LAI file

# the data file name
filename = 'files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf'

# a layer (doesn't matter so much which: use for geometry info)
layer = 'Lai_1km'
```

```

# the full dataset specification
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%(filename,layer)

# make a raster mask
# from the layer IRELAND in world.shp
mask = raster_mask(file_spec,\n
                    target_vector_file = "files/data/world.shp",\n
                    attribute_filter = "NAME = 'IRELAND'")

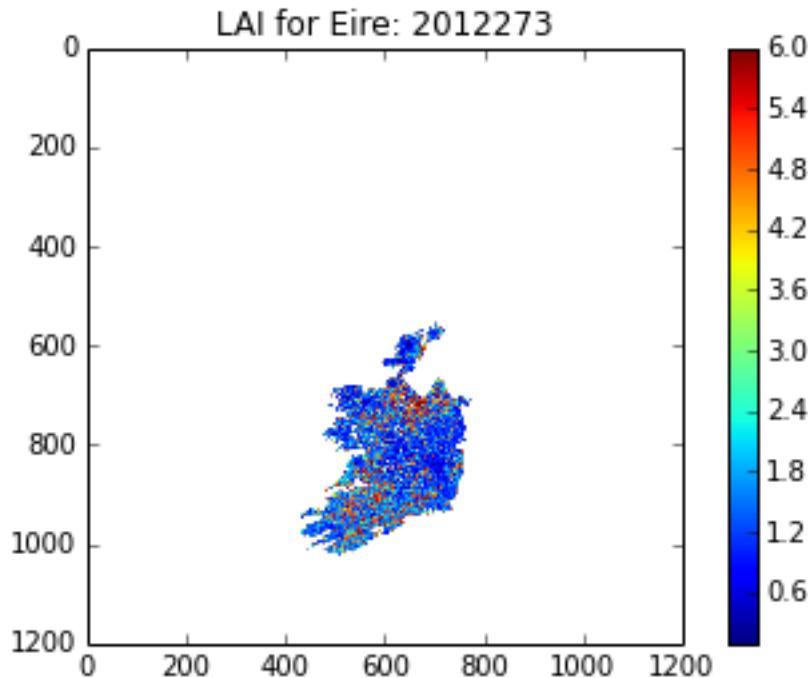
# get the LAI data
data = getLAI(filename)

# reset the data mask
# 'mask' is True for Ireland
# so take the opposite
data['Lai_1km'] = ma.array(data['Lai_1km'],mask=mask)
data['LaiStdDev_1km'] = ma.array(data['Lai_1km'],mask=mask)

plt.title('LAI for Eire: 2012273')
plt.imshow(data['Lai_1km'],vmax=6)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x7153878>

```



4.6 Exercise 4.3

Apply the concepts above to generate a 3D masked numpy data array of LAI and std LAI for Eire for the year 2012.

Plot your results and make a move of LAI.

Plot average LAI for Eire as a function of day of year for 2012.

**CHAPTER
FIVE**

SUMMARY

In this session, we have learned to use some geospatial tools using GDAL in Python. A good set of [working notes on how to use GDAL](#) has been developed that you will find useful for further reading, as well as looking at the advanced section.

We have also very briefly introduced dealing with vector datasets in `ogr`, but this was mainly through the use of a pre-defined function that will take an ESRI shapefile (vector dataset), warp this to the projection of a raster dataset, and produce a mask for a given layer in the vector file.

If there is time in the class, we will develop some exercises to examine the datasets we have generated and/or to explore some different datasets or different locations.

GDAL AND OGR LIBRARIES

In the previous session, we used the GDAL library to open HDF files. GDAL is not limited to a single file format, but can actually cope with many different raster file formats seamlessly. For *vector* data (i.e., data that is stored by features, each being made up of fields containing different types of information, one of them being a *geometry*, such as polygon, line or point), GDAL has a sister library called OGR. The usefulness of these two libraries is that they allow the user to deal with many of the different file formats in a consistent way.

It is important to note that both GDAL and OGR come with a suite of command line tools that let you do many complex tasks on the command line directly. A listing of the GDAL command line tools is available [here](#), but bear in mind that many blogs etc. carry out examples of using GDAL tools in practice. For OGR, most of the library can be accessed using [ogr2ogr](#), but as usual, you might find more useful information on [blogs](#) etc.

6.1 GDAL data type

GDAL provides a very handy way of dealing with raster data in many different formats, not only by making access to the data easy, but also abstracting the nuances and complications of different file formats. In GDAL, a raster file is made up of the actual raster data (i.e., the values of each pixel of LAI that we saw before), and of some *metadata*. Metadata is data that describes something, and in this case it could be the projection, the location of corners and pixel spacing, etc.

6.1.1 The GeoTransform

GDAL stores information about the location of each pixel using the GeoTransform. The GeoTransform contains the coordinates (in some projection) of the upper left (UL) corner of the image (taken to be the **borders of the pixel** in the UL corner, not the center), the pixel spacing and an additional rotation. By knowing these figures, we can calculate the location of each pixel in the image easily. Let's see how this works with an easy example. We have prepared a GeoTIFF file (GeoTIFF is the more ubiquitous file format for EO data) of the MODIS landcover product for the UK. The data has been extracted from the HDF-EOS files that are similar to the LAI product that we have seen before, and converted. The file is 'lc_h17v03.tif <https://raw.github.com/jgomezdans/geogg122/master/ChapterX_GDAL/lc_h17v03.tif>'. We will open the file in Python, and have a look at finding a particular location.

Assume we are interested in locating Kinder Scout, a moorland in the Peak District National Park. Its coordinates are 1.871417W, 53.384726N. In the MODIS integerised sinusoidal projection, the coordinates are (-124114.3, 5936117.4) (you can use the [MODLAND tile calculator website](#) to do this calculation yourself).

```
import gdal # Import GDAL library
g = gdal.Open ( "lc_h17v03.tif" ) # Open the file
if g is None:
    print "Could not open the file!"
geo_transform = g.GetGeoTransform ()
print geo_transform
print g.RasterXSize, g.RasterYSize
```

```
(-1111950.519667, 463.3127165279167, 0.0, 6671703.118, 0.0, -463.3127165279165)
2400 2400
```

In the previous code, we open the file (we just use the filename), and then query the object for its GeoTransform, which we then print out. The 6-element tuple comprises

1. The Upper Left *easting* coordinate (i.e., *horizontal*)
2. The E-W pixel spacing
3. The rotation (0 degrees if image is “North Up”)
4. The Upper left *northing* coordinate (i.e., *vertical*)
5. The rotation (0 degrees)
6. The N-S pixel spacing, negative as we will be counting from the UL corner

We have also seen that the dataset is of size 2400x2400, using RasterXSize and RasterYSize. The goal is to find the pixel number (i, j) , $0 \leq i, j < 2400$ that corresponds to Kinder Scout. To do this, we use the following calculations:

```
pixel_x = (-124114.3 - geo_transform[0])/geo_transform[1] \
    # The difference in distance between the UL corner (geot[0] \
    #and point of interest. Scaled by geot[1] to get pixel number

pixel_y = (5936117.4 - geo_transform[3])/(geo_transform[5]) # Like for pixel_x, \
    #but in vertical direction. Note the different elements of geot \
    #being used

print pixel_x, pixel_y

2132.11549009 1587.66572071
```

So the pixel number is a floating point number, which we might need to round off as an integer. Let's plot the entire raster map (with minimum value 0 to ignore the ocean) using `'plt.imshow'` and plot the location of Kinder Scout using `'plt.plot'`. We will also use `'plt.annotate'` to add a label with an arrow:

```
lc = g.ReadAsArray() # Read raster data
# Now plot the raster data using gist_earth palette
plt.imshow ( lc, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
# Plot location of our area of interest as a red dot ('ro')
plt.plot ( pixel_x, pixel_y, 'ro' )
# Annotate
plt.annotate('Kinder Scout', xy=(pixel_x, pixel_y), \
    xycoords='data', xytext=(-150, -60), \
    textcoords='offset points', size=12, \
    bbox=dict(boxstyle="round4,pad=.5", fc="0.8"), \
    arrowprops=dict(arrowstyle="->", \
    connectionstyle="angle,angleA=0,angleB=-90,rad=10", \
    color='w'), )
# Remove vertical and horizontal ticks
plt.xticks([])
plt.yticks([])

([], <a list of 0 Text yticklabel objects>)
```

Try it out in some other places!

Find the longitude and latitude of some places of interest in the British isles (West of Greenwich!) and using the MODLAND MODIS tile calculator and the geotransform, repeat the above experiment. Note that the MODIS

calculator calculates both the projected coordinates in the MODIS sinusoidal projection, as well as the pixel number, so it is a helpful way to check whether you got the right result.

Park name

Longitude [deg]

Latitude [deg]

Dartmoor national park

-3.904

50.58

New forest national park

-1.595

50.86

Exmoor national park

-3.651

51.14

Pembrokeshire coast national park

-4.694

51.64

Brecon beacons national park

-3.432

51.88</td>

Pembrokeshire coast national park

-4.79

51.99

Norfolk and suffolk broads

1.569

52.62

Snowdonia national park

-3.898 </td><td>52.9</td>

Peak district national park

-1.802

53.3

Yorkshire dales national park

-2.157

54.23

North yorkshire moors national park

-0.8855

54.37

Lake district national park

-3.084

54.47
Galloway forest park
-4.171
54.87
Northumberland national park
-2.228
55.28
Loch lomond and the trossachs national park
-4.593
56.24
Tay forest park
-4.025
56.59</td>
Cairngorms national park
-3.545
57.08

6.2 The projection

Projections in GDAL objects are stored can be accessed by querying the dataset using the `GetProjection()` method. If we do that on the currently opened dataset (stored in variable `g`), we get:

```
print g.GetProjection()
```

```
PROJCS["unnamed",GEOGCS["Unknown datum based upon the custom spheroid",DATUM["Not_specified_based_
```

The above is the description of the projection (in this case, MODIS sinusoidal) in WKT (well-known text) format. There are a number of different ways of specifying projections, the most important being

- WKT
- Proj4
- EPSG codes

The site spatialreference.org allows you to search a large collection of projections, and get the representation that you want to use.

6.3 Saving files

So far, we have read data from files, but lets see how we can save raster data **to** a new file. We will use the previous landcover map as an example. We will write a method to save the data in a format provided by the user. The procedure is fairly straightforward: we get a handler to a driver (e.g. a GeoTIFF or Erdas Imagine format), we create the output file (giving a filename, number of rows, columns, bands, the data type), and then add the relevant metadata (projection, geotransform, ...). We then select a band from the output and copy the array that we want to write to that band.

```

g = gdal.Open ( "lc_h17v03.tif" ) # Open original file
# Get the x, y and number of bands from the original file
x_size, y_size, n_bands = g.RasterXSize, g.RasterYSize, g.RasterCount
data = g.ReadAsArray ()
driver = gdal.GetDriverByName ( "HFA" ) # Get a handler to a driver
# Maybe try "GeoTIFF" here
# Next line creates the output dataset with
# 1. The filename ("test_lc_h17v03.img")
# 2. The raster size (x_size, y_size)
# 3. The number of bands
# 4. The data type (in this case, Byte.
#     Other typical values might be gdal.GDT_Int16
#     or gdal.GDT_Float32)

dataset_out = driver.Create ( "test_lc_h17v03.img", x_size, y_size, n_bands, \
                             gdal.GDT_Byte )
# Set the output geotransform by reading the input one
dataset_out.SetGeoTransform ( g.GetGeoTransform() )
# Set the output projection by reading the input one
dataset_out.SetProjection ( g.GetProjectionRef() )
# Now, get band # 1, and write our data array.
# Note that the data array needs to have the same type
# as the one specified for dataset_out
dataset_out.GetRasterBand ( 1 ).WriteArray ( data )
# This bit forces GDAL to close the file and write to it
dataset_out = None

```

The output file should hopefully exist in this directory. Let's use `gdalinfo` <<http://www.gdal.org/gdalinfo.html>> to find out about it

```
!gdalinfo test_lc_h17v03.img
```

```

Driver: HFA/Erdas Imagine Images (.img)
Files: test_lc_h17v03.img
Size is 2400, 2400
Coordinate System is:
PROJCS["Sinusoidal",
    GEOGCS["GCS_Unknown datum based upon the custom spheroid",
        DATUM["Not_specified_based_on_custom_spheroid",
            SPHEROID["Custom_spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,6671703.117999999783933)
Pixel Size = (463.312716527916677,-463.312716527916507)
Corner Coordinates:
Upper Left  (-1111950.520, 6671703.118) ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-1111950.520, 5559752.598) ( 15d33'26.06"W, 50d 0' 0.00"N)
Upper Right  (      0.000, 6671703.118) ( 0d 0' 0.01"E, 60d 0' 0.00"N)
Lower Right  (      0.000, 5559752.598) ( 0d 0' 0.01"E, 50d 0' 0.00"N)
Center       (-555975.260, 6115727.858) ( 8d43' 2.04"W, 55d 0' 0.00"N)
Band 1 Block=64x64 Type=Byte, ColorInterp=Undefined
    Description = Layer_1
    Metadata:
        LAYER_TYPE=athematic

```

So the previous code works. Since this is something we typically do (read some data from one or more files, manipulate it and save the result in output files), it makes a lot of sense to try to put this code in a method that is more or less generic, that we can test and then re-use. Here's a first attempt at it:

```
def save_raster ( output_name, raster_data, dataset, driver="GTiff" ):
    """
    A function to save a 1-band raster using GDAL to the file indicated
    by ``output_name``. It requires a GDAL-accessible dataset to collect
    the projection and geotransform.

    Parameters
    -----
    output_name: str
        The output filename, with full path and extension if required
    raster_data: array
        The array that we want to save
    dataset: str
        Filename of a GDAL-friendly dataset that we want to use to
        read geotransform & projection information
    driver: str
        A GDAL driver string, like GTiff or HFA.
    """

    # Open the reference dataset
    g = gdal.Open ( dataset )
    # Get the Geotransform vector
    geo_transform = g.GetGeoTransform ()
    x_size = g.RasterXSize # Raster xsize
    y_size = g.RasterYSize # Raster ysize
    srs = g.GetProjectionRef () # Projection
    # Need a driver object. By default, we use GeoTIFF
    driver = gdal.GetDriverByName ( driver )
    dataset_out = driver.Create ( output_name, x_size, y_size, 1, \
                                  gdal.GDT_Float32 )
    dataset_out.SetGeoTransform ( geo_transform )
    dataset_out.SetProjection ( srs )
    dataset_out.GetRasterBand ( 1 ).WriteArray ( \
        raster_data.astype(np.float32) )
    dataset_out = None
```

Now try modifying that method so that you can

1. Select the output data type different to Float32
2. Provide a given projection and geotransform (e.g. if you don't have a GDAL filename)

6.4 Reprojecting things

Previously, we have used the [MODLAND](#) grid converter site to go from latitude/longitude pairs to MODIS projection. However, in practice, we might want to use a range of different projections, and convert many points at the same time, so how do we go about that?

In GDAL/OGR, most projection-related tools are in the `osr` package, which needs to be imported like e.g. `gdal` itself. The main tools are the `osr.SpatialReference` object, which defines a projection object (with no projection to start with), and the `osr.CoordinateTransformation` object.

Once you instantiate `osr.SpatialReference`, it holds no projection information. You need to use methods to set it up, using EPSG codes, Proj4 strings, or whatever. These methods typically start by `ImportFrom` (e.g. `ImportFromEPSG`, `ImportFromProj4`, ...).

The `CoordinateTransformation` requires a source and destination spatial references that have been configured. Once this is done, it exposes the method `TransformPoint` to convert coordinates from the source to the destination projection.

Let's see how this works by converting some latitude/longitude pairs to the Ordnance Survey's [National Grid](#) projection. The projection is also available in [spatialreference.org](#), where we can gleam its EPSG code (27700).

The EPSG code for longitude latitude is 4326. Let's see this in practice:

```
from osgeo import osr

# Define the source projection, WGS84 lat/lon.
wgs84 = osr.SpatialReference() # Define a SpatialReference object
wgs84.ImportFromEPSG( 4326 ) # And set it to WGS84 using the EPSG code

# Now for the target projection, Ordnance Survey's British National Grid
osng = osr.SpatialReference() # define the SpatialReference object
# In this case, we get the projection from a Proj4 string
osng.ImportFromEPSG( 27700 )
# or, if using the proj4 representation
osng.ImportFromProj4 ( "+proj=tmerc +lat_0=49 +lon_0=-2 " + \
                      "+k=0.9996012717 +x_0=400000 +y_0=-100000 " + \
                      "+ellps=airy +datum=OSGB36 +units=m +no_defs" )

# Now, we define a coordinate transformation object, *from* wgs84 *to* OSNG
tx = osr.CoordinateTransformation( wgs84, osng )
# We loop over the lines of park_data,
#       using the split method to split by newline characters
park_name, lon, lat = "Snowdonia national park", -3.898,      52.9

# Actually do the transformation using the TransformPoint method
osng_x, osng_y, osng_z = tx.TransformPoint ( lon, lat )
# Print out
print park_name, lon, lat, osng_x, osng_y

Snowdonia national park -3.898 52.9 272430.180112 335304.936823
```

You can test the result is reasonable by feeding the data for `osng_x` and `osng_y` in the OS own coordinate conversion website and making sure that the calculated longitude latitude pair is the same as the one you started with.

6.5 Reprojecting whole rasters

6.5.1 Using command line tools

The easiest way to reproject a raster file is to use GDAL's '`gdalwarp <http://www.gdal.org/gdalwarp.html>`' tool. As an example, let's say we want to reproject the landcover file from earlier on into latitude/longitude (WGS84):

```
!gdalwarp -of GTiff -t_srs "EPSG:4326" -ts 2400 2400 test_lc_h17v03.img lc_h17v03_wgs84.tif

Output dataset lc_h17v03_wgs84.tif exists,
but some commandline options were provided indicating a new dataset
should be created. Please delete existing dataset and run again.
```

We see here that the command takes a number of arguments:

1. `-of GTiff` is the output format (in this case GeoTIFF)
2. `-t_srs "EPSG:4326"` is the **to** projection (the **from** projection is already specified in the source dataset), in this case, lat/long WGS84, known by its **EPSG code**
3. `-ts 2400 2400` instructs `gdalwarp` to use an output of size 2400*2400.
4. `test_lc_h17v03.img` is the **input dataset**
5. `lc_h17v03_wgs84.tif` is the **output dataset**

Note that `gdalwarp` will reproject the data, and decide on the pixel size based on some considerations. This can result in the size of the raster changing. If you wanted to still keep the same raster size, we use the `-ts 2400 2400` option, or select an appropriate pixel size using `-tr xres yres` (note it has to be in the target projection, so degrees in this case). We can use `gdalinfo` to see what we've done.

```
!gdalinfo test_lc_h17v03.img
```

```
Driver: HFA/Erdas Imagine Images (.img)
Files: test_lc_h17v03.img
Size is 2400, 2400
Coordinate System is:
PROJCS["Sinusoidal",
    GEOGCS["GCS_Unknown datum based upon the custom spheroid",
        DATUM["Not_specified_based_on_custom_spheroid",
            SPHEROID["Custom_spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,6671703.117999999783933)
Pixel Size = (463.312716527916677,-463.312716527916507)
Corner Coordinates:
Upper Left  (-1111950.520, 6671703.118)  ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-1111950.520, 5559752.598)  ( 15d33'26.06"W, 50d 0' 0.00"N)
Upper Right  (      0.000, 6671703.118)  ( 0d 0' 0.01"E, 60d 0' 0.00"N)
Lower Right  (      0.000, 5559752.598)  ( 0d 0' 0.01"E, 50d 0' 0.00"N)
Center       ( -555975.260, 6115727.858)  ( 8d43' 2.04"W, 55d 0' 0.00"N)
Band 1 Block=64x64 Type=Byte, ColorInterp=Undefined
    Description = Layer_1
    Metadata:
        LAYER_TYPE=athematic
```

```
!gdalinfo lc_h17v03_wgs84.tif
```

```
Driver: GTiff/GeoTIFF
Files: lc_h17v03_wgs84.tif
Size is 2400, 2400
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (-19.99999994952233,59.99999994611805)
Pixel Size = (0.008333919248404,-0.004166959624202)
Metadata:
    AREA_OR_POINT=Area
Image Structure Metadata:
    INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (-20.0000000, 60.0000000)  ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-20.0000000, 49.9992969)  ( 20d 0' 0.00"W, 49d59'57.47"N)
Upper Right  ( 0.0014062, 60.0000000)  ( 0d 0' 5.06"E, 60d 0' 0.00"N)
Lower Right  ( 0.0014062, 49.9992969)  ( 0d 0' 5.06"E, 49d59'57.47"N)
Center       ( -9.9992969, 54.9996484)  ( 9d59'57.47"W, 54d59'58.73"N)
Band 1 Block=2400x3 Type=Byte, ColorInterp=Gray
    Description = Layer_1
```

```
Metadata:
  LAYER_TYPE=athematic
```

Let's see how different these two datasets are:

```
g = gdal.Open( "lc_h17v03_wgs84.tif" )
wgs84 = g.ReadAsArray()
g = gdal.Open("test_lc_h17v03.img")
modis = g.ReadAsArray()
plt.subplot( 1, 2, 1 )
plt.imshow( modis, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.subplot( 1, 2, 2 )
plt.imshow( wgs84, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )

<matplotlib.image.AxesImage at 0xe566950>
```

6.5.2 Reprojecting using the Python bindings

The previous section demonstrated how you can reproject raster files using command line tools. Sometimes, you might want to do this from inside a Python script. Ideally, you would have a python method that would perform the projection for you. GDAL allows this by defining in-memory raster files. These are normal GDAL datasets, but that don't exist on the filesystem, only in the computer's memory. They are a convenient "scratchpad" for quick intermediate calculations. GDAL also makes available a function, `gdal.ReprojectImage` that exposes most of the abilities of `gdalwarp`. We shall combine these two tricks to carry out the reprojection. As an example, we shall look at the case where the landcover data for the British Isles mentioned in the previous section needs to be reprojected to the Ordnance Survey National Grid, an appropriate projection for the UK.

The main complication comes from the need of `gdal.ReprojectImage` to operate on GDAL datasets. In the previous section, we saved the data to a GeoTIFF file, so this gives us a starting dataset. We still need to create the output dataset. This means that we need to define the geotransform and size of the output dataset before the projection is made. This entails gathering information on the extent of the original dataset, projecting it to the destination projection, and calculating the number of pixels and geotransform parameters from there. This is a (heavily commented) function that performs just that task:

```
def reproject_dataset( dataset, \
                      pixel_spacing=463., epsg_from=4326, epsg_to=27700 ):
    """
    A sample function to reproject and resample a GDAL dataset from within
    Python. The idea here is to reproject from one system to another, as well
    as to change the pixel size. The procedure is slightly long-winded, but
    goes like this:

    1. Set up the two Spatial Reference systems.
    2. Open the original dataset, and get the geotransform
    3. Calculate bounds of new geotransform by projecting the UL corners
    4. Calculate the number of pixels with the new projection & spacing
    5. Create an in-memory raster dataset
    6. Perform the projection
    """

    # Define the UK OSNG, see <http://spatialreference.org/ref/epsg/27700/>
    osng = osr.SpatialReference()
    osng.ImportFromEPSG(epsg_to)
    wgs84 = osr.SpatialReference()
    wgs84.ImportFromEPSG(epsg_from)
    tx = osr.CoordinateTransformation(wgs84, osng)
    # Up to here, all the projection have been defined, as well as a
    # transformation from the from to the to :
    # We now open the dataset
    g = gdal.Open(dataset)

    # Get the Geotransform vector
```

```
geo_t = g.GetGeoTransform ()
x_size = g.RasterXSize # Raster xsize
y_size = g.RasterYSize # Raster ysize

# Work out the boundaries of the new dataset in the target projection
(ulx, uly, ulz) = tx.TransformPoint( geo_t[0], geo_t[3])
(lrx, lry, lrz) = tx.TransformPoint( geo_t[0] + geo_t[1]*x_size, \
                                     geo_t[3] + geo_t[5]*y_size )
print ulx, uly, ulz
print lrx, lry, lrz
# See how using 27700 and WGS84 introduces a z-value!
# Now, we create an in-memory raster
mem_drv = gdal.GetDriverByName( 'MEM' )
# The size of the raster is given the new projection and pixel spacing
# Using the values we calculated above. Also, setting it to store one band
# and to use Float32 data type.
dest = mem_drv.Create('', int((lrx - ulx)/pixel_spacing), \
                      int((uly - lry)/pixel_spacing), 1, gdal.GDT_Float32)
# Calculate the new geotransform
new_geo = ( ulx, pixel_spacing, geo_t[2], \
            uly, geo_t[4], -pixel_spacing )
# Set the geotransform
dest.SetGeoTransform( new_geo )
dest.SetProjection( osng.ExportToWkt() )
# Perform the projection/resampling
res = gdal.ReprojectImage( g, dest, \
                           wgs84.ExportToWkt(), osng.ExportToWkt(), \
                           gdal.GRA_Bilinear )
return dest
```

The function returns a GDAL in-memory file object, where you can `ReadAsArray` etc. As it stands, `reproject_dataset` does not write to disk. However, we can save the in-memory raster to any format supported by GDAL very conveniently by making a copy of the dataset. This takes a few lines of code:

```
# Do in memory reprojection
reprojected_dataset = reproject_dataset ( "lc_h17v03_wgs84.tif" )
# Output driver, as before
driver = gdal.GetDriverByName ( "GTiff" )
# Create a copy of the in memory dataset `reprojected_dataset`, and save it
dst_ds = driver.CreateCopy( "test_lc_h17v03_OSNG.tif", reprojected_dataset, 0 )
dst_ds = None # Flush the dataset to disk

-595472.202548 1261034.77555 -55.2326775854
543532.18509 12933.1712342 -43.993001678
```

Let's see how the different projections look like by plotting them side by side

```
plt.subplot ( 1, 3, 1 )
plt.title ( "MODIS sinusoidal" )
plt.imshow ( modis, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.subplot ( 1, 3, 2 )
plt.imshow ( wgs84, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
g = gdal.Open("test_lc_h17v03_OSNG.tif" )
osng = g.ReadAsArray()
plt.title ( "WGS84, Lat/Long" )
plt.subplot ( 1, 3, 3 )
plt.imshow ( osng, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.title("OSNG")

<matplotlib.text.Text at 0x2b9eac0b4ed0>
```

WORKING WITH VECTOR DATA: OGR

7.1 Vector data

Sometimes, geospatial data is acquired and recorded for particular geometric objects such as polygons or lines. An example is a road layout, where each road is represented as a geometric object (a line, with points given in a geographical projection), with a number of added *features* associated with it, such as the road name, whether it is a toll road, or whether it is dual-carriageway, etc. This data is quite different to a raster, where the entire scene is tessellated into pixels, and each pixel holds a value (or an array of value in the case of multiband rasterfiles).

If you are familiar with databases, vector files are effectively a database, where one of the fields is a geometry object (a line in our previous road example, or a polygon if you consider a cadastral system). We can thus select different records by writing queries on the features. Some of these queries might be spatial (e.g. check whether a point is inside a particular country polygon).

The most common format for vector data is the **ESRI Shapfile**, which is a multifile format (i.e., several files are needed in order to access the data). We'll start by getting hold of a shapefile that contains the countries of the world as polygons, together with information on country name, capital name, population, etc. The file is available here.

Figure 7.1: World

We will download the file with `wget` (or `curl` if you want to), and uncompress it using `unzip` in the shell:

```
# Downloads the data using wget
!wget http://aprsworld.net/gisdata/world/world.zip
# or if you want to use curl...
#! curl http://aprsworld.net/gisdata/world/world.zip -o world.zip
!unzip -o -x world.zip

--2013-10-22 15:47:43-- http://aprsworld.net/gisdata/world/world.zip
Resolving aprsworld.net... 72.251.203.219
Connecting to aprsworld.net|72.251.203.219|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3436277 (3.3M) [application/zip]
Saving to: `world.zip.1'

100%[=====] 3,436,277 3.28M/s in 1.0s

2013-10-22 15:47:45 (3.28 MB/s) - `world.zip.1' saved [3436277/3436277]

Archive: world.zip
  inflating: world.dbf
  inflating: world.shp
  inflating: world.shx
```

We need to import `ogr`, and then open the file. As with GDAL, we get a handler to the file, (`g` in this case). OGR files can have different layers, although Shapefiles only have one. We need to select the layer using `GetLayer(0)` (selecting the first layer).

```
from osgeo import ogr

g = ogr.Open( "world.shp" )
layer = g.GetLayer( 0 )
```

In order to see a field (the field NAME) we can loop over the features in the layer, and use the GetField('NAME') method. We'll only do ten features here:

```
n_feat = 0
for feat in layer:

    print feat.GetField('NAME')
```

```
    n_feat += 1
    if n_feat == 10:
        break
```

```
GUATEMALA
BOLIVIA
PARAGUAY
URUGUAY
SURINAME
FRENCH GUIANA
WESTERN SAHARA
GAMBIA
MOROCCO
MALI
```

If you wanted to see the different layers, we could do this using:

```
layerDefinition = layer.GetLayerDefn()
```

```
for i in range(layerDefinition.GetFieldCount()):
    print "Field %d: %s" % ( i+1, layerDefinition.GetFieldDefn(i).GetName() )

Field 1: NAME
Field 2: CAPITAL
Field 3: APPROX
Field 4: AREA
Field 5: SOURCETHM
```

Each feature, in addition to the fields shown above, will have a Geometry field. We get a handle to this using the GetGeometryRef() method. Geometries have many methods, such as ExportToKML() to export to KML (Google Maps/Earth format):

```
the_geometry = feat.GetGeometryRef()
the_geometry.ExportToKML()
```

```
'<Polygon><outerBoundaryIs><LinearRing><coordinates>-12.0443,14.669667 -11.87845,14.8252 -11.7645
```

Many of the methods that don't start with __ are interesting. Let's see what these are. typically, the interesting methods start with an upper case letter, so we'll only show those:

```
for m in dir( the_geometry ):
    if m[0].isupper():
        print m
```

```
AddGeometry
AddGeometryDirectly
AddPoint
AddPoint_2D
Area
AssignSpatialReference
```

Boundary
Buffer
Centroid
Clone
CloseRings
Contains
ConvexHull
Crosses
Destroy
Difference
Disjoint
Distance
Empty
Equal
Equals
ExportToGML
ExportToJson
ExportToKML
ExportToWkb
ExportToWkt
FlattenTo2D
GetArea
GetBoundary
GetCoordinateDimension
GetDimension
GetEnvelope
GetEnvelope3D
GetGeometryCount
GetGeometryName
GetGeometryRef
GetGeometryType
GetPoint
GetPointCount
GetPoint_2D
GetPoints
GetSpatialReference
GetX
GetY
GetZ
Intersect
Intersection
Intersects
IsEmpty
IsRing
IsSimple
IsValid
Length
Overlaps
PointOnSurface
Segmentize
SetCoordinateDimension
SetPoint
SetPoint_2D
Simplify
SimplifyPreserveTopology
SymDifference
SymmetricDifference
Touches
Transform
TransformTo
Union
UnionCascaded
Within

WkbSize

You'll notice that many of these mechanisms e.g. Overlaps or Touches are effectively geoprocessing operations (they operate on geometries and return True if one geometry overlaps or touches, respectively, the other). Other operations, such as Buffer return a buffered version of the same geometry. This allows you to actually do fairly complicated geoprocessing operations with OGR. However, if you want to do geoprocessing in earnest, you should really be using [Shapely](#).

A particularly useful webpage for this section is [available in the OGR cookbook](#). Have a look through that if you want more in depth information.

7.2 Selecting attributes and/or data extents

OGR provides an easy way to select attributes on a given layer. This is done using a SQL-like syntax (you can read more on [OGR's SQL subset here](#)). The main point is that the *attribute filter* is applied to a complete layer. For example, let's say that we want to select only countries with a population (field APPROX) larger than 90 000 000 inhabitants:

```
g = ogr.Open ( "world.shp" )
lyr = g.GetLayer( 0 )
lyr.SetAttributeFilter ( "APPROX > 90000000" )
for feat in lyr:
    print feat.GetFieldAsString ( "NAME" ) + " has %d inhabitants" % \
        feat.GetFieldAsInteger("APPROX")

PAKISTAN has 123490000 inhabitants
JAPAN has 124710000 inhabitants
RUSSIAN FEDERATION has 150500000 inhabitants
INDIA has 873850000 inhabitants
BANGLADESH has 120850000 inhabitants
BRAZIL has 159630000 inhabitants
NIGERIA has 91700000 inhabitants
CHINA has 1179030000 inhabitants
INDONESIA has 186180000 inhabitants
JOHNSTON ATOLL has 256420000 inhabitants
KINGMAN REEF - PALMYRA ATOLL has 256420000 inhabitants
UNITED STATES has 256420000 inhabitants
```

So we get a list of populous countries (note that Johnston Atoll and Palmyra are part of the US, and report the sample population as the US!)

An additional way to filter the data is by geographical extent. Let's say we wanted a list of all the countries in (broadly speaking) Europe, *i.e.* a geographical extent in longitude from 14W to 37E, and in latitude from 72N to 38N. We can use SetSpatialFilterRect to do this:

```
g = ogr.Open ( "world.shp" )
lyr = g.GetLayer( 0 )
lyr.SetSpatialFilterRect ( -14, 37, 38, 72)
for feat in lyr:
    print feat.GetFieldAsString ( "NAME" ) + " ---- " + feat.GetFieldAsString ( "CAPITAL" )

ALGERIA ---- ALGIERS
BELGIUM ---- BRUSSELS
LUXEMBOURG ---- LUXEMBOURG
SAN MARINO ---- SAN MARINO
AUSTRIA ---- VIENNA
CZECH REPUBLIC ---- PRAGUE
SLOVENIA ---- LJUBLJANA
HUNGARY ---- BUDAPEST
SLOVAKIA ---- BRATISLAVA
YUGOSLAVIA ---- BELGRADE [BEOGRADE]
```

BOSNIA AND HERZEGOVINA ---- SARAJEVO
 ALBANIA ---- TIRANE
 MACEDONIA, THE FORMER YUGOSLAV REPUBLIC ---- SKOPJE
 LITHUANIA ---- VILNIUS
 LATVIA ---- RIGA
 BULGARIA ---- SOFIA
 BELARUS ---- MINSK
 MOLDOVA, REPUBLIC OF ---- KISHINEV
 IRELAND ---- DUBLIN
 ICELAND ---- REYKJAVIK
 SPAIN ---- MADRID
 SWEDEN ---- STOCKHOLM
 FINLAND ---- HELSINKI
 TURKEY ---- ANKARA
 RUSSIAN FEDERATION ---- MOSCOW
 GREECE ---- ATHENS
 PORTUGAL ---- LISBON
 POLAND ---- WARSAW
 NORWAY ---- OSLO
 GERMANY ---- BERLIN
 ESTONIA ---- TALLINN
 TUNISIA ---- TUNIS
 CROATIA ---- ZAGREB
 ROMANIA ---- BUCURESTI
 UKRAINE ---- KIEV
 NETHERLANDS ---- AMSTERDAM
 JERSEY ---- SAINT HELIER
 GUERNSEY ---- SAINT PETER PORT
 FAROE ISLANDS ---- TORSHAVN
 DENMARK ---- COPENHAGEN
 MONACO ---- MONACO
 ANDORRA ---- ANDORRA LA VELLA
 LIECHTENSTEIN ---- VADUZ
 SWITZERLAND ---- BERN
 ISLE OF MAN ---- DOUGLAS
 UNITED KINGDOM ---- LONDON
 FRANCE ---- PARIS
 VATICAN CITY (HOLY SEE) ---- VATICAN CITY
 ITALY ---- ROME

7.3 Saving a vector file

Saving a vector file using OGR requires a number of steps:

1. Definition of the format
2. Definition of the layer projection and geometry type (e.g. lines, polygons...)
3. Definition of the data type of the different fields
4. Creation of a feature, population of the different fields, and setting a geometry
5. Addition of the feature to the layer
6. Destruction of the feature

This appears quite involved, but let's see how this works. Note that when you generate a new vector file, OGR will fail if the file already exists. You might want to use `os.remove()` to get rid of the file if it exists.

Let's see how this is done with an example which is a snippet that creates a GeoJSON file with the location of the different national parks. GeoJSON is a nice geographic format, and [github allows you to display it easily as a map](#).

```
# National park information, separated by TABs
import os
from osgeo import ogr,osr

parks = """Dartmoor national park\t-3.904\t50.58
New forest national park\t-1.595\t50.86
Exmoor national park\t-3.651\t51.14
Pembrokeshire coast national park\t-4.694\t51.64
Brecon beacons national park\t-3.432\t51.88
Pembrokeshire coast national park\t-4.79\t51.99
Norfolk and suffolk broads\t1.569\t52.62
Snowdonia national park\t-3.898\t52.9
Peak district national park\t-1.802\t53.3
Yorkshire dales national park\t-2.157\t54.23
North yorkshire moors national park\t-0.8855\t54.37
Lake district national park\t-3.084\t54.47
Galloway forest park\t-4.171\t54.87
Northumberland national park\t-2.228\t55.28
Loch lomond and the trossachs national park\t-4.593\t56.24
Tay forest park\t-4.025\t56.59
Cairngorms national park\t-3.545\t57.08"""

# See if the file exists from a previous run of this snippet
if os.path.exists ( "parks.json" ):
    # It does exist, so remove it
    os.remove ( "parks.json" )

# We need the output projection to be set to Lat/Long
latlong = osr.SpatialReference()
latlong.ImportFromEPSG( 4326 )

# Invoke the GeoJSON driver
drv = ogr.GetDriverByName( 'GeoJSON' )
# This is the output filename
dst_ds = drv.CreateDataSource( 'parks.json' )
# This is a single layer dataset. The layer needs to be of points
# and needs to have the WGS84 projection, which we defined above
dst_layer = dst_ds.CreateLayer('', srs =latlong , \
                                geom_type=ogr.wkbPoint )

# We just need a field with the Park's name, and its type is a String
field_defn=ogr.FieldDefn( 'name', ogr.OFTString )
dst_layer.CreateField( field_defn )

# Algorithm is as follows:
# 1. Loop over lines
# 2. Split line into park name, longitude, latitude
# 3. Create WKT of the point
# 4. Set the attribute name to name of park
# 5. Clean up

for park_id, line in enumerate( parks.split( "\n" ) ):
    # Get the relevant information
    park_name, lon, lat = line.split("\t")
    # Create a geopraphical representation of the current park
    wkt = "POINT ( %f %f )" % ( float(lon), float(lat) )
    # Create a feature, using the attributes/fields that are
    # required for this layer
    feat = ogr.Feature(feature_def=dst_layer.GetLayerDefn())
    # Feed the WKT into a geometry
    p = ogr.CreateGeometryFromWkt( wkt )
    feat.SetField("name", park_name)
    dst_layer.CreateFeature(feat)
```

```

# Feed the geometry into a WKT
feat.SetGeometryDirectly( p )
# Set the name field to its value
feat.SetField( "name", park_name )
# Attach the feature to the layer
dst_layer.CreateFeature( feat )
# Clean up
feat.Destroy()

# Close file
dst_ds = None

```

You can see the result of this on [github](#).

Additionally, note that if we had defined a coordinate transformation as in the raster session, we could apply this transformation to an OGR geometry entity (in the snippet above, `p` would be such), and it would be reprojected.

Exercise Modify the above snippet to output a GeoJSON file for the Peak District National Park, whose UTM30N (EPSG code: [32630](#)) co-ordinates are 577659, 5911841.

7.4 Rasterising

A very frequent problem one finds is how to mask out an area in a raster file that is defined as polygon in a shapefile. For example, if you have a raster of the world's population density, and you want to extract all the pixels that belong to one particular country, how do you go about that? One way around this is to *rasterise* the polygon(s), which translates into “burning” pixels that fall within the polygon with a number, resulting in a mask.

The way to do this is to use GDAL’s `RasterizeLayer` method. The method takes a handle to a GDAL dataset (one that you create yourself, with the right projection and geotransform, as you’ve seen above), and a OGR layer. The syntax for `RasterizeLayer` is

```
err = gdal.RasterizeLayer ( raster_ds, [raster_band_no], ogr_layer, burn_values=[burn_val] )
```

where `raster_ds` is the GDAL raster datasource (note that it needs to be georeferenced, *i.e.* it requires projection and geotransform), `raster_band_no` is the band of the GDAL dataset where we want to burn pixels, `ogr_layer` is the vector layer object, and `burn_val` is the value that we want to burn.

Let’s use `gdal.RasterizeLayer` in conjunction with all that we have covered above. Say we want to create a mask that only selects the UK or Ireland in `world.shp`, and we want to apply this mask to the MODIS landcover product that we used in the GDAL session (`h17v03.tif`), file `lc_h17v03.tif`. We find that in this case, `world.shp` is in longitude/latitude, and the MODIS data is in the MODIS projection, so we will reproject the vector data to match the MODIS data (so the latter is not interpolated and artifacts introduced). To make this efficient and avoid saving to disk, we shall use *in-memory vector and rasters*, and we will output a numpy array as our mask. Note then the steps:

1. Create the projection conversion object (as for GDAL before)
2. Create an in memory **raster** dataset to store the mask, using `lc_h17v03.tif` as a reference for geotransforms, array size and projection.
3. Create an in memory **vector** dataset to hold the features that will be reprojected
4. Open `world.shp` and apply an `AttributeFilter` to select a country
5. Select a geometry from `world.shp`, project it and store it in the destination in memory vector layer
6. Once this is done, use `gdal.RasterizeLayer` with both in-memory raster and vector datasets
7. Read the in memory raster into an array

This is a particularly good exercise that will stress all that we have learned so far.

```
from osgeo import ogr,osr
import gdal

reference_filename = "lc_h17v03.tif"
target_vector_file = "world.shp"
attribute_filter = "NAME = 'IRELAND'"
burn_value = 1

# First, open the file that we'll be taking as a reference
# We will need to gleam the size in pixels, as well as projection
# and geotransform.

g = gdal.Open( reference_filename )

# We now create an in-memory raster, with the appropriate dimensions
drv = gdal.GetDriverByName('MEM')
target_ds = drv.Create('', g.RasterXSize, g.RasterYSize, 1, gdal.GDT_Byte)
target_ds.SetGeoTransform( g.GetGeoTransform() )

# We set up a transform object as we saw in the previous notebook.
# This goes from WGS84 to the projection in the reference datasets

wgs84 = osr.SpatialReference() # Define a SpatialReference object
wgs84.ImportFromEPSG( 4326 ) # And set it to WGS84 using the EPSG code

# Now for the target projection, Ordnance Survey's British National Grid
to_proj = osr.SpatialReference() # define the SpatialReference object
# In this case, we get the projection from a Proj4 string

# or, if using the proj4 representation
to_proj.ImportFromWkt( g.GetProjectionRef() )
target_ds.SetProjection ( to_proj.ExportToWkt() )
# Now, we define a coordinate transformation object, *from* wgs84 *to* OSNG
tx = osr.CoordinateTransformation( wgs84, to_proj )

# We define an output in-memory OGR dataset
# You could also do select a driver for an eg "ESRI Shapefile" here
# and give it a sexier name than out!

drv = ogr.GetDriverByName( 'Memory' )
dst_ds = drv.CreateDataSource( 'out' )
# This is a single layer dataset. The layer needs to be of polygons
# and needs to have the target files' projection
dst_layer = dst_ds.CreateLayer('', srs = to_proj, geom_type=ogr.wkbPolygon )

# Open the original shapefile, get the first layer, and filter by attribute
vector_ds = ogr.Open( target_vector_file )
lyr = vector_ds.GetLayer( 0 )
lyr.SetAttributeFilter( attribute_filter )

# Get a field definition from the original vector file.
# We don't need much more detail here
feature = lyr.GetFeature(0)
field = feature.GetFieldDefnRef( 0 )
# Apply the field definition from the original to the output
dst_layer.CreateField( field )
feature_defn = dst_layer.GetLayerDefn()
# Reset the original layer so we can read all features
lyr.ResetReading()
for feat in lyr:
    # For each feature, get the geometry
    geom = feat.GetGeometryRef()
```

```

# transform it to the reference projection
geom.Transform ( tx )
# Create an output feature
out_geom = ogr.Feature ( feature_defn )
# Set the geometry to be the reprojected/transformed geometry
out_geom.SetGeometry ( geom )
# Add the feature with its geometry to the output yaer
dst_layer.CreateFeature(out_geom)
# Clear things up
out_geom.Destroy
geom.Destroy
# Done adding geometries
# Reset the output layer to the 0th geometry
dst_layer.ResetReading()

# Now, we rastertize the output vector in-memory file
# into the in-memory output raster file

err = gdal.RasterizeLayer(target_ds, [1], dst_layer,
                           burn_values=[burn_value])
if err != 0:
    print("error:", err)

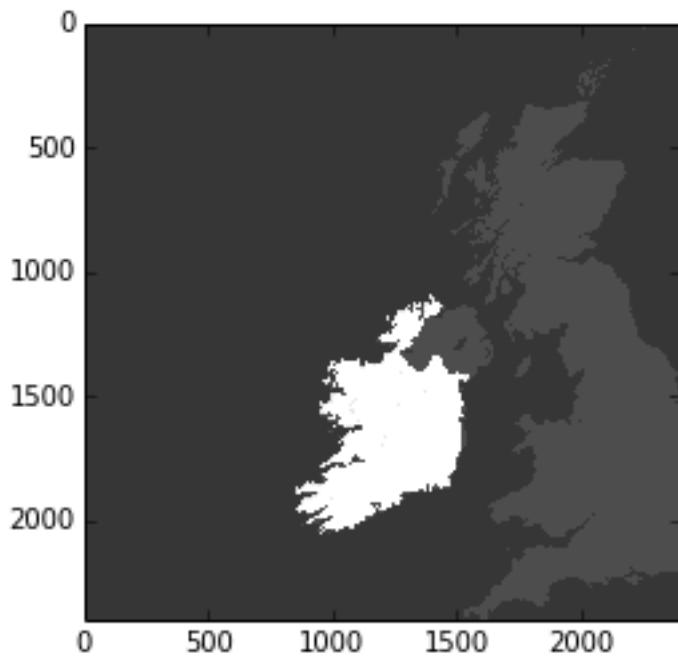
# Read the data from the raster, this is your mask
data = target_ds.ReadAsArray()

# Plotting to see whether this makes sense.

ndata = g.ReadAsArray()
plt.imshow ( ndata, interpolation='nearest', cmap=plt.cm.gray, vmin=0, vmax=1, alpha=0.3 )
plt.hold ( True )

plt.imshow ( data, interpolation='nearest', cmap=plt.cm.gray, alpha=0.7 )
plt.grid ( False )
plt.show()

```



7.5 Using matplotlib to plot geometries

Using matplotlib to plot geometries from OGR can be quite tedious. Here's an example of plotting a map of Angola from the `world.shp`. In the same vein of recommending Shapely and Fiona above for serious geoprocessing of vector data, you are encouraged to use `descartes` for plotting vector data!

```
import matplotlib.path as mpath
import matplotlib.patches as mpatches

# Extract first layer of features from shapefile using OGR
ds = ogr.Open('world.shp')
lyr = ds.GetLayer(0)

# Prepare figure
plt.ioff()
plt.subplot(1,1,1)
ax = plt.gca()

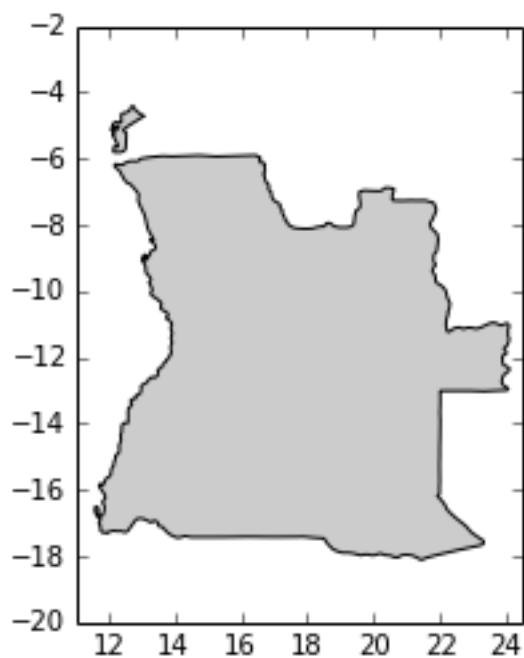
paths = []
lyr.ResetReading()

lyr.SetAttributeFilter ( " NAME = 'ANGOLA' " )
ax.set_xlim(11, 24.5)
ax.set_ylim(-20, -2)
# Read all features in layer and store as paths

for feat in lyr:

    for geom in feat.GetGeometryRef():
        envelope = np.array(geom.GetEnvelope())
        # check if geom is polygon
        if geom.GetGeometryType() == ogr.wkbPolygon:
            codes = []
            all_x = []
            all_y = []
            for i in range(geom.GetGeometryCount()):
                # Read ring geometry and create path
                r = geom.GetGeometryRef(i)
                x = [r.GetX(j) for j in range(r.GetPointCount())]
                y = [r.GetY(j) for j in range(r.GetPointCount())]
                # skip boundary between individual rings
                codes += [mpath.Path.MOVETO] + \
                          (len(x)-1)*[mpath.Path.LINETO]
                all_x += x
                all_y += y
            path = mpath.Path(np.column_stack((all_x,all_y)), codes)
            paths.append(path)
    # Add paths as patches to axes
    for path in paths:
        patch = mpatches.PathPatch(path, \
                                    facecolor='0.8', edgecolor='black')
        ax.add_patch(patch)

ax.set_aspect(1.0)
plt.show()
```



5. FUNCTION FITTING AND INTERPOLATION

In today's session, we will be using some of the LAI datasets we examined last week (masked by national boundaries) and doing some analysis on them.

- 5.1 Making 3D datasets and Movies First, we will examine how to improve our data reading function by extracting only the area we are interested in. This involves querying the 'country' mask to find its limits and passing this information through to the reader.
- 5.2 Interpolation Then we will look at methods to interpolate and smooth over gaps in datasets using various methods.
- 5.3 Function Fitting Finally, we will look at fitting models to datasets, in this case a model describing LAI phenology.

8.1 5.1 Making 3D datasets and Movies

First though, we will briefly go over once more the work we did on downloading data (ussssing `wget`), generating 3D masked datasets, and making movies.

This time, we will concentrate more on generating functions that we can re-use for other purposes.

8.1.1 5.1.1 Downloading data

We start by filtering the file `files/data/robot.txt` to get only lines (containing urls) for a particular tile and year.

We might easily do this in unix:

```
# filter LAI MODIS files for this year and tile (in bash)
tile='h17v03'
year='2005'
ofile=files/data/modis_lai_${tile}_${year}.txt

grep $tile < files/data/robot.txt | grep \/$year > $ofile

# have a look at the first few
head -4 < $ofile

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.01/MCD15A2.A2005001.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.09/MCD15A2.A2005009.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.17/MCD15A2.A2005017.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.25/MCD15A2.A2005025.h17v03.00
```

Now download the datasets:

```
tile='h17v03'
year='2005'
ofile=files/data/modis_lai_${tile}_${year}.txt
```

```
# go into the directory we want the data
pushd files/data
# get the urls from the file
# --cut-dirs=4 this time as there are 4 layers of directory we
# wish to ignore with this dataset
wget --quiet -nc -nH --cut-dirs=4 -i ../$ofile
# go back to where we were
popd

/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter5_Interpolation/files/data ...
/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter5_Interpolation
```

8.1.2 5.1.2 Read from an ASCII file

The ASCII file `files/data/modis_lai_{tile}_{year}.txt` contains lines of urls.

Each line (each url) is a string such as:

```
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.25/MCD15A2.A2005025...
```

The *filename* here is `MCD15A2.A2005025.h17v03.005.2007353055037.hdf`, so we can split the url on the field / to get this:

Let's read the filenames from the text file that has the urls in it and load it into a list that we will call `filelist`:

```
import numpy as np

tile = 'h17v03'
year = '2005'

# specify the file with the urls in
ifile= 'files/data/modis_lai_%s_%s.txt'%(tile,year)

# one way to read the data from the file
fp = open(ifile)
lines = fp.readlines()
filelist = []
for url in lines:
    filename = url.split('/')[-1].strip()
    filelist.append(filename)
fp.close()

# show the first few
print filelist[:5]

['MCD15A2.A2005001.h17v03.005.2007350235547.hdf', 'MCD15A2.A2005009.h17v03.005.2007351235445.hdf']

# a neater way:
fp = open(ifile)
filelist = [url.split('/')[-1].strip() for url in fp.readlines()]
fp.close()

# show the first few
print filelist[:5]

['MCD15A2.A2005001.h17v03.005.2007350235547.hdf', 'MCD15A2.A2005009.h17v03.005.2007351235445.hdf']

# an even neater way using np.loadtxt
# But don't worry if you don't quite get this one yet!

# define a function get_filename(f)
# When a function is 'small' its easier to use a lambda definition!
```

```

get_filename = lambda f: f.split('/')[-1]
filelist = np.loadtxt(ifile,dtype='str',converters={0:get_filename})

print filelist[:5]

['MCD15A2.A2005001.h17v03.005.2007350235547.hdf'
 'MCD15A2.A2005009.h17v03.005.2007351235445.hdf'
 'MCD15A2.A2005017.h17v03.005.2007352033411.hdf'
 'MCD15A2.A2005025.h17v03.005.2007353055037.hdf'
 'MCD15A2.A2005033.h17v03.005.2007355050158.hdf']

```

8.1.3 5.1.3 Read Just The Data We Want

Last time, we generated a function to read MODIS LAI data.

We have now included such a function in the directory ‘files/python <files/python>’ called ‘get_lai.py <files/python/get_lai.py>’.

The only added sophistication is that when we call `ReadAsArray`, we give it the starting cols, rows, and number of cols and rows to read (e.g. `xsize=600, yoff=300, xoff=300, ysize=600`):

```

# Now we have a list of filenames
# load read_lai
import sys
sys.path.insert(0,'files/python')

from get_lai import get_lai

help(get_lai)

Help on function get_lai in module get_lai:

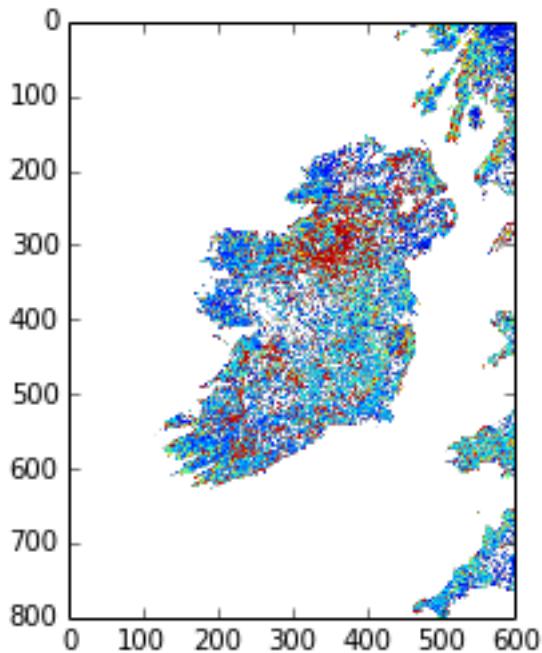
get_lai(filename, qc_layer='FparLai_QC', scale=[0.1, 0.1], mincol=0, minrow=0, ncol=None, nrow=None,
        xsize=600, yoff=300, xoff=300, ysize=600)

# e.g. for reading a single file:

lai_file0 = get_lai('files/data/%s'%filelist[20],ncol=600,mincol=300,minrow=400,nrow=800)
plt.imshow(lai_file0['Lai_1km'])

<matplotlib.image.AxesImage at 0x10342b50>

```



```
print type(lai_file0)
print lai_file0.keys()

<type 'dict'>
['Lai_1km', 'LaiStdDev_1km']
```

The function returns a dictionary with has keys ['Lai_1km', 'LaiStdDev_1km', 'FparLai_QC']:

```
print lai_file0['Lai_1km'].shape

(800, 600)
```

Each of these datasets is of shape (1200, 1200), but we have read only 600 (columns) and 800 (rows) in this case. Note that the numpy indexing is (rows,cols).

We know how to create a mask from a vector dataset from thelast session:

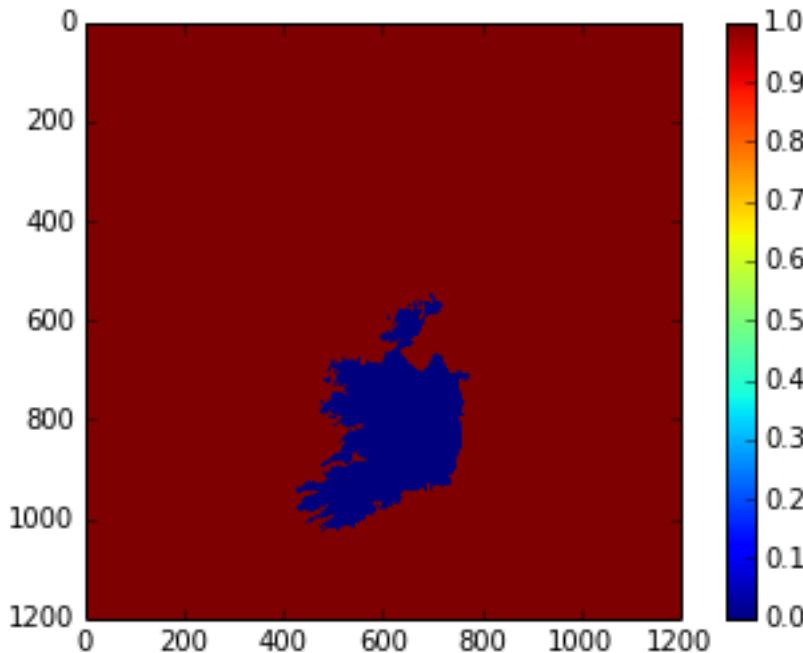
```
from raster_mask import raster_mask

# make a raster mask
# from the layer IRELAND in world.shp
filename = filelist[0]
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%('files/data/%s'%filename,'Lai_1km')

mask = raster_mask(file_spec,\n                  target_vector_file = "files/data/world.shp",\n                  attribute_filter = "NAME = 'IRELAND'")

plt.imshow(mask)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar instance at 0x10ae5bd8>
```



In this case, the data we want is only a small section of the whole spatial dataset.

It would be convenient to extract *only* the part we want.

We can use `numpy.where()` to help with this:

```
# The mask is False for the area we want
rowpix,colpix = np.where(mask == False)

print rowpix,colpix
[ 548  548  548 ..., 1024 1025 1025] [693 694 695 ..., 476 473 474]
```

`rowpix` and `colpix` are lists of pixel coordinates where the condition we specified is True (i.e. where `mask` is False).

If we wanted to find the bounds of this area, we simply need to know the minimum and maximum column and row in these lists:

```
mincol,maxcol = min(colpix),max(colpix)
minrow,maxrow = min(rowpix),max(rowpix)

# think about why the + 1 here!!!
# what if maxcol and mincol were the same?
ncol = maxcol - mincol + 1
nrow = maxrow - minrow + 1

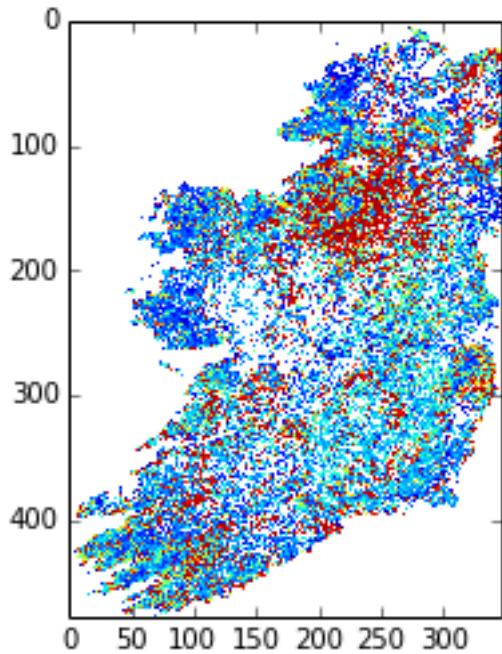
print minrow,mincol,nrow,ncol
548 422 478 348
```

We could use this information to extract *only* the area we want when we read the data:

```
lai_file0 = get_lai('files/data/%s'%filelist[20],\
                    ncol=ncol,nrow=nrow,mincol=mincol,minrow=minrow)

plt.imshow(lai_file0['Lai_1km'],interpolation='none')
```

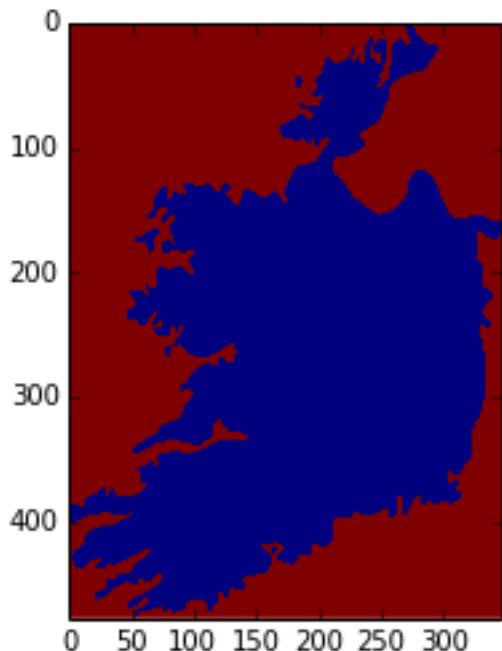
```
<matplotlib.image.AxesImage at 0x10ddc2d0>
```



Now, lets extract this portion of the mask:

```
small_mask = mask[minrow:minrow+nrow,mincol:mincol+ncol]  
  
plt.imshow(small_mask,interpolation='none')
```

```
<matplotlib.image.AxesImage at 0x2ae9e41a3750>
```



And combine the country mask with the small dataset:

As a recap, we can use the function `raster_mask` that we gave you last time to develop a raster mask (!) from an ESRI shapefile ([files/data/world.shp](#) here).

We can then combine this mask with the QC-derived mask in the LAI dataset.

The LAI mask (that will be `lai.mask` in the code below) is `False` for good data, as is the country mask.

To combine them, we want some operator X for which:

```
True X True == True
True X False == True
False X True == True
False X False == False
```

The operator to use then is an *or*, here, a bitwise or, `|`.

```
lai_file0 = get_lai('files/data/%s'%filelist[20], \
                    ncol=ncol, nrow=nrow, mincol=mincol, minrow=minrow)

layer = 'Lai_1km'
lai = lai_file0[layer]
small_mask = mask[minrow:minrow+nrow, mincol:mincol+ncol]

# combined mask
new_mask = small_mask | lai.mask

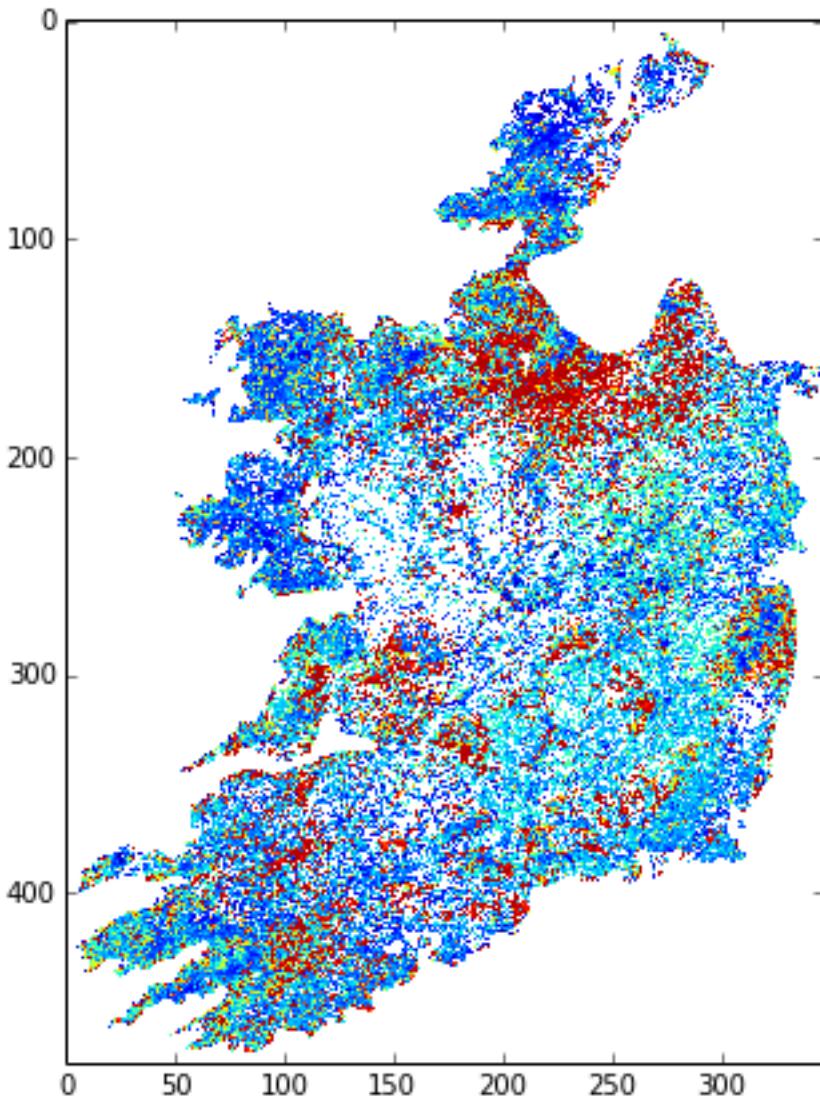
plt.figure(figsize=(7,7))
plt.imshow(new_mask, interpolation='none')

lai = ma.array(lai, mask=new_mask)

plt.figure(figsize=(7,7))
plt.imshow(lai, interpolation='none')

<matplotlib.image.AxesImage at 0x2ae9e4275f50>
```





We should be used to writing loops around such functions.

In this case, we read *all* of the files in `filelist` and put the data into the dictionary called `lai` here.

Because there are multiple layers in the datasets, we loop over layer and append to each list individually:

```
# load 'em all ...
# for United Kingdom here

import numpy.ma as ma
from raster_mask import raster_mask

country = 'UNITED KINGDOM'

# make a raster mask
# from the layer UNITED KINGDOM in world.shp
filename = filelist[0]
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%('files/data/%s'%filename,'Lai_1km')

mask = raster_mask(file_spec,
                    target_vector_file = "files/data/world.shp",
                    attribute_filter = "NAME = '%s'"%country)
# extract just the area we want
```

```
# by getting the min/max rows/cols
# of the data mask
# The mask is False for the area we want
rowpix,colpix = np.where(mask == False)
mincol,maxcol = min(coli),max(coli)
minrow,maxrow = min(rowpix),max(rowpix)
ncol = maxcol - mincol + 1
nrow = maxrow - minrow + 1
# and make a small mask
small_mask = mask[minrow:maxrow,mincol:maxcol+1]

# data_fields with empty lists
data_fields = {'LaiStdDev_1km':[],'Lai_1km':[]}

# make a dictionary and put the filenames in it
# along with the mask and min/max info
lai = {'filenames':np.sort(filelist),\
        'minrow':minrow,'mincol':mincol,\n        'mask':small_mask}

# combine the dictionaries
lai.update(data_fields)

# loop over each filename
for f in np.sort(lai['filenames']):
    this_lai = get_lai('files/data/%s'%f,\n                      mincol=mincol,ncol=ncol,\n                      minrow=minrow,nrow=nrow)
    for layer in data_fields.keys():
        # apply the mask
        new_mask = this_lai[layer].mask | small_mask
        this_lai[layer] = ma.array(this_lai[layer],mask=new_mask)
        lai[layer].append(this_lai[layer])

# have a look at one of these

i = 20

import pylab as plt

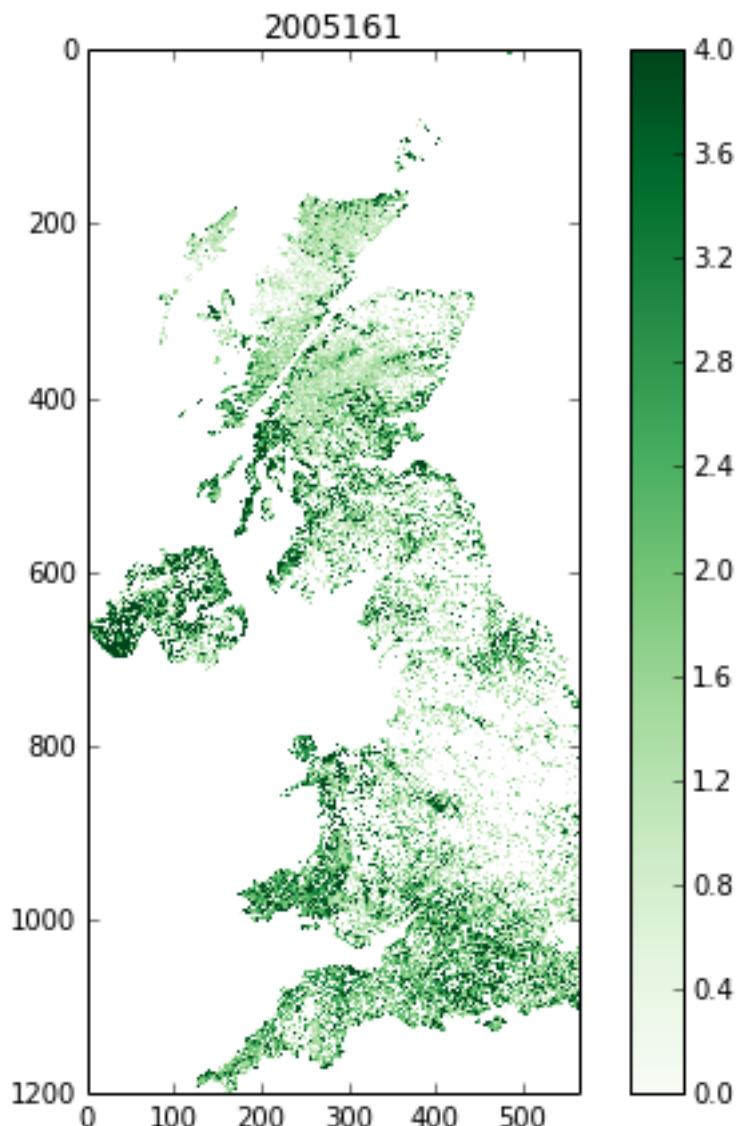
# just see what the shape is ...
print lai['Lai_1km'][i].shape

root = 'files/images/lai_uk'

cmap = plt.cm.Greens

f = lai['filenames'][i]
fig = plt.figure(figsize=(7,7))
# get some info from filename
file_id = f.split('/')[-1].split('.')[5][1:]
print file_id
plt.imshow(lai['Lai_1km'][i],cmap=cmap,interpolation='none',vmax=4.,vmin=0.0)
# plot a jpg
plt.title(file_id)
plt.colorbar()
plt.savefig('files/images/lai_uk_%s.jpg'%file_id)

(1200, 566)
2005161
```



```
# that's quite good, so put as a function:
import numpy.ma as ma
import numpy as np
import sys
sys.path.insert(0,'files/python')
from get_lai import get_lai
from raster_mask import raster_mask

def read_lai(filelist,datadir='files/data',country=None):
    '''
    Read MODIS LAI data from a set of files
    in the list filelist. Data assumed to be in
    directory datadir.

    Parameters:
    filelist : list of LAI files

    Options:
    datadir : data directory
    country : country name (in files/data/world.shp)

    Returns:

```

```
lai dictionary
'''
if country:
    # make a raster mask
    # from the layer UNITED KINGDOM in world.shp
    file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
    file_spec = file_template%'files/data/%s'%filelist[0], 'Lai_1km')

    mask = raster_mask(file_spec,
                        target_vector_file = "files/data/world.shp",
                        attribute_filter = "NAME = '%s'%"country)
    # extract just the area we want
    # by getting the min/max rows/cols
    # of the data mask
    # The mask is False for the area we want
    rowpix,colpix = np.where(mask == False)
    mincol,maxcol = min(coli),max(coli)
    minrow,maxrow = min(rowpix),max(rowpix)
    ncol = maxcol - mincol + 1
    nrow = maxrow - minrow + 1
    # and make a small mask
    small_mask = mask[minrow:minrow+nrow,mincol:mincol+ncol]

else:
    # no country
    mincol = 0
    maxcol = 0
    ncol = None
    nrow = None

# data_fields with empty lists
data_fields = {'LaiStdDev_1km':[], 'Lai_1km':[]}

# make a dictionary and put the filenames in it
# along with the mask and min/max info
lai = {'filenames':np.sort(filelist), \
        'minrow':minrow,'mincol':mincol, \
        'mask':small_mask}

# combine the dictionaries
lai.update(data_fields)

# loop over each filename
for f in np.sort(lai['filenames']):
    this_lai = get_lai('files/data/%s'%f, \
                       mincol=mincol,ncol=ncol, \
                       minrow=minrow,nrow=nrow)
    for layer in data_fields.keys():
        # apply the mask
        if country:
            new_mask = this_lai[layer].mask | small_mask
            this_lai[layer] = ma.array(this_lai[layer],mask=new_mask)
        lai[layer].append(this_lai[layer])
    for layer in data_fields.keys():
        lai[layer] = ma.array(lai[layer])

return lai

# test this ... the one in the file
# does a cutout of the data area as well
# which will keep the memory
# requirements down
from get_lai import read_lai
```

```

lai = read_lai(filelist,country='IRELAND',verbose=True)

# have a look at one of these

i = 20

# just see what the shape is ...
print lai['Lai_1km'][i].shape

root = 'files/images/lai_eire'

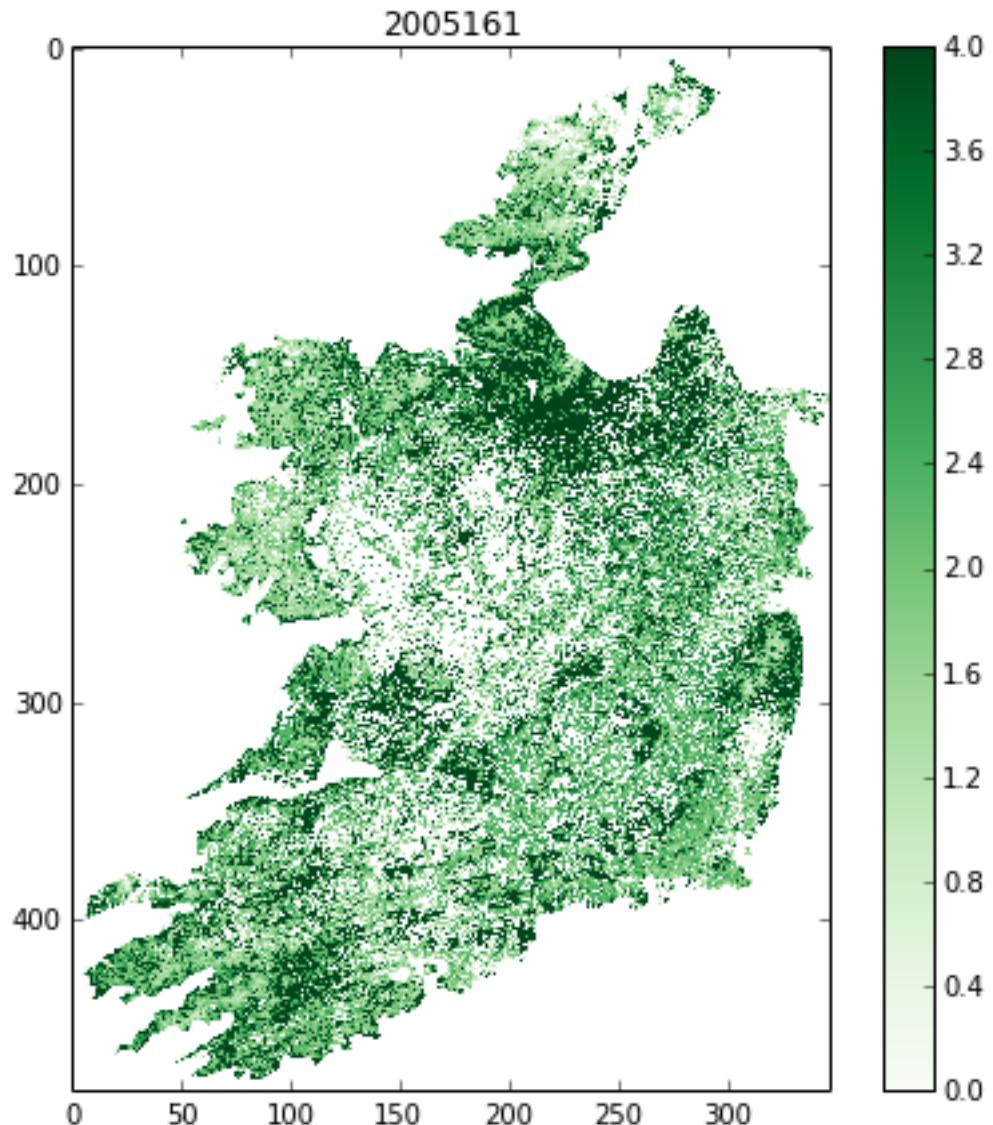
cmap = plt.cm.Greens

f = lai['filenames'][i]
fig = plt.figure(figsize=(7,7))
# get some info from filename
file_id = f.split('/')[-1].split('.')[ -5 ][1:]
print file_id
plt.imshow(lai['Lai_1km'][i],cmap=cmap,interpolation='none',vmax=4.,vmin=0.0)
# plot a jpg
plt.title(file_id)
plt.colorbar()
plt.savefig('%s_%s.jpg'%(root,file_id))

creating mask of IRELAND
...
MCD15A2.A2005001.h17v03.005.2007350235547.hdf
...
MCD15A2.A2005009.h17v03.005.2007351235445.hdf
...
MCD15A2.A2005017.h17v03.005.2007352033411.hdf
...
MCD15A2.A2005025.h17v03.005.2007353055037.hdf
...
MCD15A2.A2005033.h17v03.005.2007355050158.hdf
...
MCD15A2.A2005041.h17v03.005.2007357014602.hdf
...
MCD15A2.A2005049.h17v03.005.2007360165724.hdf
...
MCD15A2.A2005057.h17v03.005.2007361230641.hdf
...
MCD15A2.A2005065.h17v03.005.2007365024202.hdf
...
MCD15A2.A2005073.h17v03.005.2008001043631.hdf
...
MCD15A2.A2005081.h17v03.005.2008003173048.hdf
...
MCD15A2.A2005089.h17v03.005.2008005154542.hdf
...
MCD15A2.A2005097.h17v03.005.2008007175837.hdf
...
MCD15A2.A2005105.h17v03.005.2008018085544.hdf
...
MCD15A2.A2005113.h17v03.005.2008021020137.hdf
...
MCD15A2.A2005121.h17v03.005.2008021193749.hdf
...
MCD15A2.A2005129.h17v03.005.2008024061330.hdf
...
MCD15A2.A2005137.h17v03.005.2008032075236.hdf
...
MCD15A2.A2005145.h17v03.005.2008033192556.hdf
...
MCD15A2.A2005153.h17v03.005.2008035054421.hdf
...
MCD15A2.A2005161.h17v03.005.2008036173810.hdf
...
MCD15A2.A2005169.h17v03.005.2008039132812.hdf
...
MCD15A2.A2005177.h17v03.005.2008042090537.hdf
...
MCD15A2.A2005185.h17v03.005.2008044115459.hdf
...
MCD15A2.A2005193.h17v03.005.2008046140018.hdf
...
MCD15A2.A2005201.h17v03.005.2008050015227.hdf
...
MCD15A2.A2005209.h17v03.005.2008052203557.hdf
...
MCD15A2.A2005217.h17v03.005.2008055145215.hdf
...
MCD15A2.A2005225.h17v03.005.2008057010213.hdf
...
MCD15A2.A2005233.h17v03.005.2008060214119.hdf
...
MCD15A2.A2005241.h17v03.005.2008063115631.hdf
...
MCD15A2.A2005249.h17v03.005.1998144165707.hdf
...
MCD15A2.A2005257.h17v03.005.2008067051936.hdf
...
MCD15A2.A2005265.h17v03.005.2008069073121.hdf
...
MCD15A2.A2005273.h17v03.005.2008071050025.hdf
...
MCD15A2.A2005281.h17v03.005.2008072202421.hdf
...
MCD15A2.A2005289.h17v03.005.2008074194126.hdf
...
MCD15A2.A2005297.h17v03.005.2008077061121.hdf

```

```
... MCD15A2.A2005305.h17v03.005.2008080055607.hdf
... MCD15A2.A2005313.h17v03.005.2008083165435.hdf
... MCD15A2.A2005321.h17v03.005.2008084043211.hdf
... MCD15A2.A2005329.h17v03.005.2008086063619.hdf
... MCD15A2.A2005337.h17v03.005.2008087175845.hdf
... MCD15A2.A2005345.h17v03.005.2008088144615.hdf
... MCD15A2.A2005353.h17v03.005.2008091004441.hdf
... MCD15A2.A2005361.h17v03.005.2008091025114.hdf
... done
(478, 348)
2005161
```



```
# make a movie

import pylab as plt
import os

# just see what the shape is ...
print lai['Lai_1km'].shape

root = 'files/images/lai_country_eire'

cmap = plt.cm.Greens
```

```
for i,f in enumerate(lai['filenames']):
    fig = plt.figure(figsize=(7,7))
    # get some info from filename
    file_id = f.split('/')[-1].split('.')[5][1:]
    print file_id
    plt.imshow(lai['Lai_1km'][i],cmap=cmap,interpolation='none',vmax=4.,vmin=0.0)
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('%s_%s.jpg'%(root,file_id))
    plt.close(fig)

cmd = 'convert -delay 100 -loop 0 {0}_*.jpg {0}_movie.gif'.format(root)
os.system(cmd)

(46, 478, 348)
2005001
2005009
2005017
2005025
2005033
2005041
2005049
2005057
2005065
2005073
2005081
2005089
2005097
2005105
2005113
2005121
2005129
2005137
2005145
2005153
2005161
2005169
2005177
2005185
2005193
2005201
2005209
2005217
2005225
2005233
2005241
2005249
2005257
2005265
2005273
2005281
2005289
2005297
2005305
2005313
2005321
2005329
2005337
2005345
2005353
2005361
```

0

```
# The movie making works, so pack that into a function

import pylab as plt
import os

root = 'files/images/lai_eire'

def make_movie(lai,root,layer='Lai_1km',vmax=4.,vmin=0.,do_plot=False):
    """
    Make an animated gif from MODIS LAI data in
    dictionary 'lai'.

    Parameters:
    lai      : data dictionary
    root    : root file /directory name of frames and movie

    layer   : data layer to plot
    vmax    : max value for plotting
    vmin    : min value for plotting
    do_plot: set True if you want the individual plots
              to display

    Returns:
    movie name

    """
    cmap = plt.cm.Greens

    for i,f in enumerate(lai['filenames']):
        fig = plt.figure(figsize=(7,7))
        # get some info from filename
        file_id = f.split('/')[-1].split('.')[1:-5]
        print file_id
        plt.imshow(lai[layer][i],cmap=cmap,interpolation='none',\
                   vmax=vmax,vmin=vmin)
        # plot a jpg
        plt.title(file_id)
        plt.colorbar()
        plt.savefig('%s_%s.jpg'%(root,file_id))
        if not do_plot:
            plt.close(fig)

    cmd = 'convert -delay 100 -loop 0 {0}_*.jpg {0}_movie.gif'.format(root)
    os.system(cmd)
    return '{0}_movie.gif'.format(root)

# test it

lai_uk = read_lai(filelist,country='UNITED KINGDOM')
root = 'files/images/lai_UK'
movie = make_movie(lai_uk,root)
print movie

2005001
2005009
2005017
2005025
2005033
2005041
```

```
2005049
2005057
2005065
2005073
2005081
2005089
2005097
2005105
2005113
2005121
2005129
2005137
2005145
2005153
2005161
2005169
2005177
2005185
2005193
2005201
2005209
2005217
2005225
2005233
2005241
2005249
2005257
2005265
2005273
2005281
2005289
2005297
2005305
2005313
2005321
2005329
2005337
2005345
2005353
2005361
files/images/lai_UK_movie.gif
```

8.2 5.2 Interpolation

8.2.1 5.2.1 Univariate interpolation

So, we can load the data we want from multiple MODIS hdf files that we have downloaded from the NASA server into a 3D masked numpy array, with a country boundary mask (projected int the raster data coordinate system) from a vector dataset.

Let's start to explore the data then.

You should have an array of LAI for Ireland:

```
type(lai['Lai_1km'])
numpy.ma.core.MaskedArray
```

Let's plot the LAI for some given pixels.

First, we might like to identify which pixels actually have any data.

A convenient function for this would be `np.where` that returns the indices of items that are `True`.

Since the data mask is `False` for good data, we take the complement `~` so that good data are ‘`True`’:

```
data = lai['Lai_1km']
np.where(~data.mask)

(array([ 3,  3,  3, ..., 39, 39, 39]),
 array([326, 328, 329, ..., 472, 472, 475]),
 array([ 82, 145,  83, ...,  86,  87,   51]))
```

An example good pixel this is (3,329,145). Let’s look at this for all time periods:

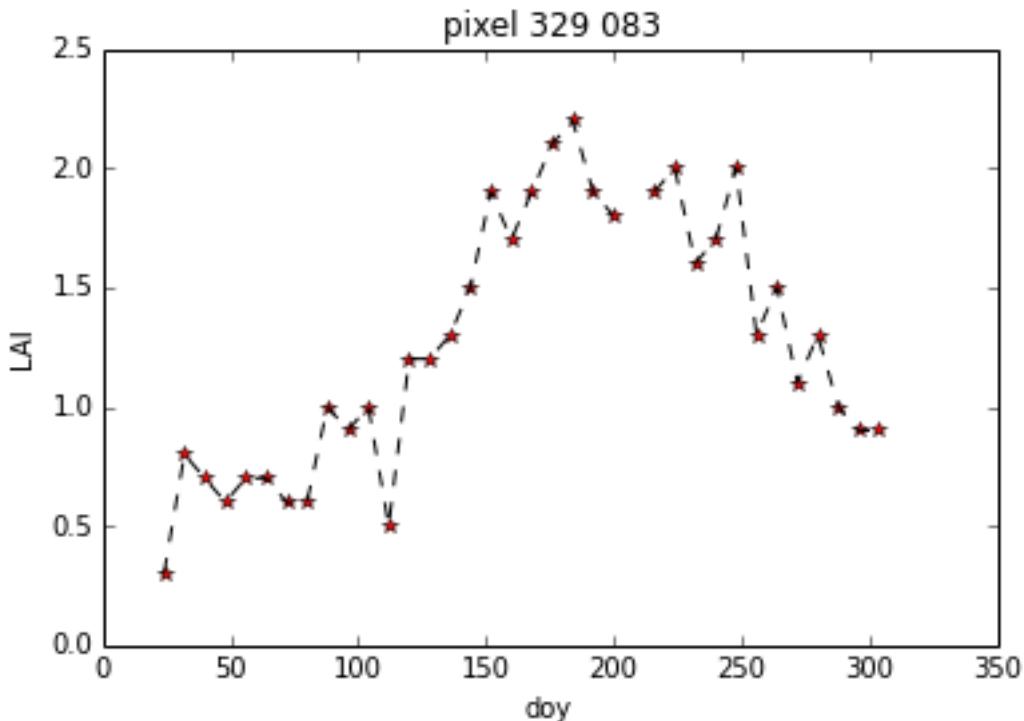
```
data = lai['Lai_1km']

r = 329
c = 83

pixel = data[:,r,c]

# plot red stars at the data points
plt.plot(np.arange(len(pixel))*8,pixel,'r*')
# plot a black (k) dashed line (--)
plt.plot(np.arange(len(pixel))*8,pixel,'k--')
plt.xlabel('doy')
plt.ylabel('LAI')
plt.title('pixel %03d %03d'%(r,c))

<matplotlib.text.Text at 0x11e77950>
```



The data follow the trend of what we might expect for LAI development, but they are clearly a little noisy.

We also have access to uncertainty information (standard deviation):

```
# copy the data in case we change it any
data = lai['Lai_1km'].copy()
```

```

sd    = lai['LaiStdDev_1km'].copy()

r = 329
c = 83

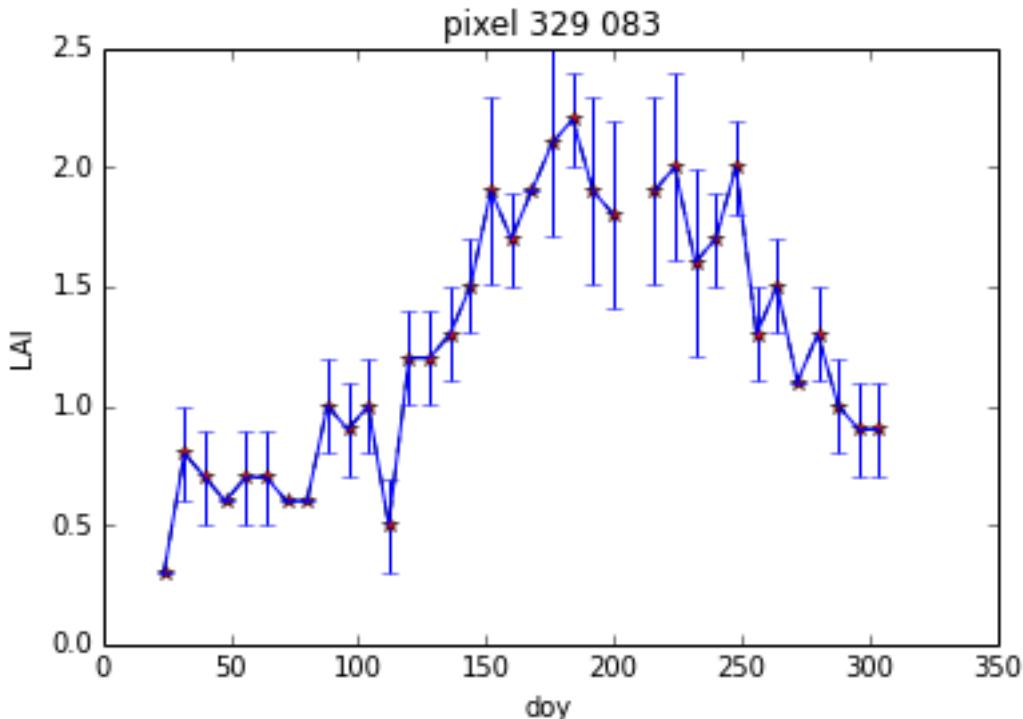
pixel    = data[:,r,c]
pixel_sd = sd[:,r,c]

x = np.arange(len(pixel))*8

# plot red stars at the data points
plt.plot(x,pixel,'r*')
# plot a black (k) dashed line (--)
plt.plot(x,pixel,'k--')
# plot error bars:
# 1.96 because that is the 95% confidence interval
plt.errorbar(x,pixel,yerr=pixel_sd*1.96)
plt.xlabel('doy')
plt.ylabel('LAI')
plt.title('pixel %03d %03d' %(r,c))

<matplotlib.text.Text at 0x2ae9e53409d0>

```



We would generally expect LAI to be quite smoothly varying over time. Visualising the data with 95% confidence intervals is quite useful as we can now ‘imagine’ some smooth line that would generally go within these bounds.

Some of the uncertainty estimates are really rather small though, which are probably not reliable.

Let’s inflate them:

```

data = lai['Lai_1km'].copy()
sd    = lai['LaiStdDev_1km'].copy()

r = 329
c = 83

pixel    = data[:,r,c]

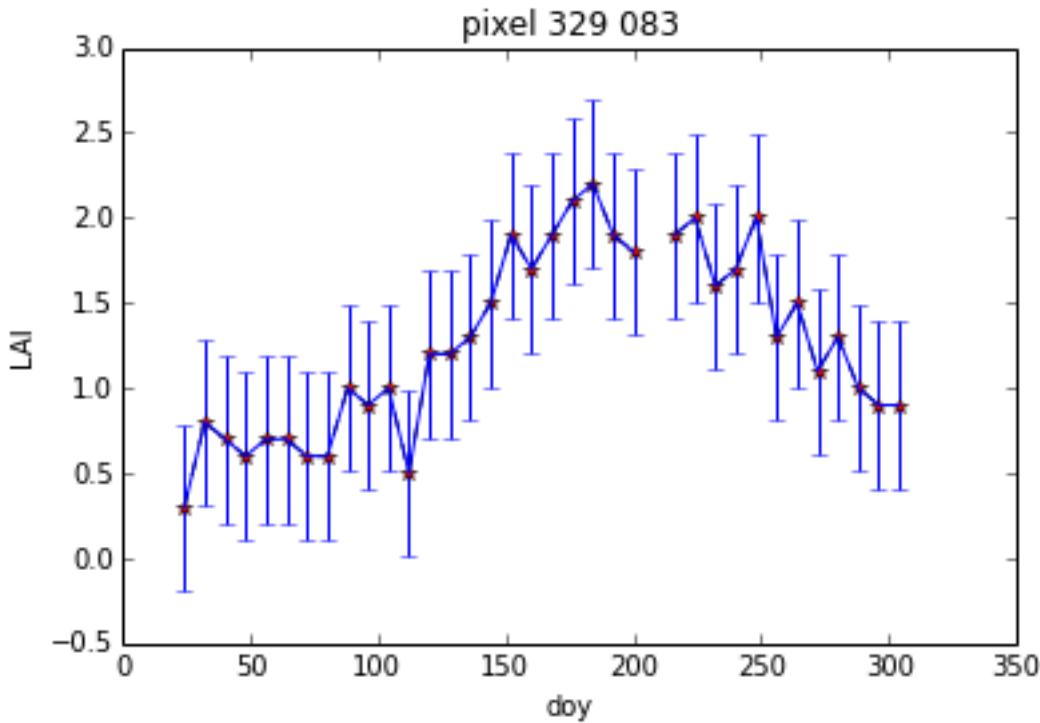
```

```
pixel_sd = sd[:,r,c]
# threshold
thresh = 0.25
pixel_sd[pixel_sd<thresh] = thresh

x = np.arange(len(pixel))*8

# plot red stars at the data points
plt.plot(x,pixel,'r*')
# plot a black (k) dashed line (--)
plt.plot(x,pixel,'k--')
# plot error bars:
# 1.96 because that is the 95% confidence interval
plt.errorbar(x,pixel,yerr=pixel_sd*1.96)
plt.xlabel('doy')
plt.ylabel('LAI')
plt.title('pixel %03d %03d'%(r,c))

<matplotlib.text.Text at 0x2ae9e5554910>
```



This is perhaps a bit more realistic ...

The data now have some missing values (data gaps) and, as we have noted, are a little noisy.

A Python module we can use for many scientific functions is ‘scipy’
<http://docs.scipy.org/doc/scipy>‘, in particular here, the ‘scipy’ interpolation functions
<http://docs.scipy.org/doc/scipy/reference/interpolate.html>‘.

We need to make a careful choice of the interpolation functions.

We might, in many circumstances simply want something that interpolates between data points, i.e. that goes through the data points that we have.

Many interpolators will not provide extrapolation, so in the example above we could not get an estimate of LAI prior to the first sample and after the last.

The best way to deal with that would be to have multiple years of data.

Instead here, we will repeat the dataset three times to mimic this:

```

from scipy import interpolate

pixel = data[:,r,c]

# original x,y
y_ = pixel
x_ = (np.arange(len(y_))*8.+1)[~pixel.mask]
y_ = y_[~pixel.mask]

# extend: using np.tile() to repeat data
y_extend = np.tile(y_, 3)
# extend: using vstack to stack 3 different arrays
x_extend = np.hstack((x_-46*8, x_, x_+46*8))

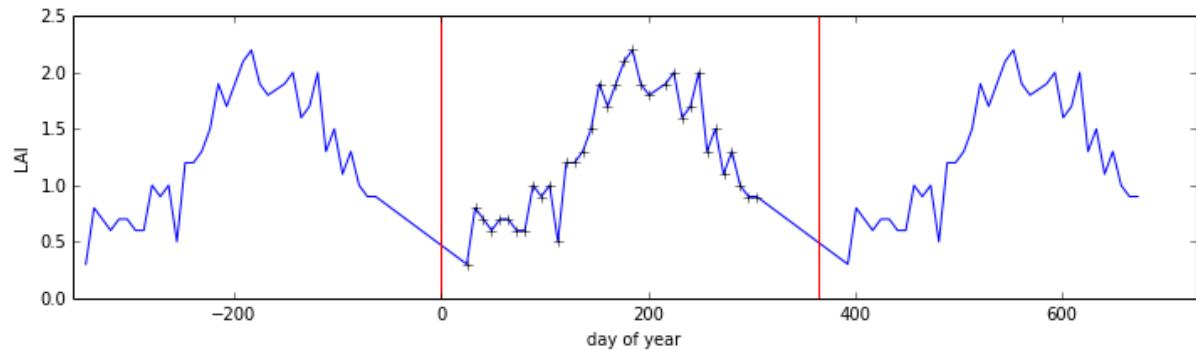
```

```

# plot the extended dataset
plt.figure(figsize=(12, 3))
plt.plot(x_extend,y_extend,'b')
plt.plot(x_,y_, 'k+')
plt.plot([0.,0.],[0.,2.5],'r')
plt.plot([365.,365.],[0.,2.5],'r')
plt.xlim(-356,2*365)
plt.xlabel('day of year')
plt.ylabel('LAI')

```

<matplotlib.text.Text at 0x2ae9e6108550>



```

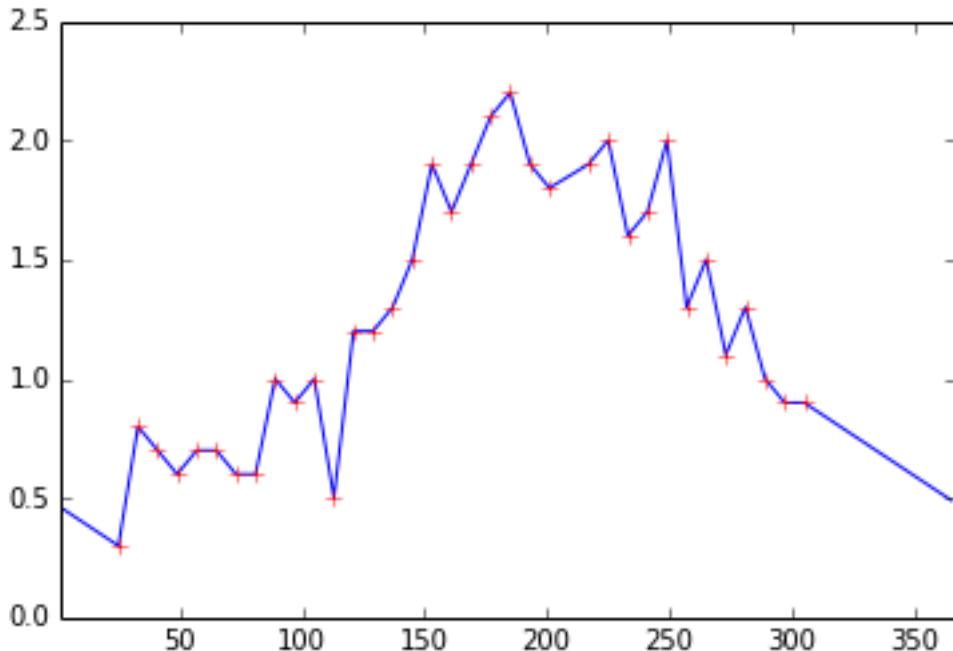
# define xnew at 1 day interval
xnew = np.arange(1.,366.)

# linear interpolation
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
ynew = f(xnew)

plt.plot(xnew,ynew)
plt.plot(x_,y_, 'r+')
plt.xlim(1,366)

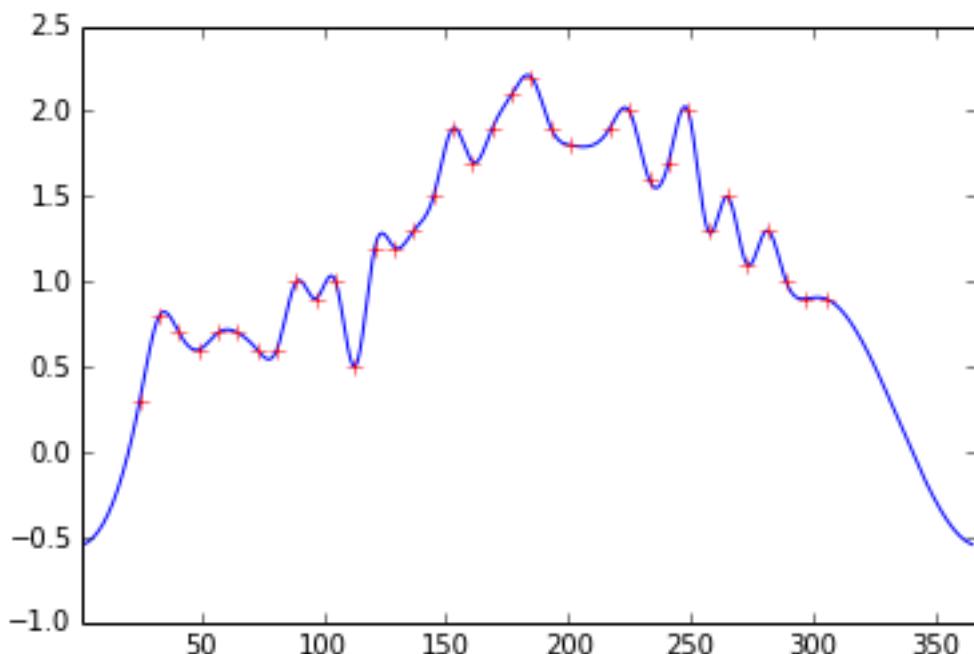
(1, 366)

```



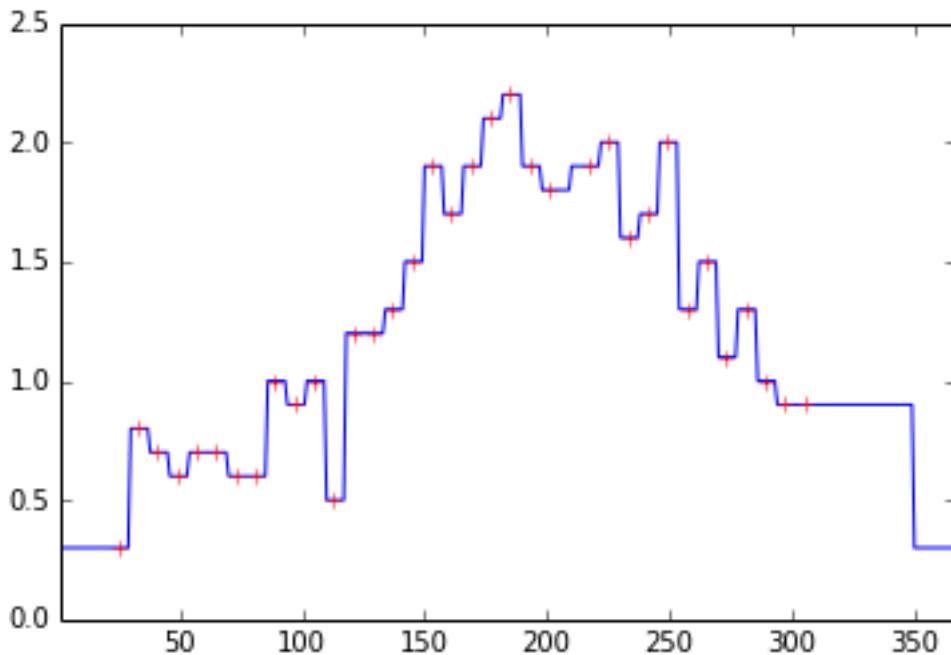
```
# cubic interpolation
f = interpolate.interp1d(x_extend,y_extend,kind='cubic')
ynew = f(xnew)
plt.plot(xnew,ynew)
plt.plot(x_,y_,'r+')
plt.xlim(1,366)

(1, 366)
```



```
# nearest neighbour interpolation
f = interpolate.interp1d(x_extend,y_extend,kind='nearest')
ynew = f(xnew)
plt.plot(xnew,ynew)
plt.plot(x_,y_,'r+')
plt.xlim(1,366)
```

(1, 366)



Depending on the problem you are trying to solve, different interpolation schemes will be appropriate. For categorical data (e.g. ‘snow’, coded as 1 and ‘no snow’ coded as 0), for instance, a nearest neighbour interpolation might be a good idea.

8.2.2 5.2.2 Smoothing

One issue with the schemes above is that they go exactly through the data points, but a more realistic description of the data might be one that incorporated the uncertainty information we have. Visually, this is quite easy to imagine, but how can we implement such ideas?

One way of thinking about this is to think about other sources of information that we might bring to bear on the problem. One such would be that we expect the function to be ‘quite smooth’. This allows us to consider applying smoothness as an additional constraint to the solution.

Many such problems can be phrased as convolution operations.

Convolution is a form of digital filtering that combines two sequences of numbers y and w to give a third, the result z that is a filtered version of y , where for each element j of y :

$$z_j = \sum_{i=-n}^{i=n} w_i y_{j+i}$$

where n is the half width of the filter w . For a smoothing filter, the elements of this will sum to 1 (so that the magnitude of y is not changed).

To illustrate this in Python:

```
# a simple box smoothing filter
# filter width 11
w = np.ones(11)
# normalise
w = w/w.sum()
# half width
n = len(w)/2

# Take the linear interpolation of the LAI above as the signal
```

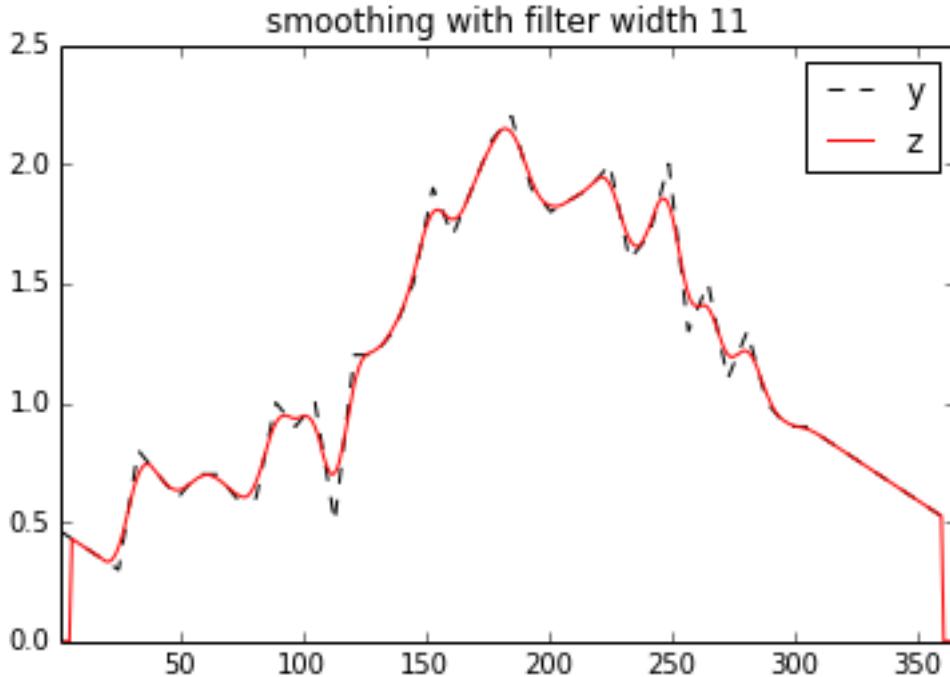
```
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

# where we will put the result
z = np.zeros_like(y)

# This is a straight implementation of the
# equation above
for j in xrange(n,len(y)-n):
    for i in xrange(-n,n+1):
        z[j] += w[n+i] * y[j+i]

plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
plt.legend(loc='best')
plt.title('smoothing with filter width %d'%len(w))

<matplotlib.text.Text at 0x2ae9e579e3d0>
```



As we suggested, the result of convolving y with the filter w (of width 31 here) is z , a smoothed version of y .

You might notice that the filter is only applied once we are n samples into the signal, so we get ‘edge effects’. There are various ways of dealing with edge effects, such as repeating the signal (as we did above, for much the same reason), reflecting the signal, or assuming the signal to be some constant value (e.g. 0) outside of its defined domain.

If we make the filter wider (width 31 now):

```
# a simple box smoothing filter
# filter width 31
w = np.ones(31)
# normalise
w = w/w.sum()
# half width
n = len(w)/2
```

```

# Take the linear interpolation of the LAI above as the signal
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

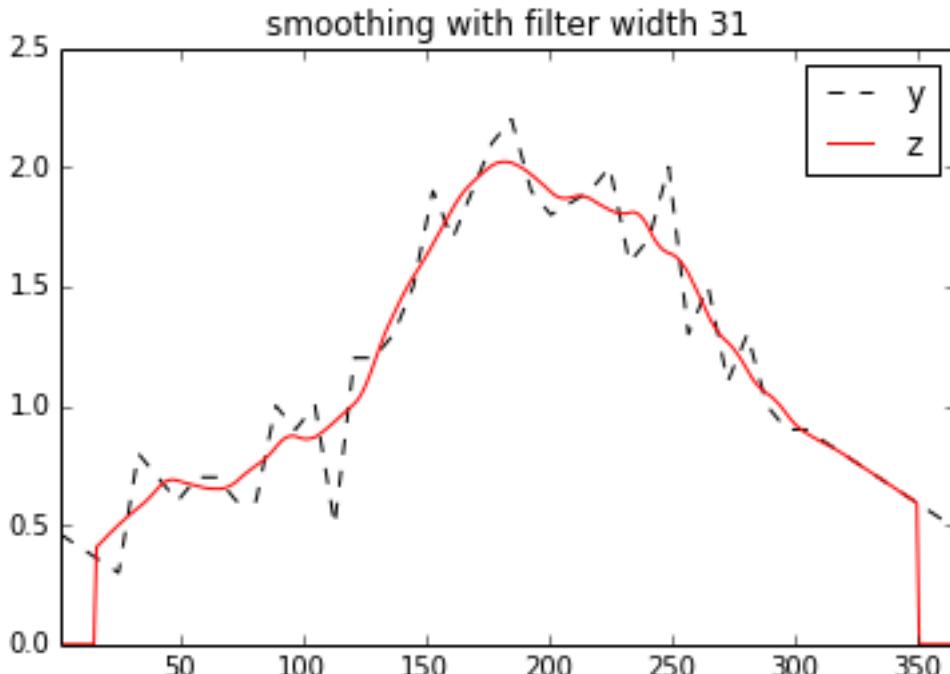
# where we will put the result
z = np.zeros_like(y)

# This is a straight implementation of the
# equation above
for j in xrange(n,len(y)-n):
    for i in xrange(-n,n+1):
        z[j] += w[n+i] * y[j+i]

plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
plt.legend(loc='best')
plt.title('smoothing with filter width %d'%len(w))

<matplotlib.text.Text at 0x2ae9e61f8cd0>

```



Then the signal is ‘more’ smoothed.

There are *many* filters implemented in ‘scipy.signal <<http://docs.scipy.org/doc/scipy/reference/signal.html>>‘ __ that you should look over.

A very commonly used smoothing filter is the Savitsky-Golay filter for which you define the window size and filter order.

As with most filters, the filter width controls the degree of smoothing (see examples above). The filter order (related to polynomial order) in essence controls the shape of the filter and defines the ‘peakiness’ of the response.

```

import sys
sys.path.insert(0,'files/python')
# see http://wiki.scipy.org/Cookbook/SavitzkyGolay
from savitzky_golay import *

```

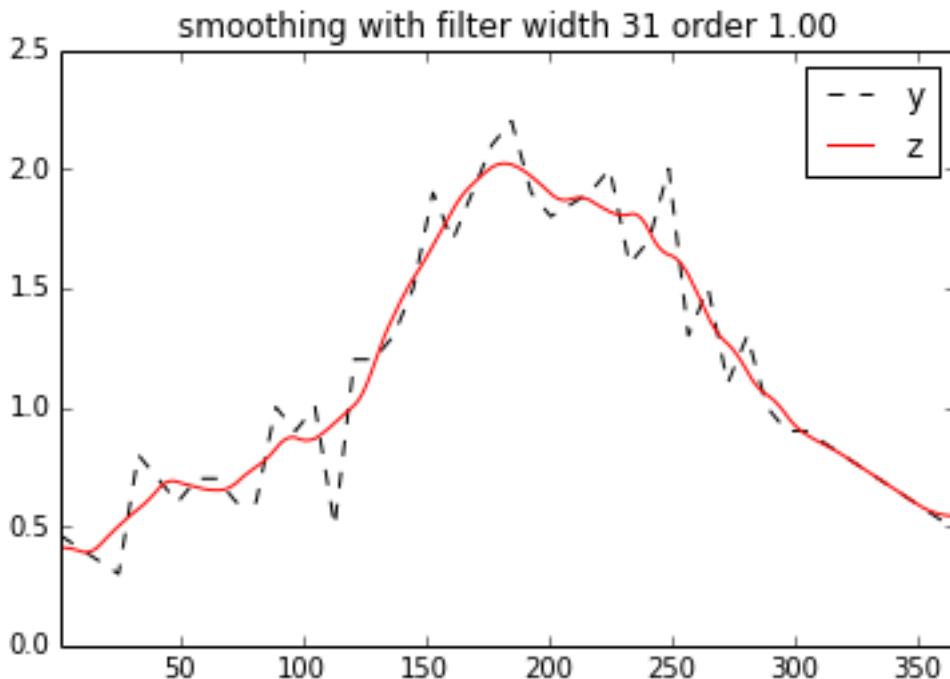
```
window_size = 31
order = 1

# Take the linear interpolation of the LAI above as the signal
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

z = savitzky_golay(y,window_size,order)

plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
plt.legend(loc='best')
plt.title('smoothing with filter width %d order %.2f'%(window_size,order))

<matplotlib.text.Text at 0x2ae9e65e6890>
```



```
import sys
sys.path.insert(0,'files/python')
# see http://wiki.scipy.org/Cookbook/SavitzkyGolay
from savitzky_golay import *

window_size = 61
order = 2

# Take the linear interpolation of the LAI above as the signal
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

z = savitzky_golay(y,window_size,order)

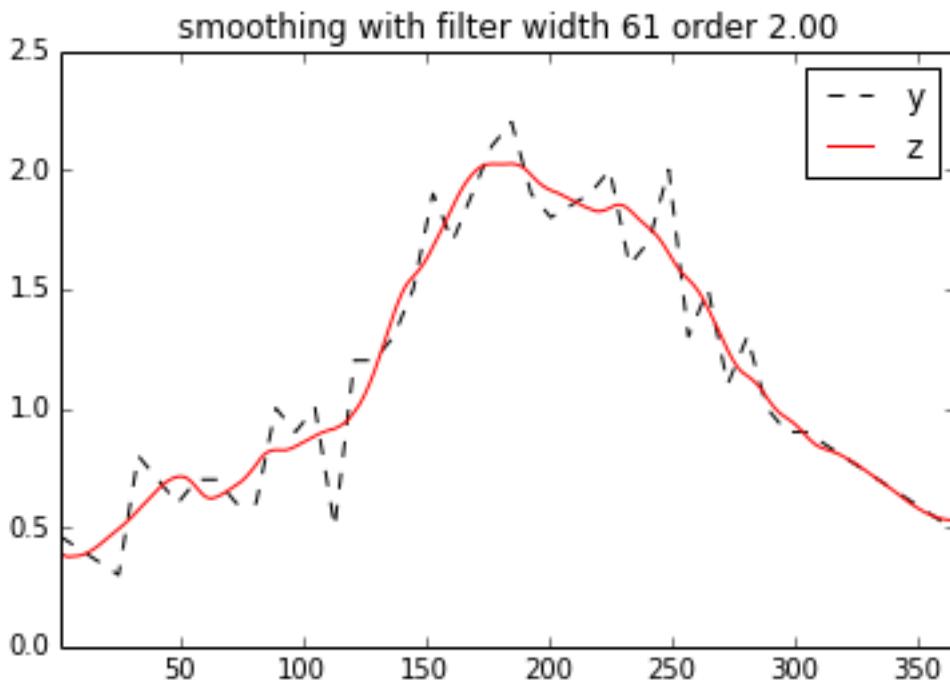
plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
```

```

plt.legend(loc='best')
plt.title('smoothing with filter width %d order %.2f'%(window_size,order))

<matplotlib.text.Text at 0x2ae9e663bb50>

```



If the samples y have uncertainty (standard deviation σ_j for sample j) associated with them, we can incorporate this into smoothing, although many of the methods in `scipy` and `numpy` do not directly allow for this.

Instead, we call an optimal interpolation scheme (a regulariser) here that achieves this. This also has the advantage of giving an estimate of uncertainty for the smoothed samples.

In this case, the parameters are: `order` (as above, but only integer in this implementation) and `wsd` which is an estimate of the variation (standard deviation) in the signal that control smoothness.

```

tile = 'h17v03'
year = '2005'

# specify the file with the urls in
ifile= 'files/data/modis_lai_%s_%s.txt'%(tile,year)

fp = open(ifile)
filelist = [url.split('/')[-1].strip() for url in fp.readlines()]
fp.close()
import sys
sys.path.insert(0,'files/python')

from get_lai import *

try:
    data = lai['Lai_1km']
    sd = lai['LaiStdDev_1km']
except:
    lai = read_lai(filelist,country='IRELAND')
    data = lai['Lai_1km']
    sd = lai['LaiStdDev_1km']

thresh = 0.25
sd[sd

```

```
r = 472
c = 84
from smoothn import *

# this is about the right amount of smoothing here
gamma = 5.

pixel = data[:,r,c]
pixel_sd = sd[:,r,c]

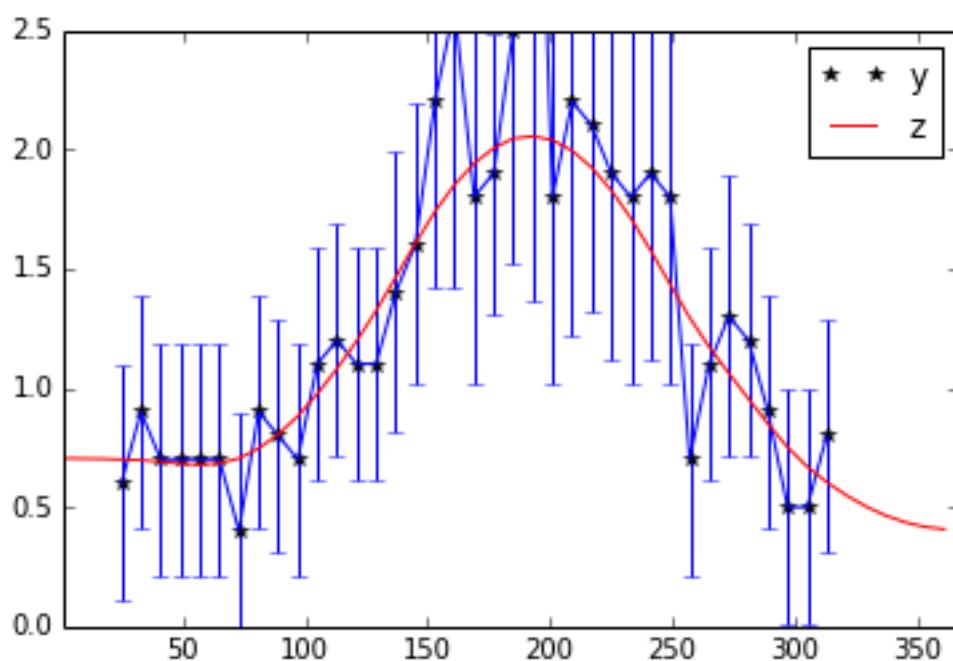
x = np.arange(46)*8+1

order = 2
z = smoothn(pixel,s=gamma, sd=pixel_sd, smoothOrder=2.0)[0]

# plot
plt.plot(x,pixel,'k*',label='y')
plt.errorbar(x,pixel,pixel_sd*1.96)
plt.plot(x,z,'r',label='z')
# lower and upper bounds of 95% CI

plt.xlim(1,366)
plt.ylim(0.,2.5)
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2b75e4be2790>
```



```
# test it on a new pixel

r = 472
c = 86

gamma = 5

pixel = data[:,r,c]
pixel_sd = sd[:,r,c]

x = np.arange(46)*8+1
```

```

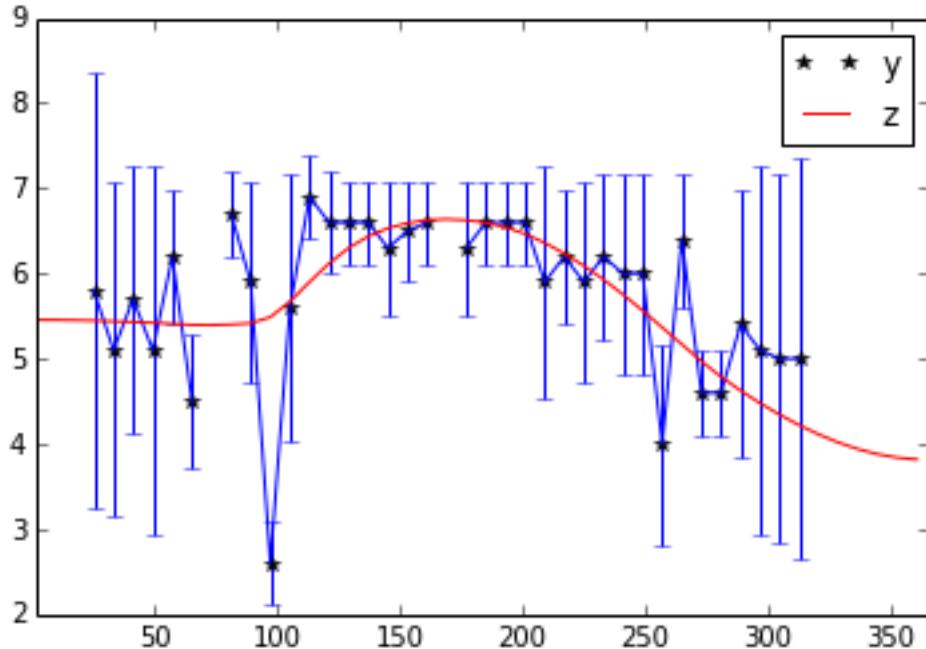
order = 2
z = smoothn(pixel,s=gamma, sd=pixel_sd,smoothOrder=2.0) [0]

# plot
plt.plot(x,pixel,'k*',label='y')
plt.errorbar(x,pixel,pixel_sd*1.96)
plt.plot(x,z,'r',label='z')

plt.xlim(1,366)
plt.legend(loc='best')
z.ndim

```

1



```

# and test it on a new pixel

r = 472
c = 84

#r = 9
#c = 277
gamma = 5.

pixel = data[:,r,c]
pixel_sd = sd[:,r,c]

x = np.arange(46)*8+1

order = 2
# solve for gamma - degree of smoothness
zz = smoothn(pixel, sd=pixel_sd, smoothOrder=2.0)
z = zz[0]
print zz[1],zz[2]

gamma = zz[1]

# plot
plt.plot(x,pixel,'k*',label='y')
plt.errorbar(x,pixel,pixel_sd*1.96)

```

```

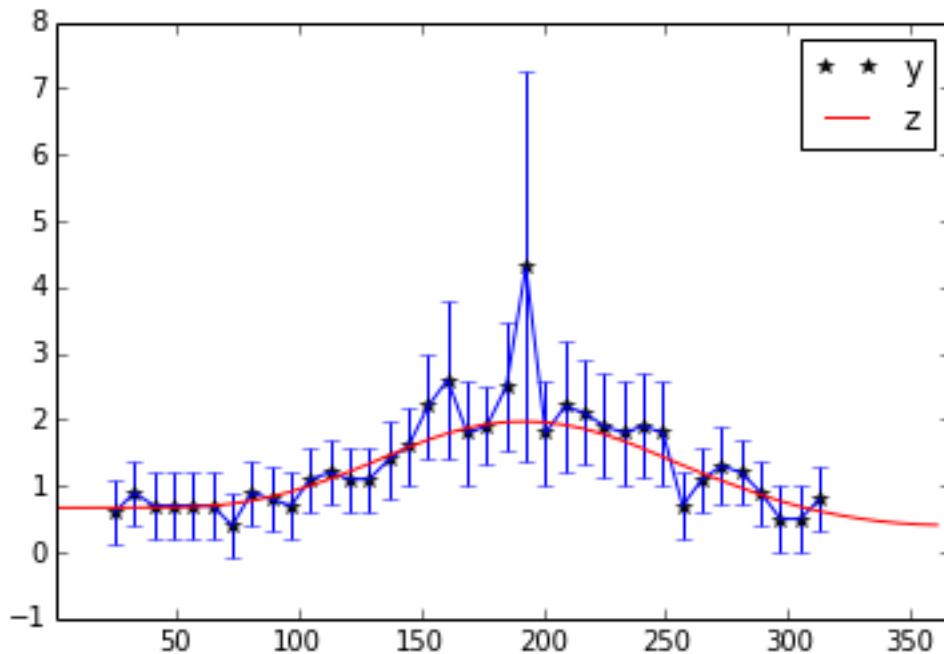
plt.plot(x,z,'r',label='z')

plt.xlim(1,366)
plt.legend(loc='best')

7.56265788653 True

<matplotlib.legend.Legend at 0x2b75e6878890>

```



To apply this approach to our 3D dataset, we could simply loop over all pixels.

Note that *any* per-pixel processing will be slow ... but this is quite a fast smoothing method, so is feasible here.

```

# we have put in an axis control to smoothn
# here so it will only smooth over day
# This will take a few minutes to process
# we switch on verbose mode to get some feedback
# on progress

# make a mask of pixels where there is at least 1 sample
# over the time period
mask = (data.mask.sum(axis=0) == 0)
mask = np.array([mask]*data.shape[0])

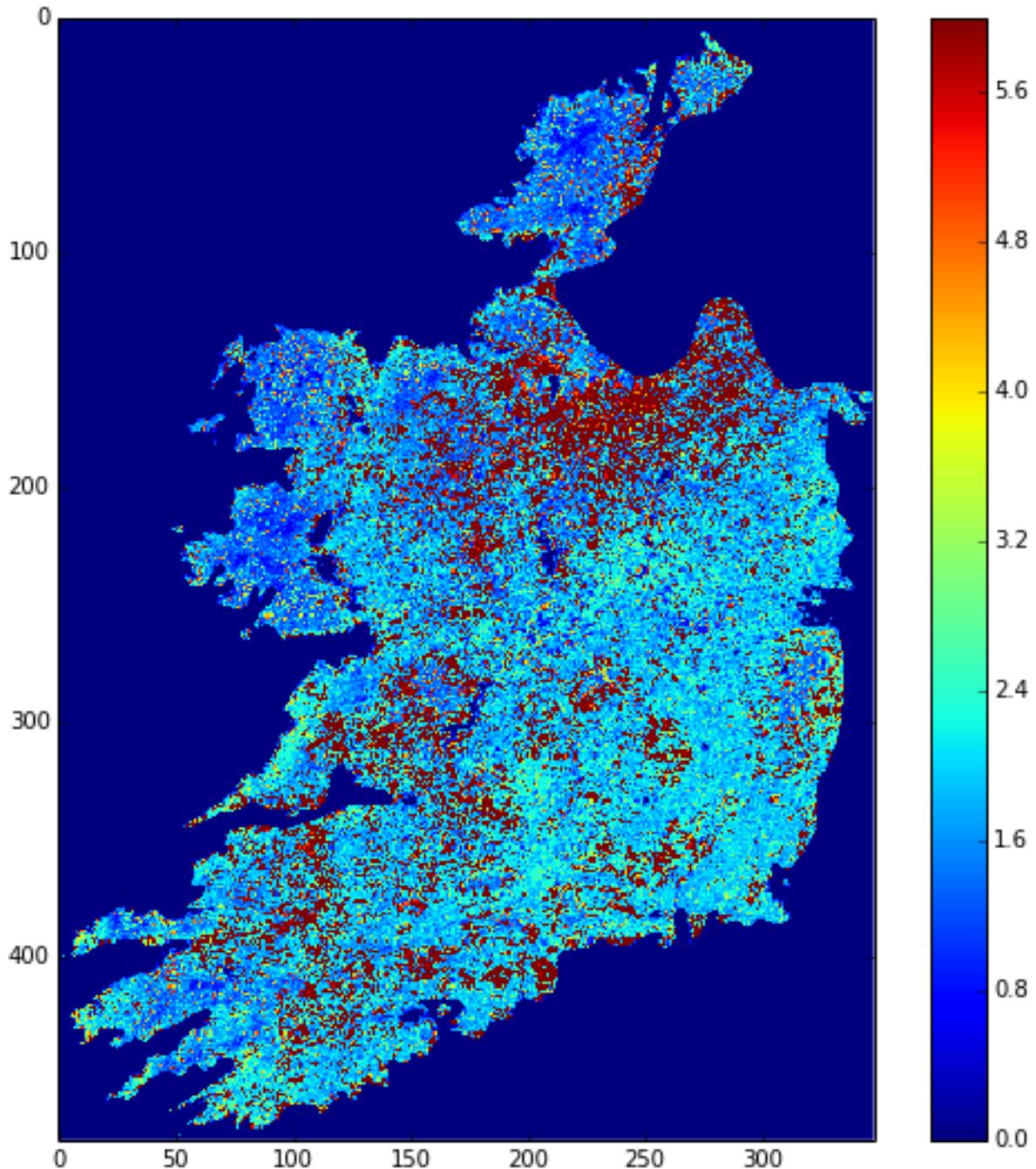
z = smoothn(data,s=5.0, sd=sd, smoothOrder=2.0, axis=0, TolZ=0.05, verbose=True)[0]
z = ma.array(z,mask=mask)

tol 1.0 nit 0
tol 1.03767913976 nit 1
tol 0.695375818129 nit 2
tol 0.55340286659 nit 3
tol 0.379048609608 nit 4
tol 0.297133997656 nit 5
tol 0.211254020382 nit 6
tol 0.161703395437 nit 7
tol 0.118022633002 nit 8
tol 0.089141179031 nit 9
tol 0.0662378920796 nit 10

```

```
plt.figure(figsize=(9,9))
plt.imshow(z[20],interpolation='none',vmax=6)
plt.colorbar()

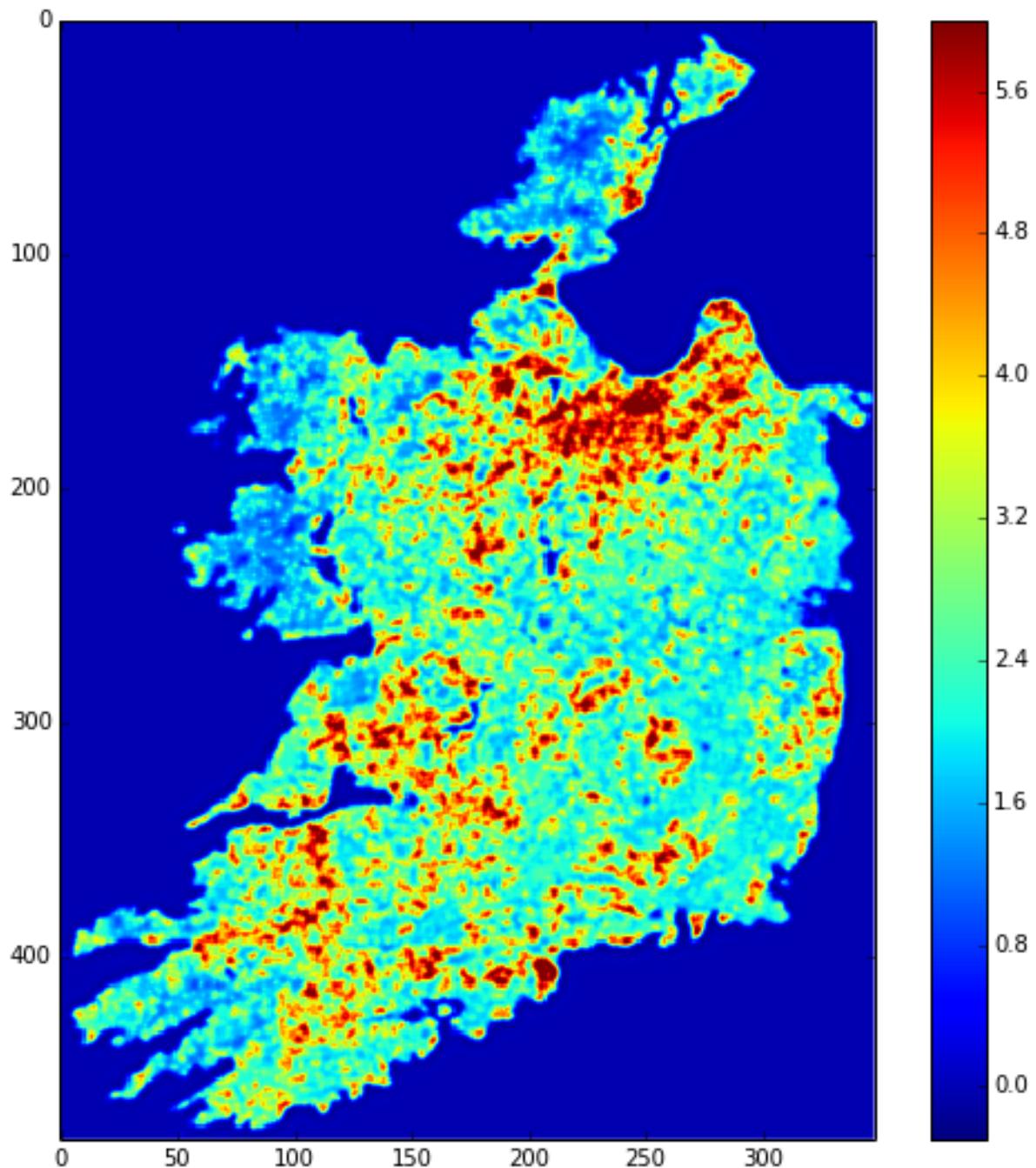
<matplotlib.colorbar.Colorbar instance at 0x2b75d2d9fd40>
```



```
# similarly, take frame 20
# and smooth that

ZZ = smoothn(z[20],smoothOrder=2.)
# self-calibrated smoothness term
s = ZZ[1]
print 's =',s
Z = ZZ[0]
plt.figure(figsize=(9,9))
plt.imshow(Z,interpolation='none',vmax=6)
```

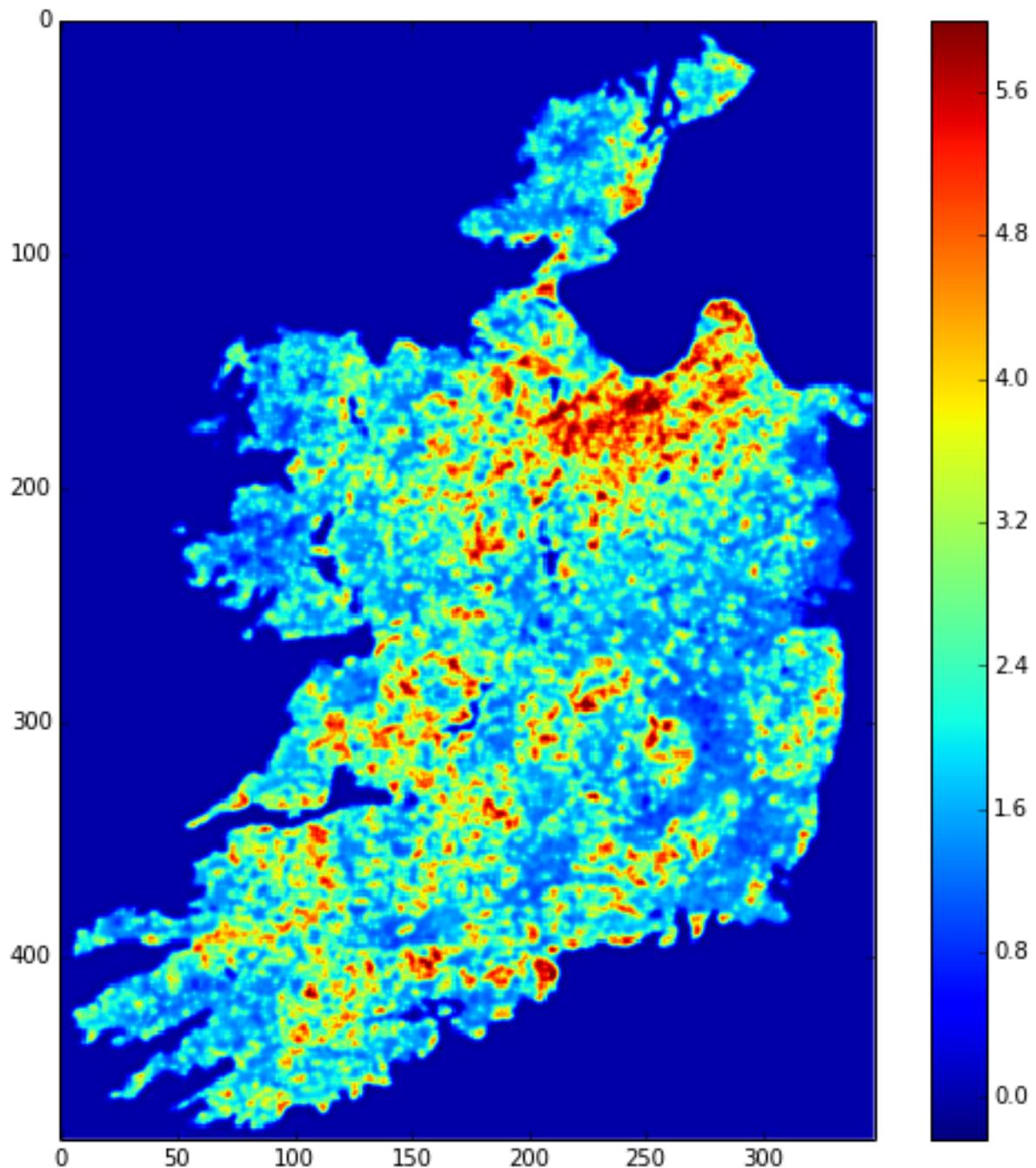
```
plt.colorbar()  
  
s = 0.731142059593  
  
<matplotlib.colorbar.Colorbar instance at 0x1857ee18>
```



```
# similarly, take frame 20  
# and smooth that  
  
ZZ = smoothn(z,s=s,smoothOrder=2.,axis=(1,2),verbose=True)  
  
Z = ZZ[0]  
plt.figure(figsize=(9,9))  
plt.imshow(Z[30],interpolation='none',vmax=6)  
plt.colorbar()
```

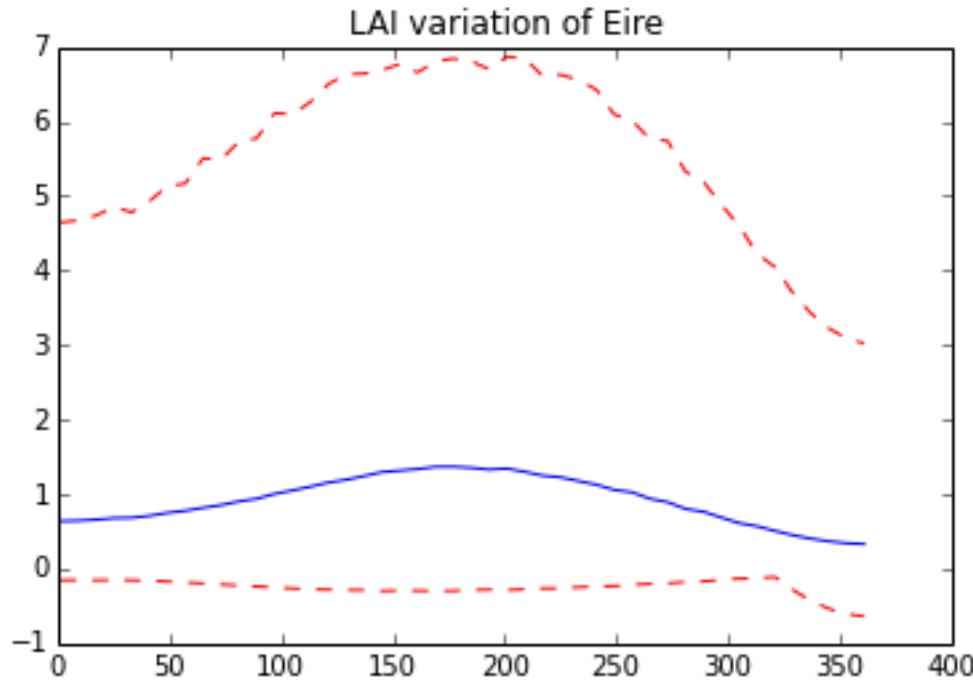
```
tol 1.0 nit 0
```

```
<matplotlib.colorbar.Colorbar instance at 0x23289dd0>
```



```
x = np.arange(46)*8+1.  
try:  
    plt.plot(x,np.mean(Z,axis=(1,2)))  
    plt.plot(x,np.min(Z,axis=(1,2)), 'r--')  
    plt.plot(x,np.max(Z,axis=(1,2)), 'r--')  
except:  
    plt.plot(x,np.mean(Z,axis=2).mean(axis=1))  
    plt.plot(x,np.min(Z,axis=2).min(axis=1), 'r--')  
    plt.plot(x,np.max(Z,axis=2).max(axis=1), 'r--')  
  
plt.title('LAI variation of Eire')
```

```
<matplotlib.text.Text at 0x232b2ed0>
```



```
# or doing this pixel by pixel ...
# which is slower than using axis

order = 2

# pixels that have some data
mask = (~data.mask).sum(axis=0)

odata = np.zeros((46,) + mask.shape)

rows,cols = np.where(mask>0)

len_x = len(rows)
order = 2
gamma = 5.

for i in xrange(len_x):
    r,c = rows[i],cols[i]
    # progress bar
    if i%(len_x/20) == 0:
        print '... %4.2f percent'%(i*100./float(len_x))
    pixel    = data[:,r,c]
    pixel_sd = sd[:,r,c]

    zz = smoothn(pixel,s=gamma, sd=pixel_sd,smoothOrder=order,TolZ=0.05)
    odata[:,rows[i],cols[i]] = zz[0]

...
0.00 percent
...
5.00 percent
...
10.00 percent
...
15.00 percent
...
20.00 percent
...
25.00 percent
...
30.00 percent
...
35.00 percent
...
40.00 percent
```

```

... 45.00 percent
... 50.00 percent
... 55.00 percent
... 60.00 percent
... 65.00 percent
... 70.00 percent
... 75.00 percent
... 80.00 percent
... 85.00 percent
... 90.00 percent
... 95.00 percent

import pylab as plt
import os

root = 'files/images/lai_eire_colourZ'

for i,f in enumerate(lai['filenames']):
    fig = plt.figure(figsize=(7,7))
    # get some info from filename
    file_id = f.split('/')[-1].split('.')[1:-5][1]
    print file_id
    plt.imshow(Z[i],interpolation='none',vmax=6.,vmin=0.0)
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('%s_%s.jpg'%(root,file_id))
    plt.close(fig)

2005001
2005009
2005017
2005025
2005033
2005041
2005049
2005057
2005065
2005073
2005081
2005089
2005097
2005105
2005113
2005121
2005129
2005137
2005145
2005153
2005161
2005169
2005177
2005185
2005193
2005201
2005209
2005217
2005225
2005233
2005241
2005249
2005257
2005265

```

```
2005273
2005281
2005289
2005297
2005305
2005313
2005321
2005329
2005337
2005345
2005353
2005361

cmd = 'convert -delay 100 -loop 0 {0}_*.jpg {0}_movie2.gif'.format(root)
os.system(cmd)

0
```

8.2.3 5.3 Function fitting

Sometimes, instead of applying some arbitrary smoothing function to data, we want to extract particular information from the time series.

One way to approach this is to fit some function to the time series at each location.

Let us suppose that we wish to characterise the phenology of vegetation in Ireland.

One way we could do this would be to look in the lai data for the most rapid changes.

Another would be to explicitly fit some mathematical function to the LAI data that would expect to describe typical LAI trajectories.

One example of such a function is the double logistic. A logistic function is:

$$\hat{y} = p_0 - p_1 \left(\frac{1}{1 + e^{p_2(t-p_3)}} + \frac{1}{1 + e^{p_4(t-p_5)}} - 1 \right)$$

We can give a function for a double logistic:

```
def dbl_logistic_model ( p, t ):
    """A double logistic model, as in Sobrino and Juliean, or Zhang et al"""
    return p[0] - p[1]* ( 1./(1+np.exp(p[2]*(t-p[3]))) + \
                         1./(1+np.exp(-p[4]*(t-p[5]))) ) - 1 )

tile = 'h17v03'
year = '2005'

# specify the file with the urls in
ifile= 'files/data/modis_lai_%s_%s.txt'%(tile,year)

fp = open(ifile)
filelist = [url.split('/')[-1].strip() for url in fp.readlines()]
fp.close()
import sys
sys.path.insert(0,'files/python')

from get_lai import *

try:
```

```

data = lai['Lai_1km']
sd = lai['LaiStdDev_1km']
except:
    lai = read_lai(filelist, country='IRELAND')
    data = lai['Lai_1km']
    sd = lai['LaiStdDev_1km']

thresh = 0.25
sd[sd# test pixel
r = 472
c = 84

y = data[:,r,c]
mask = ~y.mask
y = np.array(y[mask])
x = (np.arange(46)*8+1.) [mask]
unc = np.array(sd[:,r,c] [mask])

```

And see what this looks like:

```

# define x (time)
x_full = np.arange(1,366)

# some default values for the parameters
p = np.zeros(6)

# some stats on y
ysd = np.std(y)
ymean = np.mean(y)

# some rough guesses at the parameters

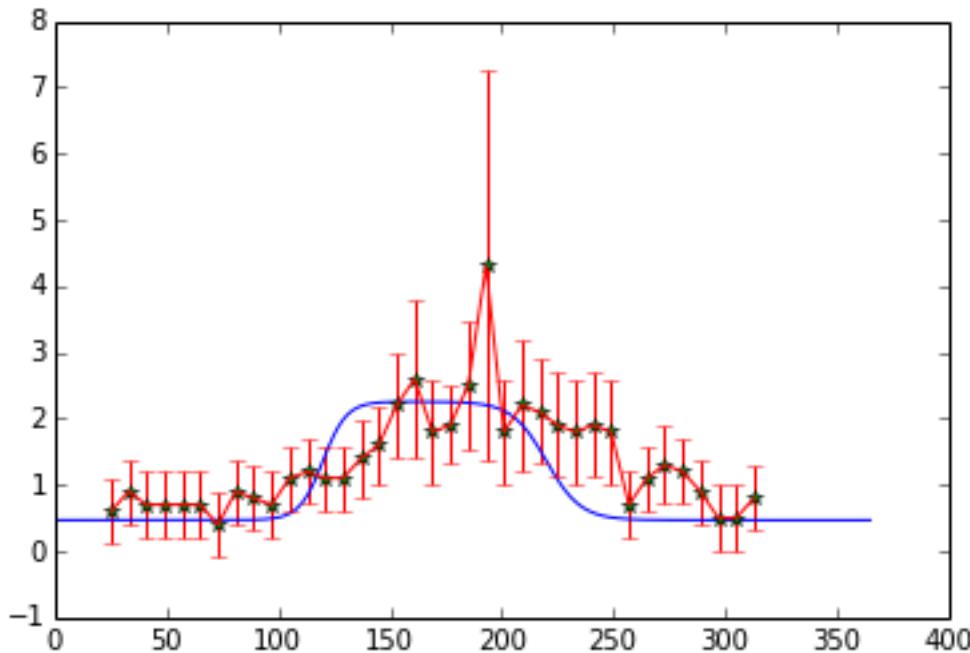
p[0] = ymean - 1.151*ysd;      # minimum (1.151 is 75% CI)
p[1] = 2*1.151*ysd           # range
p[2] = 0.19                   # related to up slope
p[3] = 120                     # midpoint of up slope
p[4] = 0.13                   # related to down slope
p[5] = 220                     # midpoint of down slope

y_hat = dbl_logistic_model(p,x_full)

plt.clf()
plt.plot(x_full,y_hat)
plt.plot(x,y,'*')
plt.errorbar(x,y,unc*1.96)

<Container object of 3 artists>

```



We could manually ‘tweak’ the parameters until we got a better ‘fit’ to the observations.

First though, let’s define a measure of ‘fit’:

$$Z_i = \frac{\hat{y}_i - y_i}{\sigma_i}$$

$$Z^2 = \sum_i Z_i^2 = \sum_i \left(\frac{\hat{y}_i - y_i}{\sigma_i} \right)^2$$

and implement this as a mismatch function where we have data points:

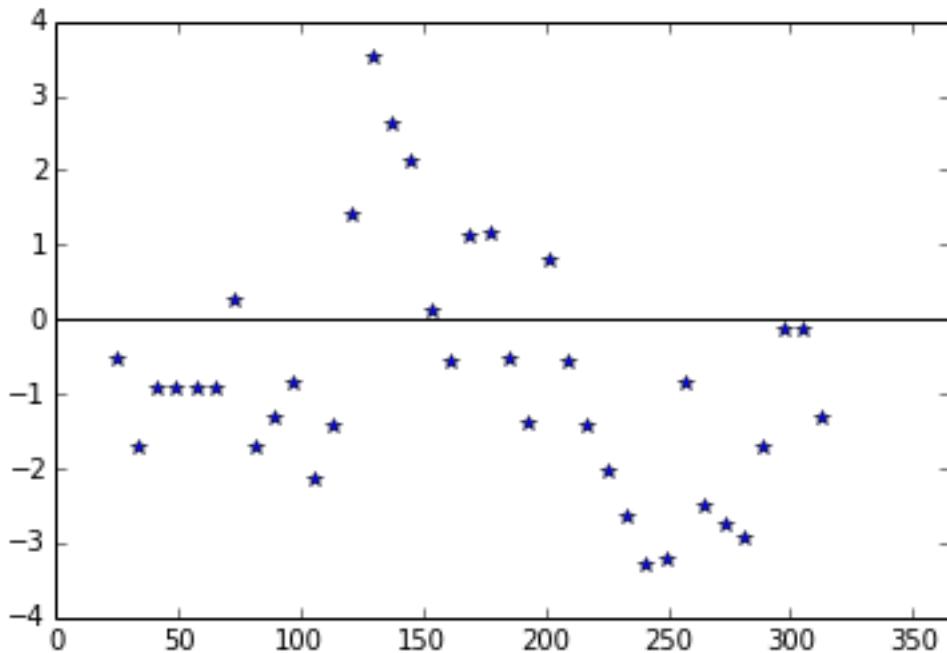
```
def mismatch_function(p, x, y, unc):
    y_hat = dbl_logistic_model(p, x)
    diff = (y_hat - y) / unc
    return diff
```

```
Z = mismatch_function(p,x,y,unc)
```

```
plt.plot([1,365.],[0,0.], 'k-')
plt.xlim(0,365)
plt.plot(x,Z, '.*')
```

```
print 'Z^2 =', (Z**2).sum()
```

```
Z^2 = 113.325251358
```



Now lets change p a bit:

```

p[0] = ymean - 1.151*ysd;      # minimum (1.151 is 75% CI)
p[1] = 2*1.151*ysd            # range
p[2] = 0.19                    # related to up slope
p[3] = 140                     # midpoint of up slope
p[4] = 0.13                    # related to down slope
p[5] = 220                     # midpoint of down slope

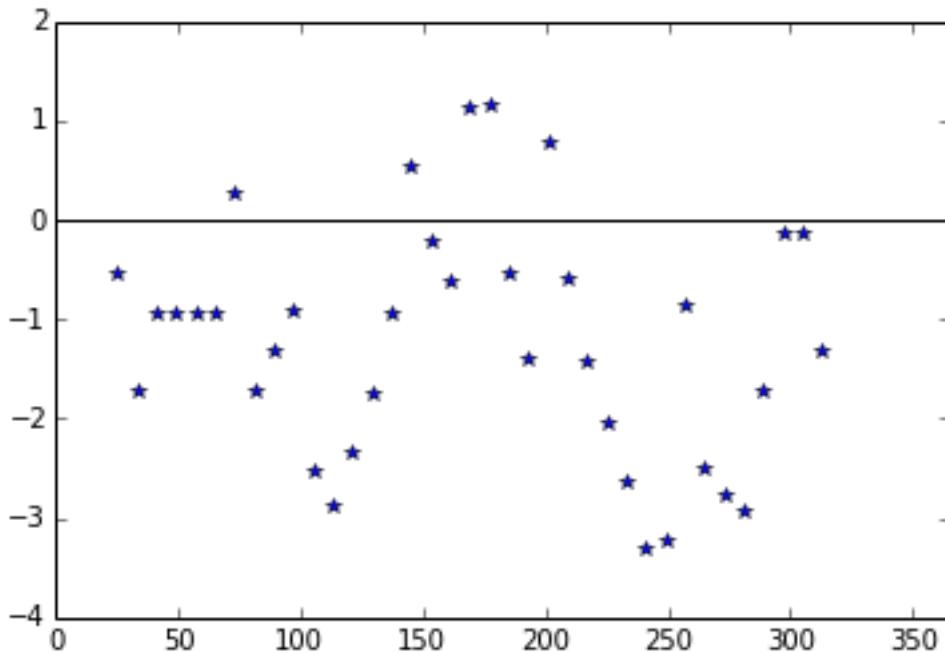
Z = mismatch_function(p,x,y,unc)

plt.plot([1,365.],[0,0.], 'k-')
plt.xlim(0,365)
plt.plot(x,Z, '*')

print 'Z^2 =', (Z**2).sum()

Z^2 = 105.478274642

```



We have made the mismatch go down a little ...

Clearly it would be tedious (and impractical) to do a lot of such tweaking, so we can use methods that seek the minimum of some function.

One such method is implemented in `scipy.optimize.leastsq`:

```
from scipy import optimize

# initial estimate is in p
print 'initial parameters:',p[0],p[1],p[2],p[3],p[4],p[5]

# set some bounds for the parameters
bound = np.array([(0.,10.), (0.,10.), (0.01,1.), (50.,300.), (0.01,1.), (50.,300.)])

# test pixel
r = 472
c = 84

y = data[:,r,c]
mask = ~y.mask
y = np.array(y[mask])
x = (np.arange(46)*8+1.) [mask]
unc = np.array(sd[:,r,c] [mask])

# define function to give Z^2

def sse(p,x,y,unc):
    '''Sum of squared error'''
    # penalise p[3] > p[5]
    err = np.max([0.,(p[3] - p[5])])*1e4
    return (mismatch_function(p,x,y,unc)**2).sum() +err

# we pass the function:
#
# sse           : the name of the function we wrote to give
#                   sum of squares of Z_i
# p             : an initial estimate of the parameters
```

```

# args=(x,y,unc)      : the other information (other than p) that
#                      mismatch_function needs
# approx_grad          : if we dont have a function for the gradient
#                      we have to get the solver to approximate it
#                      which takes time ... see if you can work out
#                      d_sse / dp and use that to speed this up!

psolve = optimize.fmin_l_bfgs_b(sse,p,approx_grad=True,iprint=-1,\n                                args=(x,y,unc),bounds=bound)

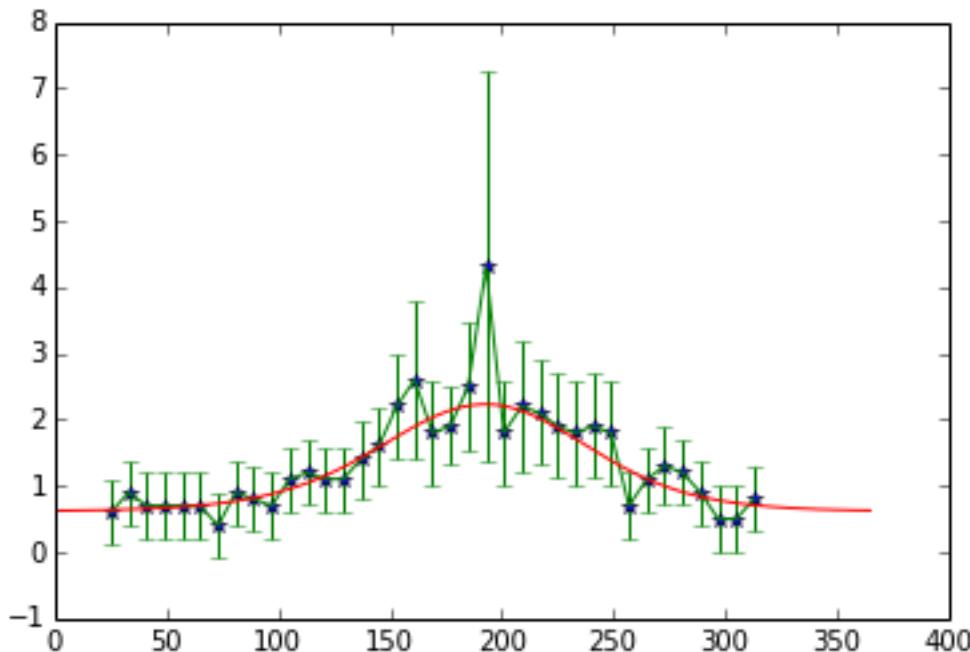
print psolve[1]
pp = psolve[0]
plt.plot(x,y,'*')
plt.errorbar(x,y,unc*1.96)
y_hat = dbl_logistic_model(pp,x_full)
plt.plot(x_full,y_hat)

print 'solved parameters: ',pp[0],pp[1],pp[2],pp[3],pp[4],pp[5]

# if we define the phenology as the parameter p[3]
# and the 'length' of the growing season:
print 'phenology',pp[3],pp[5]-pp[3]

initial parameters: 1.04703335612 1.62445180627 0.19 140.0 0.13 220.0
23.2882812239
solved parameters:  0.614031423522 2.92120123969 0.0357337165804 159.497584338 0.0381324783625 22
phenology 159.497584338 67.5051369225

```



```

# and run over each pixel ... this will take some time

# pixels that have some data
mask = (~data.mask).sum(axis=0)

pdata = np.zeros((7,) + mask.shape)

rows,cols = np.where(mask>0)
len_x = len(rows)

# lets just do some random ones to start with

```

```
#rows = rows[::10]
#cols = cols[::10]

len_x = len(rows)

for i in xrange(len_x):
    r,c = rows[i],cols[i]
    # progress bar
    if i%(len_x/40) == 0:
        print '... %4.2f percent' %(i*100./float(len_x))

    y = data[:,r,c]
    mask = ~y.mask
    y = np.array(y[mask])
    x = (np.arange(46)*8+1.) [mask]
    unc = np.array(sd[:,r,c] [mask])

    # need to get an initial estimate of the parameters

    # some stats on y
    ysd = np.std(y)
    ymean = np.mean(y)

    p[0] = ymean - 1.151*ysd;      # minimum (1.151 is 75% CI)
    p[1] = 2*1.151*ysd           # range
    p[2] = 0.19                   # related to up slope
    p[3] = 140                     # midpoint of up slope
    p[4] = 0.13                   # related to down slope
    p[5] = 220                     # midpoint of down slope

    # set factr to quite large number (relative error in solution)
    # as it'll take too long otherwise
    psolve = optimize.fmin_l_bfgs_b(sse,p,approx_grad=True,iprint=-1,
                                    args=(x,y,unc),bounds=bound,factr=1e12)

    pdata[:-1,rows[i],cols[i]] = psolve[0]
    pdata[-1,rows[i],cols[i]] = psolve[1] # sse

    ... 0.00 percent
    ... 2.50 percent
    ... 5.00 percent
    ... 7.50 percent
    ... 10.00 percent
    ... 12.50 percent
    ... 15.00 percent
    ... 17.50 percent
    ... 19.99 percent
    ... 22.49 percent
    ... 24.99 percent
    ... 27.49 percent
    ... 29.99 percent
    ... 32.49 percent
    ... 34.99 percent
    ... 37.49 percent
    ... 39.99 percent
    ... 42.49 percent
    ... 44.99 percent
    ... 47.49 percent
    ... 49.99 percent
    ... 52.49 percent
    ... 54.99 percent
```

```
... 57.49 percent
... 59.98 percent
... 62.48 percent
... 64.98 percent
... 67.48 percent
... 69.98 percent
... 72.48 percent
... 74.98 percent
... 77.48 percent
... 79.98 percent
... 82.48 percent
... 84.98 percent
... 87.48 percent
... 89.98 percent
... 92.48 percent
... 94.98 percent
... 97.47 percent
... 99.97 percent

plt.figure(figsize=(10,10))
plt.imshow(pdata[3],interpolation='none',vmin=137,vmax=141)
plt.title('green up day')
plt.colorbar()

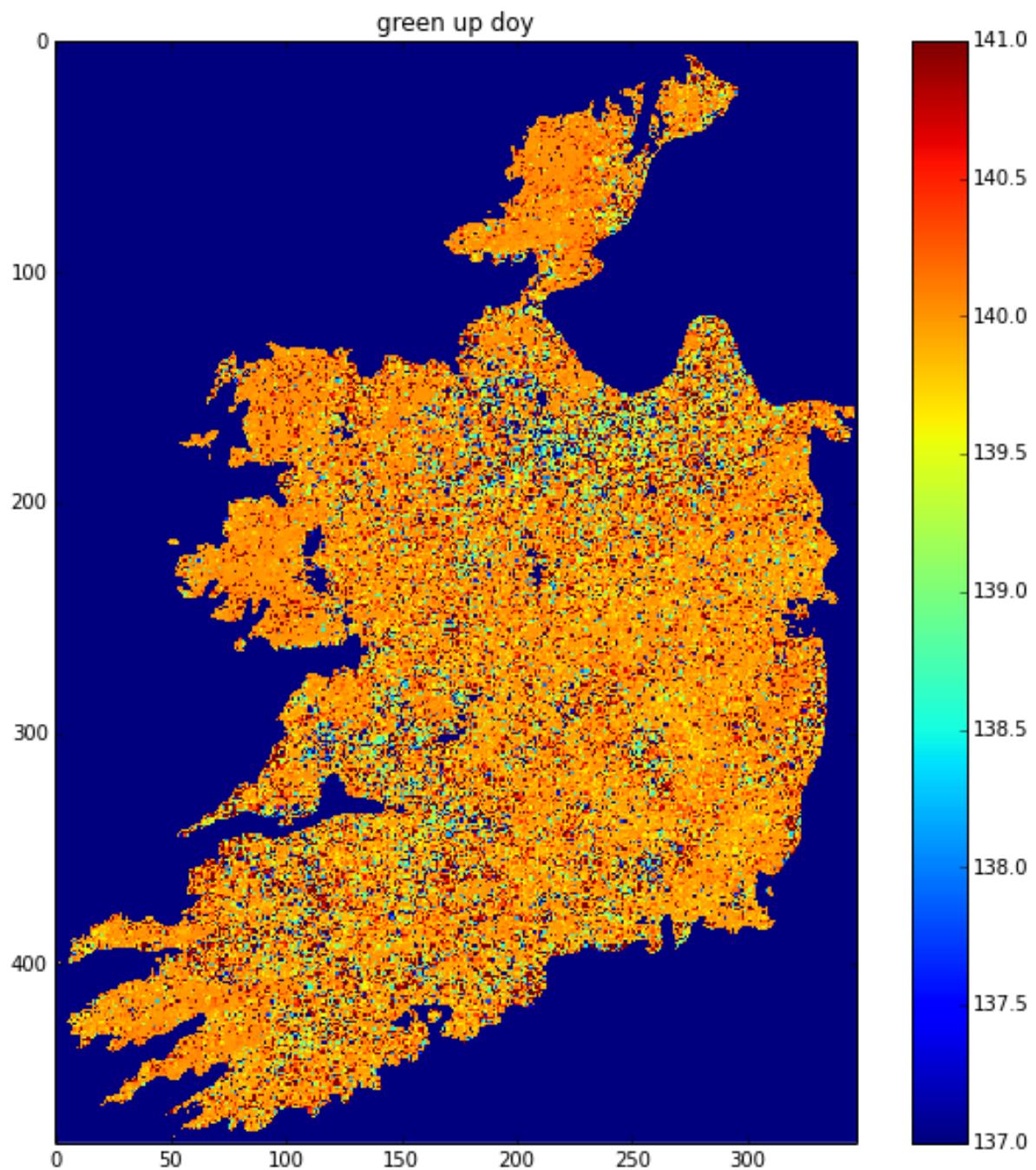
plt.figure(figsize=(10,10))
plt.imshow(pdata[5]-pdata[3],interpolation='none',vmin=74,vmax=84)
plt.title('season length')
plt.colorbar()

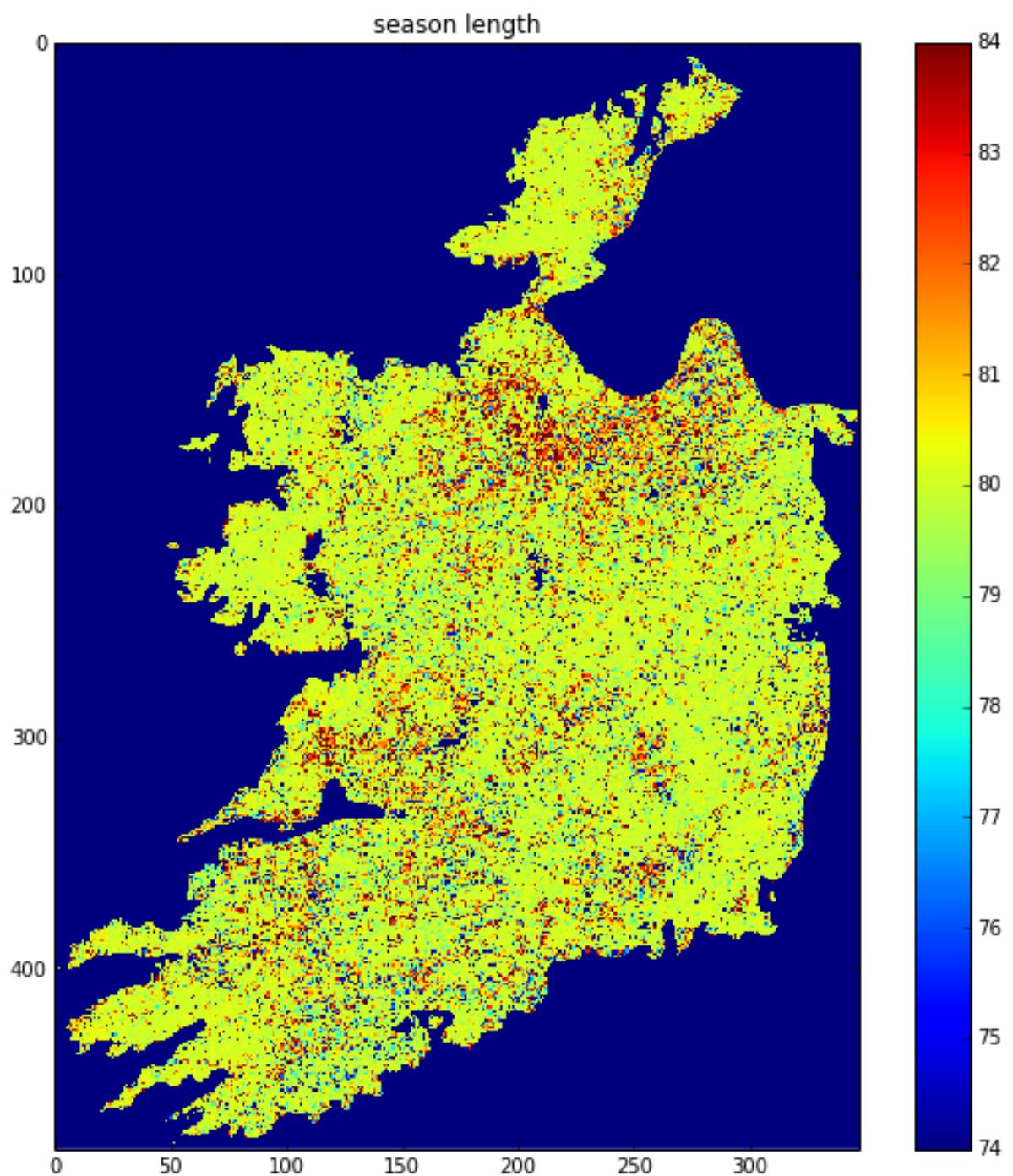
plt.figure(figsize=(10,10))
plt.imshow(pdata[0],interpolation='none',vmin=0.,vmax=6.)
plt.title('min LAI')
plt.colorbar()

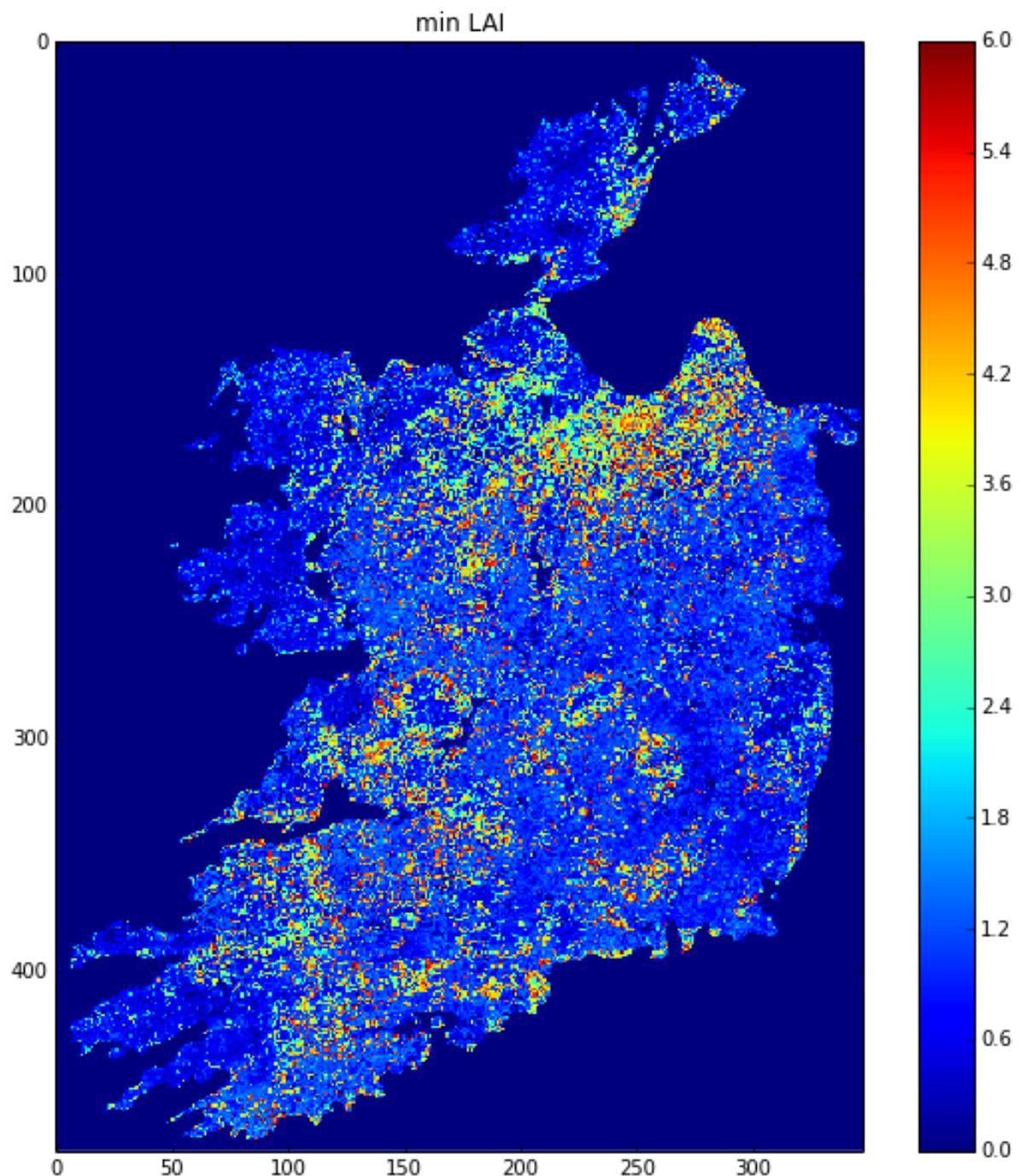
plt.figure(figsize=(10,10))
plt.imshow(pdata[1]+pdata[0],interpolation='none',vmin=0.,vmax=6.)
plt.title('max LAI')
plt.colorbar()

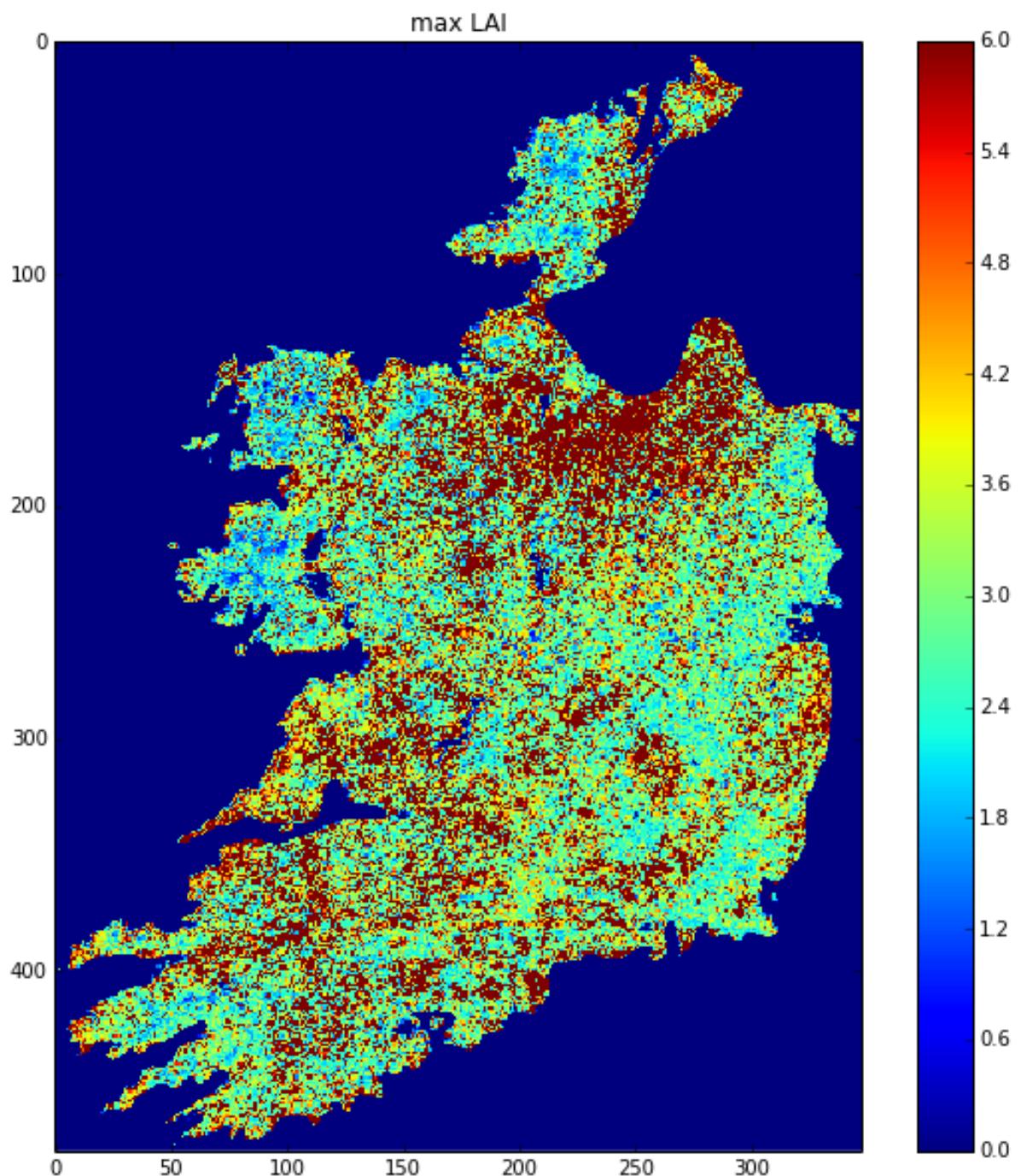
plt.figure(figsize=(10,10))
plt.imshow(np.sqrt(pdata[-1]),interpolation='none',vmax=np.sqrt(500))
plt.title('RSSE')
plt.colorbar()

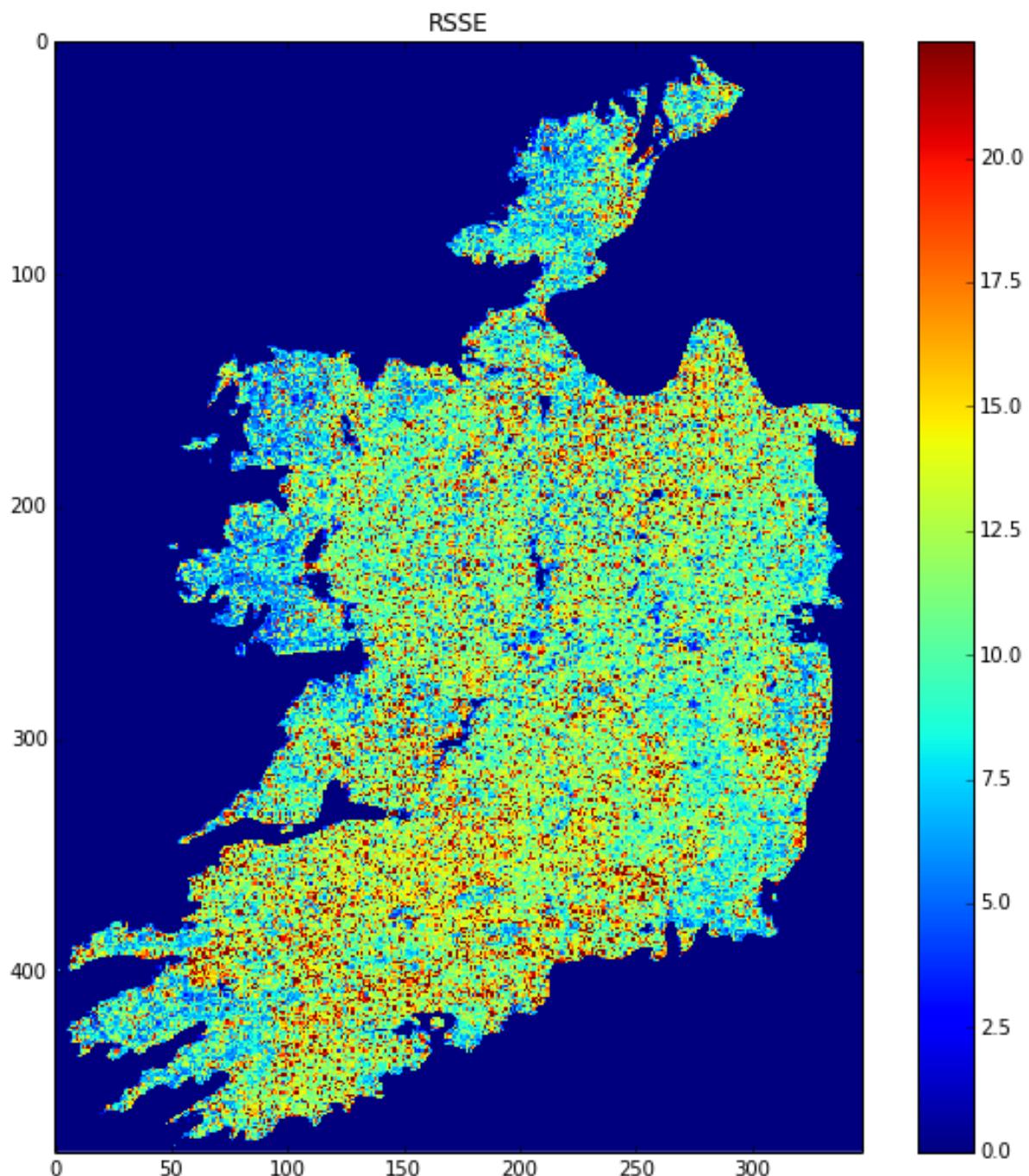
<matplotlib.colorbar.Colorbar instance at 0x2b75e5deefc8>
```











```
# check a few pixels
c = 200

for r in xrange(200,400,25):
    y = data[:,r,c]
    mask = ~y.mask
    y = np.array(y[mask])
    x = (np.arange(46)*8+1.)[mask]
    unc = np.array(sd[:,r,c][mask])

    x_full = np.arange(1,366)

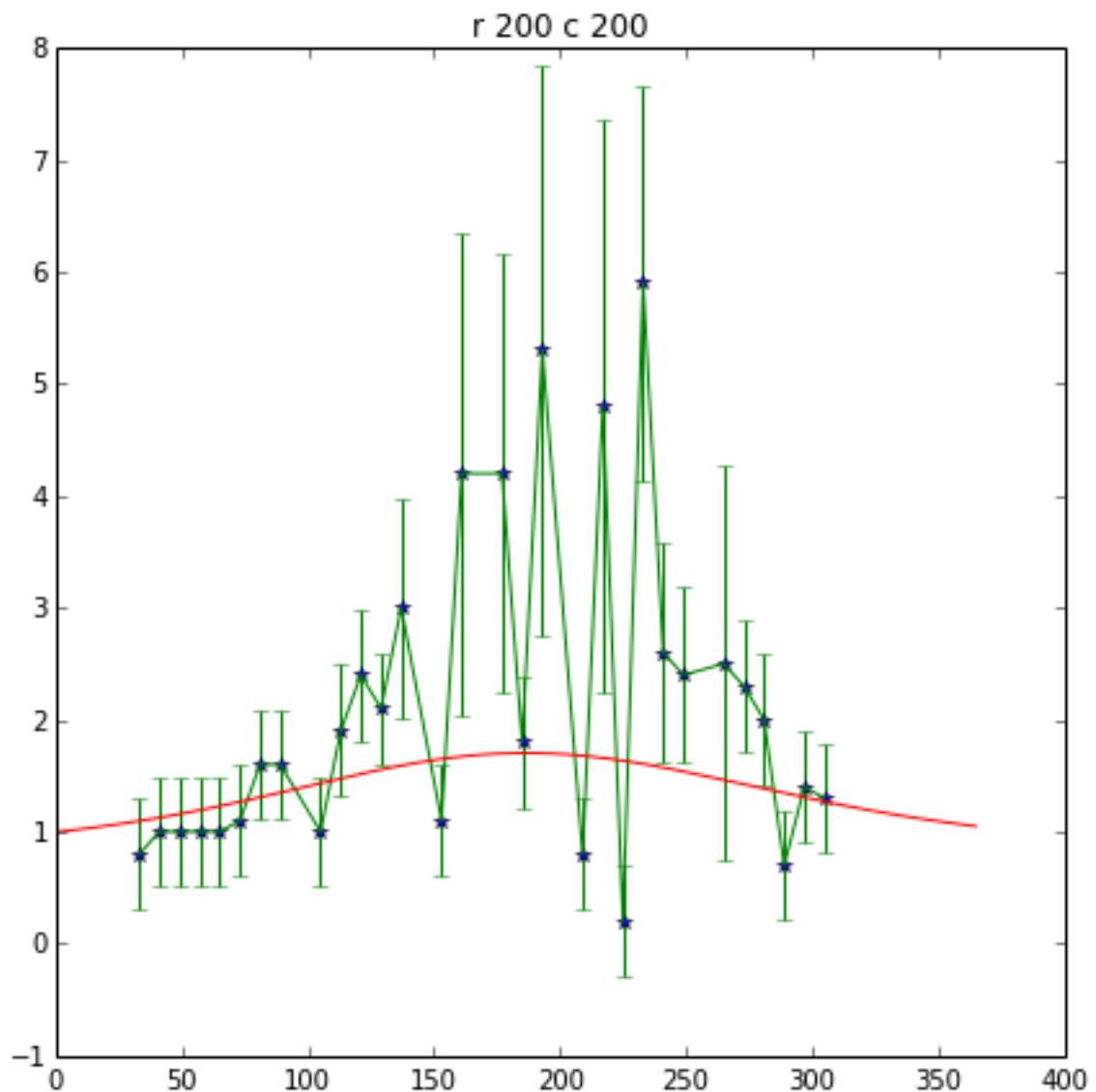
    # some default values for the parameters
    pp = pdata[:-1,r,c]
```

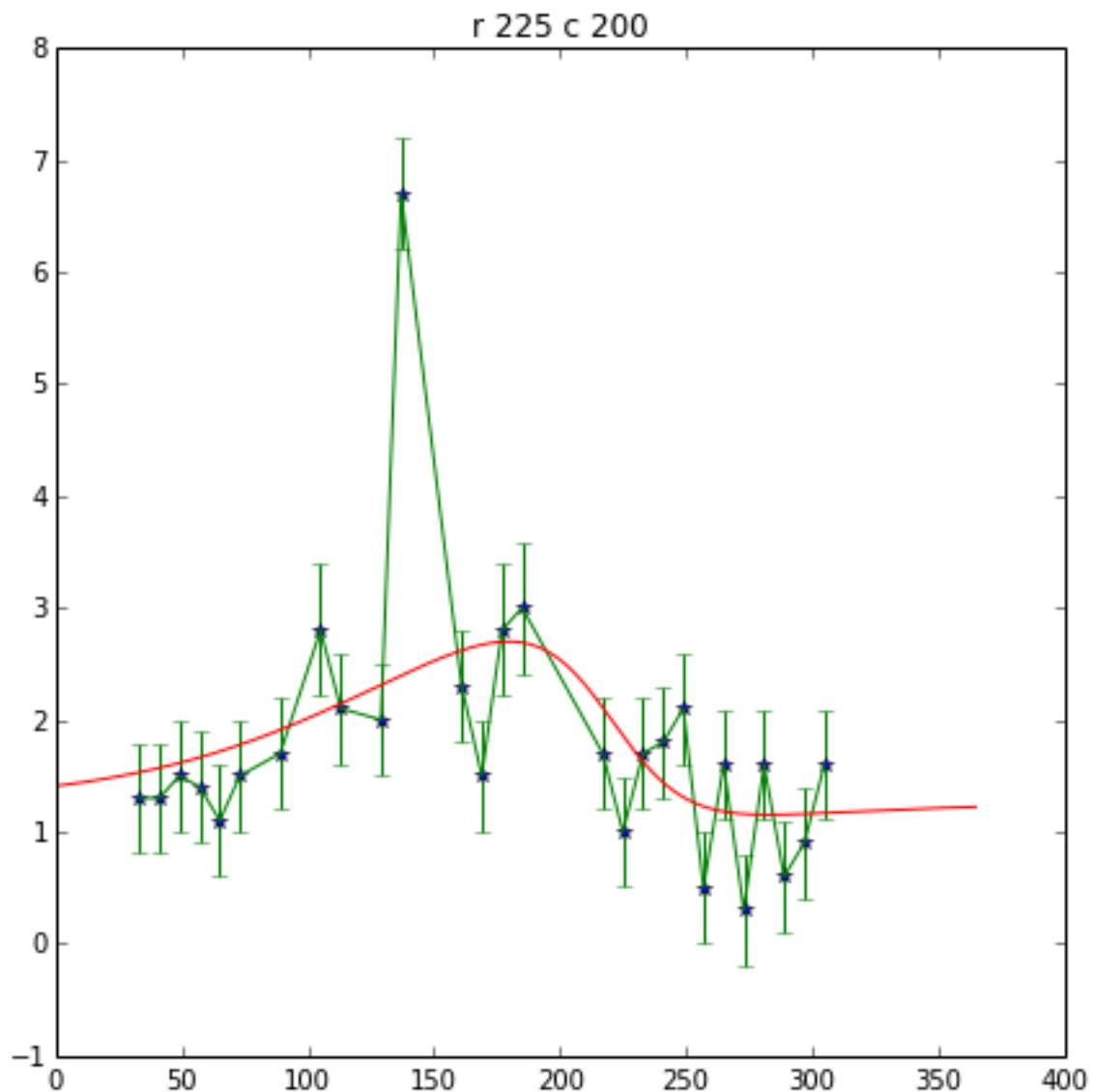
```
plt.figure(figsize=(7,7))
plt.title('r %d c %d'%(r,c))
plt.plot(x,y,'*')
plt.errorbar(x,y,unc*1.96)
y_hat = dbl_logistic_model(pp,x_full)
plt.plot(x_full,y_hat)

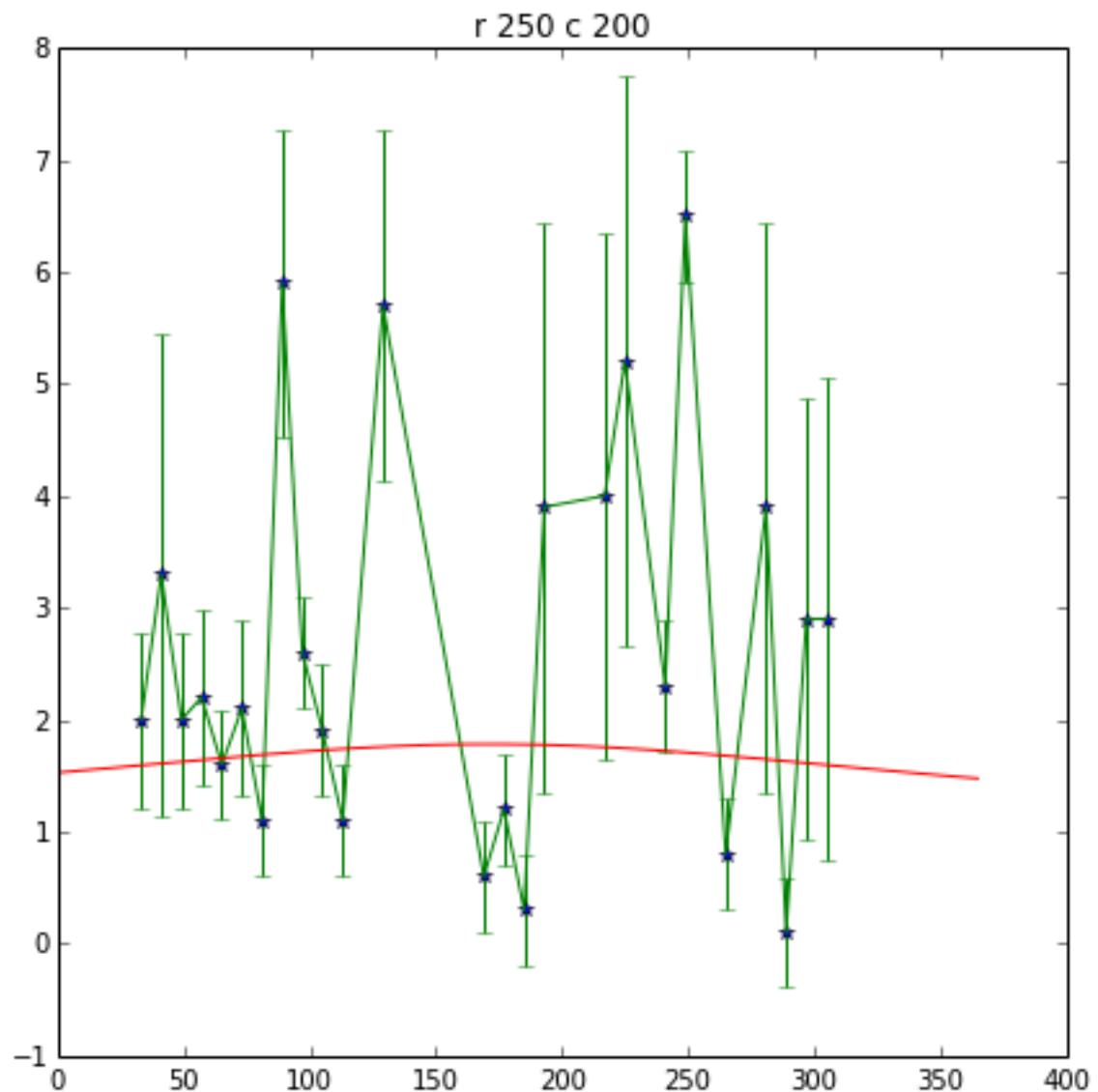
print 'solved parameters: ',pp[0],pp[1],pp[2],pp[3],pp[4],pp[5]

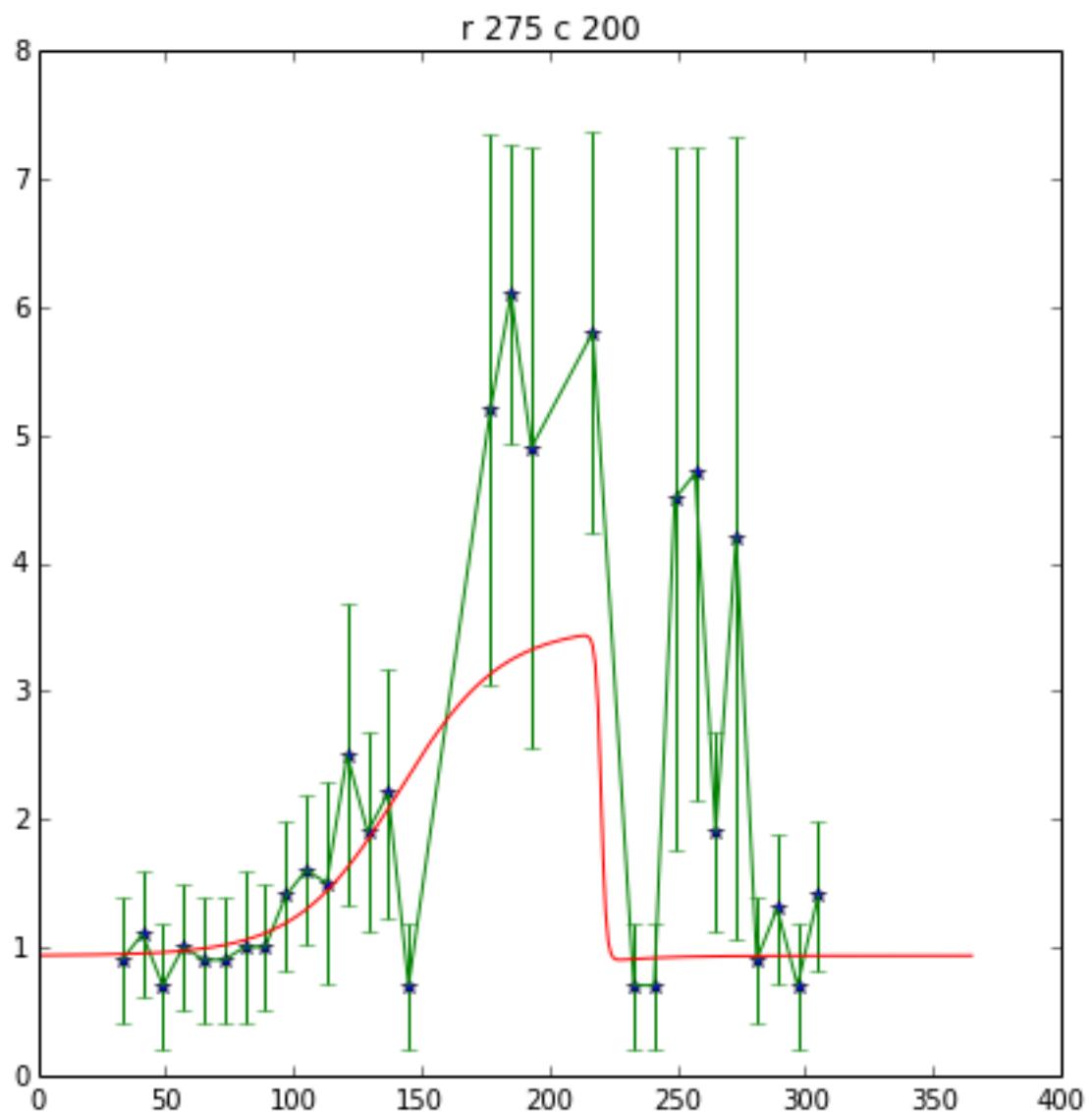
# if we define the phenology as the parameter p[3]
# and the 'length' of the growing season:
print 'phenology',pp[3],pp[5]-pp[3]

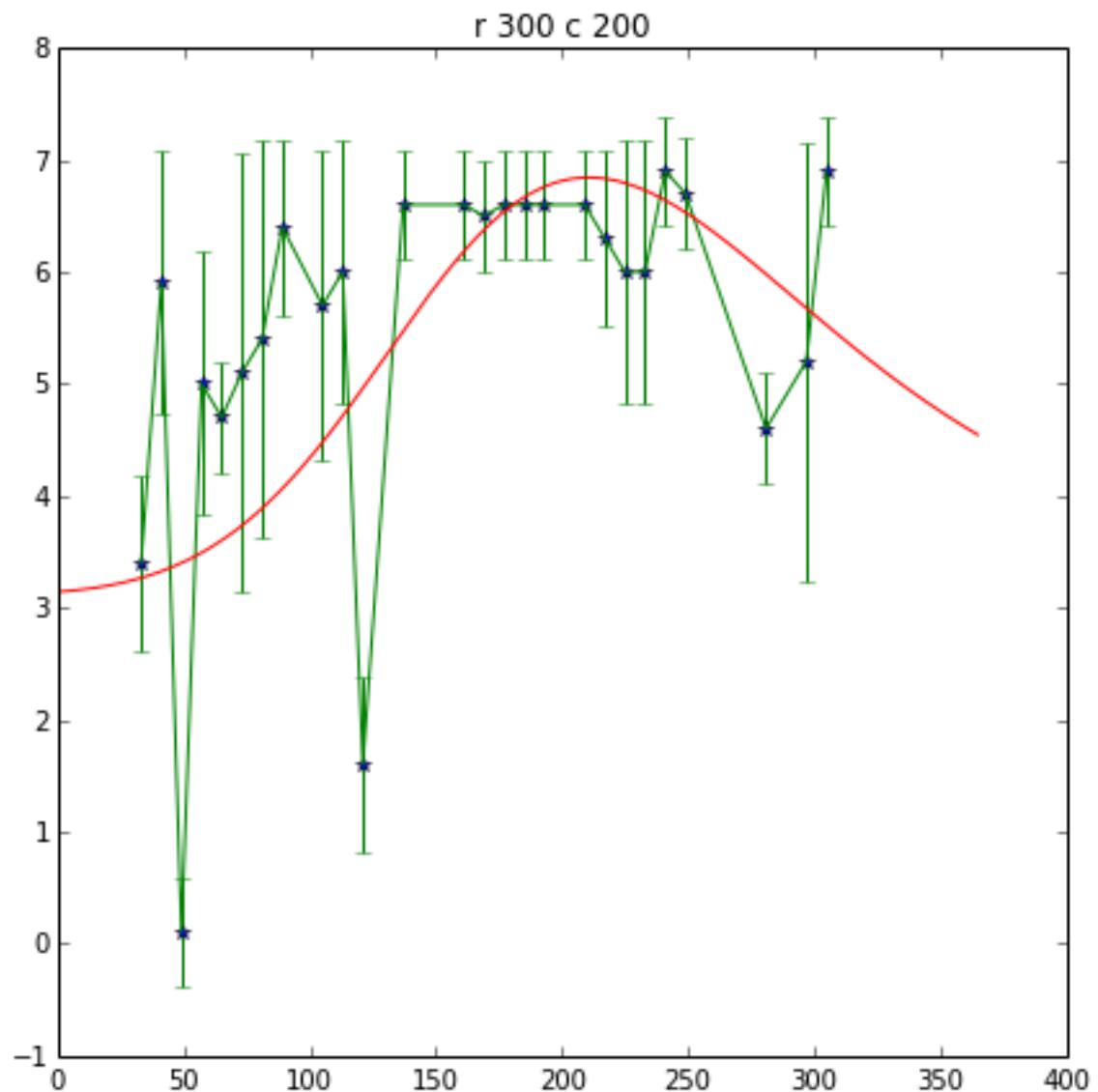
solved parameters:  0.837899998448 2.82427082241 0.0164922480786 139.933827392 0.015218312802 219
phenology 139.933827392 79.6862154345
solved parameters:  1.25111026954 2.3270241822 0.0190331997407 137.746147061 0.0650489530419 220.
phenology 137.746147061 82.3029586718
solved parameters:  1.24735745202 2.64641924245 0.01 139.923207754 0.0103960017819 220.21982824
phenology 139.923207754 80.2966204854
solved parameters:  0.932759824658 2.57001493023 0.0499114191497 141.274250409 1.0 220.241108243
phenology 141.274250409 78.9668578339
solved parameters:  3.3118926764 10.0 0.0227526645775 141.59060759 0.0129963184052 218.391152509
phenology 141.59060759 76.8005449187
solved parameters:  1.00690819411 2.5680266473 0.0244446244614 139.049623755 0.0196297496409 220.
phenology 139.049623755 81.1447880828
solved parameters:  1.41787972842 0.735065812424 0.183737093991 139.730989265 0.0642603310649 219
phenology 139.730989265 80.2444864894
solved parameters:  1.03836639365 1.58477670269 0.0296244335923 139.895942883 0.0436631925744 220
phenology 139.895942883 80.1405620773
```

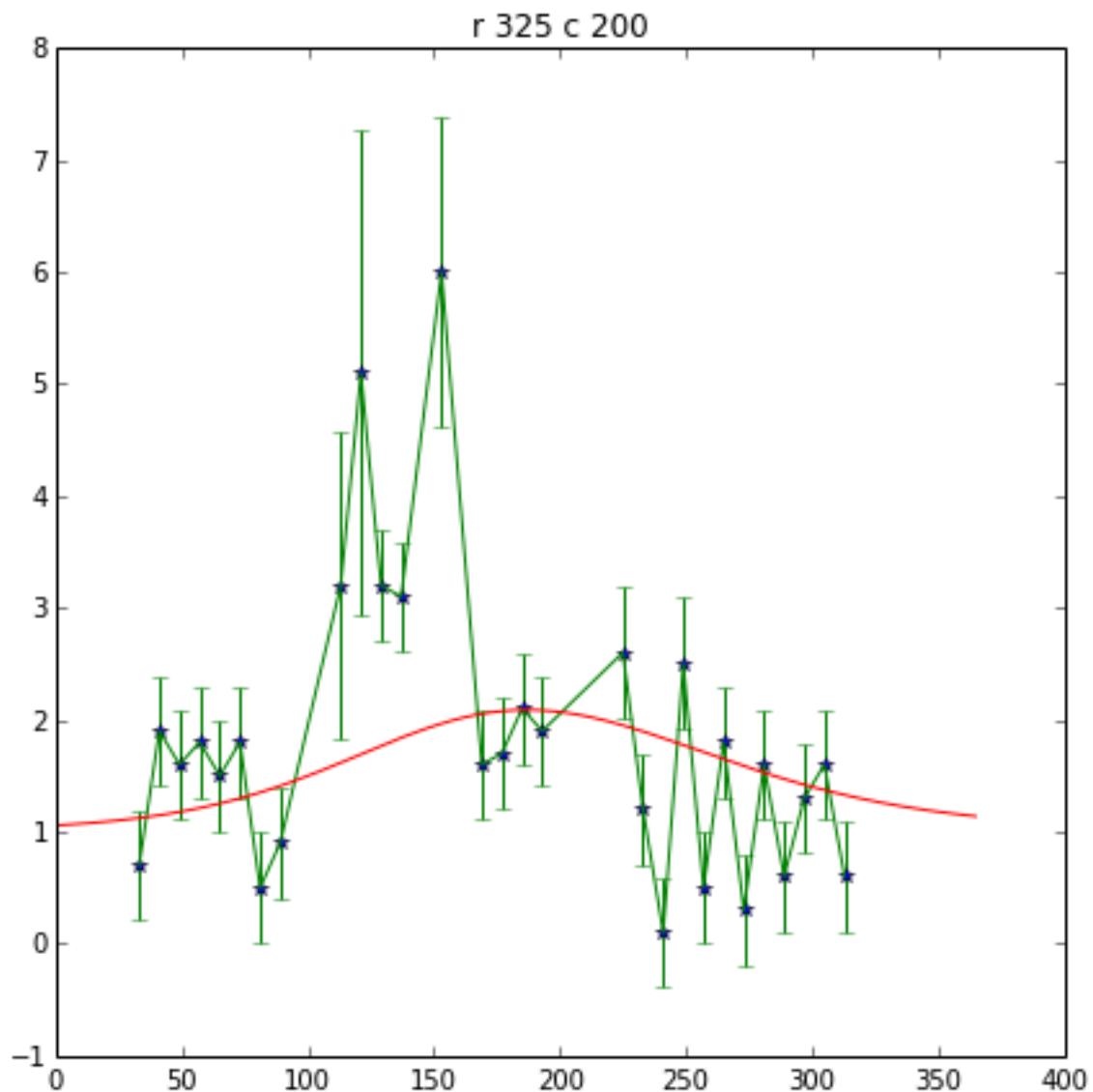


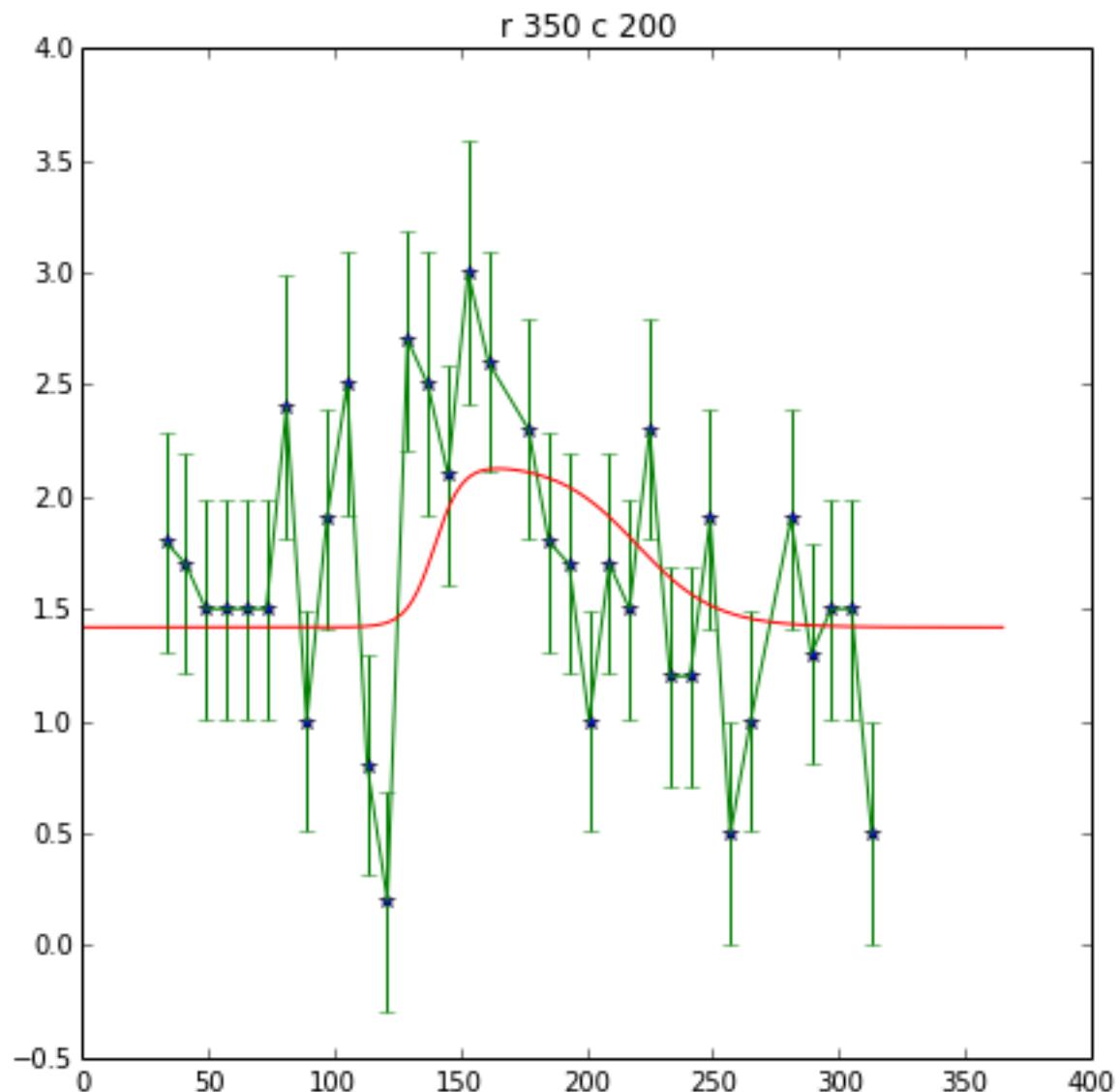


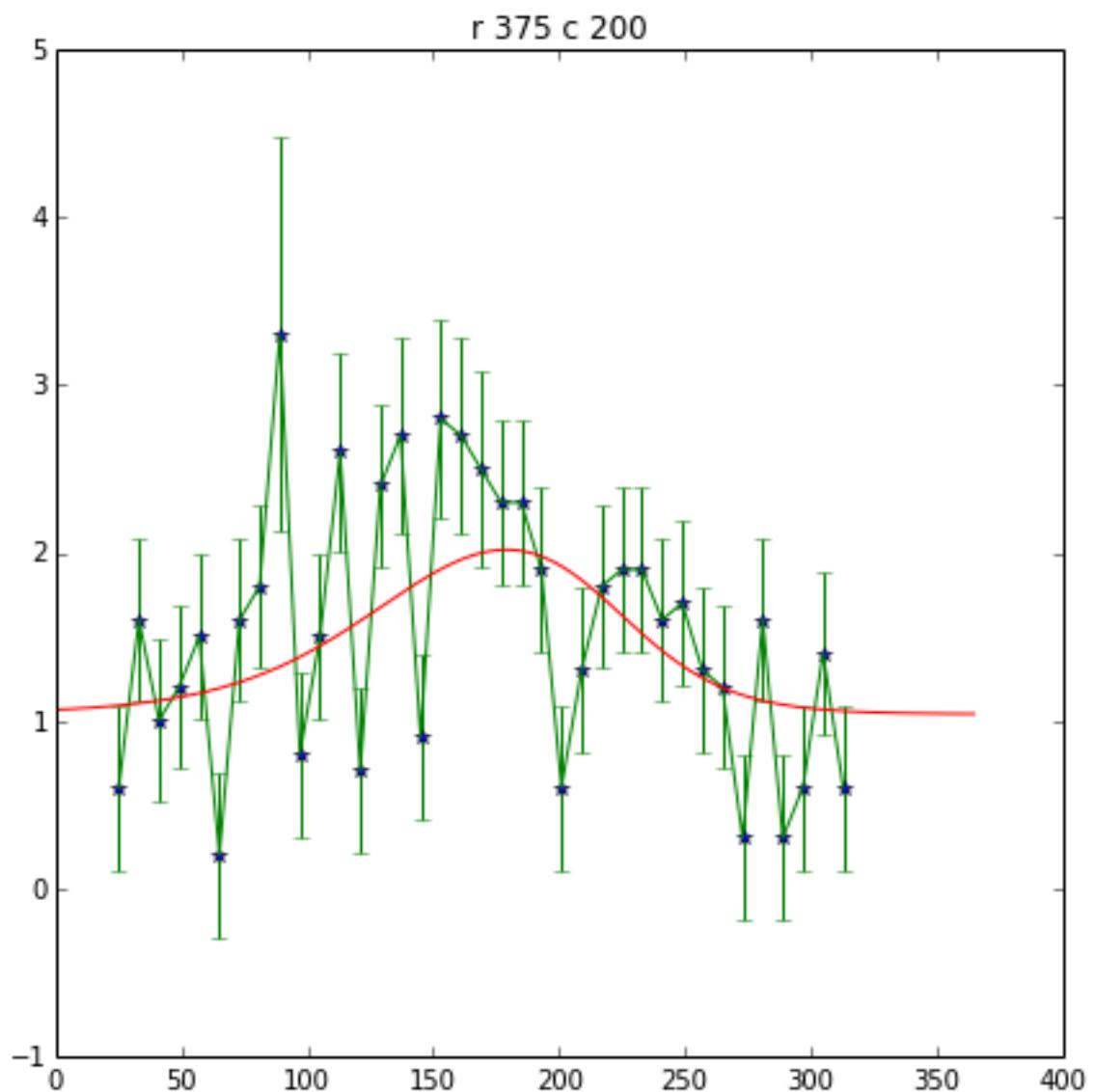












**CHAPTER
NINE**

ENVI

The purpose of this week's practical (Reading Week) is for you to become familiar with using the software package ENVI.

You should do one by following the [Classification](#) practical.

There is no direct supervision for this session.

You are expected to get on with this yourselves.

6A. ASSESSED PRACTICAL

10.1 6.1 Introduction

10.1.1 6.1.1 Site, Data and Task overview

These notes describe the practical you must submit for assessment in this course.

The practical comes in two parts: (1) data preparation; (2) modelling.

It is important that you complete both parts of this exercise, as you will need to make use of the code and results in the work you submit for assessment for this course.

- **Data Preparation**

The first task you must complete is to produce a dataset of the proportion of HUC catchment 13010001 (Rio Grande headwaters in Colorado, USA) that is covered by snow for **two consecutive years**, along with associated datasets on temperature (in C) and river discharge at the Del Norte monitoring station. You **may not** use data from the year 2005, as this is given to you in the illustrations above.

The dataset you produce must have a value for the mean snow cover, temperature and discharge in the catchment for every day over each year.

Your write up **must** include fully labelled graph(s) of snow cover, temperature and discharge for the catchment for each year (with units as appropriate), along with some summary statistics (e.g. mean or median, minimum, maximum, and the timing of these).

You **must** provide evidence of how you got these data (i.e. the code and commands you ran to produce the data).

- **Modelling**

You will have prepared two years of data above. Use one of these years to calibrate the (snowmelt) hydrological model (described below) and one year to test it.

The model parameter estimate *must* be objective (i.e. you can't just arbitrarily choose a set) and ideally optimal, and you *must* state the equation of the cost function you will try to minimise and explain the approach used.

You **must** state the values of the model parameters that you have estimated and show evidence for how you went about calculating them. Ideally, you should also state the uncertainty in these parameter estimates (not critical to pass this section though).

You **must** quantify the goodness of fit between your measured flow data and that produced by your model, both for the calibration exercise and the validation.

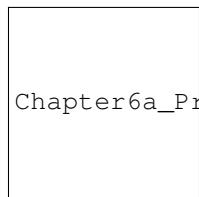
You **must** work individually on this task. If you do not, it will be treated as plagiarism. By reading these instructions for this exercise, we assume that you are aware of the UCL rules on plagiarism. You can find more information on this matter in your student handbook. If in doubt about what might constitute plagiarism, ask one of the course convenors.

What you are going to do is to build, calibrate and test a (snowmelt) hydrological model, driven by observations in the Rio Grande Headwaters in Colorado, USA.

You will need to process two years of data (N.B. *not* 2005 as that is given in the illustrations).

The purpose of the model is to describe the streamflow at the Del Norte measurement station, just on the edge of the catchment.

The average climate for Del Norte is:



Chapter6a_Practical/files/images/usco0103climatedelnorte

Further general information is available from various [websites](#), including [NOAA](#).

You can visualise the site [here](#).

First then, we should look at the streamflow data. These data are in the file delnorte.dat for the years 2000 to 2010 inclusive. You can get further data from <http://waterdata.usgs.gov> if you wish.

```
# load a pre-cooked version of the data for 2005 (NB -- Dont use this year!!!
# except perhaps for testing)

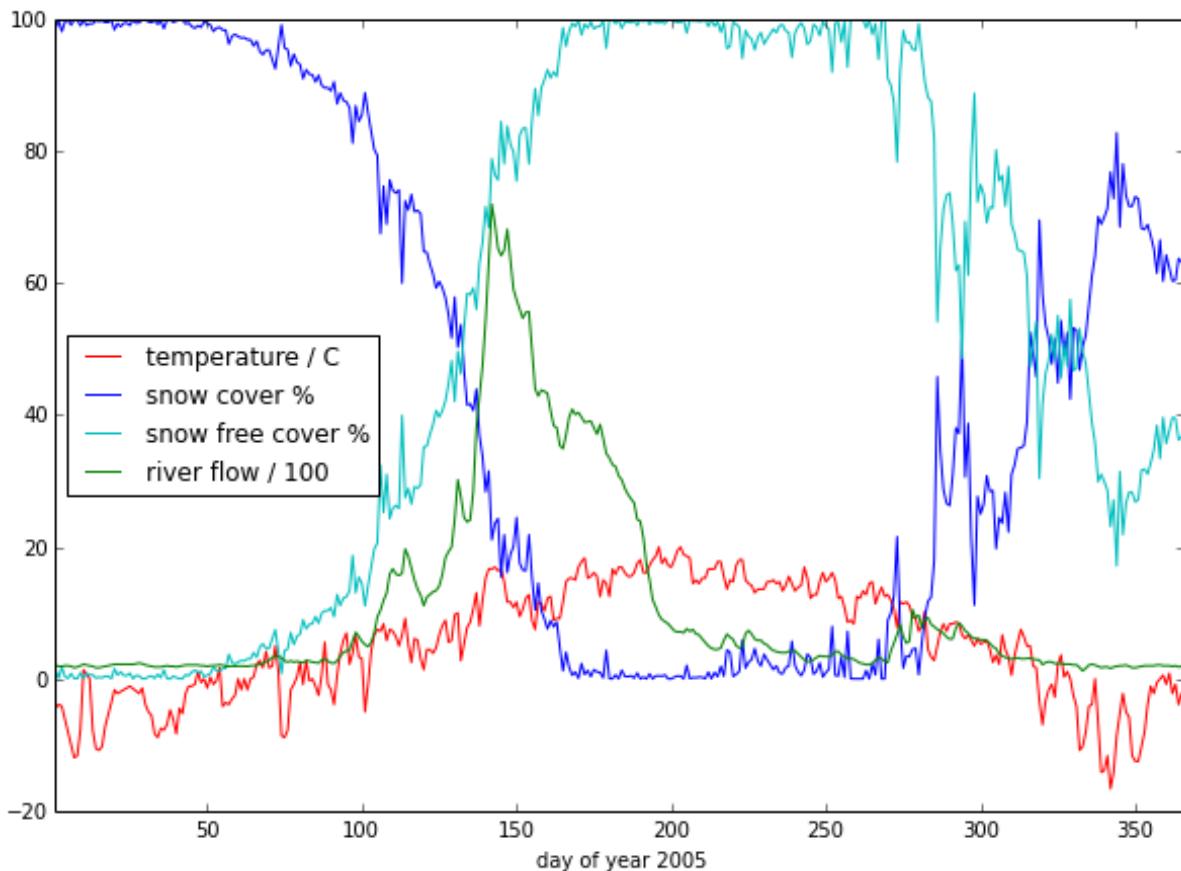
# load the data from a pickle file
import pickle
pkl_file = open('files/data/data.pkl', 'rb')
data = pickle.load(pkl_file)
pkl_file.close()

# set up plot
plt.figure(figsize=(10,7))
plt.xlim(data['doy'][0],data['doy'][-1]+1)
plt.xlabel('day of year 2005')

# plot data
plt.plot(data['doy'],data['temp'],'r',label='temperature / C')
plt.plot(data['doy'],data['snowprop']*100,'b',label='snow cover %')
plt.plot(data['doy'],100-data['snowprop']*100,'c',label='snow free cover %')
plt.plot(data['doy'],data['flow']/100.,'g',label='river flow / 100')

plt.legend(loc='best')

<matplotlib.legend.Legend at 0x1a2cae10>
```



we have plotted the streamflow (scaled) in green, the snow cover in blue, and the non snow cover in cyan and the temperature in red. It should be apparent that the hydrology is snow melt dominated, and to describe this (i.e. to build the simplest possible model) we can probably just apply some time lag function to the snow cover.

10.1.2 6.1.2 The Model

We will build a mass balance model, in terms of ‘snow water equivalent’:

The basis of a model is going to be something of the form:

$$SWE = k * \text{snowProportion}$$

where SWE is the ‘snow water equivalent’, the amount of snow in the entire snow pack in the catchment. snowProportion here then, is the proportion of snow cover in the catchment. We lump together density and volume terms into the coefficient k .

SWE then is the ‘mass’ (of water) that is available for melting on a particular day. We can obtain snowProportion from satellite data, so we only need the area / density term k , which we can suppose to be constant over time.

The simplest model of snowmelt is one where we assume that a proportion of this SWE is released (melted) as a function of temperature. In its simplest form, this is simply a temperature threshold:

```
meltDays = np.where(temperature > tempThresh) [0]
```

On these melt days then, we add $k * \text{snowProportion}$ of water into the system. For the present, we will ignore direct precipitation. So:

```
for d in meltDays: water = K * snowProportion[d]
```

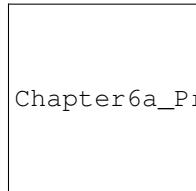
Now we have a mechanism to release snow melt into the catchment, but there will always be some delay in the water reaching the monitoring station from far away regions, compared to nearby areas. The function that describes this delay can be called a network response function. It is often modelled as a Laplace function (an

exponential). The idea is that if we have a ‘flash’ input to the catchment, this network response function will give us what we would measure as a hydrograph at the monitoring station (or elsewhere).

We can parameterise this with a decay factor, p, so that if the amount of water on day d is 1, the amount on day d+1 is p, on d+2, p^2 etc:

```
n = np.arange(len(snowProportion)) - d m = p ** n m[n<0] = 0
```

so here, m is the decay function:



Chapter6a_Practical/files/images/laplace.png

for day 150. This model will transfer a large amount of water of the peak day, then less and less as time goes by. So, a simple model then is of the form:

```
def model_accum(data,tempThresh,k,p):
    meltDays = np.where(data['temp'] > tempThresh) [0]
    accum = data['snowprop']*0.
    for d in meltDays:
        water = k * data['snowprop'][d]
        n = np.arange(len(data['snowprop'])) - d
        m = p ** n
        m[n<0] = 0
        accum += m * water
    return accum

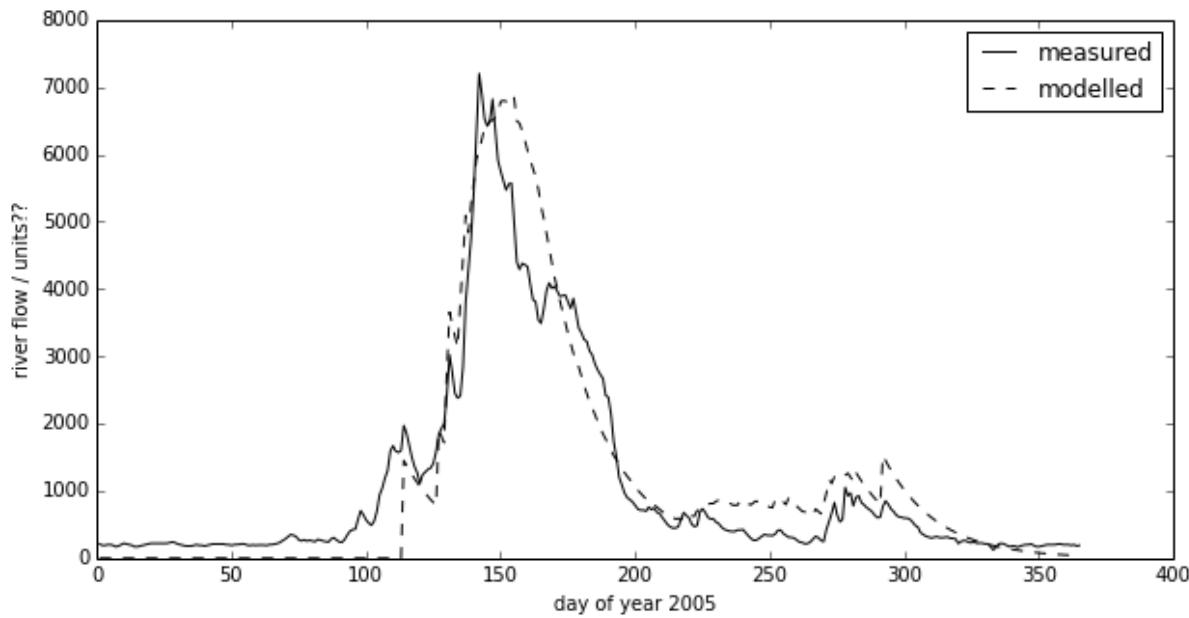
tempThresh = 8.5
k = 2000.0
p = 0.95

# test it
accum = model_accum(data,tempThresh,k,p)
```

This is a very simple model. It has three parameters (tempThresh, k, p) and is driven only by temperature and snow cover data. And yet, we see that even with a rough guess at what the parameters ought to be, we can get a reasonable match with the observed flow data:

```
plt.figure(figsize=(10,5))
plt.plot(data['doy'],data['flow'],'k',label='measured')
plt.plot(data['doy'],accum,'k--',label='modelled')
plt.ylabel('river flow / units??')
plt.xlabel('day of year 2005')
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x1c0b4ed0>
```



10.2 6.2 Data Preparation

10.2.1 6.2.1 Statement of the task

The first task you must complete is to produce a dataset of the proportion of HUC catchment 13010001 (Rio Grande headwaters) that is covered by snow for **two years**, along with associated datasets on temperature and river discharge at the Del Norte monitoring station.

The dataset you produce must have a value for the mean snow cover, temperature and discharge in the catchment for every day over each year.

Your write up **must** include a fully labelled graph of snow cover, temperature and discharge for the catchment for each year, along with some summary statistics (e.g. mean or median, minimum, maximum, and the timing of these).



You should aim to complete this task soon after Reading Week.

10.2.2 6.2.2 Some Advice

You would probably want to use a **daily** snow product for this task, such as that available from MODIS, so make sure you know what that is and explore the characteristics of the dataset.

You will notice from the figure above (the figure should give you some clue as to a suitable data product) that there will be areas of each image for which you have no information (described in the dataset QC). You will need to decide what to do about ‘missing data’. For instance, you might consider interpolating over missing values.

The simplest thing might be to produce a mean snow cover over what samples are available (ignoring the missing values). But whilst that may be sufficient to pass this section, it is far from ideal.

Whilst you only need to produce an average daily value for the catchment, a better approach would be to try to estimate snow cover for each pixel in the catchment (e.g. so you could do spatially explicit modelling with such data). I stress that this is not strictly necessary, but would be an interesting thing to do if you feel able.

However you decide to process the data, you must give a rationale for why you have taken the approach you have done.

You will notice that if you use MODIS data, you have access to both data from Terra (MOD10A) and Aqua (MYD10A), which potentially gives you two samples per day. Think about how to take that into account. Again, the simplest thing to do might be to just use one of these. That is likely to be sufficient, but it would be much better to include both datasets.

You should be able to hunt around to find the temperature and discharge data you want, but we take you through finding them in the advice below.

10.2.3 6.2.3 Data Advice

6.2.3.1 MODIS snow cover data

For MODIS data, you will need to work out which data product you want and how to download it. To help you with this, we have included urls of the MODIS Terra snow data product MOD10A1 and Aqua product MYD10A1 in the files files/data/robot_snow.????.txt:

```
!ls -l files/data/robot_snow.????.txt

-rw-rw-r-- 1 plewis plewis 9034752 Nov 6 09:12 files/data/robot_snow.2000.txt
-rw-rw-r-- 1 plewis plewis 10820568 Nov 6 09:12 files/data/robot_snow.2001.txt
-rw-rw-r-- 1 plewis plewis 16321938 Nov 6 09:12 files/data/robot_snow.2002.txt
-rw-rw-r-- 1 plewis plewis 22423068 Nov 6 09:12 files/data/robot_snow.2003.txt
-rw-rw-r-- 1 plewis plewis 7633374 Nov 6 09:12 files/data/robot_snow.2004.txt
-rw-rw-r-- 1 plewis plewis 18872448 Nov 6 09:12 files/data/robot_snow.2005.txt
-rw-rw-r-- 1 plewis plewis 11433078 Nov 6 09:12 files/data/robot_snow.2006.txt
-rw-rw-r-- 1 plewis plewis 22663686 Nov 6 09:12 files/data/robot_snow.2007.txt
-rw-rw-r-- 1 plewis plewis 22668990 Nov 6 09:12 files/data/robot_snow.2008.txt
-rw-rw-r-- 1 plewis plewis 22705317 Nov 6 09:12 files/data/robot_snow.2009.txt
-rw-rw-r-- 1 plewis plewis 22712370 Nov 6 09:13 files/data/robot_snow.2010.txt
-rw-rw-r-- 1 plewis plewis 17129166 Nov 6 09:13 files/data/robot_snow.2011.txt
-rw-rw-r-- 1 plewis plewis 22756824 Nov 6 09:13 files/data/robot_snow.2012.txt
-rw-rw-r-- 1 plewis plewis 18104898 Nov 6 09:13 files/data/robot_snow.2013.txt

!head -10 < files/data/robot_snow.2007.txt

ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h00v08.005.2008309053908
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h00v09.005.2008309053510
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h00v10.005.2008309053824
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v08.005.2008309053759
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v09.005.2008309053913
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v10.005.2008309053817
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v11.005.2008309053918
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h02v06.005.2008309053503
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h02v08.005.2008309053822
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h02v09.005.2008309054221
```

We can use the usual tools to explore the MODIS hdf files:

```
import gdal
target_vector_file = file
modis_file = 'files/data/MYD10A1.A2003026.h09v05.005.2008047035848.hdf'
g = gdal.Open(modis_file)
data_layer = 'MOD_Grid_Snow_500m:Fractional_Snow_Cover'

subdatasets = g.GetSubDatasets()
```

```

for fname, name in subdatasets:
    print name
    print "\t", fname

fname = 'HDF4_EOS:EOS_GRID:"%s":%s'%(modis_file,data_layer)
raster = gdal.Open(fname)

[2400x2400] Snow_Cover_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Spatial_QA MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Albedo_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m
[2400x2400] Fractional_Snow_Cover MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m

```

6.2.3.2 Boundary Data

Boundary data, such as catchments, might typically come as ESRI shapefiles or may be in other vector formats. There tends to be variable quality among different databases, but a reliable source for catchment data the USA is the [USGS](#). One set of catchments in the tile we have is the Rio Grande headwaters, which we can [see](#) has a HUC 8-digit code of 13010001. The full dataset is easily found from the [USGS](#) or locally. Literature and associated data concerning this area can be found [here](#). Associated GIS data are [here](#), including the watershed boundary data.

Data more specific to our particular catchment of interest can be found on the [Rio Grande Data Project pages](#).

You should download the file [Hydrologic_Units.zip](#) or get this locally. Obviously, you will need to unzip this file to get at the shapefile '`Hydrologic_Units/HUC_Polygons.shp`' within it.

You can explore the shape file with the following:

```

ogrinfo files/data/Hydrologic_Units/HUC_Polygons.shp HUC_Polygons | head -89 | tail -16

OGRFeature(HUC_Polygons):2
  HUC (Integer) = 13010001
  REG_NAME (String) = Rio Grande Region
  SUB_NAME (String) = Rio Grande Headwaters
  ACC_NAME (String) = Rio Grande Headwaters
  CAT_NAME (String) = Rio Grande Headwaters. Colorado.
  HUC2 (Integer) = 13
  HUC4 (Integer) = 1301
  HUC6 (Integer) = 130100
  REG (Integer) = 13
  SUB (Integer) = 1301
  ACC (Integer) = 130100
  CAT (Integer) = 13010001
  CAT_NUM (String) = 13010001
  Shape_Leng (Real) = 313605.66409400001
  Shape_Area (Real) = 3458016895.23000001907

```

This tells us that we want **HUC feature 2** (catchment 13010001).

We can produce a mask with `raster_mask`, but in this case, we need to use a function `raster_mask2`:

```

import sys
sys.path.insert(0,'files/python')

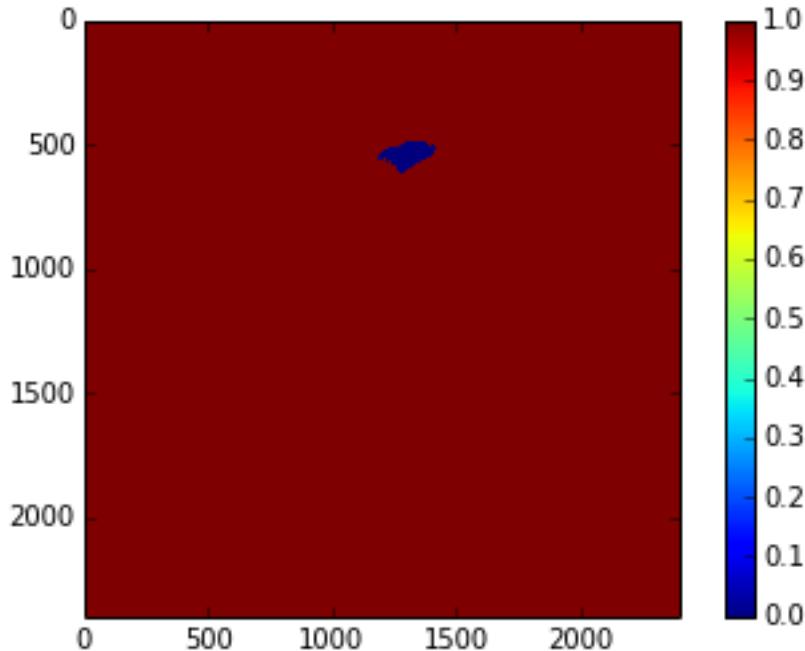
from raster_mask import *

m = raster_mask2(fname,\n                 target_vector_file="files/data/Hydrologic_Units/HUC_Polygons.shp",\\
                  attribute_filter=2)

```

```
plt.imshow(m)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2b6b7fcd6cb0>
```



The catchment is only a very small portion of the dataset, so you should make sure that you perform masking when you read the dataset in and only extract the area of data that you want.

6.2.3.3 Discharge Data

The river discharge data are in the file `files/data/delnorte.dat <files/data/delnorte.dat>`.

If you examine the file:

```
!head -35 < files/data/delnorte.dat

# ----- WARNING -----
# The data you have obtained from this automated U.S. Geological Survey database
# have not received Director's approval and as such are provisional and subject to
# revision. The data are released on the condition that neither the USGS nor the
# United States Government may be held liable for any damages resulting from its use.
# Additional info: http://waterdata.usgs.gov/nwis/help/?provisional
#
# File-format description: http://waterdata.usgs.gov/?tab_delimited_format_info
# Automated-retrieval info: http://waterdata.usgs.gov/nwis/?automated_retrieval_info
#
# Contact: gs-w_support_nwisweb@usgs.gov
# retrieved: 2011-09-30 09:35:31 EDT (caww02)
#
# Data for the following 1 site(s) are contained in this file
#   USGS 08220000 RIO GRANDE NEAR DEL NORTE, CO
#
#
# Data provided for site 08220000
#   DD parameter statistic Description
#   01 00060 00003 Discharge, cubic feet per second (Mean)
#
# Data-value qualification codes included in this output:
```

```

#      A Approved for publication -- Processing and review completed.
#      e Value has been estimated.
# agency_cd site_no datetime          01_00060_00003  01_00060_00003_cd
# 5s      15s    20d    14n   10s
USGS    08220000    2000-01-01     190      A:e
USGS    08220000    2000-01-02     170      A:e
USGS    08220000    2000-01-03     160      A:e
USGS    08220000    2000-01-04     160      A:e
USGS    08220000    2000-01-05     170      A:e
USGS    08220000    2000-01-06     180      A:e
USGS    08220000    2000-01-07     170      A:e
USGS    08220000    2000-01-08     190      A:e
USGS    08220000    2000-01-09     190      A:e

```

you will see comment lines that start with #, followed by data lines.

The easiest way to read these data would be to use:

```

file = 'files/data/delnorte.dat'
data = np.loadtxt(file, usecols=(2, 3), unpack=True, dtype=str)

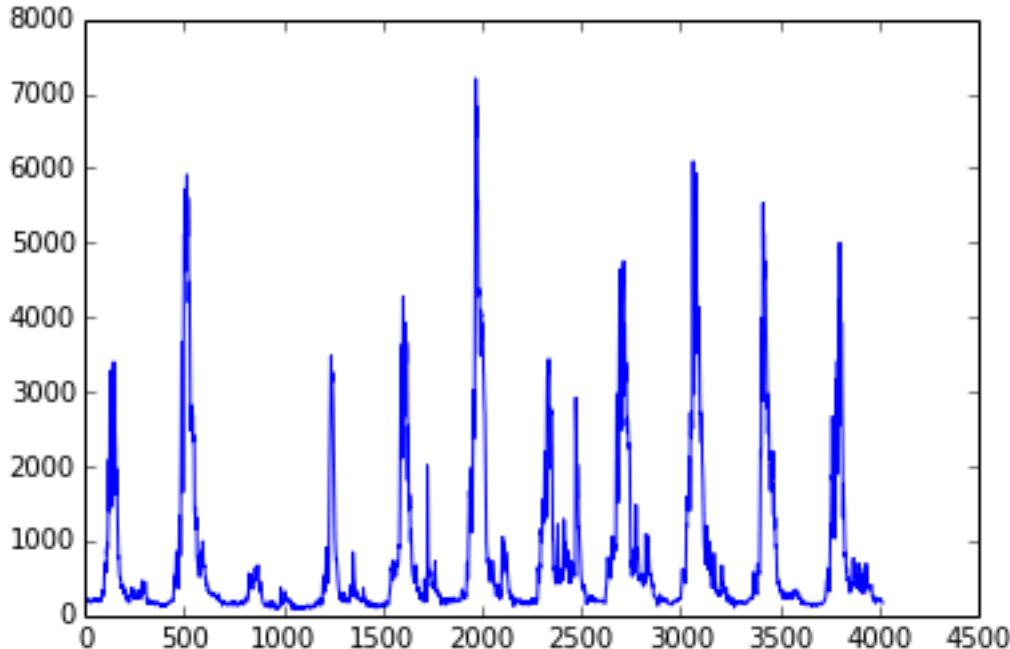
# so you have the dates in
data[0]

array(['2000-01-01', '2000-01-02', '2000-01-03', ..., '2010-12-29',
       '2010-12-30', '2010-12-31'],
      dtype='|S10')

# and the stream flow in data[1]
plt.plot(data[1].astype(float))

[<matplotlib.lines.Line2D at 0x2b6b7c573950>]

```



You will need to convert the date field (i.e. the data in `data[0]`) into the day of year.

This is readily accomplished using `datetime`:

```

import datetime
# transform the first one
ds = np.array(data[0][0].split('-')).astype(int)

```

```
print ds
year,doy = datetime.datetime(ds[0],ds[1],ds[2]).strftime('%Y %j').split()
print year,doy

[2000    1      1]
2000 001
```

6.2.3.4 Temperature data

We can directly access temperature data from [here](#).

The format of 'delNorteT.dat <files/data/delNorteT.dat>' is given here.

The first three fields are date fields (YEAR, MONTH and DAY), followed by TMAX, TMIN, PRCP, SNOW, SNDP.

You should read in the temperature data for the days and years that you want.

For temperature, you might take a **mean of TMAX and TMIN**.

Note that these are in Fahrenheit. You should convert them to Celcius.

Note also that there are missing data (values 9998 and 9999). You will need to filter these and interpolate the data in some way. A median might be a good approach, but any interpolation will suffice.

With that processing then, you should have a dataset, Temperature that will look something like (in cyan, for the year 2005):



10.3 6.3 Coursework

You need to submit your coursework in the usual manner by the usual submission date.

You **must** work individually on this task. If you do not, it will be treated as plagiarism. By reading these instructions for this exercise, we assume that you are aware of the UCL rules on plagiarism. You can find more information on this matter in your student handbook. If in doubt about what might constitute plagiarism, ask one of the course convenors.

10.3.1 6.3.1 Summary of coursework requirements

- **Data Preparation**

The first task you must complete is to produce a dataset of the proportion of HUC catchment 13010001 (Rio Grande headwaters in Colorado, USA) that is covered by snow for **two consecutive years**, along with associated datasets on temperature (in C) and river discharge at the Del Norte monitoring station.

You **may not** use data from the year 2005, as this is given to you in the illustrations above.

The dataset you produce must have a value for the mean snow cover, temperature and discharge in the catchment for every day over each year.

Your write up **must** include fully labelled graph(s) of snow cover, temperature and discharge for the catchment for each year (with units as appropriate), along with some summary statistics (e.g. mean or median, minimum, maximum, and the timing of these).

You **must** provide evidence of how you got these data (i.e. the code and commands you ran to produce the data).

- **Modelling**

You will have prepared two years of data above. Use one of these years to calibrate the (snowmelt) hydrological model (described below) and one year to test it.

The model parameter estimate *must* be objective (i.e. you can't just arbitrarily choose a set) and ideally optimal, and you *must* state the equation of the cost function you will try to minimise and explain the approach used.

You **must** state the values of the model parameters that you have estimated and show evidence for how you went about calculating them. Ideally, you should also state the uncertainty in these parameter estimates (not critical to pass this section though).

You **must** quantify the goodness of fit between your measured flow data and that produced by your model, both for the calibration exercise and the validation.

10.3.2 6.3.2 Summary of Advice

The first task involves pulling datasets from different sources. No individual part of that should be too difficult, but you must put this together from the material we have done so far. It is more a question of organisation then.

Perhaps think first about where you want to end up with on this (the ‘output’). This might for example be a dictionary with keys `temp`, `doy`, `snow` and `flow`, where each of these would be an array with 365 values (or 366 in a leap year).

Then consider the datasets you have: these are: (i) a stack of MODIS data with daily observations; (ii) temperature data in a file; (iii) flow data in a file.

It might be a little fiddly getting the data you want from the flow and temperature data files, but its not very complicated. You will need to consider flagging invalid observations and perhaps interpolating between these.

Processing the MODIS data might take a little more thought, but it is much the same process. Again, we read the datasets in, trying to make this efficient on data size by only using the area of the vector data mask as in a previous exercise. The data reading will be very similar to reading the MODIS LAI product, but you need to work out and implement what changes are necessary. As advised above you should use the `raster_mask2()` function for creating the spatial data masks. Again, you will need to interpolate or perhaps smooth between observations, and then process the snow cover proportions to get an average over the catchment.

The second task revolves around using the model that we have developed above in the function `model_accum()`. You have been through previous examples in Python where you attempt to estimate some model parameters given an initial estimate of the parameters and some cost function to be minimised. Solving the model calibration part of problem should follow those same lines then. Testing (validation) should be easy enough. Don’t forget to include the estimated parameters (and other relevant information, e.g. your initial estimate, uncertainties if available) in your write up.

There is quite a lot of data presentation here, and you need to provide *evidence* that you have done the task. Make sure you use images (e.g. of snow cover varying), graphs (e.g. modelled and predicted flow, etc.), and tables (e.g. model parameter estimates) throughout, as appropriate.

If, for some reason, you are unable to complete the first part of the practical, you should submit what you can for that first part, and continue with calibrating the model using the 2005 dataset that we used above. This would be far from ideal as you would not have completed the required elements for either part in that case, but it would generally be better than not submitting anything.

10.3.3 6.3.3 Further advice

There is plenty of scope here for going beyond the basic requirements, if you get time and are interested (and/or want a higher mark!).

You will be given credit for all additional work included in the write up, **once you have achieved the basic requirements**. So, there is no point (i.e. you will not get credit for) going off on all sorts of interesting lines of exploration here *unless* you have first completed the core task.

10.3.4 6.3.4 Structure of the Report

The required elements of the report are:

Introduction (5%)
Data Preparation (45%)
Modelling (45%)
Discussion/Conclusions (5%)

The figures in brackets indicate the percentage of marks that we will award for each section of the report.

Introduction (5%)

This should be of around 2-3 pages.

It should introduce the purpose of the study, being at a base level, ‘to build and calibrate a snow/hydrological model in Python’.

It should provide some background to building models of this sort (their purpose/role) and include some review of the types of models that might be built, with reference to the literature (journals).

A pass mark for this section will describe and explain the purpose of the study and examine some of the context to such modelling, with appropriate literature being cited. Higher or lower marks will depend on the depth that this goes into and the clarity of expression.

Data Preparation (45%)

This should contain around 3-4 pages of text, other than codes, figures and tables.

For a pass mark in this section, you must :

- introduce the study site, giving general site characteristics, with appropriate figures.
- provide an overview of the data used in the study (snow cover, temperature, flow data) and produce visualisations of the data you are using (images, graphs, tables as appropriate) alongside appropriate summary statistics.
- fully demonstrate how you got these data to this point of processing – i.e. submit appropriate Python codes and/or unix commands that when run in the sequence you describe would produce the data you have described.

The weighting here on the study site description is 5% and on the rest, 40%.

You can obtain higher marks here by going beyond the basics in your approaches to the data or modelling. You still need to demonstrate that you have done the core ‘pass’ material.

Modelling

This should contain around 3-4 pages of text, other than codes, figures and tables.

For a pass mark in this section, you must :

- provide an overview of the model that is constructed here, explaining the role of each parameter.
- explain the way in which model calibration and validation is to be undertaken.
- provide an overview of results of the calibration and validation along with relevant visualisations (images, graphs, tables as appropriate).

- fully demonstrate how you got these results – i.e. submit appropriate Python codes and/or unix commands that when run in the sequence you describe would produce the results you have described.

The weighting here is 5% for the model description and on the rest, 40%.

For a mildly improved mark, you should examine and discuss the model assumptions in the context of any modelling literature you looked at in the Introduction. For a significantly improved mark, you could try improving the model.

Discussion and Conclusions

This section should be around 2 pages. It should provide a discussion and analysis of your results and you should draw appropriate conclusions from these.

You can also use this section to critique the model/data/methods, and suggest ways that you would improve things. If you do this, you must give some indication of how that would be achieved. You will get no credit for simply saying ‘next time I would make the code more efficient’, for example.

Very good/excellent marks would normally require you to cite appropriate literature.

A sufficient effort for a pass would make a reasonable effort at discussing these results in the context of some literature and draw a few (non trivial) conclusions from the study.

10.3.5 6.3.5 Computer Code

General requirements

You will obviously need to submit computer codes as part of this assessment. Some flexibility in the style of these codes is to be expected. For example, some might write a class that encompasses the functionality for all tasks. Some people might have multiple versions of codes with different functionality. All of these, and other reasonable variations are allowed.

All codes needed to demonstrate that you have performed the core tasks are required to be included in the submission. You should include all codes that you make use of in the main body of the text in the main body. Any other codes that you want to refer to (e.g. something you tried out as an enhancement and didn’t quite get there) you can include in appendices.

All codes should be well-commented. Part of the marks you get for code will depend on the adequacy of the commenting.

Degree of original work required and plagiarism

If you use a piece of code verbatim that you have taken from the course pages or any other source, **you must acknowledge this** in comments in your text. **Not to do so is plagiarism.** Where you have taken some part (e.g. a few lines) of someone else’s code, **you should also indicate this.** If some of your code is heavily based on code from elsewhere, **you must also indicate that.**

Some examples. You may recognise this snippet of code from above.

The first example is guilty of strong plagiarism, it does not seek to acknowledge the source of this code, even though it is just a direct copy, pasted into a method called `model()`:

```
def model(tempThresh=9.0, K=2000.0, p=0.96):
    '''need to comment this further ...
    '''

    import numpy as np
    meltDays = np.where(temperature > tempThresh)[0]
    accum = snowProportion*0.
    for d in meltDays:
        water = K * snowProportion[d]
```

```

n = np.arange(len(snowProportion)) - d
m = p ** n
m[np.where(n<0)] = 0
accum += m * water
return accum

```

This is **not** acceptable.

This should probably be something along the lines of:

```

def model(tempThresh=9.0, K=2000.0, p=0.96):
    '''need to comment this further ...

    This code is taken directly from
    "Modelling delay in a hydrological network"
    by P. Lewis http://www2.geog.ucl.ac.uk/~plewis/geogg122/DelNorte.html
    and wrapped into a method.
    '''

    # my code: make sure numpy is imported
    import numpy as np

    # code below verbatim from Lewis
    meltDays = np.where(temperature > tempThresh)[0]
    accum = snowProportion*0.
    for d in meltDays:
        water = K * snowProportion[d]
        n = np.arange(len(snowProportion)) - d
        m = p ** n
        m[np.where(n<0)] = 0
        accum += m * water
    # my code: return accumulator
    return accum

```

Now, we acknowledge that this is in essence a direct copy of someone else's code, and clearly state this. We do also show that we have added some new lines to the code, and that we have wrapped this into a method.

In the next example, we have seen that the way m is generated is in fact rather inefficient, and have re-structured the code. It is partially developed from the original code, and acknowledges this:

```

def model(tempThresh=9.0, K=2000.0, p=0.96):
    '''need to comment this further ...

    This code after the model developed in
    "Modelling delay in a hydrological network"
    by P. Lewis
    http://www2.geog.ucl.ac.uk/~plewis/geogg122/DelNorte.html

    My modifications have been to make the filtering more efficient.
    '''

    # my code: make sure numpy is imported
    import numpy as np

    # code below verbatim from Lewis unless otherwise indicated
    meltDays = np.where(temperature > tempThresh)[0]
    accum = snowProportion*0.

    # my code: pull the filter block out of the loop
    n = np.arange(len(snowProportion))
    m = p ** n

    for d in meltDays:
        water = K * snowProportion[d]

    # my code: shift the filter on by one day

```

```
# ...do something clever to shift it on by one day

    accum += m * water
# my code: return accumulator
return accum
```

This example makes it clear that significant modifications have been made to the code structure (and probably to its efficiency) although the basic model and looping comes from an existing piece of code. It clearly highlights what the actual modifications have been. Note that this is not a working example!!

Although you are supposed to do this piece of work on your own, there might be some circumstances under which someone has significantly helped you to develop the code (e.g. written the main part of it for you & you've just copied that with some minor modifications). You **must** acknowledge in your code comments if this has happened. On the whole though, this should not occur, as you **must** complete this work on your own.

If you take a piece of code from somewhere else and all you do is change the variable names and/or other cosmetic changes, you **must** acknowledge the source of the original code (with a URL if available).

Plagiarism in coding is a tricky issue. One reason for that is that often the best way to learn something like this is to find an example that someone else has written and adapt that to your purposes. Equally, if someone has written some tool/library to do what you want to do, it would generally not be worthwhile for you to write your own but to concentrate on using that to achieve something new. Even in general code writing (i.e. when not submitting it as part of your assessment) you and anyone else who ever has to read your code would find it of value to make reference to where you found the material to base what you did on. The key issue to bear in mind in this work, as it is submitted ‘as your own work’ is that, to avoid being accused of plagiarism and to allow a fair assessment of what you have done, you must clearly acknowledge which parts of it are your own, and the degree to which you could claim them to be your own.

For example, based on ... is absolutely fine, and you would certainly be given credit for what you have done. In many circumstances ‘taken verbatim from ...’ would also be fine (provided it is acknowledged) but then you would be given credit for what you had done with the code that you had taken from elsewhere (e.g. you find some elegant way of doing the graphs that someone has written and you make use of it for presenting your results).

The difference between what you submit here and the code you might write if this were not a piece submitted for assessment is that you the vast majority of the credit you will gain for the code will be based on the degree to which you demonstrate that you can write code to achieve the required tasks. There would obviously be some credit for taking codes from the coursenotes and bolting them together into something that achieves the overall aim: provided that worked, and you had commented it adequately and acknowledge what the extent of your efforts had been, you should be able to achieve a pass in that component of the work. If there was no original input other than bolting pieces of existing code together though, you be unlikely to achieve more than a pass. If you get less than a pass in another component of the coursework, that then puts you in danger of an overall fail.

Provided you achieve the core tasks, the more original work that you do/show (that is of good quality), the higher the mark you will get. Once you have achieved the core tasks, even if you try something and don't quite achieve it, it is probably worth including, as you may get marks for what you have done (or that fact that it was a good or interesting thing to try to do).

Documentation

Note: All methods/functions and classes must be documented for the code to be adequate. Generally, this will contain:

- some text on the purpose of the method (/function/class)
- some text describing the inputs and outputs, including reference to any relevant details such as datatype, shape etc where such things are of relevance to understanding the code.
- some text on keywords, e.g.:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.
```

Keyword arguments:

```
real -- the real part (default 0.0)
imag -- the imaginary part (default 0.0)
```

```
Example taken verbatim from:
http://www.python.org/dev/peps/pep-0257/
"""
if imag == 0.0 and real == 0.0: return complex_zero
```

You should look at the document on good docstring conventions when considering how to document methods, classes etc.

To demonstrate your documentation, you **must** include the help text generated by your code after you include the code. e.g.:

```
def print_something(this, stderr=False):
    '''This does something.

    Keyword arguments:
    stderr -- set to True to print to stderr (default False)
    '''

    if stderr:
        # import sys.stderr
        from sys import stderr

        # print to stderr channel, converting this to str
        print >> stderr, str(this)

        # job done, return
        return

    # print to stdout, converting this to str
    print str(this)

    return
```

Then the help text would be:

```
help(print_something)

Help on function print_something in module __main__:

print_something(this, stderr=False)
    This does something.

    Keyword arguments:
    stderr -- set to True to print to stderr (default False)
```

The above example represents a ‘good’ level of commenting as the code broadly adheres to the style suggestions and most of the major features are covered. It is not quite ‘very good/excellent’ as the description of the purpose of the method (rather important) is trivial and it fails to describe the input this in any way. An excellent piece would do all of these things, and might well tell us about any dependencies (e.g. requires sys if stderr set to True).

An inadequate example would be:

```
def print_something(this, stderr=False):
    '''This prints something'''
    if stderr:
        from sys import stderr
        print >> stderr, str(this)
        return
    print str(this)
```

It is inadequate because it still only has a trivial description of the purpose of the method, it tells us nothing about inputs/outputs and there is no commenting inside the method.

Word limit

There is no word limit per se on the computer codes, though as with all writing, you should try to be succinct rather than overly verbose.

Code style

A good to excellent piece of code would take into account issues raised in the [style guide](#). The ‘degree of excellence’ would depend on how well you take those points on board.

7.0 FIRE/ENSO TELECONNECTIONS

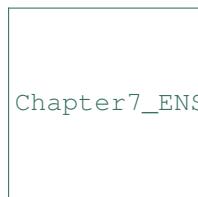
11.1 7.1 Introduction

There is much public and scientific interest in monitoring and predicting the activity of wildfires and such topics are often in the media.

Part of this interest stems from the role fire plays in issues such as land cover change, deforestation and forest degradation and Carbon emissions from the land surface to the atmosphere, but also of concern are human health impacts. The impacts of fire should not however be considered as wholly negative, as it plays a significant role in natural ecosystem processes.

For many regions of the Earth, there are large inter-annual variations in the timing, frequency and severity of wildfires. Whilst anthropogenic activity accounts for a large and probably increasing proportion of fires started, this is not in itself a new phenomenon.

Fires spread where: (i) there is an ignition source (lightning or man, mostly); (ii) sufficient combustible fuel to maintain the fire. The latter is strongly dependent on fuel loads and moisture content, as well as meteorological conditions. Generally then, when conditions are drier (and there is sufficient fuel and appropriate weather conditions), we would expect fire spread to increase. If the number of ignitions remained approximately constant, this would mean more fires. Many models of fire activity predict increases in fire frequency in the coming decades, although there may well be different behaviours in different parts of the world.



Chapter7_ENSO/files/images/492949main_Figure-2-Wildfires_s3.jpg

Satellite data has been able to provide us with increasingly useful tools for monitoring wildfire activity, particularly since 2000 with the MODIS instruments on the NASA Terra and Aqua (2002) satellites. A suite of ‘fire’ products have been generated from these data that have been used in a large number of publications and practical/management projects.

There is growing evidence of ‘teleconnection’ links between fire occurrence and large scale climate patterns, such as ENSO.

The proposed mechanisms are essentially that such climatic patterns are linked to local water status and temperature and thus affect the ability of fires to spread. For some regions of the Earth, empirical models built from such considerations have quite reasonable predictive skill, meaning that fire season severity might be predicted some months ahead of time.

11.2 7.2 A Practical Exercise

11.2.1 7.2.1 In This Session

In this session, you will be working in groups (of 3 or 4) to build a computer code in python to explore links between fire activity and Sea Surface Temperature anomalies.

This is a team exercise, but does not form part of your formal assessment for this course. You should be able to complete the exercise in the 3 hour session, if you work effectively as a team. Staff will be on hand to provide pointers.

You should be able to complete the exercise using coding skills and python modules that you have previously experience of, though we will also provide some pointers to get you started.

11.2.2 7.2.2 Statement of the problem

Using monthly fire count data from MODIS Terra, develop and test a predictive model for the number of fires per unit area per year driven by Sea Surface Temperature anomaly data.

11.2.3 7.2.3 Datasets

We suggest that the datasets you use of this analysis, following Chen et al. (2011), are:

- MODIS Terra fire counts (2001-2011) (MOD14CMH). The particular dataset you will want from the file is ‘SUBDATASET_2 [360x720] CloudCorrFirePix (16-bit integer)’.
- Climate index data from NOAA

If you ever wish to take this study further, you can find various other useful datasets such as these.

Fire Data

The MOD14CMH CMG data are available from the [UMD ftp server](#) but the data you will need are also directly available from /data/geospatial_10/ucfajlg/MOD14CMH/. Note that, if you are on the UCL system, you do not need to copy the data, just use them from where they are.

If for any reason, you *did* want to copy or update them, use the following unix command:

```
wget 'ftp://fire:burnt@fuoco.geog.umd.edu/modis/C5/cmg/monthly/hdf/*'
```

The data are in files/data and are in HDF format, so you should know how to read them into a numpy array in python.

```
!ls -l files/data/*hdf | head -10
```

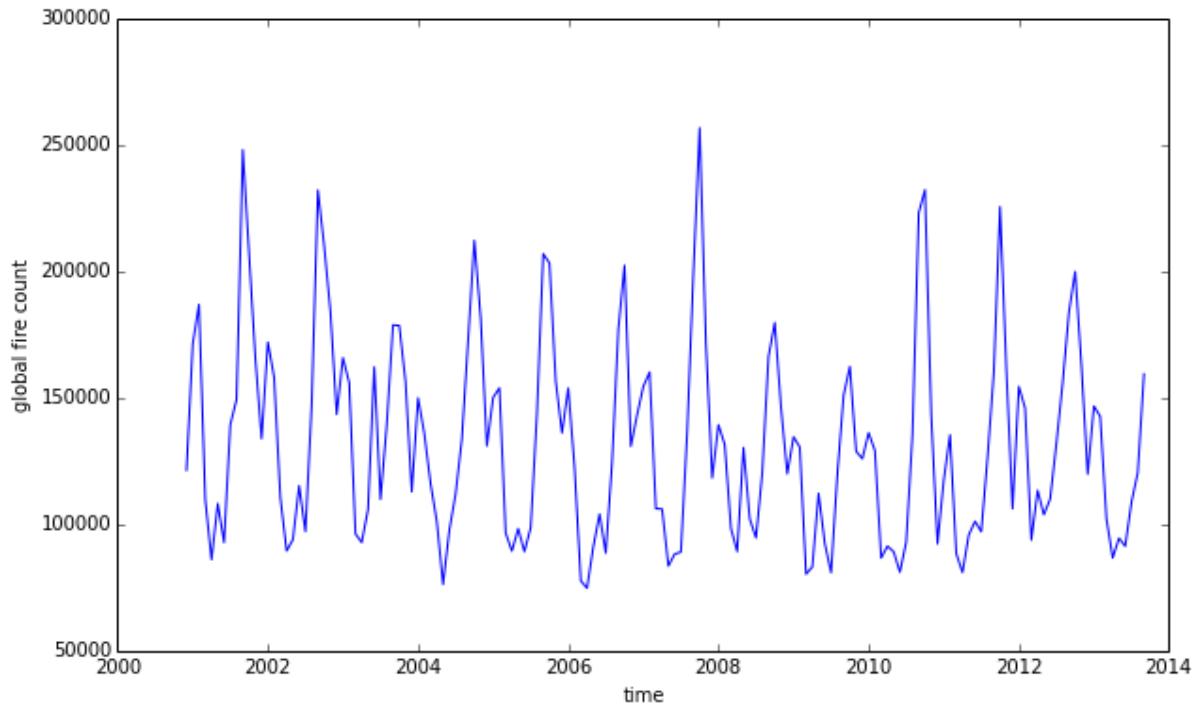
```
-rw-rw-r-- 1 plewis plewis 961993 Nov  1 2007 files/data/MOD14CMH.200011.005.01.hdf
-rw-rw-r-- 1 plewis plewis 923238 Nov  1 2007 files/data/MOD14CMH.200012.005.01.hdf
-rw-rw-r-- 1 plewis plewis 953227 Mar 22 2008 files/data/MOD14CMH.200101.005.01.hdf
-rw-rw-r-- 1 plewis plewis 951917 Apr 23 2008 files/data/MOD14CMH.200102.005.01.hdf
-rw-rw-r-- 1 plewis plewis 959282 Apr 23 2008 files/data/MOD14CMH.200103.005.01.hdf
-rw-rw-r-- 1 plewis plewis 943794 Apr 23 2008 files/data/MOD14CMH.200104.005.01.hdf
-rw-rw-r-- 1 plewis plewis 988588 Apr 23 2008 files/data/MOD14CMH.200105.005.01.hdf
-rw-rw-r-- 1 plewis plewis 898137 Mar 22 2008 files/data/MOD14CMH.200106.005.01.hdf
-rw-rw-r-- 1 plewis plewis 962767 Mar 22 2008 files/data/MOD14CMH.200107.005.01.hdf
-rw-rw-r-- 1 plewis plewis 982616 Nov  5 2007 files/data/MOD14CMH.200108.005.01.hdf
ls: write error: Broken pipe
```

If you are **really** stuck on reading the data, or just want to move on to the next parts, you can use ‘files/python/reader.py <files/python/reader.py>’ which will create a masked array in data, and an array of years (year) and months (month):

```
run files/python/reader

plt.figure(figsize=(10, 6))
x = year + month/12.
y = np.sum(data, axis=(1, 2))
plt.plot(x, y)
plt.ylabel('global fire count')
plt.xlabel('time')

<matplotlib.text.Text at 0x2b4488089490>
```



This dataset is at 0.5 degree resolution and we want to perform tha analysis as 5 degrees.

We need to shrink the dataset by a factor of 10 then.

There are different ways to achive this, but one way would be to reorganise the data:

```
rdata = [data[:, i::10, j::10] for i in xrange(10) for j in xrange(10)]
rdata = ma.array(rdata)

print rdata.shape
```

(100, 154, 36, 72)

So, we have made the dataset which as (154, 360, 720) into a shape (100, 154, 36, 72).

We can now get the total fire counts easily at 5 degrees by summing over those 100 cells (axis=0):

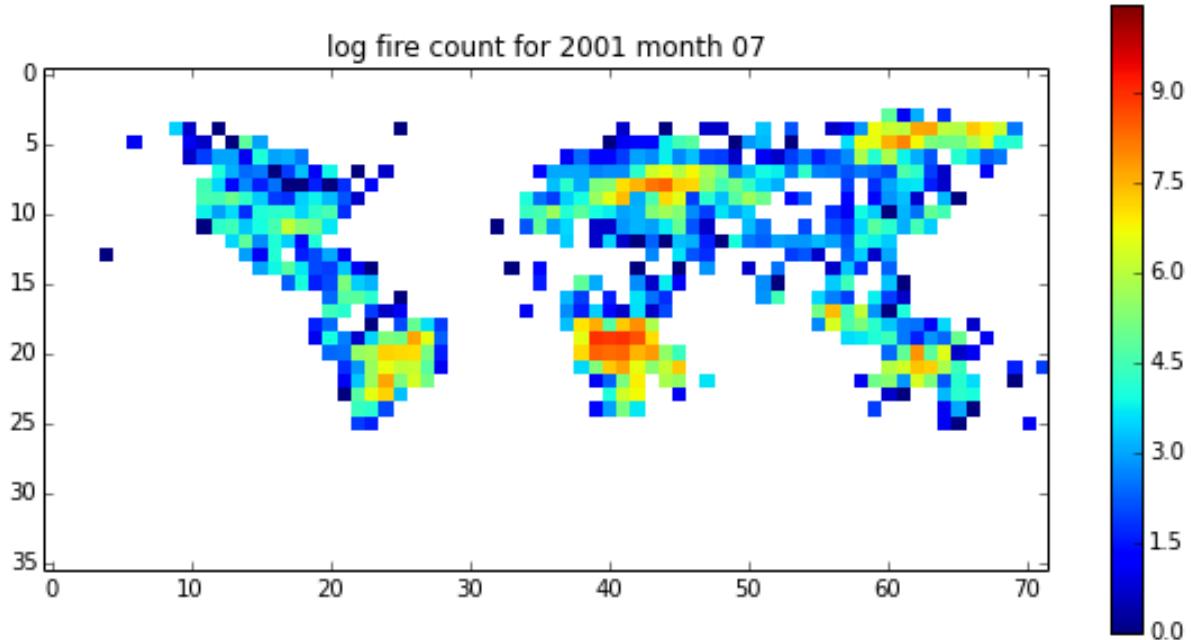
```
fdata = rdata.sum(axis=0)
print fdata.shape

lf = np.log(fdata)
vmax = np.max(lf[lf>0])

plt.figure(figsize=(10, 5))
plt.imshow(lf[8], interpolation='nearest', vmax=vmax)
plt.colorbar()
plt.title('log fire count for %d month %02d' % (year[8], month[8]))
```

(154, 36, 72)

<matplotlib.text.Text at 0x1e19add0>



```
# or even make a movie
lf = np.log(fdata)
vmax = np.max(lf[lf>0])

root = 'files/images/'
for i in xrange(lf.shape[0]):
    fig = plt.figure(figsize=(10,5))
    plt.imshow(np.log(fdata[i]), interpolation='nearest', vmax=vmax)
    plt.colorbar()
    file_id = '%d month %02d'%(year[i],month[i])
    plt.title('log fire count for %s'%file_id)
    plt.savefig('%s_%s.jpg'%(root,file_id.replace(' ', '_')))
    plt.close(fig)

-c:2: RuntimeWarning: divide by zero encountered in log
-c:2: RuntimeWarning: invalid value encountered in log
-c:8: RuntimeWarning: divide by zero encountered in log
-c:8: RuntimeWarning: invalid value encountered in log

cmd = 'convert -delay 100 -loop 0 {0}_*month*.jpg {0}fire_movie3.gif'.format(root)
os.system(cmd)

0
```

The information we want is the peak fire count and to know which month this occurred in.

To do this, we might reorder the data first:

```
nlatlon = fdata.shape[1:]
min_year = year[0]
max_year = year[-1]
# number of years
nyears = max_year - min_year + 1

# set up a big array
```

```

f2data = np.zeros((12,nyears)+nlatlon)
f2datam = np.ones((12,nyears)+nlatlon).astype(bool)

for i,(y,m) in enumerate(zip(year-year[0],month-1)):
    f2data[m,y] = fdata[i]
    f2datam[m,y] = (fdata[i] <= 0)
# mask it
f2data = ma.array(f2data,mask=f2datam)
print f2data.shape

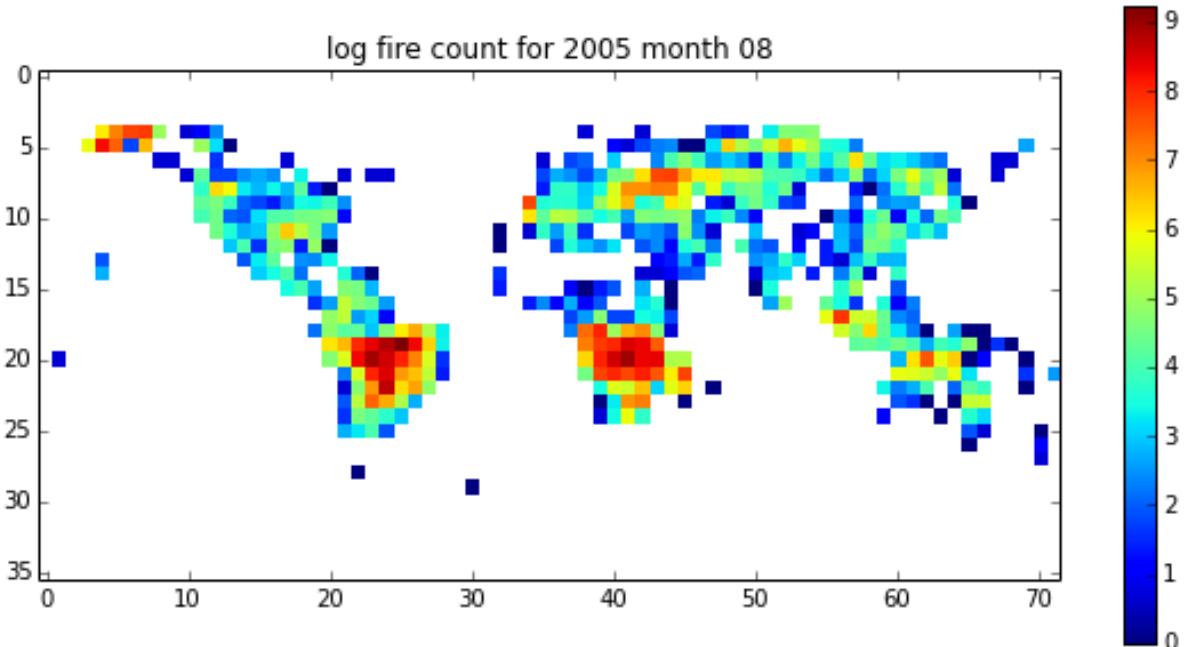
# test it
m = 8
y = 2005
plt.figure(figsize=(10,5))
plt.imshow(np.log(f2data[m-1,y-year[0]]),interpolation='nearest')
plt.colorbar()
plt.title('log fire count for %d month %02d'%(y,m))

(12, 14, 36, 72)

-c:22: RuntimeWarning: divide by zero encountered in log

<matplotlib.text.Text at 0x2b4489c91ad0>

```



```

# which month has the highest fire count
# NB 0-based here but we use a masked array

# total fire count summed over month (axis 0)
fmask = f2data.sum(axis=0) == 0

# which month (axis 0) has the max value?
fire_month = np.argmax(f2data,axis=0)

# masked array of this
fire_month = ma.array(fire_month,mask=fmask)

y = 2005
plt.figure(figsize=(10,5))
plt.imshow(fire_month[y-year[0]],interpolation='nearest')

```

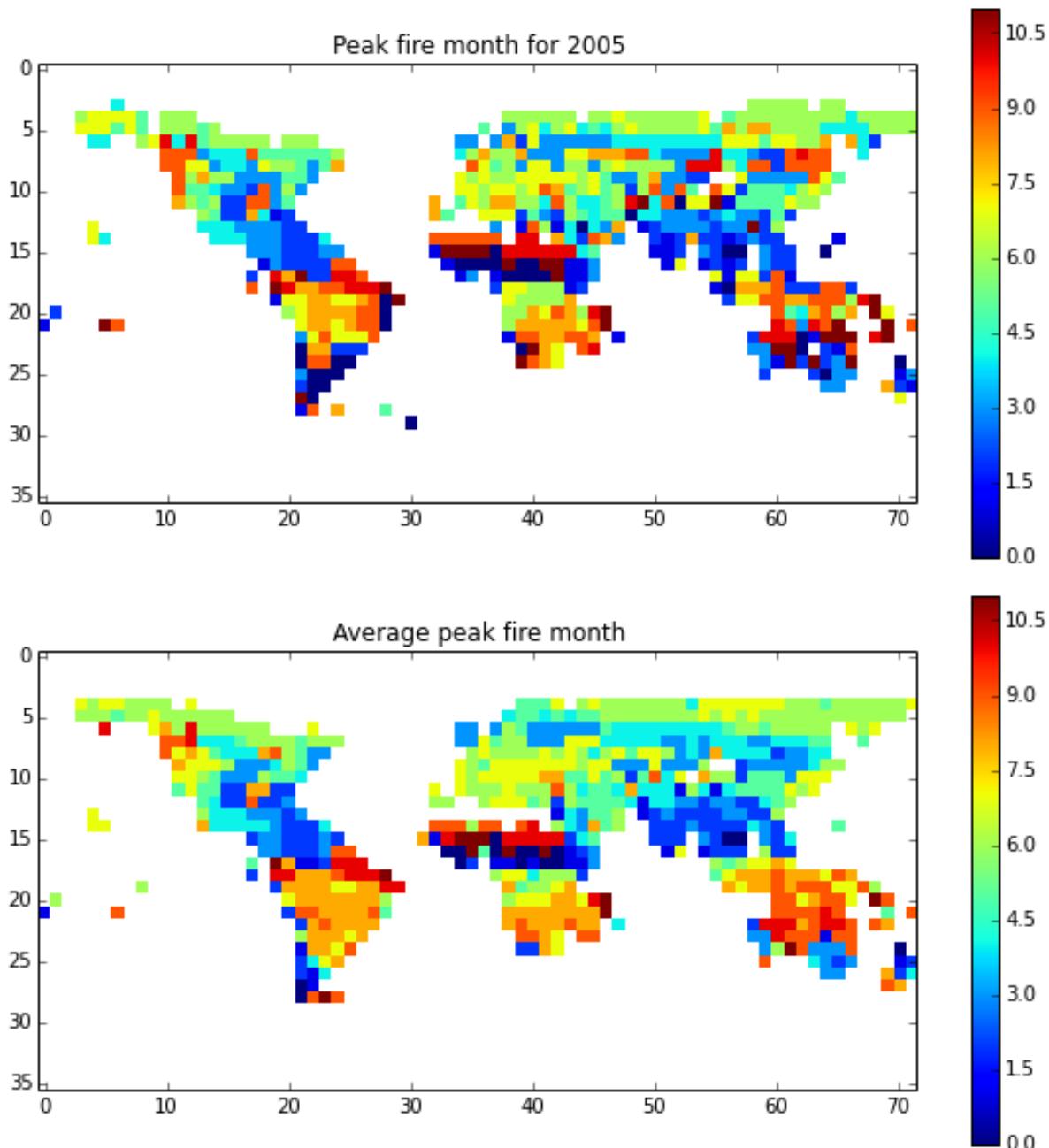
```

plt.colorbar()
plt.title('Peak fire month for %d'%(y))

# suppose this is the same for all years:
av_fire_month = np.median(fire_month, axis=0).astype(int)
plt.figure(figsize=(10,5))
plt.imshow(av_fire_month, interpolation='nearest')
plt.colorbar()
plt.title('Average peak fire month')

<matplotlib.text.Text at 0x134d27d0>

```



```

# and now get the fire count for that month
# lets try this by hand first

```

```
peak_count = np.zeros_like(f2data[0])
```

```

y = 2001
m = 0

fmask = (av_fire_month == m)
peak_count[y-year[0]][fmask] = f2data[m,y-year[0]][fmask]

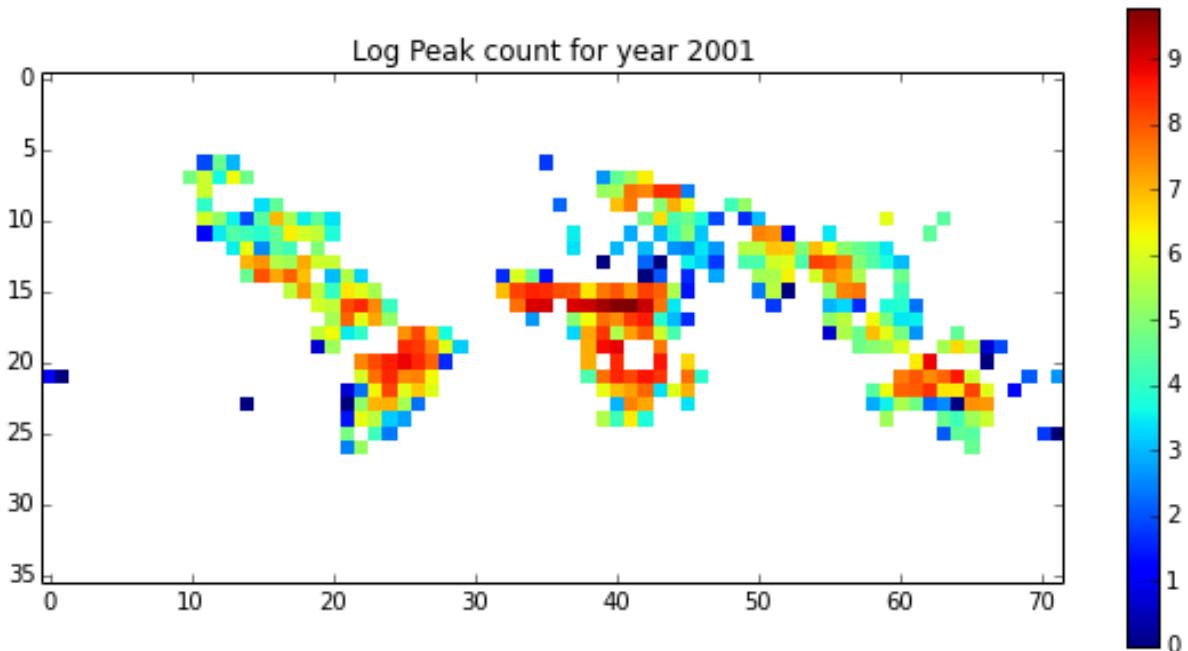
# and now extend it
peak_count = np.zeros_like(f2data[0])

for m in xrange(f2data.shape[0]):
    fmask = (av_fire_month == m)
    for y in xrange(f2data.shape[1]):
        peak_count[y][fmask] = f2data[m,y][fmask]

# test it
y = 1
plt.figure(figsize=(10,5))
plt.imshow(np.log(peak_count[y]),interpolation='nearest')
plt.colorbar()
plt.title('Log Peak count for year %d'%(min_year+y))

<matplotlib.text.Text at 0x2b44901b2090>

```



In summary, we have developed the following datasets:

```

print 'peak_count',peak_count.shape
print 'av_fire_month',av_fire_month.shape
print 'min_year',min_year

peak_count (14, 36, 72)
av_fire_month (36, 72)
min_year 2000

```

Climate Data

The climate data you will want will be some form of Sea Surface Temperature (SST) anomaly measure. There is a long list of such measures on <http://www.esrl.noaa.gov/psd/data/climateindices/list>.

Examples would be [AMO](#) or [ONI](#). Note that some of these measures are smoothed and others not.

Suppose we had selected AMO and we want to read directly from the url:

```
import urllib2

url = 'http://www.esrl.noaa.gov/psd/data/correlation/amon.us.data'

req = urllib2.Request ( url )
raw_data = urllib2.urlopen(req).readlines()

# we notice from inspection that
# we want data from rows 1 to -4
raw_data[:2]

[' 1948      2013n',
 ` 1948    -0.006   -0.018    0.037   -0.061    0.005    0.064   -0.030   -0.013   -0.043
raw_data[-10:-4]

[' 2008    0.051    0.150    0.185    0.071    0.193    0.287    0.237    0.201    0.228
` 2009   -0.032   -0.137   -0.139   -0.103   -0.039    0.152    0.259    0.182    0.086
` 2010    0.068    0.201    0.313    0.457    0.486    0.476    0.482    0.559    0.481
` 2011    0.173    0.134    0.082    0.119    0.172    0.206    0.126    0.180    0.183
` 2012   -0.041    0.028    0.048    0.109    0.191    0.332    0.412    0.468    0.482
` 2013    0.155    0.144    0.186    0.168    0.132    0.078    0.218    0.226    0.290

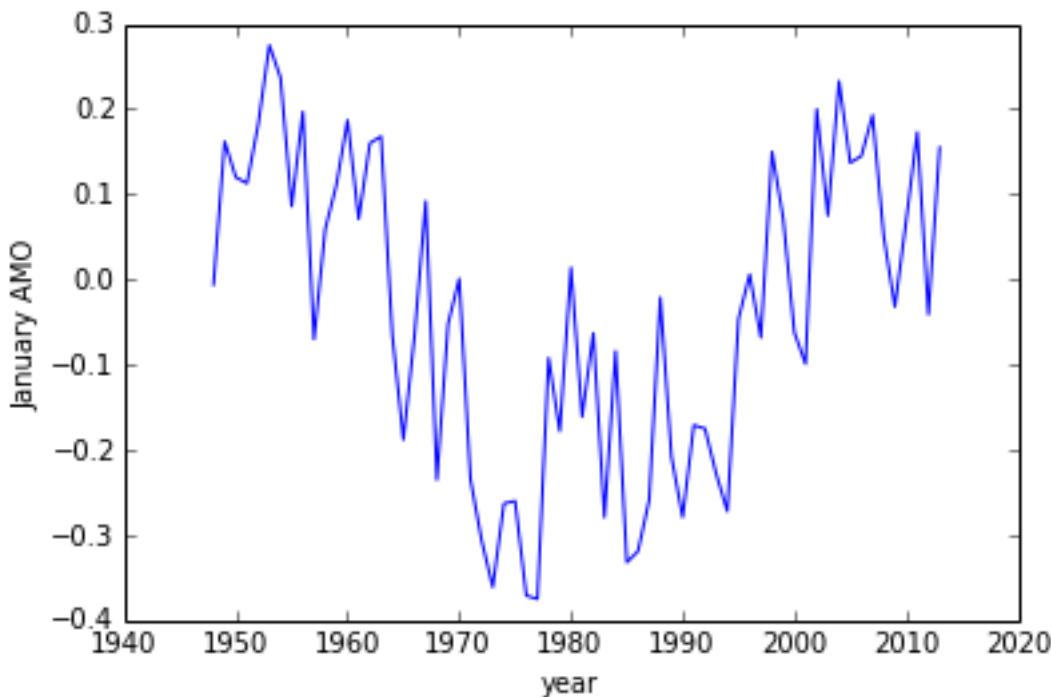
cdata = np.array([r.split() for r in raw_data[1:-4]]).astype(float)

cmask = (cdata < -50 )
cdata = ma.array(cdata,mask=cmask).T
cyears = cdata[0]
cdata = cdata[1:]

# now we have the climate data as a masked array
# column 0 is years, column 1 is Jan etc.

plt.plot(cyears,cdata[0])
plt.xlabel('year')
plt.ylabel('January AMO')
print cdata.shape

(12, 66)
```



11.2.4 7.2.4 Code to perform correlation analysis

The idea here is, for a particular (or set of) SST anomaly measures, work out which ‘lag’ month gives the highest correlation coefficient with fire count.

By ‘lag’ month, we mean that e.g. if the peak fire month for a particular pixel was September, which month prior to that has a set of SST anomalies over the sample years that is most strongly correlated with fire count.

So, if we were using a single SST anomaly measure (e.g. AMO or ONI) and sample years 2001 to 2009 to build our model, then we would do a linear regression of fire count for a particular pixel over these years against e.g. AMO data for September (lag 0) then August (lag 1) then July (lag 2) etc. and see which produced the highest R^2 .

Before we get into that, let’s look again at the data structure we have:

```
# climate data
print 'cdata', cdata.shape
print 'cyears', cyears.shape

# From the fire data

print 'peak_count', peak_count.shape
print 'av_fire_month', av_fire_month.shape
print 'min_year', min_year

cdata (12, 66)
cyears (66,)
peak_count (14, 36, 72)
av_fire_month (36, 72)
min_year 2000
```

So, if we want to select data for particular years:

```
# which years (inclusive)
years = [2001, 2010]

ypeak_count = peak_count[years[0]-min_year:years[1] - min_year + 1]
ycdata = cdata[:, years[0] - cyears[0]:years[1] - cyears[0] + 1]
```

```
# check the shape
print ycdata.shape,ypeak_count.shape,av_fire_month.shape

(12, 10) (10, 36, 72) (36, 72)

We need to consider a little carefully the implementation of lag ...

# we will need to access ycdata[month - n][year]
# which is a bit fiddly as e.g. -3 will be interpreted as
# October for that same year, rather than the previous year
y = 2001 - min_year
m = 2
lag = 5
print m - lag,y

-3 1

# so one way to fix this is to decrease y by one
# if m - lag is -ve
Y = y - (m - lag < 0)
print m-lag,Y

-3 0

from scipy.stats import linregress

# examine an example row col
# for a given month over all years

c = 24
r = 19
m = av_fire_month[r,c]
# pull the data
yyears = np.arange(years[1]-years[0]+1)

R2 = np.array([linregress(\ 
    ycdata[m-n,yyears - (m - n < 0)],\
    ypeak_count[yyears - (m - n < 0),r,c]\ 
) [2] for n in xrange(12)]) 

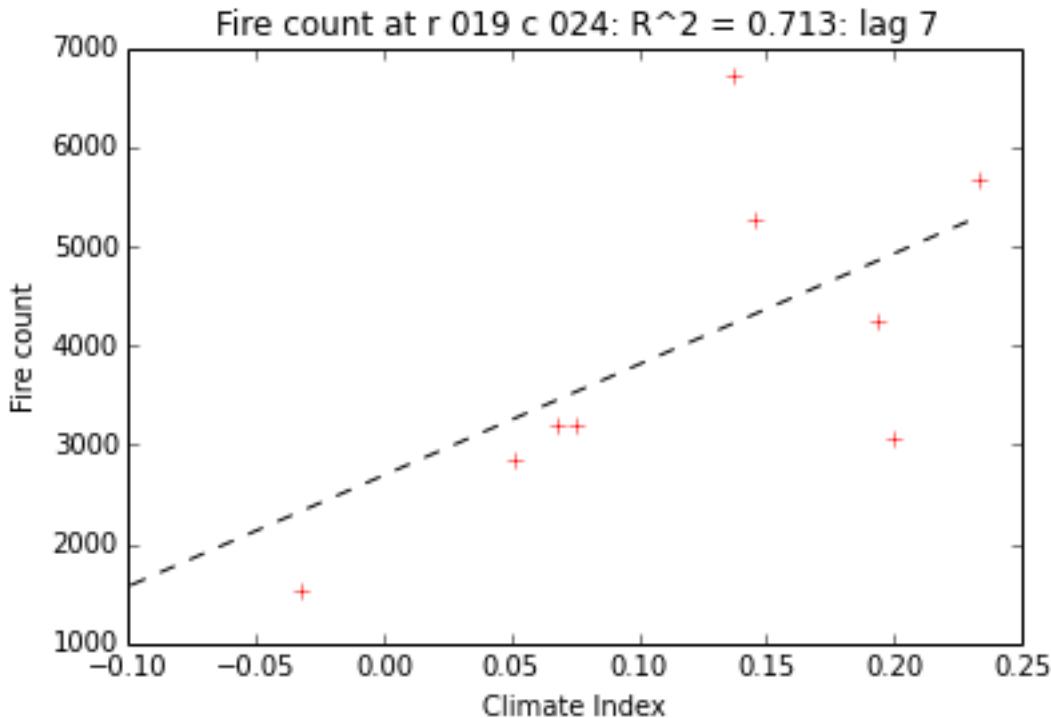
n = np.argmax(R2)

x = ycdata[m-n,yyears - (m - n < 0)]
y = ypeak_count[yyears - (m - n < 0),r,c]
slope,intercept,R,p,err = linregress(x,y)

print slope,intercept,p,err
plt.plot(ycdata[m-n],y,'r+')
plt.xlabel('Climate Index')
plt.ylabel('Fire count')
plt.plot([x.min(),x.max()],\
         [intercept+slope*x.min(),intercept+slope*x.max()],'k--')
plt.title('Fire count at r %03d c %03d: R^2 = %.3f: lag %d'%(r,c,R2[n],n))

11175.1548059 2691.89246835 0.0207363379213 3889.85050593

<matplotlib.text.Text at 0x166679d0>
```



```
# looper

data_mask = ypeak_count.sum(axis=0)>100

rs,cs = np.where(data_mask)

results = {'intercept':0,'slope':0,'p':0,'R':0,'stderr':0,'lag':0}
for k in results.keys():
    results[k] = np.zeros_like(av_fire_month).astype(float)
    results[k] = ma.array(results[k],mask=~data_mask)

for r,c in zip(rs,cs):
    m = av_fire_month[r,c]
    # pull the data
    yyears = np.arange(years[1]-years[0]+1)
    R2 = np.array([
        linregress(
            ycdata[m-n,yyears - (m - n < 0)], \
            ypeak_count[yyears - (m - n < 0),r,c]\n        )[2] for n in xrange(12)])
    n = np.argmax(R2)
    results['lag'][r,c] = n
    x = ycdata[m-n,yyears - (m - n < 0)]
    y = ypeak_count[yyears - (m - n < 0),r,c]
    results['slope'][r,c],results['intercept'][r,c],\
    results['R'][r,c],results['p'][r,c],\
    results['stderr'][r,c] = linregress(x,y)

plt.figure(figsize=(10,4))
plt.imshow(results['R'],interpolation='nearest')
plt.colorbar()
plt.title('R')

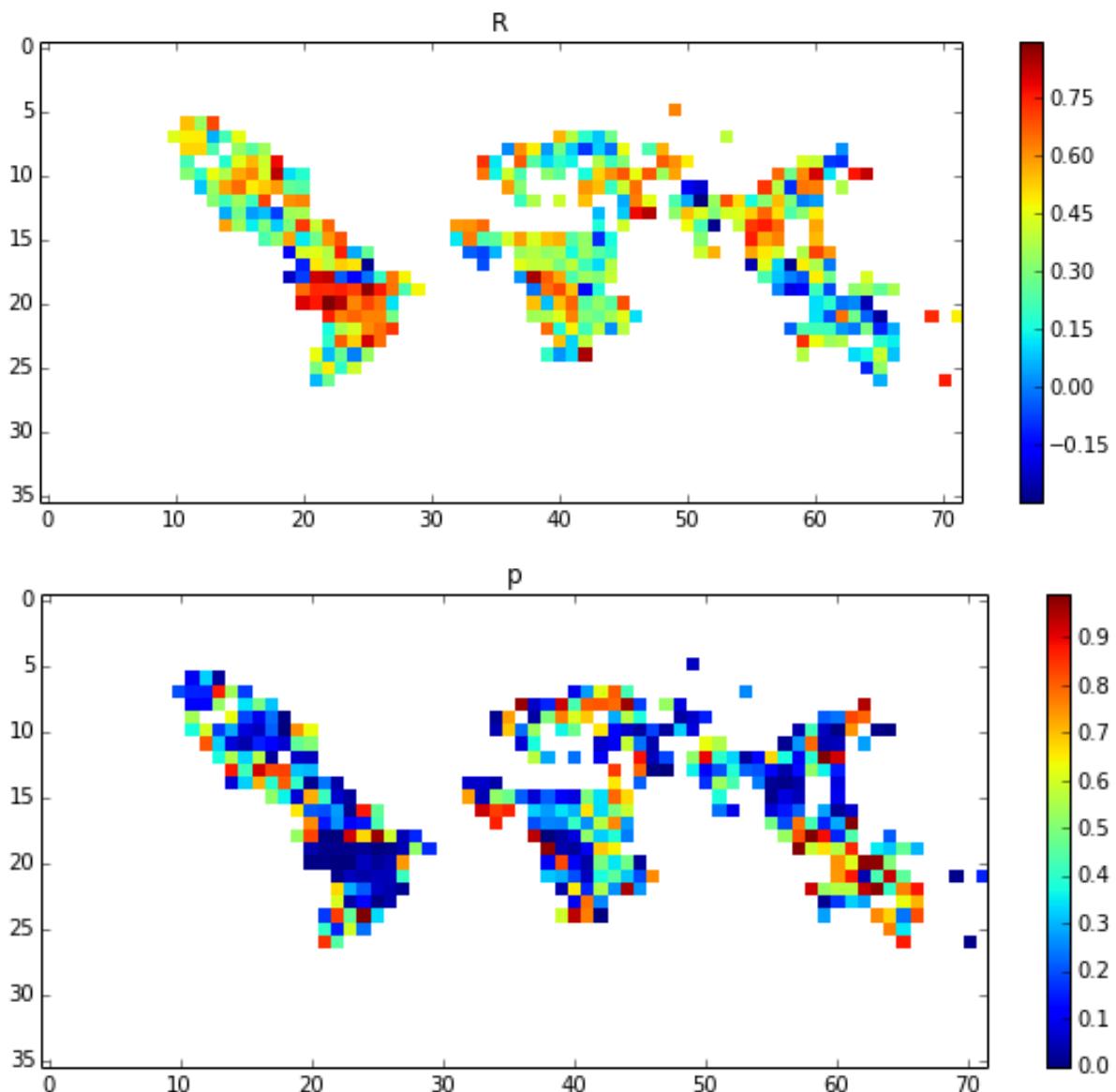
plt.figure(figsize=(10,4))
plt.imshow(results['p'],interpolation='nearest')
plt.colorbar()
```

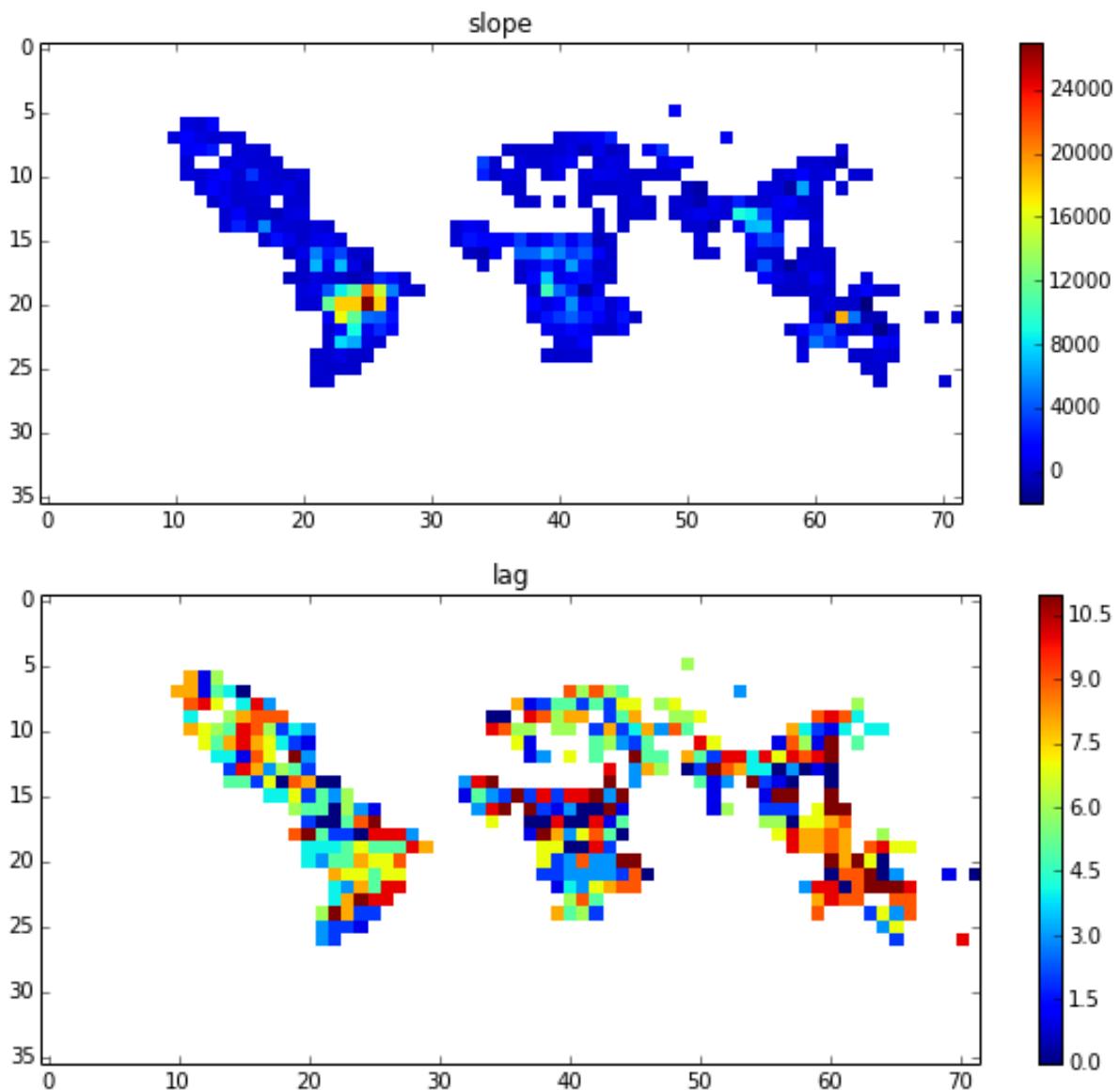
```
plt.title('p')

plt.figure(figsize=(10, 4))
plt.imshow(results['slope'], interpolation='nearest')
plt.colorbar()
plt.title('slope')

plt.figure(figsize=(10, 4))
plt.imshow(results['lag'], interpolation='nearest')
plt.colorbar()
plt.title('lag')

<matplotlib.text.Text at 0x6d69890>
```





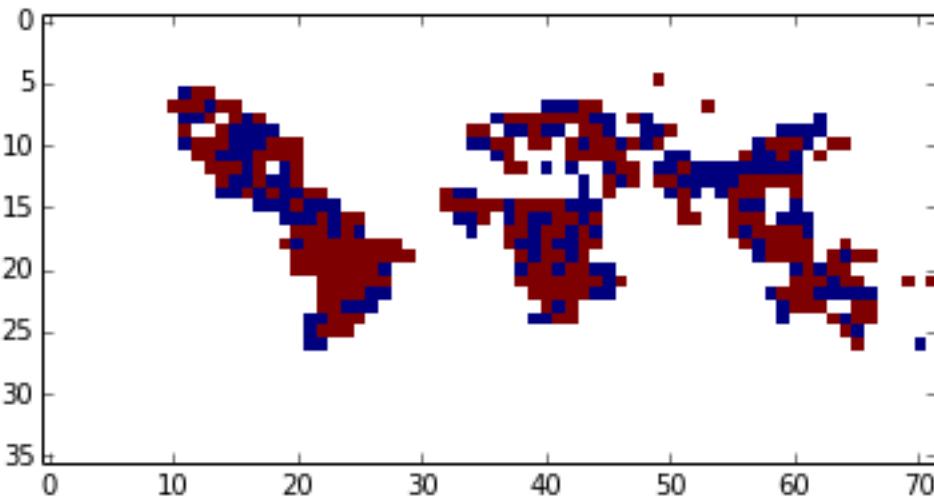
which we can now predict:

```
# prediction year
pyear = 2012

# which month?
M = av_fire_month - results['lag']
Y = np.zeros_like(M) + pyear
Y[M<0] -= 1

# lets look at that ...
plt.imshow(Y, interpolation='nearest')

<matplotlib.image.AxesImage at 0x7d6ab90>
```

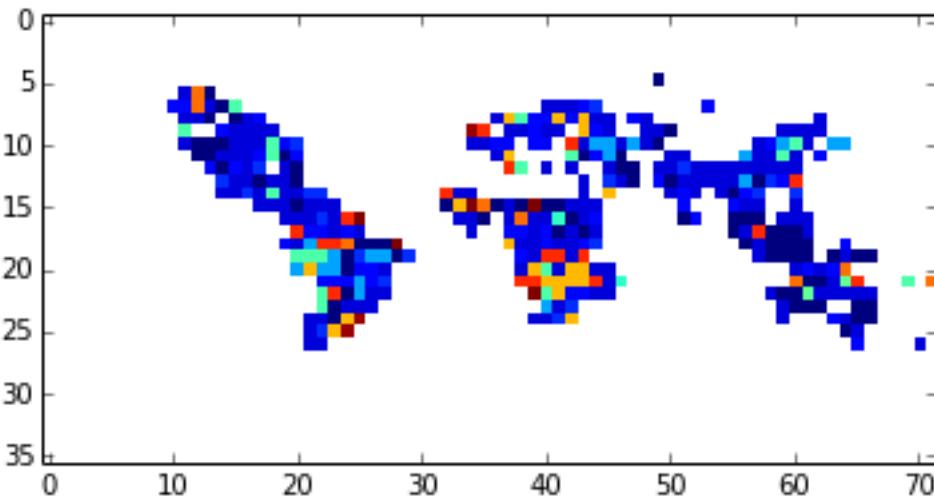


```
# climate data
scdata = np.zeros_like(Y).astype(float)

for y in [pyear,pyear-1]:
    for m in xrange(12):
        scdata[(Y == y) & (M == m)] = cdata[m,y-cyears[0]]

plt.imshow(scdata,interpolation='nearest')

<matplotlib.image.AxesImage at 0x7d5a590>
```



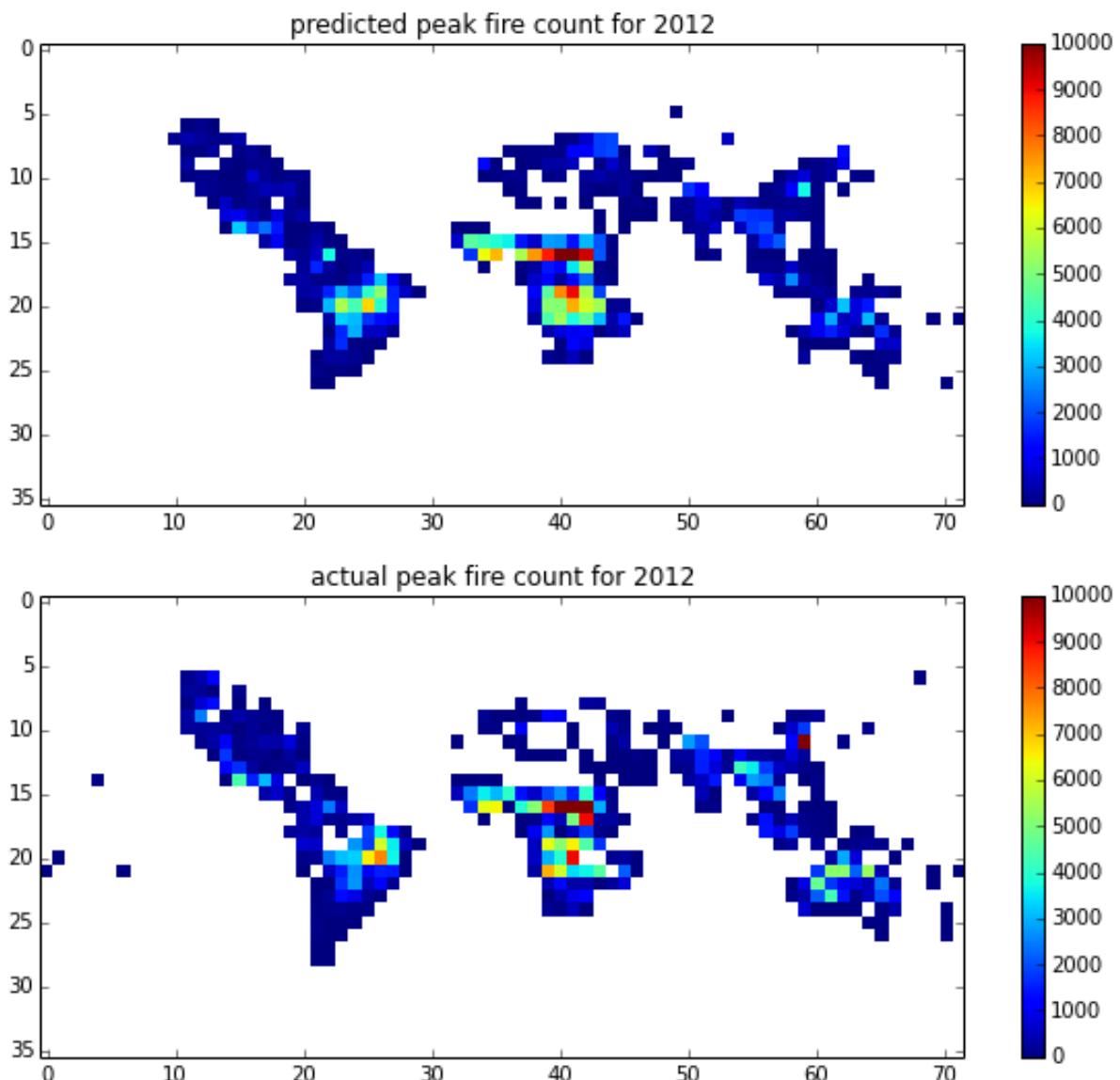
```
# now predict

fc_predict = results['intercept'] + results['slope'] * scdata

plt.figure(figsize=(10,4))
plt.imshow(fc_predict,interpolation='nearest',vmin=0,vmax=10000)
plt.colorbar()
plt.title('predicted peak fire count for %d'%pyear)

plt.figure(figsize=(10,4))
plt.imshow(peak_count[pyear-min_year],\
           interpolation='nearest',vmin=0,vmax=10000)
plt.colorbar()
plt.title('actual peak fire count for %d'%pyear)
```

<matplotlib.text.Text at 0x83ec750>



```

x = peak_count[pyear-min_year].flatten()
y = fc_predict.flatten()

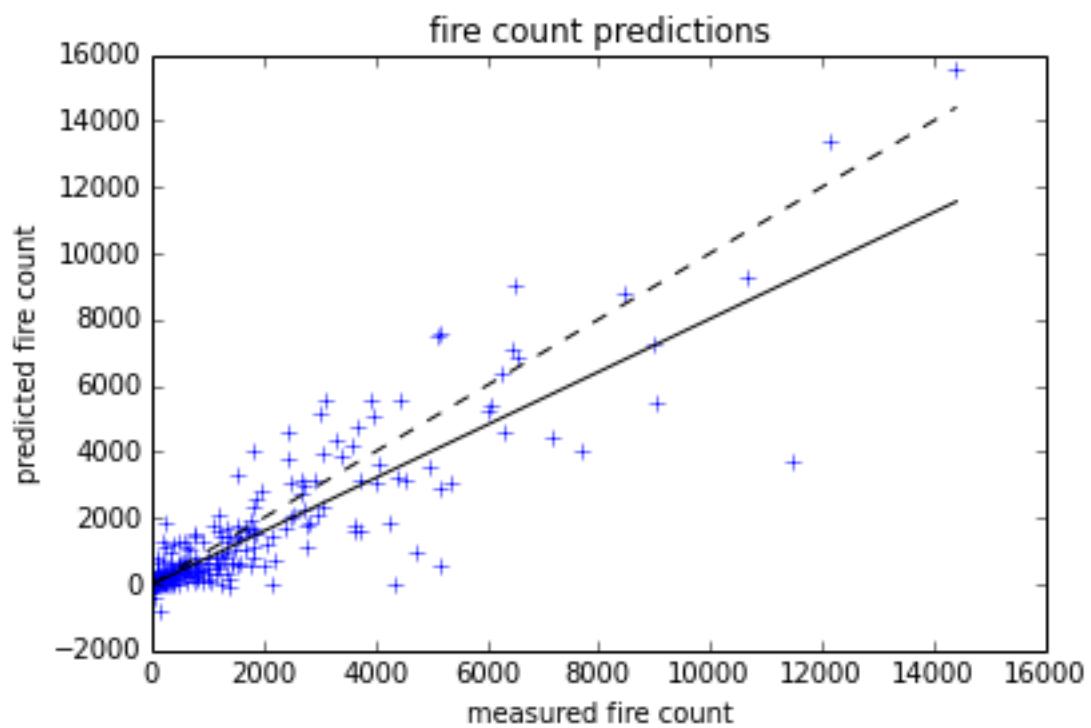
slope,intercept,R,p,err = linregress(x,y)

plt.plot(x,y,'+')
plt.xlabel('measured fire count')
plt.ylabel('predicted fire count')
cc = np.array([0.,x.max()])

plt.plot(cc,cc,'k--')
plt.plot(cc,slope*cc+intercept,'k-')
plt.title('fire count predictions')
print slope,intercept,R,p,err

<class 'numpy.ma.core.MaskedArray'>
0.802827703736 -6.97449131978 0.880785138861 0.0 0.00848080897451

```



CHAPTER
TWELVE

RECOLLISION PROBABILITY THEORY

A useful source of information quantifying vegetation amount is the Leaf Area Index (LAI). To interpret LAI from satellite data, we build models of radiative transfer.

One of the simplest such models that we can use at optical wavelengths uses what is known as ‘p theory’ or ‘recollision probability theory’ (Lewis and Disney, 2007; Huang et al., 2007).

The task today is to use p-theory to estimate LAI over some agricultural fields.

References

- P. Lewis and M. Disney (2007) Spectral invariants and scattering across multiple scales from within-leaf to canopy, *Remote Sensing of Environment* 109, 196-206.
- D. Huang, Y. Knyazikhin, R.E. Dickinson, M. Rautiainen, P. Stenberg, M. Disney, P. Lewis, A. Cescatti, Y. Tian, W. Verhoef, and R.B. Myneni (2007), Canopy spectral invariants for remote sensing and model applications, *Remote Sensing of Environment*, 106, 106-122
- Knyazikhin Y, Schull MA, Stenberg P, Mottus M, Rautiainen M, Yang Y, Marshak A, Latorre Carmona P, Kaufmann RK, Lewis P, Disney MI, Vanderbilt V, Davis AB, Baret F, Jacquemoud S, Lyapustin A, Myneni RB. (2013) Hyperspectral remote sensing of foliar nitrogen content. Proc Natl Acad Sci USA, 10.1073/pnas.1210196109.

**CHAPTER
THIRTEEN**

DATA

The dataset you have available is an airborne hyperspectral image (HYMAP) dataset for which you have a 512x512 pixel subscene taken in 125 wavebands with a spatial resolution of 4m. The data were obtained on 17 June 2000 over Barton Bendish Farms, Norfolk during the BNSC/NERC SHAC campaign.

The data are available as a compressed *flat binary* file in the directory ‘files/data <files/data>‘__.

First, uncompress the dataset:

```
# or try zcat if gzcat isn't there
!gzcat files/data/bbHYMAP.dat.gz > files/data/bbHYMAP.dat

!ls -l files/data/bbHYMAP.dat

-rw-r--r-- 1 plewis  staff  131072000 26 Nov 17:31 files/data/bbHYMAP.dat
```

The file is 131072000 bytes in 32 bit floating point format:

```
print 'nbands =', 131072000 / (512*512*4)

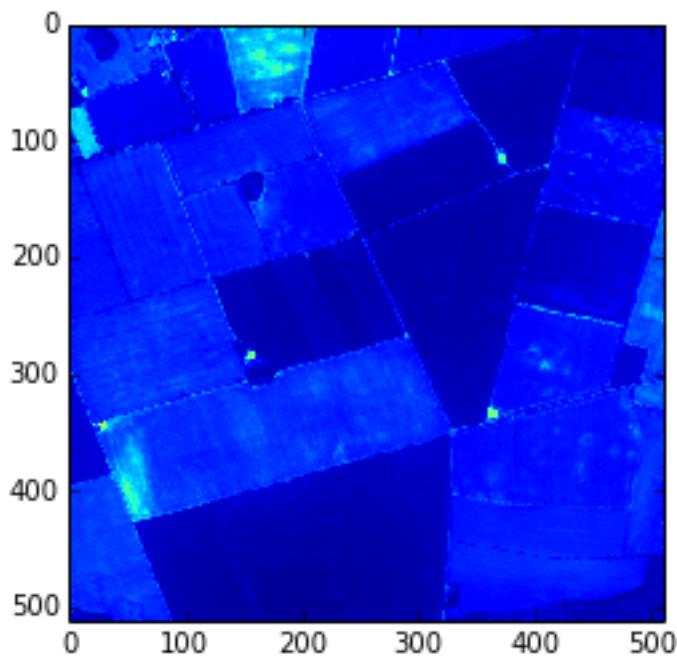
nbands = 125

# we will use memmap to read the data in
from numpy import memmap

hymap = memmap('files/data/bbHYMAP.dat', dtype=np.float32, mode='r', shape=(125, 512, 512))

plt.imshow(hymap[10], interpolation='nearest')

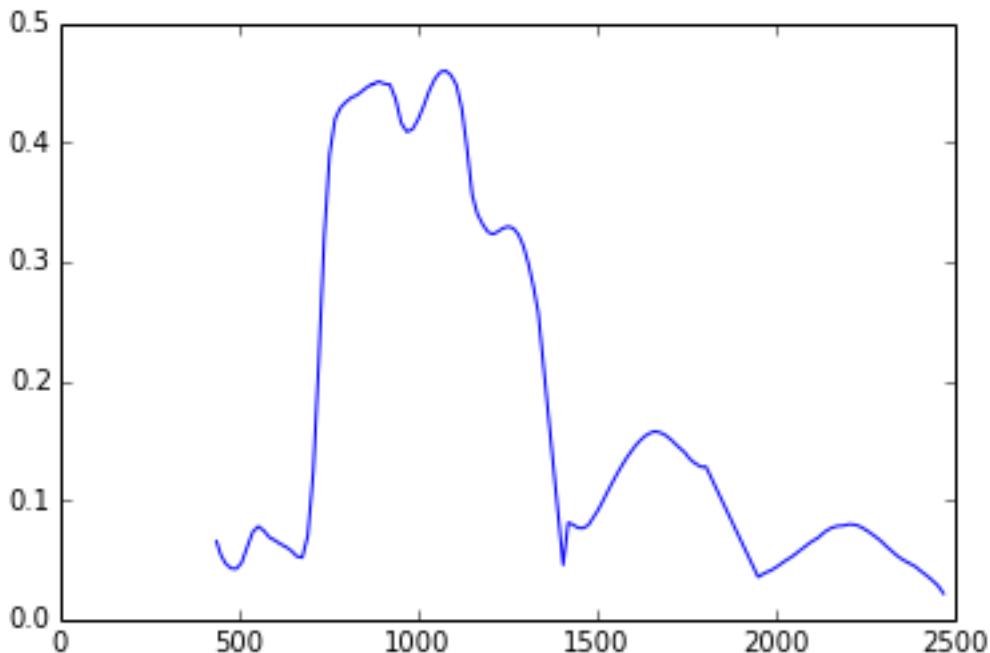
<matplotlib.image.AxesImage at 0x10598edd0>
```



The wavelengths associated with each band are stored in 'files/data/wavebands.dat'.

```
mean = hymap.mean(axis=(1,2))
wavelength = np.loadtxt('files/data/wavebands.dat')
plt.plot(wavelength,mean)
```

```
[<matplotlib.lines.Line2D at 0x10f089790>]
```



CHAPTER
FOURTEEN

THEORY

The simplest form of model in p-theory assumes that the photon recollision probability is constant with wavelength and scattering order. Under this assumption, the total scattering from the canopy, W is:

$$W = i_0 \frac{(1-p)\omega}{1-p\omega}$$

where i_0 is the canopy interception probability, ω is the leaf-level scattering (the leaf single scattering albedo) and p is the recollision probability: the probability that a photon, having intercepted a canopy element, will recollide with another element rather than escape the canopy.

We can develop from this a model of the canopy *reflectance* ρ (i.e. that portion scattered upwards, perhaps in a particular direction):

$$\rho = \frac{a\omega}{1-p\omega}$$

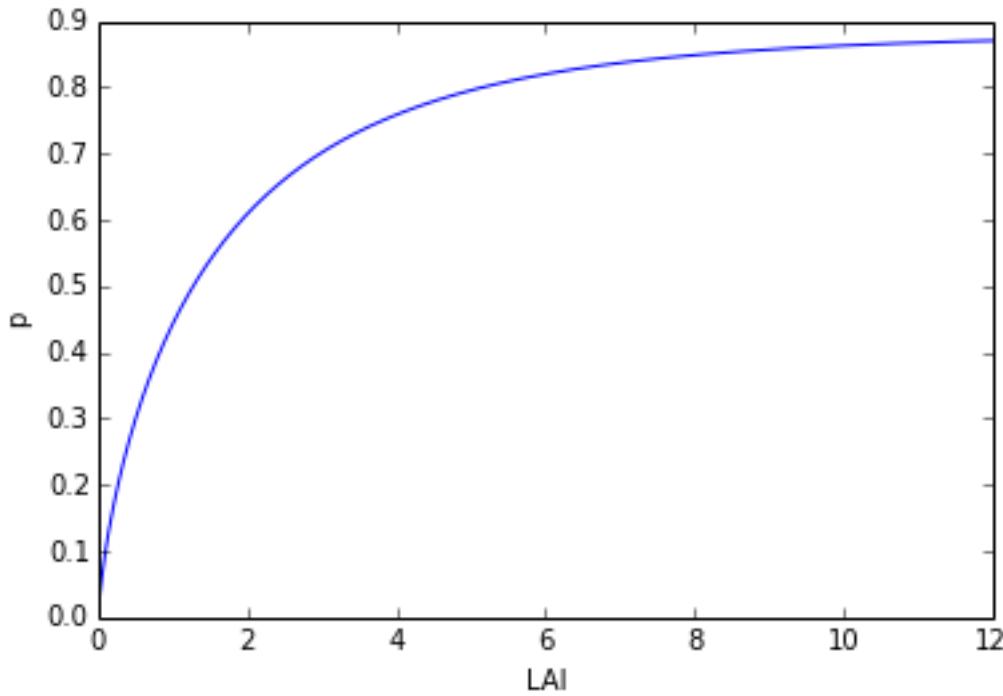
For a closed canopy, or one with only little soil influence, this will describe the spectral reflectance for some given leaf single scattering albedo spectrum and given p .

If we know ω then, and have a measurement of ρ , we can estimate p . From p , we can estimate LAI according to Lewis and Disney (2007) by:

$$p = 0.88 (1 - \exp(-0.7LAI^{0.75}))$$

```
LAI = np.arange(0,12,0.01)
p = 0.88*(1 - np.exp(-0.7 * LAI**0.75))
plt.plot(LAI,p)
plt.xlabel('LAI')
plt.ylabel('p')

<matplotlib.text.Text at 0x10f12f0d0>
```



There are several ways we could estimate p . An interesting feature exploited by Knyazikhin et al. (2013) follows from:

$$\frac{\rho}{\omega} = \frac{a}{1 - p\omega}$$

so

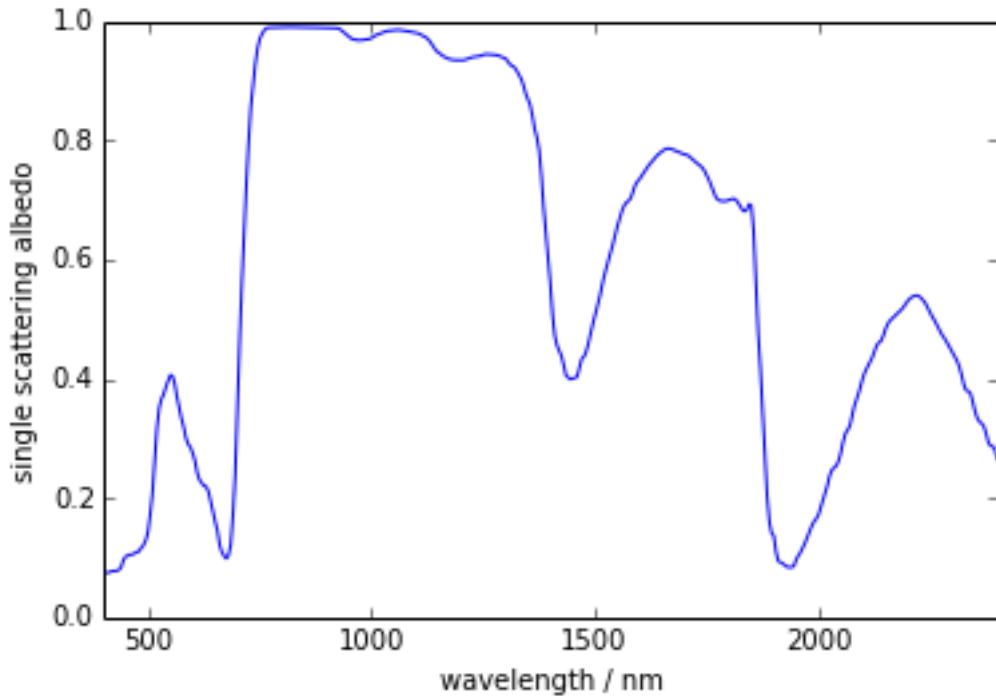
$$\frac{\rho}{\omega} = a + p\rho$$

So that if we plot $\frac{\rho}{\omega}$ as a function of ρ , this theory predicts we should see a straight line.

An example leaf single scattering albedo is given in ‘files/data/ssalbedo.dat’:

```
ssalbedo = np.loadtxt('files/data/ssalbedo.dat').T
plt.plot(ssalbedo[0], ssalbedo[1])
plt.xlabel('wavelength / nm')
plt.ylabel('single scattering albedo')
plt.xlim(ssalbedo[0][0], ssalbedo[0][-1])  

(400.0, 2400.0)
```



which is sampled every 1 nm from 400 to 2400 nm.

We will clearly need to resample this to the same wavebands as the hyperspectral data:

```
from scipy.interpolate import interp1d
from scipy.stats import linregress

f = interp1d(ssalbedo[0], ssalbedo[1])
omega = f(wavelength[wavelength<=2400])

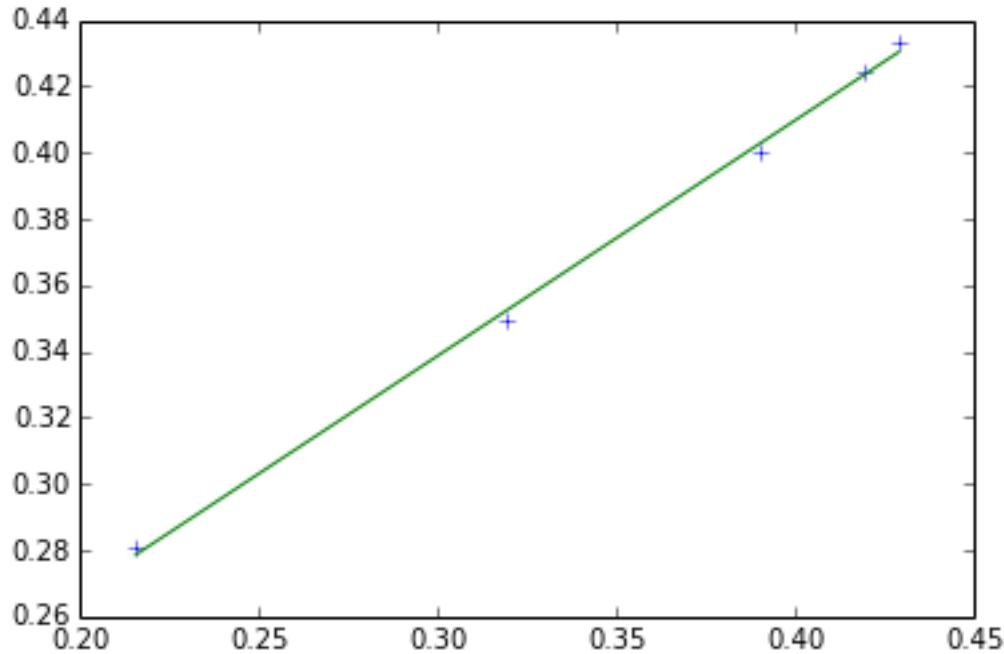
# wavelength
W = (wavelength >= 710) & (wavelength <= 790)

wave = wavelength[W]
# single scattering albedo
omega = f(wavelength[W])
# reflectance
rho = hymap[W]

# lets see if its a straight line!
mean = rho.mean(axis=(1,2))
plt.plot(mean, mean/omega, '+')

# linear fit
slope, intercept, r_value, p_value, std_err = linregress(mean, mean/omega)
x = np.array([mean[0], mean[-1]])
plt.plot(x, x*slope+intercept)
print 'slope', slope, 'intercept', intercept
```

```
slope 0.710882123721 intercept 0.125383329915
```



So, here, p is 0.71088

```
#p = 0.88*(1 - np.exp(-0.7 * LAI**0.75))
LAI = (np.log(1 - slope/0.88) / -0.7)**(4./3.)
print LAI

3.13529156174
```

Following Knyazikhin et al. (2013) we can calculate the DASF (Directional Area Scattering Function) from:

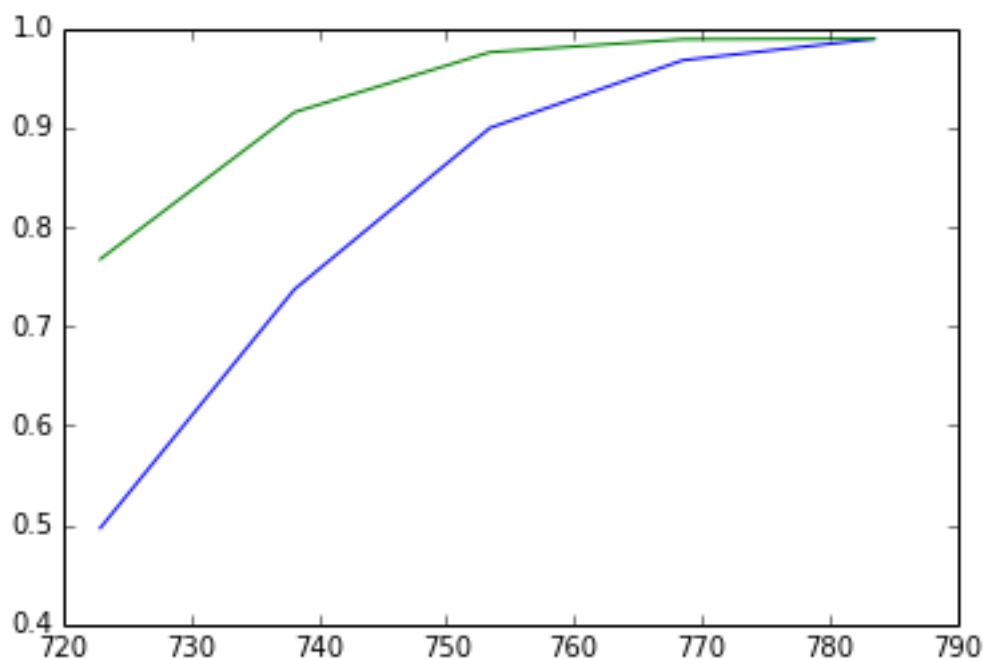
```
DASF = intercept / (1 - slope)
print DASF
```

```
0.43367546666
```

and from that, W :

```
W = mean/DASF
plt.plot(wave,W)
plt.plot(wave,omega)

[<matplotlib.lines.Line2D at 0x11b5d1410>]
```



which in turn can give access to leaf biochemistry.

**CHAPTER
FIFTEEN**

THE TASK

The output of this exercise should be a spatial dataset of LAI.

The processing for this task is quite straightforward, and you *should* be able to develop a neat algorithm given the information above.

You should work in teams for this exercise as in previous weeks, and assign tasks for people to complete after you have discussed an overall algorithm and defined any interfaces you will need.

If you finish the task quickly, you could explore the impact of the the leaf single scattering albedo assumed, and/or examine the impact of widening the wavelength range used here.

This section of the notes is intended for follow-up after the class and future reference. We will not go through this in detail in the teaching sessions.

This material and the exercises are more advanced than in the main notes. You are not *required* to do these, but may like to do so if you want to stretch yourself.

A2.1 BINARY OPERATORS

16.1 A2.1.1 Basic Logic

Quite often (data masks in data products, being the most typical example), you will need to deal with binary numbers and binary operations, so we will introduce the concepts you need here.

We came across *logical* or *Boolean* operations in the main session (with the data type `bool`). In logic, something can be `True` or `False`, and operations such as '`not` <http://en.wikipedia.org/wiki/Logical_NOT>'__,'`and` <http://en.wikipedia.org/wiki/Logical_AND>'__ and '`or` <http://en.wikipedia.org/wiki/Logical_OR>'__ have quite obvious meanings. More generally in logic (and electronics) we may come across other logical operators such as '`nand` <http://en.wikipedia.org/wiki/Logical_NAND>'__ (`not and`) and '`nor` <http://en.wikipedia.org/wiki/Logical_NOR>'__ (`not or`) and '`xor` <http://en.wikipedia.org/wiki/Logical_XOR>'__ (`exclusive or: either but not both`), but these are not defined in Python (they can of course be *derived* though).

You should first make sure that you understand the results of the `and`, `or` and `not` `bool` operators:

```
print 'False and False =',False and False
print 'False and True  =',False and True
print 'True and False =',True and False
print 'True and True   =',True and True

False and False = False
False and True  = False
True and False = False
True and True   = True

print 'False or False =',False or False
print 'False or True  =',False or True
print 'True or False =',True or False
print 'True or True   =',True or True

False or False = False
False or True  = True
True or False = True
True or True   = True

print 'not False =',not False
print 'not True  =',not True

not False = True
not True  = False
```

16.2 Exercise A2.1

As an exercise for this, you could see if you can simulate the logical combinations `xor`, `nor` and `nand`, e.g.:

```
# nand test:
# see http://en.wikipedia.org/wiki/Logical_NAND
# (A nand B) is not (A and B)

ABList = [(False,False), (False,True), (True,False), (True,True)]
for A,B in ABList:
    print '%s nand %s = %%(str(A),str(B)),not (A and B)

False nand False = True
False nand True = True
True nand False = True
True nand True = False
```

16.3 A2.1.2 Binary

16.4 Endianness

It is of value to have some understanding of binary operations and representation(s):

- you will come across these in encoded data products (such as the QA information in MODIS and other satellite products) as it is a more efficient way of encoding multiple sets of logical information
- this is the form in which the computer ultimately stores and processes information, so it is useful to have some appreciation of that
- you will sometimes need to consider how large a number representation is (e.g. byte or short integer or long integer) as this can impact computer memory and storage requirements
- you may come across different binary representations for different datasets

There are two main number representations used in computing, which depend on the interpretation of the **MSB**, Most Significant Bit or ‘high bit order’ (or Byte) and **LSB**, Least Significant Bit or ‘low bit order’ (or Byte).

Which system is used is sometimes referred to as ‘endianness’ so we may refer to a ‘big-endian’ or ‘little-endian’ representation. In a big-endian representation, the left-most byte represents the highest number. In a little-endian system, this is the lowest number.

It is probably easiest to understand this with decimal numbers:

So, in a big-endian decimal representation:

152

represents one hundred and fifty two ((1 x 10²) + (5 x 10¹) + (2 x 10⁰))

In a little-endian system, this is interpreted the other way around, so 152 is (2 x 10²) + (5 x 10¹) + (1 x 10⁰), so is actually two hundred and fifty one.

The term comes from Jonathan Swift’s [Gulliver’s travels](#), if you are interested, referring to the which end of an egg the people of Lilliput and Blefuscus believe you should open:

"(The people of Lilliput and Blefuscus have) been engaged in a most obstinate war for six-and-thirty moons past. It began upon the following occasion. It is allowed on all hands, that the primitive way of breaking eggs, before we eat them, was upon the larger end; but his present majesty's grandfather, while he was a boy, going to eat an egg, and breaking it according to the ancient practice, happened to cut one of his fingers. Whereupon the emperor his father published an edict, commanding all his subjects, upon great penalties, to break the smaller end of their eggs. The people so highly resented this law, that our histories tell us, there have been six rebellions raised on that account; wherein one emperor lost his life, and another his crown. These civil commotions

were constantly fomented by the monarchs of Blefuscu; and when they were quelled, the exiles always fled for refuge to that empire. It is computed that eleven thousand persons have at several times suffered death, rather than submit to break their eggs at the smaller end. Many hundred large volumes have been published upon this controversy: but the books of the Big-endians have been long forbidden, and the whole party rendered incapable by law of holding employments."

Endianness then (like which end of an egg you should break), is ultimately arbitrary, but we must still have conventions so that information on one computer system can be interpreted by another.

At present, probably most computers you use will be little-endian (the ‘Intel convention’ as it has come to be known), but you may also come across data stored in big-endian (so-called ‘Motorola convention’) format.

Endianness is mainly implemented at the byte level, so if you are considering a single byte, which ‘end you open the egg’ has no consequence.

For longer number representations though (e.g. a 2-byte or 4-byte integer) it can have significant consequences, and you need to be aware of the endianness of data that you might try to be reading. Many file formats will explicitly store the endianness that data were written in, so a correct interpretation will be handled by low level reading routines. But, if you get binary data that are of more than one byte that look rather odd when you display them, one possible reason is that you have assumed a different endianness to what the data were written in.

16.5 Binary numbers

When we are working with *binary numbers for a single bit*, we represent False by 0 and True by 1.

In Python, these representations can essentially be used interchangably with logical operators, (but we use the function `bool()` to convert to a boolean representation)

```
print True or False
True

print 1 or 0
1

print bool(1 or 0)
True

whatTheySay = True

if whatTheySay:
    print "it's true what they say ..."
else:
    print "it's not true what they say ..."

it's true what they say ...

whatTheySay = 1

if whatTheySay:
    print "it's true what they say ..."
else:
    print "it's not true what they say ..."

it's true what they say ...
```

A binary representation of a number is a representation in **base 2**, where we may use multiple **bits** to represent numbers.

Some number representations (such as floating point) are stored in a more complicated manner, but ASCII codes for string representation and integers are in a more straightforward binary format.

So, e.g. the (integer) **decimal** number 3 is 11 in binary (base 2):

```
3 == (1 * 2**1) + \
      (1 * 2**0)
```

```
True
```

We can use the Python function `bin()` to get this more directly:

```
print bin(3)
```

```
0b11
```

```
print 0b11
```

```
3
```

Similarly, the decimal number 101 is 1100101 in binary:

```
101 == (1 * 2**6) + \
        (1 * 2**5) + \
        (0 * 2**4) + \
        (0 * 2**3) + \
        (1 * 2**2) + \
        (0 * 2**1) + \
        (1 * 2**0)
```

```
True
```

```
print bin(101)
```

```
0b1100101
```

16.6 Exercise A2.2

1. Work out what the following decimal numbers are in binary:

7

493

127

255

1024

and check your result using the approach we took to confirming 101:

```
101 == (1 * 2**6) + \
        (1 * 2**5) + \
        (0 * 2**4) + \
        (0 * 2**3) + \
        (1 * 2**2) + \
        (0 * 2**1) + \
        (1 * 2**0)
```

2. How many bits are needed to represent each of these numbers?
3. What is the largest number you could represent in: (i) a 32 bit representation; (b) a 64 bit representation?

4. Recalling that there are 8 bits in a `byte`, what is the largest number you could represent in: (a) a single byte; (b) two bytes?

'''

Explorations in binary!

We can represent:

<i>binary numbers, e.g.</i>	<i>0b010101</i>
<i>octal numbers, e.g.</i>	<i>0o755</i>
<i>hexadecimal, e.g.</i>	<i>0x2A2FF</i>

*but if we print these,
by default they are printed as the decimal equivalents.*

*We can convert to binary, octal or hex string with
`bin()`, `oct()`, `hex()`*

'''

```
x = 0b111101101
print bin(x), 'is', x, 'in decimal'
print bin(x), 'is', oct(x), 'in octal'
print bin(x), 'is', hex(x), 'in hexadecimal'

0b111101101 is 493 in decimal
0b111101101 is 0755 in octal
0b111101101 is 0x1ed in hexadecimal
```

16.7 A2.1.2 Bitwise Operators

In Python (and most other computer languages) you have access to bitwise operators. As you might expect, these are operators that are executed on individual bits in a binary representation of a number.

The bitwise operators available in Python are:

- & bitwise and
- | bitwise or
- ^ bitwise xor (exclusive or)
- ~ bitwise ones complement
- << bitwise left shift
- >> bitwise right shift

The & operator simply performs a logical and operation on two sets of binary representations, so:

```
1 & 0 == 0
```

is the same as a logical True and False operation that we saw above .

The | operator simply performs a logical or operation on two sets of binary representations, so:

```
1 | 0 == 1
```

is the same as a logical True or False operation that we saw above.

Similarly,

```
1 ^ 0 == 1
1 ^ 1 == 0
```

and

```
~1 == 0  
~0 == 1
```

These same rules apply then to all bit fields:

```
~1010 == 0101
```

etc.

The shift operators are interesting:

left shift by 1, for example is equivalent to multiplying by 2, and right shift by 1 a division by 2.

They are also very useful in sorting out ‘bit masks’ for data products.

```
'''  
    Bitwise operators:  
'''  
  
A = 521  
B = 523  
# print as binary:  
print A, '\tA:\t', bin(A)  
print B, '\tB:\t', bin(B)  
  
# some operations  
print '\tA | B:\t', bin(A|B), '\t', A|B  
print '\tA ^ B:\t', bin(A^B), '\t\t', A^B  
print '\tA & B:\t', bin(A&B), '\t', A&B  
  
521      A:      0b1000001001  
523      B:      0b1000001011  
A | B: 0b1000001011      523  
A ^ B: 0b10          2  
A & B: 0b1000001001      521  
  
'''  
    Bitwise shift operators:  
'''  
  
A = 531  
print '\tA:\t', bin(A), '\t', A  
print '\tA>>1:\t', bin(A>>1), '\t', A>>1  
print '\tA>>1:\t', bin(A<<1), '\t', A<<1  
print '\tA>>3:\t', bin(A>>3), '\t', A>>3  
print '\tA>>3:\t', bin(A<<3), '\t', A<<3  
  
A:      0b1000010011      531  
A>>1:  0b100001001      265  
A>>1:  0b10000100110     1062  
A>>3:  0b1000010          66  
A>>3:  0b1000010011000    4248
```

As an example of data masking, consider the QA mask in the MODIS Leaf Area Index (LAI) product:

Bit number

Parameter Name

Bit combination

Interpretation

0

MODLAND_QC bits

0

Good quality (main algorithm with or without saturation)

1

Other Quality (back-up algorithm or fill values)

1

Sensor

0

Terra

1

Aqua

2

DeadDetector

0

Detectors apparently fine for up to 50% of channels

1

Dead detectors caused >50% adjacent detector retrieval

3-4

CloudState

00

Significant clouds NOT present (clear)

01

Significant clouds WERE present

10

Mixed cloud present on pixel

11

Cloud state not defined (assumed clear)

5-7

CF_QC

000

Main (RT) method used (best result possible (no saturation))

001

Main (RT) method used with saturation. (usable)

010

Main (RT) method failed due to bad geometry (empirical algorithm used)

010

Main (RT) method failed due to problems other than geometry (empirical algorithm used)

010

Pixel not produced at all.

So, the MODIS LAI QA information is contained in one byte (8 bits), which encodes data about five different categories of QA information.

When using such data, we need to make choices about what quality of data we are willing to accept.

How can we use our understanding of bitwise operations to pull out these datasets?

First, we can try to pull out the different categories.

One way to do this is using a set of ‘bit masks’ for each of the bit fields we want to cover.

If then perform a bitwise and operation with these, only relevant bits will ‘show’ through.

```
# an example QA value: fill all fields
qa = 0b11111111

print 'qa',qa
print 'binary qa',bin(qa)

# set up masks for the different parameters
mask1 = 0b00000001      # bit 0
mask2 = 0b00000010      # bit 1
mask3 = 0b00000100      # bit 2
mask4 = 0b00011000      # bit 3-4
mask5 = 0b11100000      # bit 5-7

qa1 = qa & mask1
qa2 = qa & mask2
qa3 = qa & mask3
qa4 = qa & mask4
qa5 = qa & mask5

print 'qa1',bin(qa1)
print 'qa2',bin(qa2)
print 'qa3',bin(qa3)
print 'qa4',bin(qa4)
print 'qa5',bin(qa5)

qa 255
binary qa 0b11111111
qa1 0b1
qa2 0b10
qa3 0b100
qa4 0b11000
qa5 0b11100000
```

We gave an example qa value with 1 in all bit fields, and can see that this information has been passed through above, but we still need to remove the trailing zeros to the right of the bit field.

We can easily do this with a right shift operator:

```
qa1 = (qa & 0b00000001) >> 0      # bit 0
qa2 = (qa & 0b00000010) >> 1      # bit 1
qa3 = (qa & 0b00000100) >> 2      # bit 2
qa4 = (qa & 0b00011000) >> 3      # bit 3-4
qa5 = (qa & 0b11100000) >> 5      # bit 5-7

print 'qa1',bin(qa1)
print 'qa2',bin(qa2)
print 'qa3',bin(qa3)
print 'qa4',bin(qa4)
print 'qa5',bin(qa5)

qa1 0b1
qa2 0b1
qa3 0b1
qa4 0b11
qa5 0b111
```

And now we see that we have *just* the information we required.

There are many ways to achieve the same result in filtering such QA masks, but it is quite easy to make a mistake in doing so, so the *clearest* way to do this is generally as above:

- develop a bit mask with 1 in the required fields
 - e.g. `mask4 = 0b00011000 # bit 3-4`
 - e.g. `mask1 = 0b00000001 # bit 0`
- do a bitwise and (&) between the data and the bit mask
- and right shift the result by the ‘starting’ bit
 - e.g. `(qa & 0b00011000) >> 3 # bit 3-4`

It is always a good idea to test the application of such masks by setting 1 in all fields, as you can easily see if you get the result you should be expecting.

16.8 Exercise A2.3

Develop some code to repeat the QA bit masking done above, but make the code generate the bit masks itself from knowledge of the first and last of the bit fields you require (assuming they are sequential).

If possible, do this in a function.

Demonstrate its operation with several example bit masks.

CHAPTER
SEVENTEEN

A2.2 EXCEPTION HANDLING

In Python, the mechanism for trapping errors (i.e. when something goes wrong in running a block of code) is:

`try:`

`...`

`except:`

`...`

e.g.:

```
a = 1
b = 0
print a/b
```

```
-----
```

```
ZeroDivisionError                                Traceback (most recent call last)
```

```
<ipython-input-20-e0a9b2305aea> in <module>()
      1 a = 1
      2 b = 0
----> 3 print a/b
```

```
ZeroDivisionError: integer division or modulo by zero
```

```
a = 1
b = 0
```

```
'''
```

*What do we want to happen if we try to divide by zero?
In this case, we decide we want to set the result to zero*

```
'''
```

`try:`

`print a/b`

`except ZeroDivisionError:`

`print 0`

```
0
```

Note the indentation in the code. This is what defines the hierarchy of conditional statements in Python (rather than some explicit ‘start’ and ‘end’ statements, as in some other languages).

We can trap *explicit* types of error, e.g.:

```
x = 100.
del x

try:
    print x
except NameError:
    print "name 'x' is not defined"
```

name 'x' is not defined

or we can be non-specific and trap all errors in a code block:

```
try:  
    print '2' + 2      # adding a string to integer doesn't make sense  
except:  
    print 'you made an error'
```

you made an error

You can trap multiple forms of error:

```
import sys  
  
filename = 'files/data/progress.dat'  
  
try:  
    f = open(filename)  
    s = f.readlines()  
    print s[0]  
    print s[0]/10  
except IOError as e:  
    # trap a specific IOError  
    print "I/O error reading from %s"%(filename)  
except:  
    # trap any further errors  
    print "Error in processing data from file %s"%filename
```

AS I walk'd through the wilderness of this world,

Error in processing data from file files/data/progress.dat

If you want to know more on this subject, you can follow this up with a more detailed [tutorial](#).

A2.3 MORE COMPLEX SPLITTING EXAMPLES

```
# We can use separators other than whitespace with split.

data = "1964,1220\n1974,2470\n1984,2706\n1994,4812\n2004,2707"
print "data.split('\\n'):\n\t", data.split('\n')
# note '\n' in here as '\n' would have printed a newline
# \ is an escape character (as in unix)

data.split(`n`):
    [`1964,1220`, `1974,2470`, `1984,2706`, `1994,4812`, `2004,2707`]

# using a generator expressions

fdata1 = (i.split() for i in data.split('\n'))

print list(fdata1)

[['1964,1220'], ['1974,2470'], ['1984,2706'], ['1994,4812'], ['2004,2707']]

# two listcomps to split first
# on \n and then each of these on comma

fdata1 = [i.split() for i in data.split('\n')]
fdata2 = [j[0].split(',') for j in fdata1]

print fdata2

[['1964', '1220'], ['1974', '2470'], ['1984', '2706'], ['1994', '4812'], ['2004', '2707']]

fdata1 = [i.split() for i in data.split('\n')]
fdata2 = [j[0].split(',') for j in fdata1]
fdata = [[int(l) for l in k] for k in fdata2]

print fdata

[[1964, 1220], [1974, 2470], [1984, 2706], [1994, 4812], [2004, 2707]]

# or all at once, which works
# but now the meaning of code is too obscure
# so this is poor style

fdata = [[int(l) for l in k] for k in [j[0].split(',') \
    for j in [i.split() for i in data.split('\n')]]]

print fdata

[[1964, 1220], [1974, 2470], [1984, 2706], [1994, 4812], [2004, 2707]]
```

```
# It might be comprehensible if laid out better

fdata = [[int(l) \
          for l in k] \
          for k in [j[0].split(',') \
                    for j in [i.split() \
                               for i in data.split('\n')]]]

# even so you might find this confusing
# and prefer to do it in smaller chunks

print fdata

[[1964, 1220], [1974, 2470], [1984, 2706], [1994, 4812], [2004, 2707]]
```

A2.4 STRING FORMATTING

You can control spaces, and precision e.g.:

```
print "%20s %s"%'hello','world')
print "%-20s %s"%'hello','world')

                           hello world
hello                  world

pi = 3.1415926536
for format in ["%3.2f", "%3.20f", "%9.5f", "%020.3f"]:
    print format, '\n\t:', format%pi

%3.2f
: 3.14
%3.20f
: 3.14159265360000006240
%9.5f
: 3.14159
%020.3f
: 0000000000000003.142

pi = 3.1415926536
for format in ["%3.2e", "%3.20e", "%9.5e", "%020.3e"]:
    print format, '\n\t:', format%pi

%3.2e
: 3.14e+00
%3.20e
: 3.14159265360000006240e+00
%9.5e
: 3.14159e+00
%020.3e
: 000000000003.142e+00

i = 100
for format in ["%i", "%3i", "%5i", "%08i"]:
    print format, '\n\t:', format%i

%i
: 100
%3i
: 100
%5i
: 100
%08i
: 00000100
```

There are other (and in some ways more elegant) ways to format strings in Python. e.g.

```
print "Hey {1}, Hello {0}".format('world', 'you')
print '{0}{1}{0}'.format('abra', 'cad')

# date formatting is a particular example
from datetime import datetime
print """
At the third stroke, it will be
{:%Y-%m-%d %H:%M:%S}
Beep
Beep
Beep
""".format(datetime.now(), 'beep')

Hey you, Hello world
abracadabra

At the third stroke, it will be
2014-09-30 11:11:37
Beep
Beep
Beep
```

There are also additional formatting codes, including:

```
# character
print "char: %c %c %c"%(100,101,102)

# octal
print "octal: %o"%(0o755)

# hex
print "hex: %X"%(0xFFFF)
print "hex: %x"%(0xFFFF)

char: d e f
octal: 755
hex: FFF
hex: fff
```

A2.5 FILE SEPARATOR: PORTABILITY

We noted earlier that in parsing a filename, you might want to split it by the file separator:

```
# example, with directory names

# glob unix style pattern matching for files and directories
import glob

# returns a list (or []) if empty
# to match the pattern given
file_list = glob.glob("files/data/*.txt")
print "file_list:\n\t",file_list

# e.g. the first string in the list
this_file = file_list[0]

# split the filename on the field '/'
print "\nthis_file.split('/'): \n\t",this_file.split('/')

# so the filename is just the last element in this list
print "\nthis_file.split('/')[-1]: \n\t",this_file.split('/')[-1]

file_list:
    ['files/data/HadSEEP_monthly_qc.txt', 'files/data/heathrowdata.txt', 'files/data/modis_files.txt']

this_file.split('/'):
    ['files', 'data', 'HadSEEP_monthly_qc.txt']

this_file.split('/')[-1]:
    HadSEEP_monthly_qc.txt
```

That's fine if you are on a unix system, but on a windows system, the file separator is the other way around, so this code is not portable.

To make aspects of your code like this portable, we can use features from Python that will tend to be in the module `os`:

```
from os import sep # import the string sep from the module os

print "On this system, the file separator is",sep
```

On this system, the file separator is /

So, better code than the above is:

```
# example, with directory names

# glob unix style pattern matching for files and directories
import glob
from os import sep
```

```
# returns a list (or [] if empty)
# to match the pattern given
file_list = glob.glob("files{0}data{0}*.txt".format(sep))
print "file_list:\n\t",file_list

# e.g. the first string in the list
this_file = file_list[0]

# split the filename on the field '/' or '\' (sep)
print "\nthis_file.split({0}):\n\t".format(sep),\
      this_file.split(sep)

# so the filename is just the last element in this list
print "\nthis_file.split({0})[-1]:\n\t".format(sep),\
      this_file.split(sep)[-1]

file_list:
['files/data/HadSEEP_monthly_qc.txt', 'files/data/heathrowdata.txt', 'files/data/modis_files.txt']

this_file.split():
['files', 'data', 'HadSEEP_monthly_qc.txt']

this_file.split()[-1]:
HadSEEP_monthly_qc.txt
```

There will always be a compromise between portability and readability of the code, so try to make sure that any portability modifications you make don't obscure the meaning of the code.

A2.7 INSTALLING YOUR OWN PACKAGES

Most of what you will want (especially to begin with) will be contained in your current Python distribution.

Sometimes though, you might need to install some new module.

Generally, the easiest way to do this is to use `easy_install`.

We will use this to install a package `pyephem` into your user area:

at a unix prompt, type:

```
easy_install --user pyephem
```

If all goes well, the text that comes up at the terminal should tell you that this has installed (e.g. in `/home/plewis/.local/lib/python2.7/site-packages/pyephem-3.7.5.1-py2.7-linux-x86_64.egg`).

We can test to see if we can load this package:

```
import ephem
```

and then do some interesting things with it.

The following piece of code might seem a little complicated right now, but it just uses

We will need to import some other things from a few other packages as well:

```
from ephem import Sun, Observer
from math import pi
from datetime import date, datetime, time

# radians to degrees
rtod = 180./pi

# observer information in pyephem
obs = Observer()

today = date.today()
# or datetime.date(2013, 3, 12)

# put in lat / lon for UCL
# https://www.google.co.uk/search?q=ucl+longitude+latitude
obs.lat = '51.5248'
obs.long = '-0.1336'

for hour in xrange(0,24):
    for minute in xrange(0,60,30):
        t = time(hour, minute, 0)
        obs.date = datetime.combine(today, t)
        sun = Sun(obs)
        zenith_elevation = float(sun.alt)*rtod # degrees

        print obs.date, zenith_elevation
```

```
2014/9/30 00:00:00 -41.1227180114
2014/9/30 00:30:00 -40.4796625312
2014/9/30 00:59:59 -39.0649015428
2014/9/30 01:30:00 -36.9463783528
2014/9/30 02:00:00 -34.2136162508
2014/9/30 02:30:00 -30.9643006022
2014/9/30 03:00:00 -27.2941667536
2014/9/30 03:30:00 -23.2912319377
2014/9/30 03:59:59 -19.0337171884
2014/9/30 04:30:00 -14.5903017653
2014/9/30 05:00:00 -10.0216590927
2014/9/30 05:30:00 -5.01932715546
2014/9/30 06:00:00 -0.134849371939
2014/9/30 06:30:00 4.09286322097
2014/9/30 06:59:59 8.55453445901
2014/9/30 07:30:00 12.938958392
2014/9/30 08:00:00 17.1514399965
2014/9/30 08:30:00 21.1181133381
2014/9/30 09:00:00 24.7672136192
2014/9/30 09:30:00 28.020436136
2014/9/30 09:59:59 30.7949255712
2014/9/30 10:30:00 33.0070257679
2014/9/30 11:00:00 34.579161145
2014/9/30 11:30:00 35.4491052641
2014/9/30 12:00:00 35.5794764999
2014/9/30 12:30:00 34.9643838142
2014/9/30 12:59:59 33.6305127561
2014/9/30 13:30:00 31.6322282163
2014/9/30 14:00:00 29.0427156717
2014/9/30 14:30:00 25.9443096378
2014/9/30 15:00:00 22.4206252902
2014/9/30 15:30:00 18.5516782953
2014/9/30 15:59:59 14.4111578562
2014/9/30 16:30:00 10.0714597095
2014/9/30 17:00:00 5.61372180666
2014/9/30 17:30:00 1.2203105187
2014/9/30 18:00:00 -2.38322183516
2014/9/30 18:30:00 -8.47981048194
2014/9/30 18:59:59 -13.0951972617
2014/9/30 19:30:00 -17.6077341991
2014/9/30 20:00:00 -21.9594218892
2014/9/30 20:30:00 -26.0849005441
2014/9/30 21:00:00 -29.9096032812
2014/9/30 21:30:00 -33.3488186792
2014/9/30 21:59:59 -36.3086640806
2014/9/30 22:30:00 -38.6902793272
2014/9/30 23:00:00 -40.3983491291
2014/9/30 23:30:00 -41.3537969714
```

Let's write this out to a file now:

```
from ephem import Sun, Observer
from math import pi
from datetime import date, datetime, time

filename = 'files/data/elevation.dat'
try:
    fp = open(filename, "w")
except:
    print "Failed to open file %s for writing"%filename

# radians to degrees
rtod = 180./pi
```

```
# observer information in pyephem
obs = Observer()

today = date.today()
# or datetime.date(2013, 3, 12)

# put in lat / lon for UCL
# https://www.google.co.uk/search?q=ucl+longitude+latitude
obs.lat = '51.5248'
obs.long = '-0.1336'

for hour in xrange(0,24):
    for minute in xrange(0,60,30):
        t = time(hour, minute, 0)
        obs.date = datetime.combine(today, t)
        sun = Sun(obs)
        elevation = float(sun.alt)*rtod # degrees
        fp.write("%s %s\n"%(obs.date,elevation))

fp.close()

# just for interest, let's plot this:

import pylab as plt

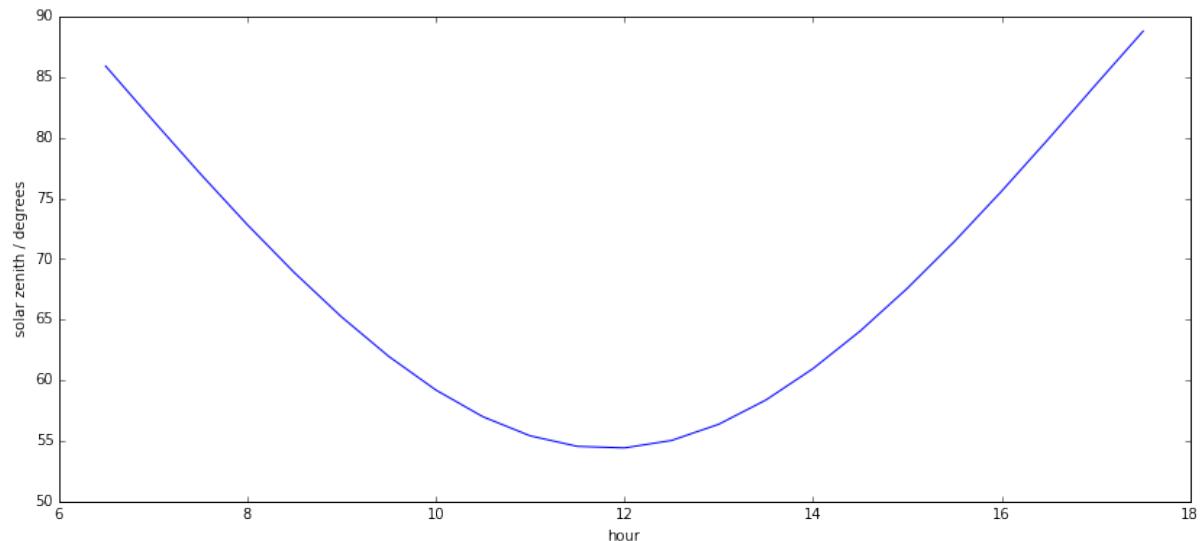
filename = 'files/data/elevation.dat'
fp = open(filename,"r")

hours = []
zeniths = []

for i in fp.readlines():
    data = i.split()
    time = [float(i) for i in data[1].split(':')]
    hour = time[0] + time[1]/60. + time[2]/(60.*60)
    zenith = 90. - float(data[2])
    if zenith <= 90.:
        hours.append(hour)
        zeniths.append(zenith)
fp.close()

# plotting
fig = plt.figure(figsize=(14, 6))
plt.plot(hours,zeniths)
plt.xlabel('hour')
plt.ylabel('solar zenith / degrees')

<matplotlib.text.Text at 0x106574710>
```



CHAPTER
TWENTYTWO

A2.8 READING FILES FROM A URL

In the main exercises, you have been advised to download a file and read it using the normal file `open` commands. If a file is available as a [URL](#), you can conveniently use the `urllib2` module in Python to access the file directly. So, in the case of the precipitation data in an earlier exercise, we could have directly read the data with:

```
import urllib2

#as files
url = "http://www.metoffice.gov.uk/hadobs/hadukp/data/monthly/HadSEEP_monthly_qc.txt"

req = urllib2.Request ( url )
raw_data = urllib2.urlopen(req).readlines()

print raw_data[:6]

['Monthly Southeast England precipitation (mm). Daily automated values used after 1996.r'
```


A2.9 ADVANCED DICTIONARIES

They are particularly useful when you want to build clear data structures, e.g. in loading a dataset such as `files/data/HadSEEP_monthly_qc.txt <files/data/HadSEEP_monthly_qc.txt>`__.

First, let's read the data from this file into the lists `label` and `rdata`, where `rdata[0]` will be the first column in the file, `rdata[1]` the second column etc.

```
fp = open('files/data/HadSEEP_monthly_qc.txt', 'r')
sdata = fp.readlines()
fp.close()

labels = sdata[3].split()
data = [i.split() for i in sdata[4:]]
j = 0
rdata = [[float(data[i][j]) \
          for i in xrange(len(data))] \
          for j in xrange(len(data[0]))]

print rdata[0]

[1873.0, 1874.0, 1875.0, 1876.0, 1877.0, 1878.0, 1879.0, 1880.0, 1881.0, 1882.0, 1883.0, 1884.0,
```

We can of course just remember that the first column is 'year', the second column data for 'JAN' etc., but we would have a much better organised dataset if we put it in a dictionary.

One way to do this would be:

```
# set up a
dataset = {}
for i,k in enumerate(labels):
    dataset[k] = rdata[i]
```

and we could now refer to the 'year' data as:

```
print dataset['YEAR']

[1873.0, 1874.0, 1875.0, 1876.0, 1877.0, 1878.0, 1879.0, 1880.0, 1881.0, 1882.0, 1883.0, 1884.0,
```

A simper and more Pythonic way is to use `zip`:

```
dataset = dict(zip(labels,rdata))

print dataset['YEAR']

[1873.0, 1874.0, 1875.0, 1876.0, 1877.0, 1878.0, 1879.0, 1880.0, 1881.0, 1882.0, 1883.0, 1884.0,
```

So, e.g. plotting data sets would be clear using the dictionary:

```
import pylab as plt

for i,JFM in enumerate([('JAN','FEB','MAR'),('APR','MAY','JUN'),\
                        ('JUL','AUG','SEP'),('OCT','NOV','DEC')):
```

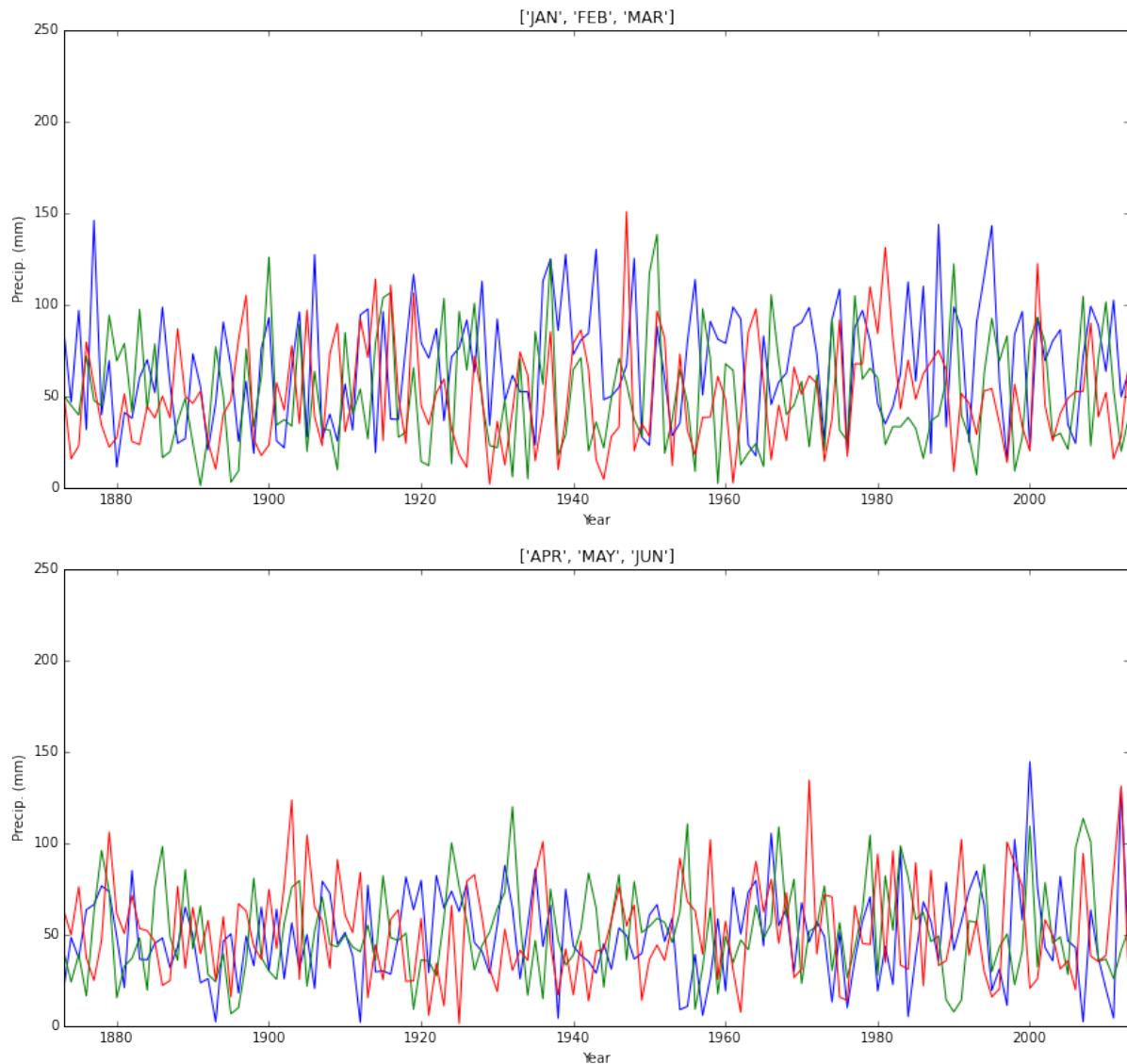
```

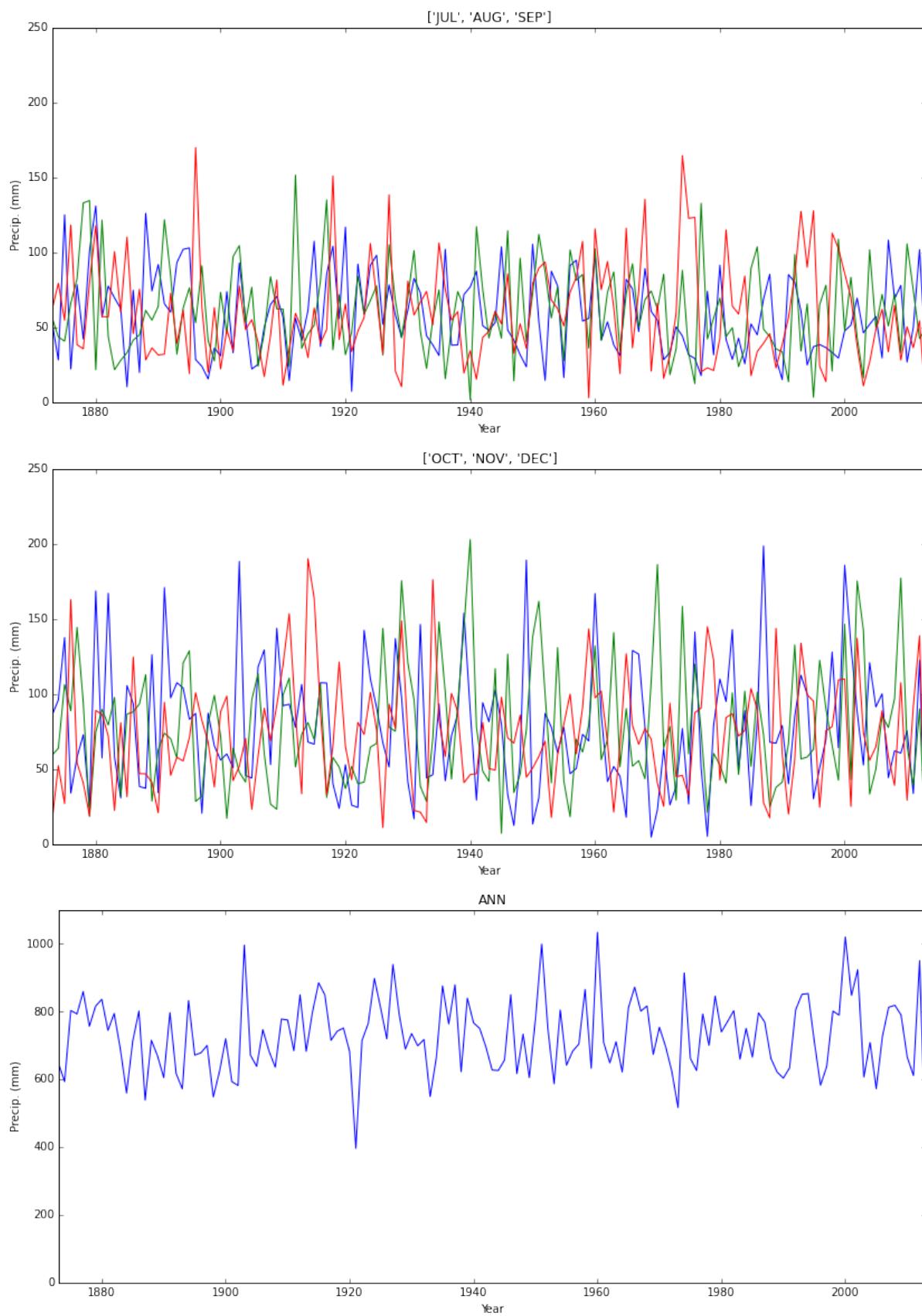
plt.figure(i, figsize=(14, 6))
plt.xlabel('Year')
plt.ylabel('Precip. (mm)')
for month in JFM:
    plt.ylim(0, 250)
    plt.xlim(dataset['YEAR'][0], dataset['YEAR'][-1])
    plt.title(str(JFM))
    plt.plot(dataset['YEAR'], dataset[month])

plt.figure(i+1, figsize=(14, 6))
plt.xlabel('Year')
plt.ylabel('Precip. (mm)')
plt.title('ANN')
plt.ylim(0, 1100)
plt.xlim(dataset['YEAR'][0], dataset['YEAR'][-1])
plt.plot(dataset['YEAR'], dataset['ANN'])

[<matplotlib.lines.Line2D at 0x107593850>]

```





CHAPTER
TWENTYFOUR

EXCERCISE

In this example, we want to read all data from the Heathrow Met data file `files/data/heathrowdata.txt` and plot each data field.

In doing this, we want to store the data in a dictionary, as this is a convenient way of access the different data fields.

At the heart of this example is the statement

```
dict(zip(labels,rdata))
```

This is a very useful combination in Python as it allows us to generate a dictionary with the keys specified by `labels` and the values `rdata`.

A few things that make this a bit more complicated are:

1. When we read the data line by line from the file, we have it stored as `data[line][sample]`, i.e. each `data[line]` is the data for that line. When we want to load it into the dictionary, we want the data the other way around. This is a bit tricky with lists, and we have to use the following, which is a general statement, so you can use it for similar rotations, but its a bit dense.

```
rdata = [[data[j][i] for j in xrange(len(data))] for i in xrange(len(data[0]))]
```

2. When parsing the file, we have to deal with some awkward aspects, such as the presence of `#` symbols here and there and the fact that `'---` is used to indicate no data. Here, we replace these awkward fields with more convenient numerical values (that won't break the conversion to `float`), or get rid of them as appropriate (using `replace()` on the strings).
3. When we come to plot the data, we have to confront the fact that we used `-999` to indicate no data, and have to filter these values out before plotting. This is easy enough to do in a `listcomp` for the `y` values by using a conditional statement:

```
y = [i for i in items[label] if i != -999]
```

But for the `x` values, we have to filter these based on a test with the `y` values ... and so we use a more complex statement:

```
x = [time[j] for j,i in enumerate(items[label]) if i != -999]
```

4. Finally, lists are not very convenient for numerical operations on the lists ... we can't simply add `[0.1,0.2]` to `[3.0,4.0]` to get `[3.1,4.2]`, because `+` is a concatenation operator for lists. Instead, we have to loop over all of the items in the lists:

```
time = [items['year'][i] + items['month'][i]/12. for i in xrange(len(items['year']))]
```

We will see next time that there are more convenient modules for doing this sort of thing, but it's of value to know how you can do it if you only have lists.

One final comment is that we have made extensive use of `listcomps` here, which make the code more compact but may be more difficult for people just starting coding to follow.

```

# Read files/data/heathrowdata.txt as above into data as float
# doing some clever things to deal with file markers
# i.e. where it says # Provisional or # we will not worry about
# that for the moment
# where it has fillers, ---, replace by -999

fp = open('files/data/heathrowdata.txt','r')
ldata = [i.replace('---','-999').replace('# Provisional','').replace('#','').split() \
          for i in fp.readlines()[7:]]
data = [[float(k) for k in j] for j in ldata]
fp.close()

# The reading by line is a bit inconvenient, so rotate the
# dataset
rdata = [[data[j][i] for j in xrange(len(data))] for i in xrange(len(data[0]))]

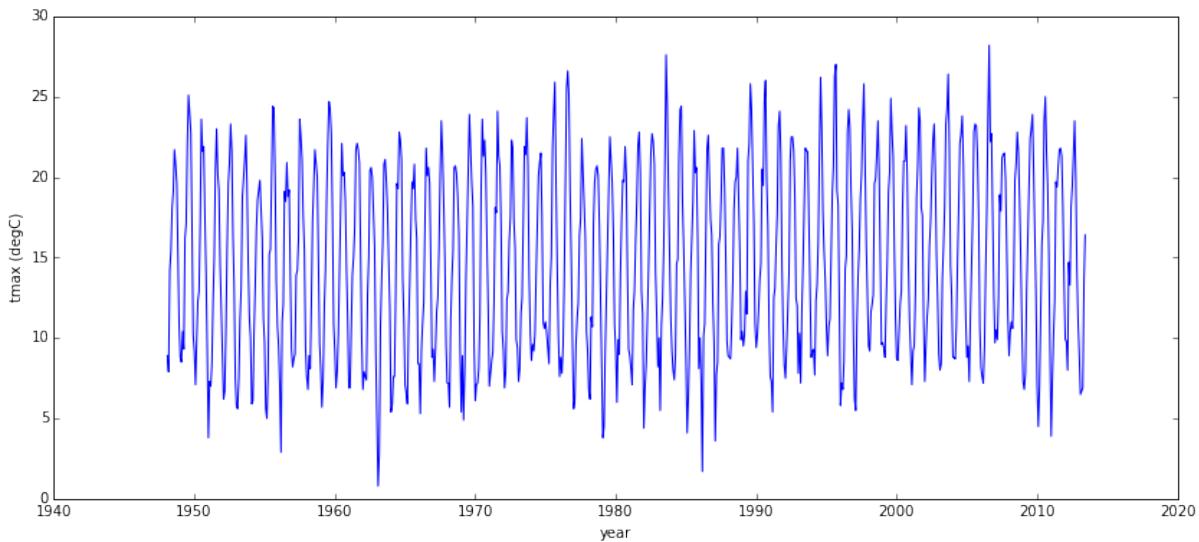
# set some labels for each column
labels = ['year','month','tmax (degC)','tmin (degC)',\
           'air frost (days)','rain (mm)','sun (hours)']

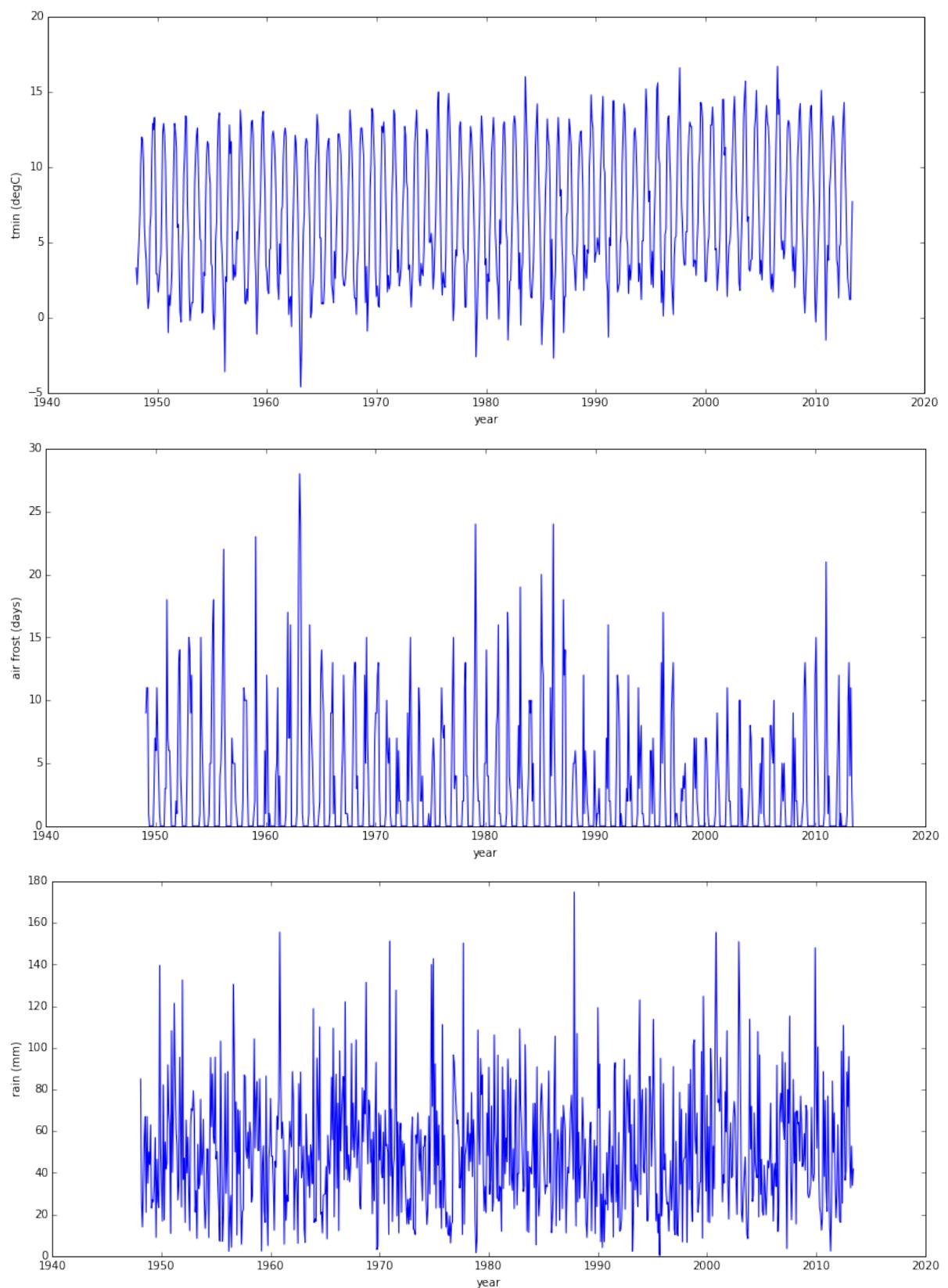
# generate a dictionary with the labels as keys
items = dict(zip(labels,rdata))

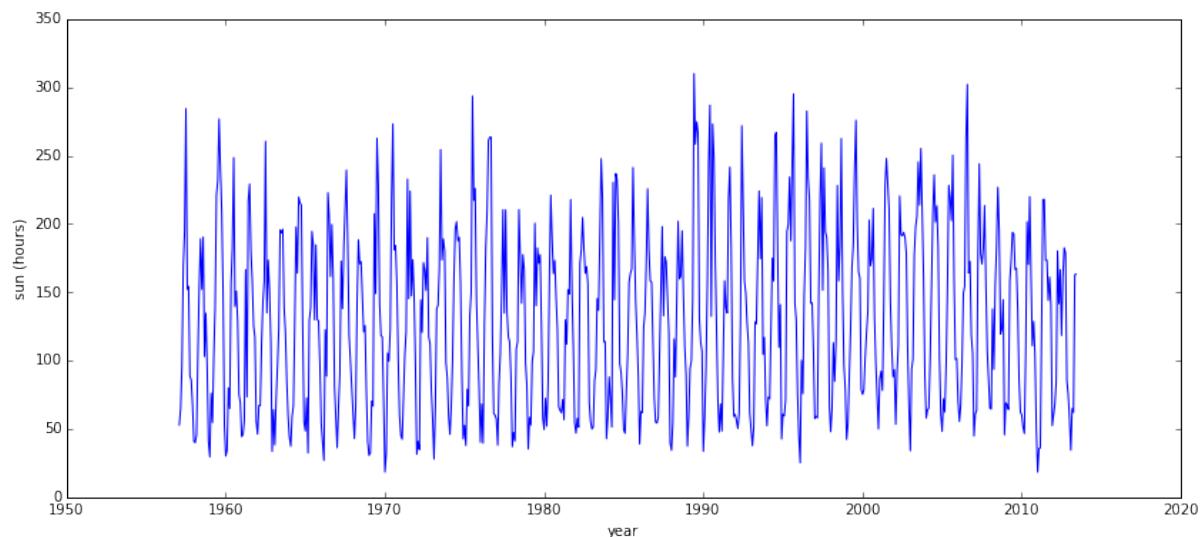
import pylab as plt
# its quite difficult to add arrays
# so we have to loop over each item
time = [items['year'][i] + items['month'][i]/12. for i in xrange(len(items['year']))]

for i,label in enumerate(labels[2:]):
    plt.figure(i,figsize=(14, 6))
    plt.xlabel('year')
    plt.ylabel(label)
    # filter out the -999 values
    x = [time[j] for j,i in enumerate(items[label]) if i != -999]
    y = [i for i in items[label] if i != -999]
    plt.plot(x,y)

```







CHAPTER
TWENTYFIVE

AE2. ADVANCED EXERCISES

- Exercise A2.1
- Exercise A2.2
- Exercise A2.3

Remember that these exercises are more advanced. You are not *required* to do these, but may like to do so if you want to stretch yourself.

25.1 Exercise A2.1

As an exercise for this, you could see if you can simulate the logical combinations xor, nor and nand, e.g.:

```
# nand test:  
# see http://en.wikipedia.org/wiki/Logical\_NAND  
# (A nand B) is not (A and B)  
  
ABList = [(False,False), (False,True), (True,False), (True,True)]  
for A,B in ABList:  
    print '%s nand %s ='%(str(A),str(B)),not (A and B)  
  
False nand False  = True  
False nand True   = True  
True nand False   = True  
True nand True    = False
```

25.2 Answer

```
""" Exercise 2.1 Scientific Computing  
  
Thu 3 Oct 2013 10:46:58 BST  
  
P. Lewis : p.lewis@ucl.ac.uk  
  
nand test  
  
see http://en.wikipedia.org/wiki/Logical\_NAND  
(A nand B) is not (A and B)  
  
It is a modification of the code in Exercise A2.1  
of Scientific Computing (https://github.com/profLewis/geogg122)  
  
Edits made from original:  
- neatened output format  
"""
```

```
# set up a list of tuples to loop over
ABList = [(False,False), (False,True), (True,False), (True,True)]
```

```
# loop over tuples and print A xor B
# with tabs (\t) for neater output
for A,B in ABList:
    print '%s\tand\t%s\t=%s\n' % (str(A), str(B)), not (A and B)
```

False	nand	False	= True
False	nand	True	= True
True	nand	False	= True
True	nand	True	= False

""" Exercise 2.1 Scientific Computing

Thu 3 Oct 2013 10:46:58 BST

P. Lewis : p.lewis@ucl.ac.uk

nor test

see http://en.wikipedia.org/wiki/Logical_NOR

(A nor B) is not (A or B)

It is a modification of the code in Exercise A2.1
of Scientific Computing (<https://github.com/profLewis/geogg122>)

Edits made from original:

- neatened output format
- modified from nand to nor

"""

```
# set up a list of tuples to loop over
ABList = [(False,False), (False,True), (True,False), (True,True)]
```

```
# loop over tuples and print A xor B
# with tabs (\t) for neater output
for A,B in ABList:
    print '%s\tnor\t%s\t=%s\n' % (str(A), str(B)), not (A or B)
```

False	nor	False	= True
False	nor	True	= False
True	nor	False	= False
True	nor	True	= False

""" Exercise 2.1 Scientific Computing

Thu 3 Oct 2013 10:46:58 BST

P. Lewis : p.lewis@ucl.ac.uk

xor test

see http://en.wikipedia.org/wiki/Logical_XOR

(A xor B) is (A or B) and not (A and B)

It is a modification of the code in Exercise A2.1
of Scientific Computing (<https://github.com/profLewis/geogg122>)

Edits made from original:

```

    - neatened output format
    - modified from nand to xor
"""

# set up a list of tuples to loop over
ABList = [(False, False), (False, True), (True, False), (True, True)]

# loop over tuples and print A xor B
# with tabs (\t) for neater output
for A,B in ABList:
    print '%s\xor\t%s\t=%s\n' % (str(A), str(B)), (A or B) and not (A and B)

False      xor      False      = False
False      xor      True       = True
True       xor      False      = True
True       xor      True       = False

```

25.3 Exercise A2.2

1. Work out what the following decimal numbers are in binary:

7
493
127
255
1024

and check your result using the approach we took to confirming 101:

```
101 == (1 * 2**6) + \
        (1 * 2**5) + \
        (0 * 2**4) + \
        (0 * 2**3) + \
        (1 * 2**2) + \
        (0 * 2**1) + \
        (1 * 2**0)
```

2. How many bits are needed to represent each of these numbers?
3. What is the largest number you could represent in: (i) a 32 bit representation; (b) a 64 bit representation?
4. Recalling that there are 8 bits in a byte, what is the largest number you could represent in: (a) a single byte; (b) two bytes?

25.4 Answer

- a. Work out what the following decimal numbers are in binary:

7
493
127
255
1024

```
for n in (7, 493, 127, 255, 1024):
    print n, '\tin binary is', bin(n)

7    in binary is 0b111
493     in binary is 0b111101101
127     in binary is 0b1111111
255     in binary is 0b11111111
1024    in binary is 0b100000000000
```

check your result using the approach we took to confirming “101“:

Long-hand way:

```
7 == (1 * 2**2) + \
      (1 * 2**1) + \
      (1 * 2**0)
```

True

```
493 == (1 * 2**8) + \
        (1 * 2**7) + \
        (1 * 2**6) + \
        (1 * 2**5) + \
        (0 * 2**4) + \
        (1 * 2**3) + \
        (1 * 2**2) + \
        (0 * 2**1) + \
        (1 * 2**0)
```

True

```
127 == (1 * 2**6) + \
        (1 * 2**5) + \
        (1 * 2**4) + \
        (1 * 2**3) + \
        (1 * 2**2) + \
        (1 * 2**1) + \
        (1 * 2**0)
```

True

```
255 == (1 * 2**7) + \
        (1 * 2**6) + \
        (1 * 2**5) + \
        (1 * 2**4) + \
        (1 * 2**3) + \
        (1 * 2**2) + \
        (1 * 2**1) + \
        (1 * 2**0)
```

True

```
1024 == (1 * 2**10) + \
          (0 * 2**9) + \
          (0 * 2**8) + \
          (0 * 2**7) + \
          (0 * 2**6) + \
          (0 * 2**5) + \
          (0 * 2**4) + \
          (0 * 2**3) + \
          (0 * 2**2) + \
          (0 * 2**1) + \
          (0 * 2**0)
```

True

More automatic/advanced way (but more complicated coding):

```
# loop over numbers
for n in (7, 493, 127, 255, 1024):

    # convert decimal to binary
    # then convert to string and ignore 0b at start
    binstr = bin(n)[2:]

    # initialise sum as accumulator
    sum = 0

    # how many bits in this case?
    nBits = len(binstr)
    print n, '\tin binary is', binstr, ": %d bits" %nBits

    # loop over each bit
    for c in xrange(nBits):

        # extract the bit and exponent
        bit = int(binstr[-(c+1)])
        exp = 2**c
        sum += exp * bit
        print '\t%d x 2^%d = %d x %d \t%d' %(bit,c,bit,exp,sum)

    print "%d == %d?" %(n,sum), n == sum
    print "=====
```

7 in binary is 111 : 3 bits
 1 x 2^0 = 1 x 1 1
 1 x 2^1 = 1 x 2 3
 1 x 2^2 = 1 x 4 7
7 == 7? True
=====

493 in binary is 111101101 : 9 bits
 1 x 2^0 = 1 x 1 1
 0 x 2^1 = 0 x 2 1
 1 x 2^2 = 1 x 4 5
 1 x 2^3 = 1 x 8 13
 0 x 2^4 = 0 x 16 13
 1 x 2^5 = 1 x 32 45
 1 x 2^6 = 1 x 64 109
 1 x 2^7 = 1 x 128 237
 1 x 2^8 = 1 x 256 493
493 == 493? True
=====

127 in binary is 1111111 : 7 bits
 1 x 2^0 = 1 x 1 1
 1 x 2^1 = 1 x 2 3
 1 x 2^2 = 1 x 4 7
 1 x 2^3 = 1 x 8 15
 1 x 2^4 = 1 x 16 31
 1 x 2^5 = 1 x 32 63
 1 x 2^6 = 1 x 64 127
127 == 127? True
=====

255 in binary is 11111111 : 8 bits
 1 x 2^0 = 1 x 1 1
 1 x 2^1 = 1 x 2 3
 1 x 2^2 = 1 x 4 7
 1 x 2^3 = 1 x 8 15

```
1 x 2^4 = 1 x 16          31
1 x 2^5 = 1 x 32          63
1 x 2^6 = 1 x 64          127
1 x 2^7 = 1 x 128         255
255 == 255? True
=====
1024      in binary is 10000000000 : 11 bits
0 x 2^0 = 0 x 1            0
0 x 2^1 = 0 x 2            0
0 x 2^2 = 0 x 4            0
0 x 2^3 = 0 x 8            0
0 x 2^4 = 0 x 16           0
0 x 2^5 = 0 x 32           0
0 x 2^6 = 0 x 64           0
0 x 2^7 = 0 x 128          0
0 x 2^8 = 0 x 256          0
0 x 2^9 = 0 x 512          0
1 x 2^10 = 1 x 1024        1024
1024 == 1024? True
=====
```

If you enjoyed this part of the exercise ;-), you could repeat this part of it, but using base 8 (octal) and or base 16 (hexadecimal), which you will often come across in computing. We might use octal e.g. in unix file permissions, or hexadecimal more widely in memory addresses (because it would be too long a number in binary!).

e.g.:

```
n = 493
# N.B., no 0b or 0x part on the string for octal, just a
# leading 0
octstr = oct(n)[1:]
sum = 0
# how many octets in this case? (assuming that's the right word)
nOct = len(octstr)
print n, '\tin octal is', octstr, ":", %d octets" %nOct

# loop over each octets
for c in xrange(nOct):
    # extract the octet and exponent
    octet = int(octstr[-(c+1)])
    exp = 8**c
    sum += exp * octet
    print '\t %d x 8^%d = %d x %d \t%d' %(octet, c, octet, exp, sum)
print "%d == %d?" %(n, sum), n == sum
print "====="

493      in octal is 755 : 3 octets
5 x 8^0 = 5 x 1            5
5 x 8^1 = 5 x 8            45
7 x 8^2 = 7 x 64           493
493 == 493? True
=====
```

b. How many bits are needed to represent each of these numbers?

```
# bin() converts to binary, but really its a string
# the first 2 characters of which are '0b'
# so find the length of the string other than the first two
# characters

for n in (7, 493, 127, 255, 1024):
    binstr = bin(n)[2:]
    print binstr, len(binstr), 'bits'
```

```

111 3 bits
111101101 9 bits
1111111 7 bits
11111111 8 bits
10000000000 11 bits

```

c. What is the largest number you could represent in: (i) a 32 bit representation; (b) a 64 bit representation?

In an *unsigned* integer representation, the largest number would be one with 1 in all bit fields.

We *could* add this up, but it's faster to notice that this is one less than two to the power of the number of bits, e.g. the largest number with 8 bits is $2^8 - 1$:

```
print "the largest (unsigned) integer with 8 bits (1 byte) is", 2**8 - 1
```

```
the largest (unsigned) integer with 8 bits (1 byte) is 255
```

```
print "the largest (unsigned) integer with 32 bits is", 2**32 - 1
print "the largest (unsigned) integer with 64 bits is", 2**64 - 1
```

```
the largest (unsigned) integer with 32 bits is 4294967295
```

```
the largest (unsigned) integer with 64 bits is 18446744073709551615
```

Sometimes, a *signed* integer representation is used, in which case the leftmost bit (usually) of the bit field is used to represent the *sign* (0 meaning +ve and 1 meaning -ve), so e.g.:

011 would be +7 in decimal but

111 would be -7 in decimal

This means that there is one fewer bit to represent the magnitude of the number, so e.g. with 8 bits (one byte) you have one bit for the sign, and 7 bits for magnitude:

```
print "the largest (signed) integer with 8 bits (1 byte) is", +2**8 - 1
print "the smallest (signed) integer with 8 bits (1 byte) is", -2**8 + 1
```

```
the largest (signed) integer with 8 bits (1 byte) is 127
```

```
the smallest (signed) integer with 8 bits (1 byte) is -127
```

Interestingly, in a signed integer representation, there are two numbers that represent zero:

```
000
```

```
100
```

which effectively mean +0 and -0. In that sense, the signed representation is a little wasteful (but you only lose one number!).

****d.** Recalling that there are 8 bits in a *byte*, what is the largest number you could represent in: (a) a single byte; (b) two bytes? **

So, one byte is 8 bits.

As we saw above, for an unsigned representation this can have integers from 0 to 255.

For a signed representation this can have integers from -127 to +127.

Two bytes is 16 bits, so:

```
print "the largest (unsigned) integer with 16 bits (2 bytes) is", 2**16 - 1
print "the largest (signed) integer with 16 bits (2 bytes) is", +2**16 - 1
print "the smallest (signed) integer with 16 bits (2 bytes) is", -2**16 + 1
```

```
the largest (unsigned) integer with 16 bits (2 bytes) is 65535
the largest (signed) integer with 16 bits (2 bytes) is 32767
the smallest (signed) integer with 16 bits (2 bytes) is -32767
```

25.5 Exercise A2.3

```
qa = 0b11111111

qa1 = (qa & 0b00000001) >> 0      # bit 0
qa2 = (qa & 0b00000010) >> 1      # bit 1
qa3 = (qa & 0b00000100) >> 2      # bit 2
qa4 = (qa & 0b00011000) >> 3      # bit 3-4
qa5 = (qa & 0b11100000) >> 5      # bit 5-7

print 'qa1',bin(qa1)
print 'qa2',bin(qa2)
print 'qa3',bin(qa3)
print 'qa4',bin(qa4)
print 'qa5',bin(qa5)

qa1 0b1
qa2 0b1
qa3 0b1
qa4 0b11
qa5 0b111
```

Develop some code to repeat the QA bit masking done above, but make the code generate the bit masks itself from knowledge of the first and last of the bit fields you require (assuming they are sequential).

If possible, do this in a function.

Demonstrate its operation with several example bit masks.

25.6 Answer

One way to do this elegantly is to use the left shift to generate the bit masks.

e.g. to put a 1 in bit 2:

```
mask2 = 0b1 << 2
print bin(mask2)
```

0b100

or to fill fields 3-4:

```
mask3 = 0b11 << 3
print bin(mask3)
```

0b11000

An easier way though is to right shift the qa and perform a bitwise and with the mask:

```
qa = 0b00011000
mask3 = 0b11
print bin((qa >> 3) & mask3)
```

0b11

We need to consider how to pass the information through to this operation.

One way would be to use a dict with the key as the starting value of the bit field and the value as the end:

```
fields = {0:0, 1:1, 2:2, 3:4, 5:7}
```

We access these as:

```
for start,finish in fields.items():
    print start,finish
```

```
0 0
1 1
2 2
3 4
5 7
```

Then form an integer which is $2^n - 1$, where n is the (inclusive) length from start to finish

```
# a test qa with only required bits
qa = 0b00011000
start = 3
finish = 4

print bin(2** (finish-start+1)-1)
```

```
0b11
```

```
# qa with all bits filled
qa = 0b11111111

for start,finish in fields.items():
    mask = 2** (finish-start+1)-1
    print start,finish,bin(mask)
```

```
0 0 0b1
1 1 0b1
2 2 0b1
3 4 0b11
5 7 0b111
```

so now we right shift the qa by start and perform a bitwise and (&) with the mask:

```
start = 3
finish = 4
qa = 0b00011000

mask = (2** (finish-start+1)-1)
maskedQA = (qa >> start) & mask
print bin(maskedQA)

0b11
```

Now we can combine all of these ideas:

```
fields = {0:0, 1:1, 2:2, 3:4, 5:7}

# qa with all bits filled
qa = 0b11111111

for start,finish in fields.items():
    mask = (2** (finish-start+1)-1)
    maskedQA = (qa >> start) & mask
    print start,finish,bin(maskedQA)
```

```
0 0 0b1
1 1 0b1
2 2 0b1
3 4 0b11
5 7 0b111
```

"""Exercise 2.1 Scientific Computing

Thu 3 Oct 2013 10:46:58 BST

P. Lewis : p.lewis@ucl.ac.uk

bit mask test

Original code

```
"""
def maskedQA(qa, fields = {0:0, 1:1, 2:2, 3:4, 5:7}):
    """Return a dictionary of masked QA bit fields

    Inputs:
    qa -- QA to be masked

    Keyword arguments:
    fields -- bit field dictionary (default {0:0, 1:1, 2:2, 3:4, 5:7})
              Here, the key is the start bit of a mask and the value
              the end bit (inclusive).

    Returns:
    Masked QA in dictionary with same keys as fields
"""

maskedQAs = {}

for start,finish in fields.items():
    mask      = (2***(finish-start+1)-1)
    maskedQAs[start] = (qa >> start) & mask

return maskedQAs

help(maskedQA)

Help on function maskedQA in module __main__:

maskedQA(qa, fields={0: 0, 1: 1, 2: 2, 3: 4, 5: 7})
    Return a dictionary of masked QA bit fields

    Inputs:
    qa -- QA to be masked

    Keyword arguments:
    fields -- bit field dictionary (default {0:0, 1:1, 2:2, 3:4, 5:7})
              Here, the key is the start bit of a mask and the value
              the end bit (inclusive).

    Returns:
    Masked QA in dictionary with same keys as fields

# testing

qaDict = maskedQA(0b11111111)
for k in qaDict.keys():
    print k,qaDict[k],bin(qaDict[k])
```

```
0 1 0b1
1 1 0b1
2 1 0b1
3 3 0b11
5 7 0b111

qaDict = maskedQA(0b01010101)
for k in qaDict.keys():
    print k,qaDict[k],bin(qaDict[k])

0 1 0b1
1 0 0b0
2 1 0b1
3 2 0b10
5 2 0b10
```


E2. EXERCISES

This notebook contains answers and notes for the following exercises:

- Exercise 2.1
 - Answer 2.1
- Exercise 2.2
 - Answer 2.2
- Exercise 2.3
 - Answer 2.3
- Exercise 2.4
 - Answer 2.4
- Exercise 2.5
 - Answer 2.5
- Exercise 2.6
 - Answer 2.6
- Exercise 2.7
 - Answer 2.7

26.1 Exercise 2.1

You were given some code:

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

```
# hunger threshold in hours
hungerThreshold = 3.0
# sleep threshold in hours
sleepThreshold = 8.0

# time since fed, in hours
timeSinceFed = 4.0
# time since sleep, in hours
timeSinceSleep = 3.0

# Note use of \ as line continuation here
# It is poor style to have code lines > 79 characters
#
# see http://www.python.org/dev/peps/pep-0008/#maximum-line-length
```

```
#  
print "Tired and hungry?", (timeSinceSleep >= sleepThreshold) and \  
      (timeSinceFed >= hungerThreshold)  
print "Just tired?", (timeSinceSleep >= sleepThreshold) and \  
      (not (timeSinceFed >= hungerThreshold))  
print "Just hungry?", (not (timeSinceSleep >= sleepThreshold)) and \  
      (timeSinceFed >= hungerThreshold)
```

```
Tired and hungry? False  
Just tired? False  
Just hungry? True
```

The code above works fine, but the large blocks of logical tests are not very clear or readable, and contain repeated items.

Modify this block of code to be clearer by assigning the individual logical tests to variables,

e.g.

```
tired = timeSinceSleep >= sleepThreshold
```

26.2 Answer 2.1

This is quite easily made clearer, and is also improved by making sure you comment what you are doing.

It is normally a good idea to put some information on what the purpose of the code is etc.

It is also a good idea if you get into the practice of describing what changes you have made to the code from the original version that this is based on, to give full attribution. We will see that this is important with regard to plagiarism in coursework submission.

Have a look at the [Python styleguide](#) for ideas on what makes good clear code.

You wouldn't have been expected to do *all* of these things (the code below is mainly to give you some ideas for what to do next time!), but hopefully you will have at least set the variables:

```
tired = (timeSinceSleep >= sleepThreshold)  
hungry = (timeSinceFed >= hungerThreshold)
```

and used them in the code for greater clarity.

```
""" Exercise 2.1 Scientific Computing
```

```
Thu 3 Oct 2013 10:46:58 BST
```

```
P. Lewis : p.lewis@ucl.ac.uk
```

```
This code prints information on whether  
I am tired and/or hungry.
```

```
It is controlled by the variables:  
    timeSinceFed  
    timeSinceSleep
```

```
which are checked against hunger and  
sleep thresholds:  
    hungerThreshold  
    sleepThreshold
```

```
All times given as float, in hours
```

*It is a modification of the code in Exercise 2.1
of Scientific Computing (<https://github.com/profLewis/geogg122>)*

Edits made from original:

- added detailed comment strings
- neatened up the comments
- made code easier to read by setting variables *tired* and *hungry*.
- added a new result for 'tired or hungry'
- added tabs (\t) to the print statements to get neater output formatting.

"""

```
# Thresholds
hungerThreshold = 3.0      # hunger threshold in hours
sleepThreshold  = 8.0       # sleep threshold in hours

# Control variables
timeSinceFed    = 4.0      # time since fed, in hour
timeSinceSleep  = 3.0       # time since sleep, in hours

# logical tests for tired and hungry
tired   = (timeSinceSleep >= sleepThreshold)
hungry  = (timeSinceFed >= hungerThreshold)

# print results
print "Tired and hungry?\t", tired and hungry
print "Tired or hungry?\t\t", tired or hungry
print "Just tired?\t\t",     tired and not hungry
print "Just hungry?\t\t",    hungry and not tired

Tired and hungry? False
Tired or hungry?      True
Just tired?          False
Just hungry?         True
```

26.3 Exercise 2.2

26.3.1 A.

A small piece of Python code that will set the variable `today` to be a string with the day of the week today is:

```
# This imports a module that we can use to access dates
from datetime import datetime

# set up a list of days of the week, starting Monday
# Note the line continuation here with \
week = ['Monday','Tuesday','Wednesday','Thursday',\
        'Friday','Saturday','Sunday']

# This part gives the day of the week
# as an integer, 0 -> Monday, 1 -> Tuesday etc.
day_number = datetime.now().weekday()

# print item day_number in the list week
print "today is",week[day_number]
```

```
today is Tuesday
```

Based on the example below, set up a diary for yourself for the week to print out what you should be doing today, using the conditional structure “if .. elif ... else“.

26.4 Answer 2.2

```
'''  
    Exercise 2.2A Scientific Computing  
  
    Mon  7 Oct 2013 10:43:38 BST  
  
    P. Lewis : p.lewis@ucl.ac.uk  
  
    This code prints a diary for today.  
  
'''  
  
'''The following code block is taken verbatim from  
    The Scientific Computing notes  
# This imports a module that we can use to access dates  
from datetime import datetime  
  
# set up a list of days of the week, starting Monday  
# Note the line continuation here with \  
week = ['Monday','Tuesday','Wednesday','Thursday',\  
       'Friday','Saturday','Sunday']  
  
# This part gives the day of the week  
# as an integer, 0 -> Monday, 1 -> Tuesday etc.  
day_number = datetime.now().weekday()  
  
# print item day_number in the list week  
print "Today is",week[day_number]  
  
'''New code: my own work  
    Following the example in the question for E2.2A  
if day_number == 0:    # Monday  
    print 'Spend the day learning Python'  
elif day_number == 1: # Tuesday  
    print 'Spend the day learning Python'  
elif day_number == 2: # Wednesday  
    print 'Go to Python class'  
elif day_number == 3: # Thursday  
    print 'Spend the day learning Python'  
elif day_number == 4: # Friday  
    print 'Spend the day learning Python'  
elif day_number == 5: # Saturday  
    print 'Spend the day learning Python'  
elif day_number == 6: # Sunday  
    print 'Spend the day learning Python'  
else:  
    print "get some sleep"  
  
Today is Tuesday  
Spend the day learning Python
```

You should see from this that this is a rather awkward way to set up something of this nature, and it is prone to

error in typing (you could easily miss a day out or type something else wrong).

26.4.1 B.

You could set up the basic calendar for the week in a list, with the first entry representing Monday, the second Tuesday etc.

```
my_diary = ['Spend the day practicing Python', \
            'Spend the day practicing Python', \
            'Do some reading in the library at UCL', \
            'Remember to wake up early to get to the Python class at UCL', \
            'Spend the day practicing Python', \
            'Remember to wake up early to go to classes at Imperial College', \
            'Work at Python exercises from home', \
            'Work at Python exercises from home']
```

Using a list of this sort, **print the diary entry for today *without* using conditional statements.**

Criticise the code you develop and make suggestions for improvement.

Answer

```
'''  
    Exercise 2.2B Scientific Computing  
  
    Mon  7 Oct 2013 10:43:38 BST  
  
    P. Lewis : p.lewis@ucl.ac.uk  
  
    This code prints a diary for today.  
  
'''  
  
'''The following code block is taken verbatim from  
    The Scientific Computing notes  
'''  
  
# This imports a module that we can use to access dates  
from datetime import datetime  
  
# set up a list of days of the week, starting Monday  
# Note the line continuation here with \  
week = ['Monday','Tuesday','Wednesday','Thursday',\  
       'Friday','Saturday','Sunday']  
  
# This part gives the day of the week  
# as an integer, 0 -> Monday, 1 -> Tuesday etc.  
day_number = datetime.now().weekday()  
  
'''New code: my own work  
    Following the example in the question for E2.2B  
'''  
  
my_diary = ['spend the day doing Python', \  
            'do some reading in the library at UCL', \  
            'remember to wake up early to get to the Python class at UCL', \  
            'spend the day doing Python', \  
            'remember to wake up early to go to classes at Imperial College', \  
            'work at Python exercises from home', \  
            'work at Python exercises from home']
```

```
print "Today is",week[day_number],'so',my_diary[day_number]
```

Today is Tuesday so do some reading in the library at UCL

This is a bit better, but it's still too easy to get the entries in the list wrong, or get confused that Monday is day 0, so we would typically use a dictionary (dict) for something of this nature.

```
'''
```

```
    Exercise 2.2B Scientific Computing
```

```
    Mon  7 Oct 2013 10:43:38 BST
```

```
    P. Lewis : p.lewis@ucl.ac.uk
```

```
    This code prints a diary for today.
```

```
'''
```

```
'''The following code block is taken verbatim from
    The Scientific Computing notes
'''
```

```
# This imports a module that we can use to access dates
from datetime import datetime
```

```
# This part gives the day of the week
# as an integer, 0 -> Monday, 1 -> Tuesday etc.
day_number = datetime.now().weekday()
```

```
'''New code: my own work
    Following the example in the question for E2.2B
'''
```

```
week = {0: 'Monday', \
        1: 'Tuesday', \
        2: 'Wednesday', \
        3: 'Thursday', \
        4: 'Friday', \
        5: 'Saturday', \
        6: 'Sunday'}
```

```
my_diary = {'Monday':      'spend the day doing Python', \
            'Tuesday':     'do some reading in the library at UCL', \
            'Wednesday':   'remember to wake up early to get to the Python class at UCL', \
            'Thursday':    'spend the day doing Python', \
            'Friday':      'remember to wake up early to go to classes at Imperial College', \
            'Saturday':    'work at Python exercises from home', \
            'Sunday':      'work at Python exercises from home'}
```

```
today = week[day_number]
print "Today is",today,'so',my_diary[today]
```

Today is Tuesday so do some reading in the library at UCL

26.5 Exercise 2.3

The data below are fields of:

0 year

- 1 month
- 2 tmax (degC)
- 3 tmin (degC)
- 4 air frost (days)
- 5 rain (mm)
- 6 sun (hours)

for Lowestoft in the UK for the year 2012, taken from Met Office data.

```
data = """ 2012   1     8.7     3.1      5    33.1    53.9
          2     7.1     1.6     13   13.8    86.6
          3    11.3     3.7      2   64.2   141.3
          4    10.9     4.3      3  108.9   151.1
          5    15.1     8.6      0   46.6   171.3
          6    17.9    10.9      0   74.4   189.0
          7    20.3    12.8      0   93.6   206.9
          8    22.0    14.0      0   59.6   217.3
          9    18.9     9.5      0   38.8   200.8
         10   13.6     7.9      0   92.7   94.7
         11   10.5     4.4      2   62.1   79.6
         12    7.9     2.4      8   95.6   41.9 """

```

```
print data
```

```
2012   1     8.7     3.1      5    33.1    53.9
2012   2     7.1     1.6     13   13.8    86.6
2012   3    11.3     3.7      2   64.2   141.3
2012   4    10.9     4.3      3  108.9   151.1
2012   5    15.1     8.6      0   46.6   171.3
2012   6    17.9    10.9      0   74.4   189.0
2012   7    20.3    12.8      0   93.6   206.9
2012   8    22.0    14.0      0   59.6   217.3
2012   9    18.9     9.5      0   38.8   200.8
2012  10   13.6     7.9      0   92.7   94.7
2012  11   10.5     4.4      2   62.1   79.6
2012  12    7.9     2.4      8   95.6   41.9
```

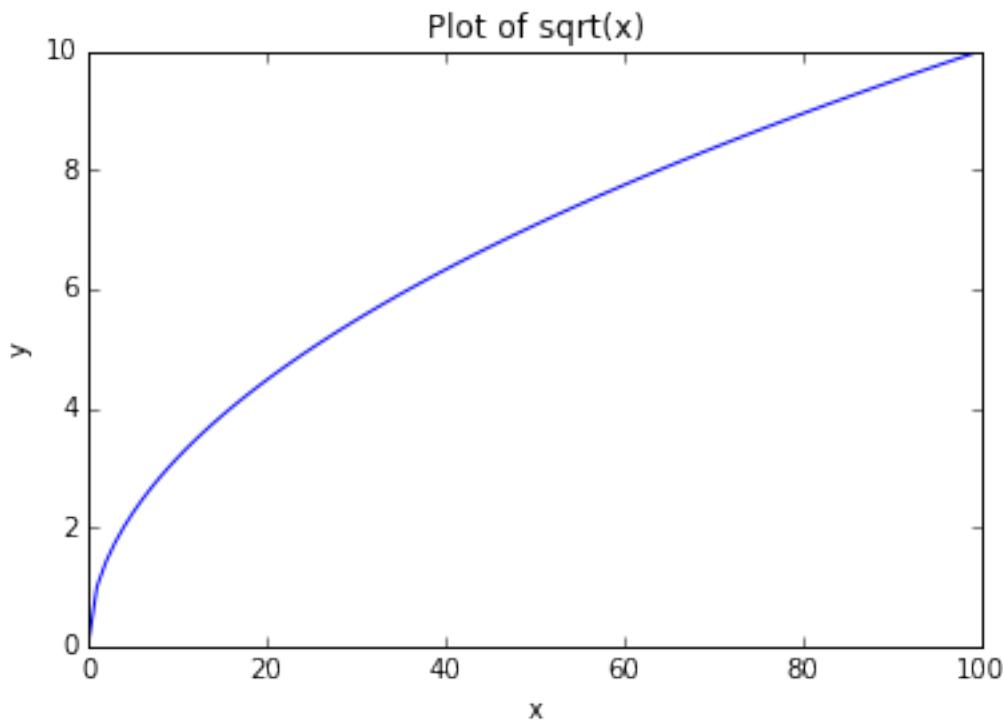
You can use the Python package pylab to simply plot data on a graph:

```
# import the pylab module
import pylab as plt

# some e.g. x and y data
x = range(100)
y = [i**0.5 for i in x]

plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plot of sqrt(x)')

<matplotlib.text.Text at 0x106501890>
```



Produce a plot of the number of sunshine hours for Lowestoft for the year 2012 using the data given above.

Hint: the data have newline characters `\n` at the end of each line of data, and are separated by white space within each line.

26.6 Answer 2.3

First, we need to split the string data by newline (`\n`) to give a list of lines:

```
ldata = data.split('\n')  
  
print ldata  
  
[ ' 2012    1     8.7     3.1      5    33.1    53.9', ' 2012    2     7.1     1.6     13    13.8    86.1']
```

Next, we need to loop over each line and split by whitespace.

This will involve something like:

```
# example using first line  
this_line = ldata[0]  
items = this_line.split()  
print items  
  
['2012', '1', '8.7', '3.1', '5', '33.1', '53.9']  
  
ldata = data.split('\n')  
  
# set empty lists to put the data in  
x = []  
y = []  
  
for this_line in ldata:  
    items = this_line.split()  
    # convert from string to float  
    x.append(float(items[1])) # the month
```

```

y.append(float(items[6])) # sunshine hrs

print x
print y

[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0]
[53.9, 86.6, 141.3, 151.1, 171.3, 189.0, 206.9, 217.3, 200.8, 94.7, 79.6, 41.9]

```

Now we just need to plot it:

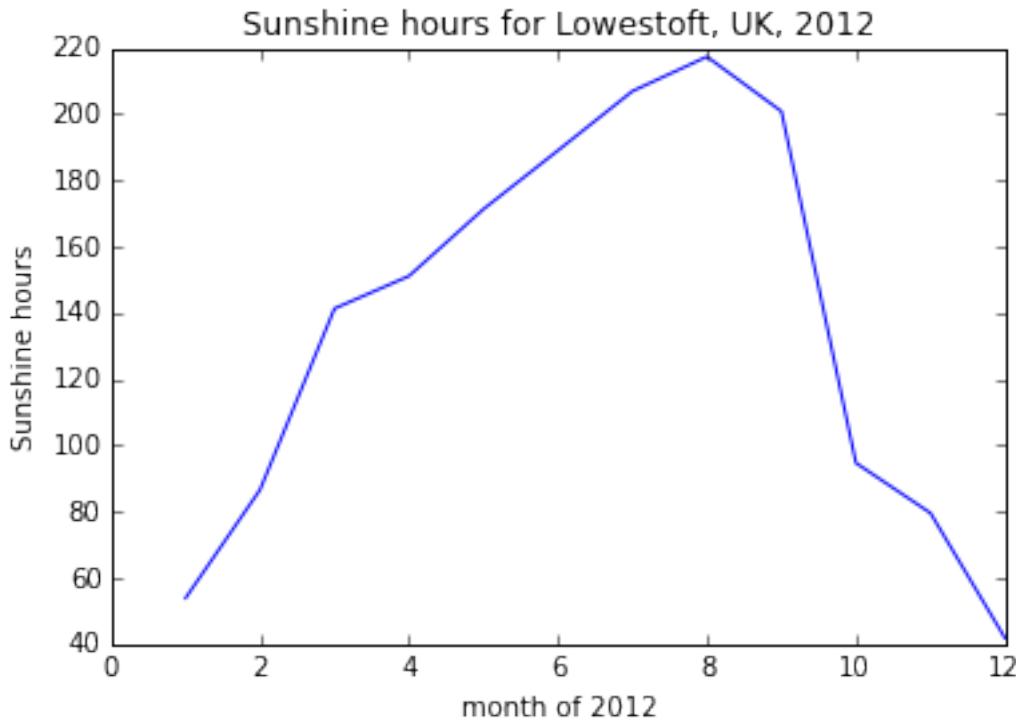
```

# import the pylab module
import pylab as plt

plt.plot(x,y)
plt.xlabel('month of 2012')
plt.ylabel('Sunshine hours')
plt.title('Sunshine hours for Lowestoft, UK, 2012')

<matplotlib.text.Text at 0x106499550>

```



We could make it a bit more compact and general, but if you do, make sure it is still easily readable.

```

import pylab as plt

items = [[float(i) \
          for i in j.split()] \
          for j in data.split('\n')]

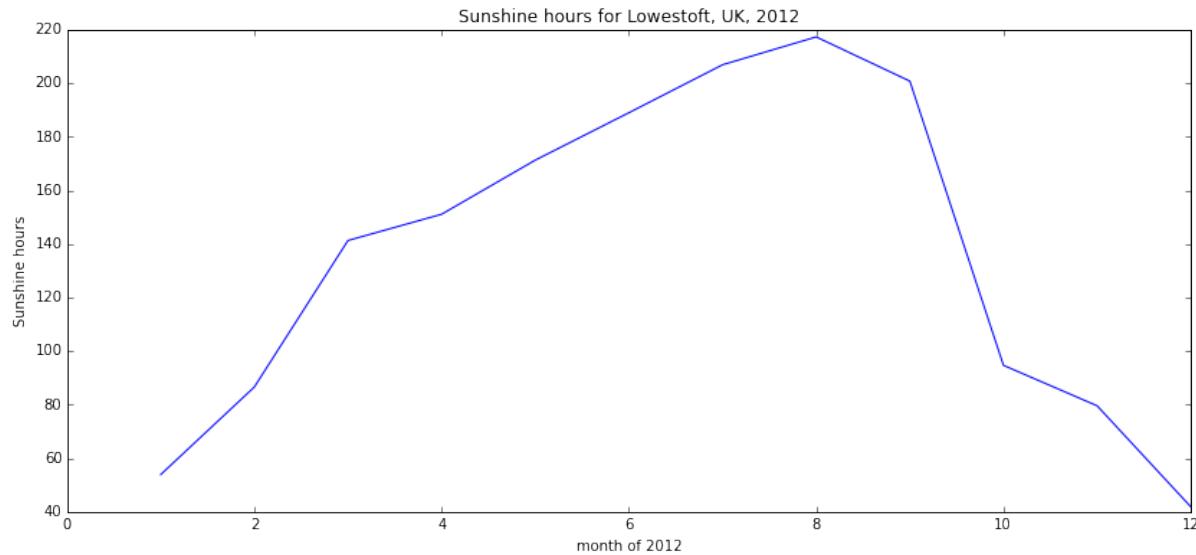
# get specific columns from items
x = [i[1] for i in items]
y = [i[6] for i in items]

# bigger graph if we want
plt.figure(figsize=(14, 6))
plt.plot(x,y)
plt.xlabel('month of 2012')
plt.ylabel('Sunshine hours')

```

```
plt.title('Sunshine hours for Lowestoft, UK, 2012')
```

```
<matplotlib.text.Text at 0x106787510>
```



You *might think* that labelling with the month name would be nicer.

Thats quite a bit more complicated, but we'll put it in here anyway for future reference:

```
import pylab as plt

items = [[float(i) \
          for i in j.split()] \
          for j in data.split('\n')]

# get specific columns from items
x = [int(i[1]) for i in items]
y = [i[6] for i in items]

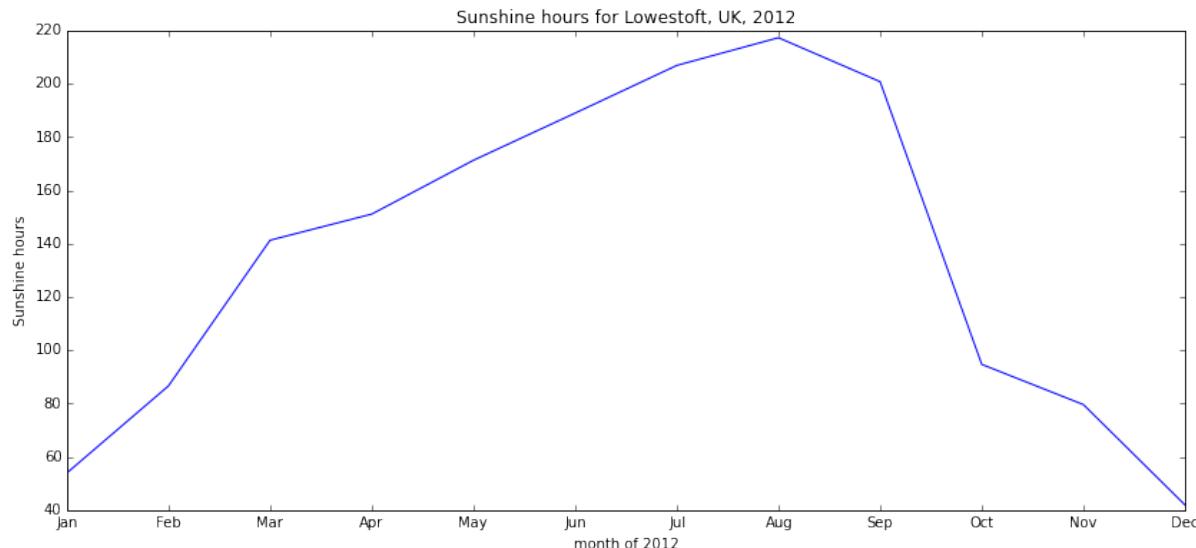
months = ["Jan", "Feb", "Mar", "Apr", \
          "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ]

# need access to ax to mess with xticks
fig, ax = plt.subplots(figsize=(14, 6))
plt.plot(x,y)

# lots of hassle to do this sort of thing ...
# but it is possible
#
# first set the limits to be the
# same as x
plt.xlim(x[0],x[-1])
# then set the xticks to the values of x
# to make sure there are 12 of them
ax.set_xticks(x)
# then replace these by the months
ax.set_xticklabels(months)

plt.xlabel('month of 2012')
plt.ylabel('Sunshine hours')
plt.title('Sunshine hours for Lowestoft, UK, 2012')

<matplotlib.text.Text at 0x106855110>
```



26.7 Exercise 2.4

The text file `files/data/modis_files.txt` contains a listing of hdf format files that are in the directory `/data/geospatial_19/ucfajlg/fire/Angola/MOD09` on the UCL Geography system. The contents of the file looks like (first 10 lines):

```
!head -10 < files/data/modis_files.txt
```

```
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004001.h19v10.005.2008109063923.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004002.h19v10.005.2008108084250.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004003.h19v10.005.2008108054126.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004004.h19v10.005.2008108112322.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004005.h19v10.005.2008108173219.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004006.h19v10.005.2008108214033.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004007.h19v10.005.2008109081257.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004008.h19v10.005.2008109111447.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004009.h19v10.005.2008109211421.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004010.h19v10.005.2008110031925.hdf
```

Your task is to create a new file `files/data/some_modis_files.txt` that contains *only* the file names for the month of August.

You will notice that the file names have a field in them such as A2004006. This is the one you will need to concentrate on, as it specifies the year (2004 here) and the day of year (doy), (006 in this example).

There are various ways to find the day of year for a particular month / year, e,g, look on a [website](#).

26.8 Answer 2.4

First then, find out which day of year range you want.

August 2004 was doy 214 to 244 inclusive, so in python:

```
range(214, 245)
```

We should know how to read data from files from the material above.

```
# ls /data/geospatial_19/ucfajlg/fire/Angola/MOD09/*.hdf > /tmp/modis_files.txt
fp = open('files/data/modis_files.txt', 'r')
for line in fp.readlines():
```

```

print line
break # break in here just so it doesnt print too much
      # dont put that in your code !!!
fp.close()

```

/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004001.h19v10.005.2008109063923.hdf

we have the filename e.g.

```
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004001.h19v10.005.2008109063923.
```

in the variable line.

Now we want to split this to find the filename, which will be the last element (-1):

```

# split the string on '/', the file separator
aline = line.split('/')
print aline

[`, `data', `geospatial_19', `ucfajlg', `fire', `Angola', `MOD09', `MOD09GA.A2004001.h19v10.005.2008109063923.hdf

# and all we want is the last item:

fname = line.split('/')[-1]
print fname

MOD09GA.A2004001.h19v10.005.2008109063923.hdf

```

Now we need to split this up, with . (dot) as the separator, and get the second item:

```

fname = line.split('/')[-1]
fbits = fname.split('.')
print fbits

datebit = fbits[1]
print datebit

['MOD09GA', `A2004001', `h19v10', `005', `2008109063923', `hdfn']
A2004001

```

The day of year is the last 3 elements of this string:

```

doy = int(datebit[-3:])

print doy

```

1

And check to see if doy is in the required range:

Putting this together:

```

date_range = range(214, 245)

# ls /data/geospatial_19/ucfajlg/fire/Angola/MOD09/*.hdf > /tmp/modis_files.txt
fp = open('files/data/modis_files.txt','r')
for line in fp.readlines():
    fname = line.split('/')[-1]
    fbits = fname.split('.')
    datebit = fbits[1]
    doy = int(datebit[-3:])
    if doy in date_range:
        print line
        break
fp.close()

```

```
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004214.h19v10.005.2010052095834.hdf
```

Now put in the part that writes it to a file:

```
ifile = 'files/data/modis_files.txt'
ofile = 'files/data/some_modis_files.txt'

ifp = open(ifile, 'r')
ofp = open(ofile, 'w')
for line in ifp.readlines():
    fname = line.split('/')[-1]
    fbits = fname.split('.')
    datebit = fbits[1]
    doy = int(datebit[-3:])
    if doy in date_range:
        ofp.write(line)
ifp.close()
ofp.close()

# or in a more compressed form

ifile = 'files/data/modis_files.txt'
ofile = 'files/data/some_modis_files.txt'

ifp = open(ifile, 'r')
ofp = open(ofile, 'w')
for line in ifp.readlines():
    doy = int(line.split('/')[-1].split('.')[1][-3:])
    if doy in date_range:
        ofp.write(line)
ifp.close()
ofp.close()
```

check from unix:

```
!head -1 < files/data/some_modis_files.txt
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004214.h19v10.005.2010052095834.hdf
!tail -1 < files/data/some_modis_files.txt
/data/geospatial_19/ucfajlg/fire/Angola/MOD09/MOD09GA.A2004244.h19v10.005.2008256095130.hdf
```

26.9 Exercise 2.5

We want to calculate the **maximum** monthly precipitation for regions of the UK for all years in the 20th Century.

```
# specify filename
filename = 'files/data/HadSEEP_monthly_qc.txt'

# read the data, chop off first 4 lines
# and store in required_data
fp = open(filename, 'r')
raw_data = fp.readlines()
fp.close()
required_data = raw_data[4:]

# set up list to store data in
data = []
```

```
# loop over each line
for line_data in required_data:
    # split on white space
    year_data = line_data.split()

    # convert data to float
    for column, this_element in enumerate(year_data):
        year_data[column] = float(this_element)
    data.append(year_data)

# select years
c20_data = []
for line in data:
    if (line[0] >= 1900) and (line[0] < 2000):
        c20_data.append(line[1:-1])

# Aside: show which month that was
month_names = [ "Jan", "Feb", "Mar", "Apr", \
    "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ]

print "In South East England"
for row in xrange(0,100,1):
    year = 1900 + row
    max_precip = max(c20_data[row])
    month = c20_data[row].index(max_precip)

    print "In the year",year,"the rainiest month was",month_names[month],"with",max_precip,"mm"
```

In South East England

In the year 1900 the rainiest month was Feb with 125.9 mm
In the year 1901 the rainiest month was Dec with 98.7 mm
In the year 1902 the rainiest month was Aug with 97.2 mm
In the year 1903 the rainiest month was Oct with 188.4 mm
In the year 1904 the rainiest month was Jan with 96.0 mm
In the year 1905 the rainiest month was Jun with 104.5 mm
In the year 1906 the rainiest month was Jan with 127.3 mm
In the year 1907 the rainiest month was Oct with 129.5 mm
In the year 1908 the rainiest month was Aug with 83.8 mm
In the year 1909 the rainiest month was Oct with 143.9 mm
In the year 1910 the rainiest month was Dec with 119.4 mm
In the year 1911 the rainiest month was Dec with 153.6 mm
In the year 1912 the rainiest month was Aug with 151.6 mm
In the year 1913 the rainiest month was Oct with 106.5 mm
In the year 1914 the rainiest month was Dec with 190.2 mm
In the year 1915 the rainiest month was Dec with 163.9 mm
In the year 1916 the rainiest month was Mar with 110.7 mm
In the year 1917 the rainiest month was Aug with 135.1 mm
In the year 1918 the rainiest month was Sep with 151.1 mm
In the year 1919 the rainiest month was Dec with 121.4 mm
In the year 1920 the rainiest month was Jul with 116.9 mm
In the year 1921 the rainiest month was Jan with 70.8 mm
In the year 1922 the rainiest month was Jul with 92.2 mm
In the year 1923 the rainiest month was Oct with 142.5 mm
In the year 1924 the rainiest month was Oct with 110.2 mm
In the year 1925 the rainiest month was Jul with 98.1 mm
In the year 1926 the rainiest month was Nov with 143.8 mm
In the year 1927 the rainiest month was Sep with 138.5 mm
In the year 1928 the rainiest month was Oct with 137.0 mm
In the year 1929 the rainiest month was Nov with 175.6 mm
In the year 1930 the rainiest month was Nov with 121.0 mm
In the year 1931 the rainiest month was Aug with 101.3 mm
In the year 1932 the rainiest month was Oct with 146.4 mm

In the year 1933 the rainiest month was Mar with 74.1 mm
In the year 1934 the rainiest month was Dec with 176.3 mm
In the year 1935 the rainiest month was Nov with 148.2 mm
In the year 1936 the rainiest month was Jan with 112.8 mm
In the year 1937 the rainiest month was Jan with 124.9 mm
In the year 1938 the rainiest month was Nov with 93.6 mm
In the year 1939 the rainiest month was Oct with 154.0 mm
In the year 1940 the rainiest month was Nov with 203.0 mm
In the year 1941 the rainiest month was Aug with 117.3 mm
In the year 1942 the rainiest month was Oct with 94.3 mm
In the year 1943 the rainiest month was Jan with 130.2 mm
In the year 1944 the rainiest month was Nov with 117.0 mm
In the year 1945 the rainiest month was Jul with 103.8 mm
In the year 1946 the rainiest month was Nov with 126.7 mm
In the year 1947 the rainiest month was Mar with 150.7 mm
In the year 1948 the rainiest month was Jan with 125.3 mm
In the year 1949 the rainiest month was Oct with 189.3 mm
In the year 1950 the rainiest month was Nov with 137.9 mm
In the year 1951 the rainiest month was Nov with 161.9 mm
In the year 1952 the rainiest month was Nov with 99.9 mm
In the year 1953 the rainiest month was Jul with 87.5 mm
In the year 1954 the rainiest month was Nov with 131.0 mm
In the year 1955 the rainiest month was May with 110.5 mm
In the year 1956 the rainiest month was Jan with 113.7 mm
In the year 1957 the rainiest month was Feb with 97.7 mm
In the year 1958 the rainiest month was Sep with 107.4 mm
In the year 1959 the rainiest month was Dec with 143.4 mm
In the year 1960 the rainiest month was Oct with 167.0 mm
In the year 1961 the rainiest month was Dec with 102.0 mm
In the year 1962 the rainiest month was Sep with 94.0 mm
In the year 1963 the rainiest month was Nov with 141.0 mm
In the year 1964 the rainiest month was Mar with 97.7 mm
In the year 1965 the rainiest month was Dec with 126.9 mm
In the year 1966 the rainiest month was Oct with 129.1 mm
In the year 1967 the rainiest month was Oct with 126.6 mm
In the year 1968 the rainiest month was Sep with 135.5 mm
In the year 1969 the rainiest month was Nov with 108.4 mm
In the year 1970 the rainiest month was Nov with 186.4 mm
In the year 1971 the rainiest month was Jun with 134.4 mm
In the year 1972 the rainiest month was Dec with 94.0 mm
In the year 1973 the rainiest month was May with 76.6 mm
In the year 1974 the rainiest month was Sep with 164.7 mm
In the year 1975 the rainiest month was Sep with 122.8 mm
In the year 1976 the rainiest month was Oct with 141.5 mm
In the year 1977 the rainiest month was Aug with 132.8 mm
In the year 1978 the rainiest month was Dec with 144.9 mm
In the year 1979 the rainiest month was Dec with 122.6 mm
In the year 1980 the rainiest month was Oct with 110.1 mm
In the year 1981 the rainiest month was Mar with 131.2 mm
In the year 1982 the rainiest month was Oct with 143.0 mm
In the year 1983 the rainiest month was May with 98.5 mm
In the year 1984 the rainiest month was Jan with 112.4 mm
In the year 1985 the rainiest month was Dec with 103.7 mm
In the year 1986 the rainiest month was Jan with 110.1 mm
In the year 1987 the rainiest month was Oct with 198.7 mm
In the year 1988 the rainiest month was Jan with 143.8 mm
In the year 1989 the rainiest month was Dec with 143.8 mm
In the year 1990 the rainiest month was Feb with 122.3 mm
In the year 1991 the rainiest month was Jun with 102.0 mm
In the year 1992 the rainiest month was Nov with 132.8 mm
In the year 1993 the rainiest month was Dec with 133.9 mm
In the year 1994 the rainiest month was Jan with 115.6 mm
In the year 1995 the rainiest month was Jan with 143.1 mm

In the year 1996 the rainiest month was Nov with 122.6 mm
In the year 1997 the rainiest month was Jun with 100.6 mm
In the year 1998 the rainiest month was Oct with 128.1 mm
In the year 1999 the rainiest month was Dec with 109.5 mm

That is quite an achievement, given the limited amount of programming you know so far.

If you go through this though, you will (should) see that it is really not very efficient.

For example:

```
we read all the data in and then filter out the years we want (what if the dataset were huge?)  
we loop over the 100 years multiple times  
we store intermediate results
```

For this exercise, you should look through the code we developed and try to make it more efficient.

Efficiency should not override clarity and understanding though, so make sure you can understand what is going on at each stage.

26.10 Answer 2.5

```
# specify filename  
filename = 'files/data/HadSEEP_monthly_qc.txt'  
  
fp = open(filename, 'r')  
  
# years range  
years = range(1900,2000)  
  
# set up list to store data in  
c20_data = []  
  
# loop over each line  
for line_data in fp.readlines()[4:]:  
    # split on white space  
    # convert data to float  
    year_data = [float(i) for i in line_data.split()]  
  
    if year_data[0] in years:  
        c20_data.append(year_data)  
  
# Aside: show which month that was  
month_names = [ "Jan", "Feb", "Mar", "Apr", "\\\n    "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ]  
  
print "In South East England"  
for row,year in enumerate(years):  
    max_precip = max(c20_data[row])  
    month = c20_data[row].index(max_precip)  
  
    print "In the year",year,"the rainiest month was",month_names[month],"with",max_precip,"mm"  
  
fp.close()
```

```
In South East England  
In the year 1900 the rainiest month was Jan with 1900.0 mm  
In the year 1901 the rainiest month was Jan with 1901.0 mm  
In the year 1902 the rainiest month was Jan with 1902.0 mm  
In the year 1903 the rainiest month was Jan with 1903.0 mm  
In the year 1904 the rainiest month was Jan with 1904.0 mm  
In the year 1905 the rainiest month was Jan with 1905.0 mm
```


In the year 1969 the雨iest month was Jan with 1969.0 mm
In the year 1970 the雨iest month was Jan with 1970.0 mm
In the year 1971 the雨iest month was Jan with 1971.0 mm
In the year 1972 the雨iest month was Jan with 1972.0 mm
In the year 1973 the雨iest month was Jan with 1973.0 mm
In the year 1974 the雨iest month was Jan with 1974.0 mm
In the year 1975 the雨iest month was Jan with 1975.0 mm
In the year 1976 the雨iest month was Jan with 1976.0 mm
In the year 1977 the雨iest month was Jan with 1977.0 mm
In the year 1978 the雨iest month was Jan with 1978.0 mm
In the year 1979 the雨iest month was Jan with 1979.0 mm
In the year 1980 the雨iest month was Jan with 1980.0 mm
In the year 1981 the雨iest month was Jan with 1981.0 mm
In the year 1982 the雨iest month was Jan with 1982.0 mm
In the year 1983 the雨iest month was Jan with 1983.0 mm
In the year 1984 the雨iest month was Jan with 1984.0 mm
In the year 1985 the雨iest month was Jan with 1985.0 mm
In the year 1986 the雨iest month was Jan with 1986.0 mm
In the year 1987 the雨iest month was Jan with 1987.0 mm
In the year 1988 the雨iest month was Jan with 1988.0 mm
In the year 1989 the雨iest month was Jan with 1989.0 mm
In the year 1990 the雨iest month was Jan with 1990.0 mm
In the year 1991 the雨iest month was Jan with 1991.0 mm
In the year 1992 the雨iest month was Jan with 1992.0 mm
In the year 1993 the雨iest month was Jan with 1993.0 mm
In the year 1994 the雨iest month was Jan with 1994.0 mm
In the year 1995 the雨iest month was Jan with 1995.0 mm
In the year 1996 the雨iest month was Jan with 1996.0 mm
In the year 1997 the雨iest month was Jan with 1997.0 mm
In the year 1998 the雨iest month was Jan with 1998.0 mm
In the year 1999 the雨iest month was Jan with 1999.0 mm

26.11 Exercise 2.6

We want to calculate the long-term average temperature (tmax degC) using observational data at one or more meteorological stations. Such data are relevant to understanding climate and its dynamics.

We choose the period 1960 to 1990 (30 years average to even out natural variability).

We can obtain monthly average data for a number of UK stations from the UK Met. Office.

Not all station records are complete enough for this calculation, so we select, for example Heathrow.

Just with the Python skills you have learned so far, you should be able to solve a problem of this nature.

26.12 Answer 2.6

Before diving into this though, you need to think through what steps you need to go through to achieve your aim?

At a ‘high’ level, this could be:

1. Get hold of the data
2. Read the data into the computer program
3. Select which years you want
4. Average the data for each month over all selected years
5. Print the results

So now we need to think about how to implement these steps.

For the first step, there are many ways you could do this.

```
!wget -O files/data/heathrowdata.txt http://www.metoffice.gov.uk/climate/uk/stationdata/heathrowdata.txt

--2014-09-30 11:13:08-- http://www.metoffice.gov.uk/climate/uk/stationdata/heathrowdata.txt
Resolving www.metoffice.gov.uk... 213.120.162.233, 213.120.162.211
Connecting to www.metoffice.gov.uk|213.120.162.233|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.metoffice.gov.uk/pub/data/weather/uk/climate/stationdata/heathrowdata.txt [f...
--2014-09-30 11:13:08-- http://www.metoffice.gov.uk/pub/data/weather/uk/climate/stationdata/heathrowdata.txt
Reusing existing connection to www.metoffice.gov.uk:80.
HTTP request sent, awaiting response... 200 OK
Length: 41280 (40K) [text/plain]
Saving to: 'files/data/heathrowdata.txt'

100%[=====] 41,280 --.-K/s in 0s

2014-09-30 11:13:09 (206 MB/s) - 'files/data/heathrowdata.txt' saved [41280/41280]

ifile = 'files/data/heathrowdata.txt'
fp = open(ifile)
raw_data = fp.readlines()
fp.close()

raw_data = raw_data[7:] # Skip headers

# set up the problem
start_year = 1960
end_year = 1990

tmax = []

for l in raw_data:
    s = l.split()
    year = int(s[0])
    if year >= start_year and year < end_year:
        month = int(s[1])
        if month == 1:
            tmax.append([])
            tmax[-1].append(float(s[2]))

for l in tmax:
    print l

[6.9, 7.9, 10.2, 14.3, 18.4, 22.1, 20.1, 20.3, 18.5, 14.2, 11.2, 6.9]
[6.9, 10.3, 13.9, 15.0, 16.8, 21.7, 22.1, 21.7, 20.9, 15.6, 9.9, 6.8]
[7.9, 7.6, 7.4, 12.7, 15.0, 20.4, 20.6, 20.0, 17.8, 15.7, 9.1, 5.0]
[0.8, 2.8, 10.7, 13.6, 16.0, 20.8, 21.1, 19.8, 18.0, 14.8, 11.8, 5.4]
[5.8, 7.6, 7.6, 12.8, 19.6, 19.3, 22.8, 22.3, 21.1, 13.7, 11.2, 7.1]
[6.6, 5.9, 10.5, 13.4, 16.8, 19.7, 19.3, 20.8, 17.4, 16.2, 8.4, 8.4]
[5.3, 9.3, 10.9, 12.2, 17.0, 21.8, 20.1, 20.6, 19.6, 14.7, 8.8, 9.3]
[7.3, 9.3, 11.6, 12.5, 15.6, 20.1, 23.5, 21.5, 18.5, 15.0, 9.7, 7.2]
[7.2, 5.7, 11.1, 13.7, 15.4, 20.6, 20.7, 20.2, 18.8, 16.5, 9.5, 5.4]
[8.9, 4.9, 8.0, 13.6, 17.1, 20.6, 23.9, 21.8, 19.6, 18.2, 9.6, 6.1]
[7.1, 7.2, 8.1, 11.4, 19.2, 23.6, 21.3, 22.3, 20.3, 15.7, 11.9, 7.0]
[7.8, 8.6, 9.1, 12.2, 18.1, 17.8, 24.1, 21.3, 20.7, 17.2, 10.4, 9.3]
[6.9, 7.9, 12.4, 12.9, 16.0, 17.5, 22.3, 21.8, 17.3, 15.6, 9.9, 9.4]
[7.3, 8.1, 11.5, 12.5, 16.9, 21.9, 21.4, 23.7, 20.7, 13.9, 10.3, 8.6]
[9.6, 9.2, 10.3, 14.0, 17.0, 20.1, 20.9, 21.5, 17.5, 11.0, 10.6, 11.0]
[10.3, 9.3, 8.4, 13.3, 15.1, 21.8, 24.1, 25.9, 19.5, 14.4, 10.1, 7.6]
[8.8, 7.8, 9.5, 13.7, 19.3, 25.5, 26.6, 25.1, 19.0, 14.9, 9.9, 5.6]
[6.1, 9.6, 11.2, 12.2, 16.4, 17.4, 22.4, 20.3, 18.0, 16.3, 10.4, 9.2]
```

```
[6.7, 6.2, 11.3, 10.7, 17.2, 19.7, 20.5, 20.7, 20.1, 17.1, 12.6, 7.8]
[3.8, 4.5, 9.2, 12.7, 15.9, 19.0, 22.5, 21.0, 19.6, 16.4, 10.8, 9.2]
[6.0, 9.9, 9.0, 14.0, 17.3, 19.8, 19.7, 21.9, 20.1, 14.0, 9.3, 8.9]
[8.1, 7.1, 11.8, 12.9, 16.1, 18.5, 22.0, 22.8, 20.3, 12.9, 11.5, 4.4]
[6.9, 8.4, 11.1, 14.3, 18.0, 21.5, 22.7, 22.2, 20.8, 14.1, 11.4, 8.2]
[10.0, 5.5, 10.8, 12.4, 15.6, 20.8, 27.6, 24.5, 19.3, 15.1, 11.3, 9.2]
[8.0, 7.4, 8.9, 14.6, 14.9, 21.3, 24.2, 24.4, 18.6, 15.8, 12.2, 8.7]
[4.1, 6.3, 9.4, 14.0, 16.4, 18.5, 22.9, 20.3, 20.6, 16.1, 8.1, 10.0]
[7.2, 1.7, 10.1, 10.9, 16.3, 21.8, 22.6, 20.0, 17.4, 16.4, 12.1, 9.7]
[3.6, 7.7, 8.5, 15.8, 16.3, 18.6, 21.8, 21.8, 19.3, 15.0, 9.8, 8.9]
[8.8, 8.7, 10.7, 13.5, 18.0, 19.7, 20.0, 21.8, 18.8, 15.5, 9.9, 10.4]
[9.5, 10.2, 12.9, 11.5, 21.0, 22.1, 25.8, 24.2, 20.7, 17.1, 10.9, 9.4]

monthly_avg = []
n_years = len(tmax)

for month in xrange( len(tmax[0]) ):
    temp = 0.
    for year in xrange(n_years):
        temp += tmax[year][month]
    monthly_avg.append(temp/n_years)

print "T results for", n_years, "years:", start_year, "to", end_year

month_names = [ "Jan", "Feb", "Mar", "Apr", \
                 "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ]

print "\n\tMonth\tTmaxAvg\n"
for i, mname in enumerate( month_names ):
    print '\t', mname, '\t', monthly_avg[i]

T results for 30 years: 1960 to 1990

    Month      TmaxAvg
    Jan       7.00666666667
    Feb        7.42
    Mar      10.2033333333
    Apr       13.11
    May      16.9566666667
    Jun      20.4666666667
    Jul       22.32
    Aug      21.8833333333
    Sep      19.2933333333
    Oct      15.3033333333
    Nov       10.42
    Dec      8.0033333333
```

26.13 Exercise 2.7

```
# 1. Read the configuration file
# into the dict modis
import ConfigParser

config = ConfigParser.ConfigParser()
config.read('files/data/modis.cfg')

# we can convert this to a normal dictionary
modis = {}
```

```

for k in config.sections():
    modis[k] = dict(config.items(k))

# 2. Now, loop over config sections
# and get the sub-dictionary which we call sub_dict

# 3. set up anb empty list to contain the
# files we want to process
wanted_files = []

for k,v in modis.items():

    sub_dict = v

    # 3a. Read the file list
    fp = open(sub_dict['file_list'], 'r')
    file_data = fp.readlines()
    fp.close()

    # 3b. find the doy range
    doy_range = range(int(sub_dict['doy_start']), \
                      int(sub_dict['doy_end']))

    # 3c. loop over each file read from
    #      sub_dict['file_list']
    for count in xrange(len(file_data)):
        # 3d. extract doy from the file name
        this_file = file_data[count]

        doy = int(this_file.split('.')[1][-3:])

        # 3e. see if doy is in the range we want?
        if doy in doy_range:

            # 3f. put the directory on the fornt
            full_name = sub_dict['dir'] + \
                        '/' + this_file
            wanted_files.append(full_name)

print "I found %d files to process"%len(wanted_files)

# I won't print the whole list as its too long
# just the first 10
for f in wanted_files[:10]:
    print f

I found 62 files to process
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004214.h19v10.005.2007299212915.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004215.h19v10.005.2007300042347.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004216.h19v10.005.2007300091257.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004217.h19v10.005.2007300153436.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004218.h19v10.005.2007300215826.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004219.h19v10.005.2007302194509.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004220.h19v10.005.2007302093547.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004221.h19v10.005.2007302222054.hdf

```

```
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004222.h19v10.005.2007303011606.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004223.h19v10.005.2007303073538.hdf
```

You should modify the example above to make it simpler, if you can spot any places for that (don't make it more complicated!).

You should then modify it so that the list of files that we want to process is printed to a file.

26.14 Answer 2.7

Probably the main place you could make the code simpler would be in the loop:

```
# 3a. Read the file list
fp = open(sub_dict['file_list'], 'r')
file_data = fp.readlines()
fp.close()

# 3b. find the doy range
doy_range = range(int(sub_dict['doy_start']), \
                  int(sub_dict['doy_end']))

# 3c. loop over each file read from
#      sub_dict['file_list']
for count in xrange(len(file_data)):
    # 3d. extract doy from the file name
    this_file = file_data[count]

    doy = int(this_file.split('.')[1][-3:])

    # 3e. see if doy is in the range we want?
    if doy in doy_range:

        # 3f. put the directory on the front
        full_name = sub_dict['dir'] + \
                    '/' + this_file
        wanted_files.append(full_name)
```

Really, we don't need the count variable and structure in here, and we could replace this by:

```
# 3a. Read the file list
fp = open(sub_dict['file_list'], 'r')

# 3b. find the doy range
doy_range = range(int(sub_dict['doy_start']), \
                  int(sub_dict['doy_end']))

# 3c. loop over each file read from
#      sub_dict['file_list']
for this_file in fp.readlines():

    # 3d. extract doy from the file name
    doy = int(this_file.split('.')[1][-3:])

    # 3e. see if doy is in the range we want?
    if doy in doy_range:

        # 3f. store the filename in the list
        wanted_files.append("%s/%s"%(sub_dict['dir'],this_file))

# close the file here now
fp.close()
```

If you wanted, you could also do some clever things with and:

```
# 3a. Read the file list
fp = open(sub_dict['file_list'], 'r')

# 3b. find the doy range
doy_range = range(int(sub_dict['doy_start']), \
                  int(sub_dict['doy_end']))

# 3c. loop over each file read from
#      sub_dict['file_list']
for this_file in fp.readlines():

    # 3d. extract doy from the file name
    doy = int(this_file.split('.')[1][-3:])

    # 3e. see if doy is in the range we want
    # and put in list if so
    doy in doy_range and \
        wanted_files.append("%s/%s"%(sub_dict['dir'],this_file))

# close the file here now
fp.close()
```

Writing the results to file should be straightforward enough for you now.

You could do it at the end:

```
# 1. Read the configuration file
# into the dict modis
import ConfigParser

# good to open the file early on
# in case it fails
ofile = 'files/data/modis_files.dat'
ofp = open(ofile, "w")

config = ConfigParser.ConfigParser()
config.read('files/data/modis.cfg')

# we can convert this to a normal dictionary
modis = {}
for k in config.sections():
    modis[k] = dict(config.items(k))

# 2. Now, loop over config sections
# and get the sub-dictionary which we call sub_dict

# 3. set up an empty list to contain the
# files we want to process
wanted_files = []

for k,v in modis.items():

    sub_dict = v

    # 3a. Read the file list
    fp = open(sub_dict['file_list'], 'r')

    # 3b. find the doy range
    doy_range = range(int(sub_dict['doy_start']), \
                      int(sub_dict['doy_end']))
```

```
# 3c. loop over each file read from
#      sub_dict['file_list']
for this_file in fp.readlines():

    # 3d. extract doy from the file name
    doy = int(this_file.split('.')[1][-3:])
    # 3e. see if doy is in the range we want
    # and put in list if so
    doy in doy_range and \
        wanted_files.append("%s/%s"%(sub_dict['dir'],this_file))
fp.close()

print "I found %d files to process"%len(wanted_files)

ofp.writelines(wanted_files)
ofp.close()

I found 62 files to process

!head -10 < files/data/modis_files.dat

/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004214.h19v10.005.2007299212915.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004215.h19v10.005.200730042347.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004216.h19v10.005.2007300091257.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004217.h19v10.005.2007300153436.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004218.h19v10.005.2007300215826.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004219.h19v10.005.20073002194509.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004220.h19v10.005.20073002093547.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004221.h19v10.005.2007302222054.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004222.h19v10.005.2007303011606.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004223.h19v10.005.2007303073538.hdf
```

Or you could print as you go along in the loop:

```
# 1. Read the configuration file
# into the dict modis
import ConfigParser

# good to open the file early on
# in case it fails
ofile = 'files/data/modis_files.dat'
ofp = open(ofile,"w")

config = ConfigParser.ConfigParser()
config.read('files/data/modis.cfg')

# we can convert this to a normal dictionary
modis = {}
for k in config.sections():
    modis[k] = dict(config.items(k))

# 2. Now, loop over config sections
# and get the sub-dictionary which we call sub_dict

# 3. set up anb empty list to contain the
# files we want to process
wanted_files = []

for k,v in modis.items():

    sub_dict = v

    # 3a. Read the file list
```

```

fp = open(sub_dict['file_list'], 'r')

# 3b. find the doy range
doy_range = range(int(sub_dict['doy_start']), \
                  int(sub_dict['doy_end']))

# 3c. loop over each file read from
#      sub_dict['file_list']
for this_file in fp.readlines():

    # 3d. extract doy from the file name
    # 3e. see if doy is in the range we want
    # and put in list if so
    if int(this_file.split('.')[1][-3:]) in doy_range and \
       ofp.write("%s/%s"%(sub_dict['dir'],this_file))

fp.close()

ofp.close()

!head -10 < files/data/modis_files.dat

/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004214.h19v10.005.2007299212915.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004215.h19v10.005.2007300042347.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004216.h19v10.005.2007300091257.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004217.h19v10.005.2007300153436.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004218.h19v10.005.2007300215826.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004219.h19v10.005.2007302194509.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004220.h19v10.005.2007302093547.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004221.h19v10.005.2007302222054.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004222.h19v10.005.2007303011606.hdf
/data/geospatial_19/ucfajlg/fire/Angola/MYD09/MYD09GA.A2004223.h19v10.005.2007303073538.hdf

```


A3. ADVANCED NOTES: SCIENTIFIC AND NUMERICAL PYTHON

27.1 A3.1 Pulling Compressed netCDF Files

Sometimes, such as when we want to pull data from netCDF files from some data site such as <http://www.globalbedo.org>, we might find that ‘older style’ formats have been used, such as netCDF3 which might not have internal compression.

To save storage space, it is common to compress such files extrenally (i.e. to gzip a file).

That makes direct reading from a url a bit more tricky, and in such cases, we may as well uncompress the file to a local temporary file.

27.1.1 Doing this in Python

What we are going to do is to write a class to download a gzipped file from a url and return a filename that can be read by other functions.

The file is available as `gzurl.py`, in the directory `files/python`.

To be able to import this, we have to put `files/python` in the path where Python looks for modules:

```
import sys,os
# put local directory into the path
sys.path.insert(0,os.path.abspath('files%spython'%os.sep))

# import module
from gzurl import gzurl

help(gzurl)

Help on class gzurl in module gzurl:

class gzurl(__builtin__.object)
|   Download gzipped url to a local file or string
|
|   Prof. P. Lewis, UCL,
|   Thu 10 Oct 2013 12:01:00 BST
|   p.lewis@ucl.ac.uk
|
|   Methods defined here:
|
|   __del__(self)
|       Destriuctor
|
|   Tidy up
|
|   __init__(self, url, filename=None, store=False, file=True)
|       initialise class instance
|
```

```
|     Parameters:
| 
|     url    : url of gzipped file
| 
|     Options:
| 
|     filename:
|         specify a filename explicitly, rather than
|         a temporary file (default None)
|     store : boolean flag to store the uncompressed
|             data in self.data (default false)
|     file  : boolean flag to store data to a file
|             (default True)
| 
|     close(self)
|         Tidy up
| 
|     read(self, url)
|         read gzipped data from url
|         and uncompress
| 
|     -----
|     Data descriptors defined here:
| 
|     __dict__
|         dictionary for instance variables (if defined)
| 
|     __weakref__
|         list of weak references to the object (if defined)
```

You can look through the file `gzurl.py` at your leisure, but it is of interest to see how we have done this.

We need to load the following modules to do this:

```
urllib2, io, gzip, tempfile
import urllib2, io, gzip, tempfile
```

The first thing we do is attempt to open a file specified from a url:

```
# codes for url specification on globalbedo.org
years = range(1998,2012)
codes = [95, 95, 97, 97, 26, 66, 54, 54, 29, 25, 53, 56, 56, 78]
XX = dict(zip(years,codes))

year = 2009

root = 'http://www.globalbedo.org/GlobAlbedo%d/mosaics/%d/0.5/monthly/' % \
       (XX[year],year)

# filename formatting string: use %02d for month eg 01 for 1
month = 1
url = root + '/GlobAlbedo.%d%02d.mosaic.5.nc.gz' % (year,month)
print url

# open file from url
f = urllib2.urlopen(url)

http://www.globalbedo.org/GlobAlbedo56/mosaics/2009/0.5/monthly//GlobAlbedo.200901.mosaic.5.nc.gz
```

We then read data from that with the statement:

```
bdata = f.read()

# which looks like this:
bdata[:50]
```

```
`x1fx8bx08x08xf5xe71Rx00x03GlobAlbedo.200901.mosaic.5.ncx00xecxdwxTUxfexc7qx10`
```

We then create a buffered I/O stream from this using `io.BytesIO`, which is the form we want the information in for the next part:

```
f = urllib2.urlopen(url)
fileobj = io.BytesIO(f.read())
```

Next, we use the module `gzip.GzipFile` which simulates the methods of a gzip file:

```
gzip.GzipFile(fileobj=fileobj)

<gzip _io.BytesIO object at 0x105aea950 0x105bd2890>
```

And then we read from this:

```
f = urllib2.urlopen(url)
data=gzip.GzipFile(fileobj=io.BytesIO(f.read())).read()
```

This is now binary of netCDF format in this case.

Next, we need to write these data to a file. In this case, we don't want to really save the data anywhere, so we want to use a temporary file.

In Python, you can create a temporary file using the module `tempfile`, which creates a temporary (unique) file on the system.

```
tmp = tempfile.NamedTemporaryFile(delete=False)
print tmp.name

/var/folders/pt/z0y8dmcd7d77cs_0hnygpwh80000gn/T/tmpN1LPzf
```

So we write the data to this file:

```
tmp.write(data)
```

Then, after we have done something with the data, we will want to tidy up and delete the file:

```
tmp.unlink(tmp.name)
```

To use this module then:

```
import sys,os
# put local directory into the path
sys.path.insert(0,os.path.abspath('files%spython'%os.sep))
# import local module gzurl
from gzurl import gzurl
from netCDF4 import Dataset

# codes for url specification on globalbedo.org
years = range(1998,2012)
codes = [95,95,97,97,26,66,54,54,29,25,53,56,56,78]
XX = dict(zip(years,codes))

year = 2009

root = 'http://www.globalbedo.org/GlobAlbedo%d/mosaics/%d/0.5/monthly/'%\
    (XX[year],year)

print root
```

```
# filename formatting string: use %02d for month eg 01 for 1
month = 1
url = root + '/GlobAlbedo.%d%02d.mosaic.5.nc.gz' %(year,month)

# read the gzipped file
f = gzurl(url)
# read the netCDF file from f.filename
nc = Dataset(f.filename,'r')
# close f
f.close()

http://www.globalbedo.org/GlobAlbedo56/mosaics/2009/0.5/monthly/
```

Alternatively, to read all of the files into the directory files/data for the year 2011 and keep them:

```
import sys,os
# put local directory into the path
sys.path.insert(0,os.path.abspath('files%spython'%os.sep))
# import local module gzurl
from gzurl import gzurl
from netCDF4 import Dataset

# codes for url specification on globalbedo.org
years = range(1998,2012)
codes = [95,95,97,97,26,66,54,54,29,25,53,56,56,78]
XX = dict(zip(years,codes))

year = 2009

root = 'http://www.globalbedo.org/GlobAlbedo%d/mosaics/%d/0.5/monthly/'%\
(XX[year],year)

for month0 in range(12):
    # filename formatting string: use %02d for month eg 01 for 1
    base = 'GlobAlbedo.%d%02d.mosaic.5.nc' %(year,month0+1)
    url = root + base + '.gz'
    # specify a local filename
    # work out how / why this works ...
    local = os.path.join('files{0}data{0}'.format(os.sep),base)

    # read the gzipped file
    print local
    f = gzurl(url,filename=local)
    # read the netCDF file from f.filename
    nc = Dataset(f.filename,'r')
    # close f
    f.close()

files/data/GlobAlbedo.200901.mosaic.5.nc
files/data/GlobAlbedo.200902.mosaic.5.nc
files/data/GlobAlbedo.200903.mosaic.5.nc
files/data/GlobAlbedo.200904.mosaic.5.nc
files/data/GlobAlbedo.200905.mosaic.5.nc
files/data/GlobAlbedo.200906.mosaic.5.nc
files/data/GlobAlbedo.200907.mosaic.5.nc
files/data/GlobAlbedo.200908.mosaic.5.nc
files/data/GlobAlbedo.200909.mosaic.5.nc
files/data/GlobAlbedo.200910.mosaic.5.nc
files/data/GlobAlbedo.200911.mosaic.5.nc
files/data/GlobAlbedo.200912.mosaic.5.nc
```

27.1.2 Doing this in unix

That's not too complicated, but you might often do this sort of thing from unix instead:

```
!rm -f files/data/GlobAlbedo.200901.mosaic.5.nc.gz
!wget -O files/data/GlobAlbedo.200901.mosaic.5.nc.gz \
      http://www.globalbedo.org/GlobAlbedo56/mosaics/2009/0.5/monthly/GlobAlbedo.200901.mosaic.5.nc.gz

--2013-10-12 20:13:51-- http://www.globalbedo.org/GlobAlbedo56/mosaics/2009/0.5/monthly/GlobAlbedo.200901.mosaic.5.nc.gz
Resolving www.globalbedo.org... 128.40.73.100
Connecting to www.globalbedo.org|128.40.73.100|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4540366 (4.3M) [application/x-gzip]
Saving to: `files/data/GlobAlbedo.200901.mosaic.5.nc.gz'

100%[=====] 4,540,366 2.78M/s in 1.6s

2013-10-12 20:13:52 (2.78 MB/s) - `files/data/GlobAlbedo.200901.mosaic.5.nc.gz' saved [4540366/4540366]

!gunzip -f files/data/GlobAlbedo.200901.mosaic.5.nc.gz
!ls -l files/data/GlobAlbedo.200901.mosaic.5.nc

-rw-r--r-- 1 plewis staff 18669672 12 Sep 17:12 files/data/GlobAlbedo.200901.mosaic.5.nc
```

27.2 A3.2 Logical combinations in numpy

Let's read in a different GlobAlbedo dataset.

This time, we will read 8 day tile data (day of year: 001, 009 etc. every 8 days).

The tile we will read is h17v03 which covers most of the UK.

```
import sys,os
sys.path.insert(0,os.path.abspath('files%spython'%os.sep))
from gzurl import gzurl
from netCDF4 import Dataset

years = range(1998,2012)
codes = [95,95,97,97,26,66,54,54,29,25,53,56,56,78]
XX = dict(zip(years,codes))

year = 2009
tile = 'h17v03'

root = 'http://www.globalbedo.org/GlobAlbedo%d/tiles/%d/%s/'%\
       (XX[year],year,tile)

# filename formatting string: use %03d for doy eg 001 for 1
doy = 145
url = root + 'GlobAlbedo.%d%03d.%s.nc.gz'%(year,doy,tile)
# see if you can make sense of this complicated formatting
filename = url.split('/')[-1].replace('.gz','')
local_file = 'files{0}data{1}'.format(os.sep,filename)

# try to read local file
try:
    nc = Dataset(local_file,'r')
except:
    f = gzurl(url,filename=local_file)
    nc = Dataset(f.filename,'r')
    f.close()
```

```
# now pull some data

vis = np.array(nc.variables['BHR_VIS'])
nir = np.array(nc.variables['BHR_NIR'])
ndvi = (nir - vis)/(nir + vis)

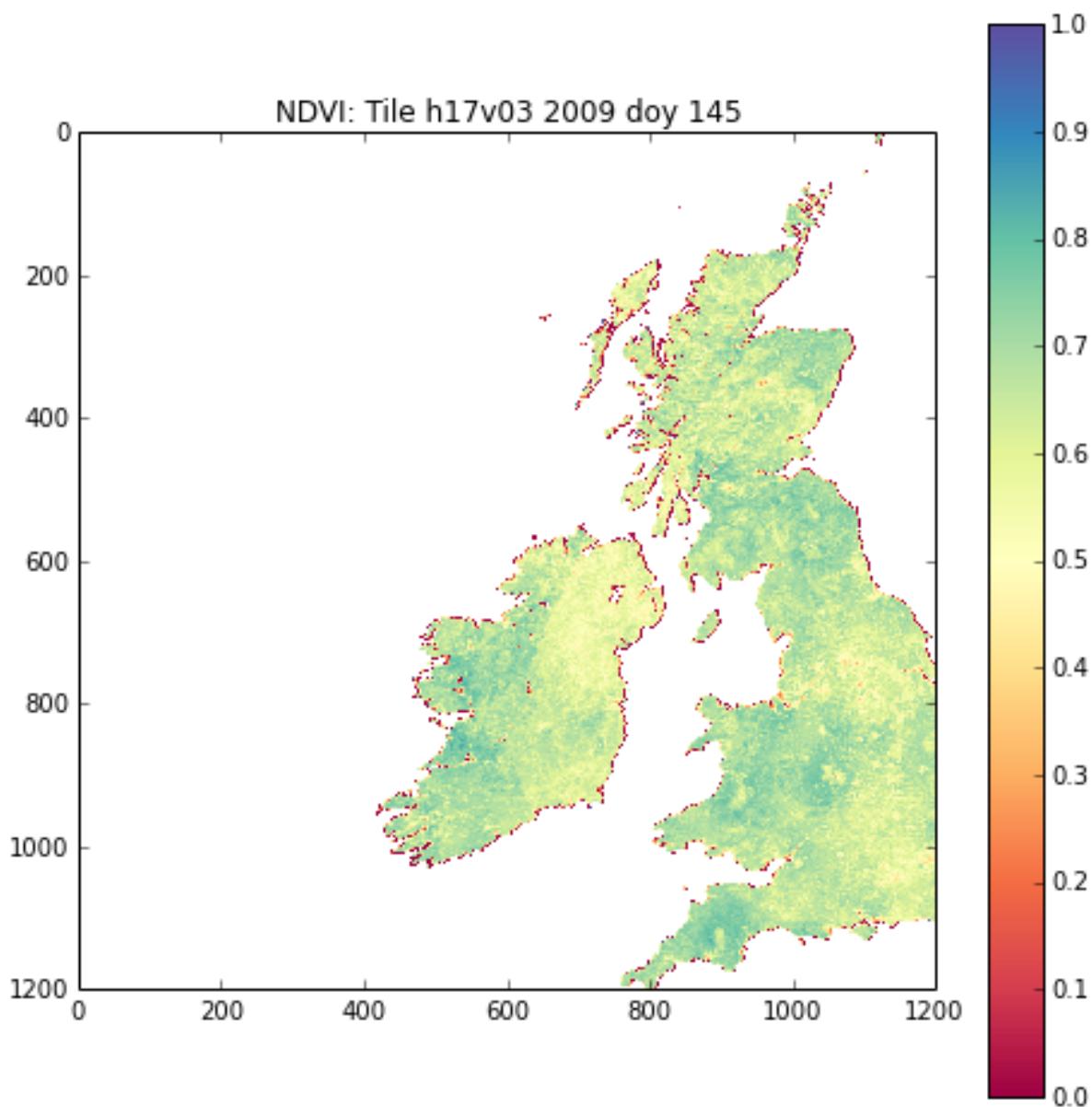
-c:5: RuntimeWarning: invalid value encountered in divide
```

Now plot it:

```
import pylab as plt

# figure size
plt.figure(figsize=(8, 8))
# title
plt.title('NDVI: Tile %s %d doy %03d' %(tile,year,doy))
# colour map
cmap = plt.get_cmap('Spectral')
# plot the figure
plt.imshow(ndvi,interpolation='none',cmap=cmap,vmin=0.,vmax=1.)
# colour bar
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x10687ce60>
```



We notice in this dataset that there are some ‘funnies’ (unreliable data) around the coastline, which are probably due to negative reflectance values.

We could try, for instance to build a mask for these, supposing them to be some other ‘invalid’ number, but in this dataset, we have some other data layers that can help:

```
nc.variables.keys()
```

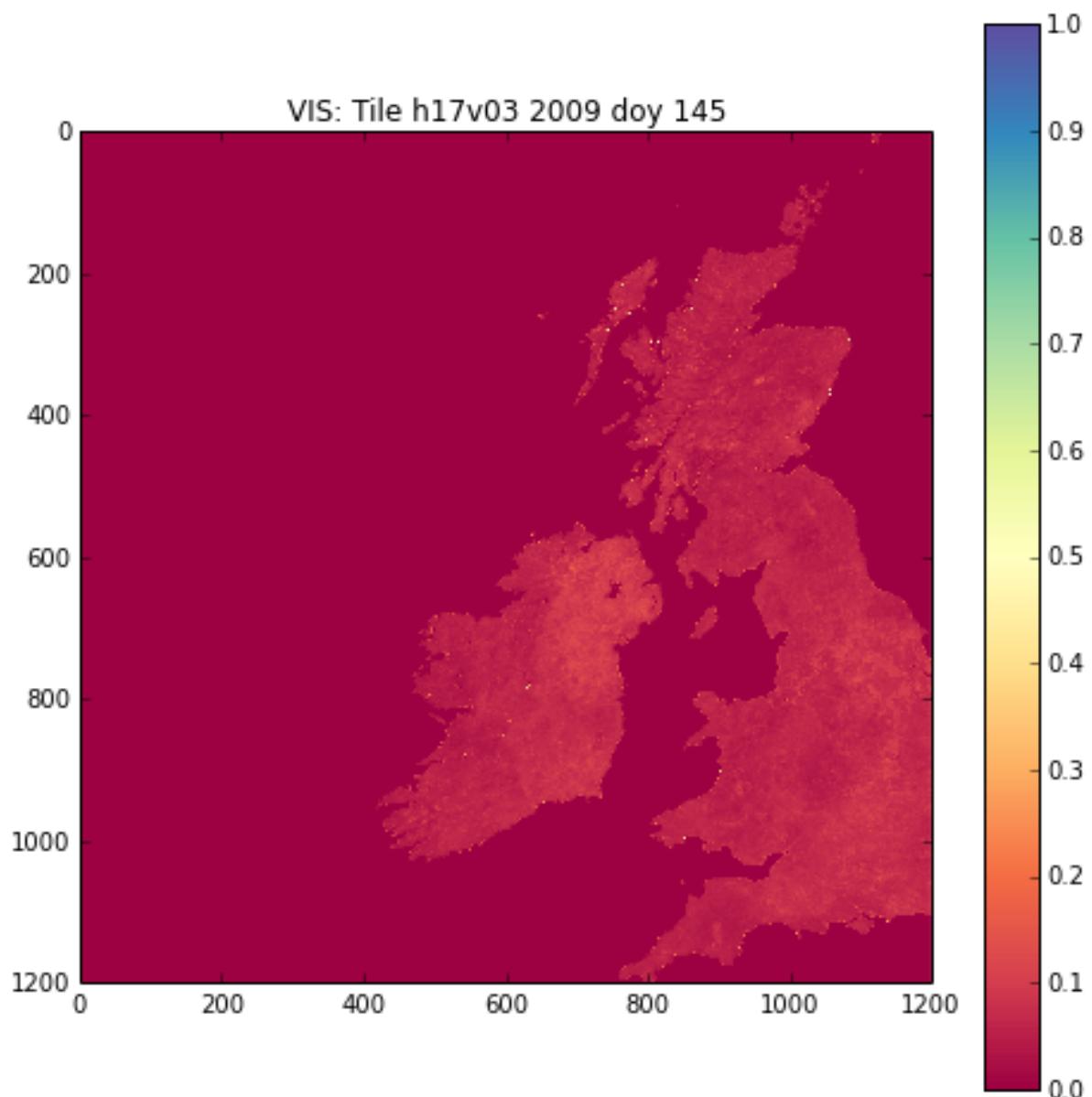
```
[u'metadata',
 u'DHR_VIS',
 u'DHR_NIR',
 u'DHR_SW',
 u'BHR_VIS',
 u'BHR_NIR',
 u'BHR_SW',
 u'DHR_sigmaVIS',
 u'DHR_sigmaNIR',
 u'DHR_sigmaSW',
 u'BHR_sigmaVIS',
 u'BHR_sigmaNIR',
 u'BHR_sigmaSW',
```

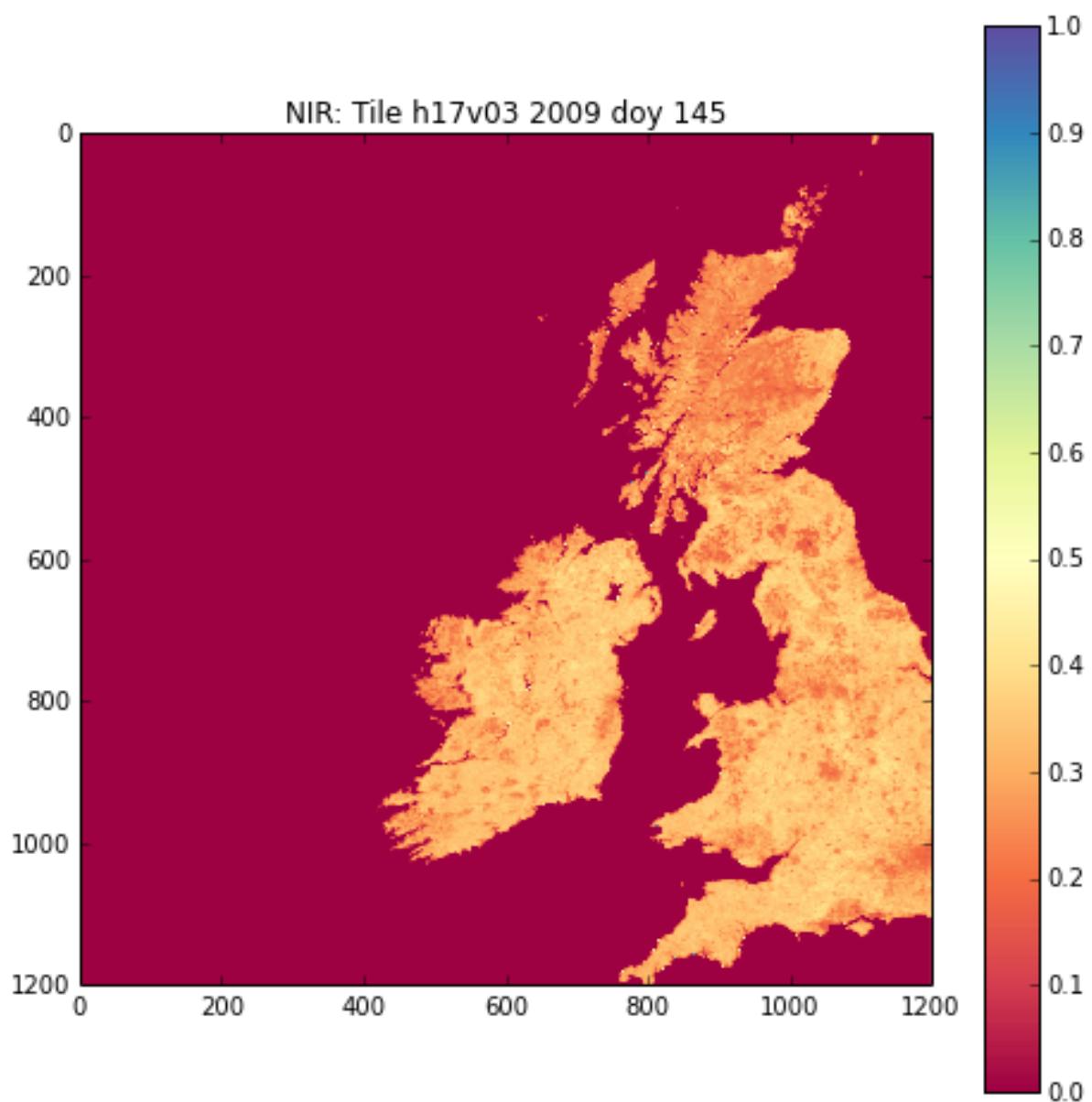
```
u'Weighted_Number_of_Samples',
u'Relative_Entropy',
u'Goodness_of_Fit',
u'Snow_Fraction',
u'Data_Mask',
u'Solar_Zenith_Angle',
u'lat',
u'lon',
u'crs']

# better have a look at the individual bands as well
# plot the vis and nir bands
plt.figure(figsize=(8,8))
plt.title('VIS: Tile %s %d doy %03d'%(tile,year,doy))
cmap = plt.get_cmap('Spectral')
plt.imshow(vis,interpolation='none',cmap=cmap,vmin=0.,vmax=1.)
plt.colorbar()

plt.figure(figsize=(8,8))
plt.title('NIR: Tile %s %d doy %03d'%(tile,year,doy))
cmap = plt.get_cmap('Spectral')
plt.imshow(nir,interpolation='none',cmap=cmap,vmin=0.,vmax=1.)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x107236d88>
```





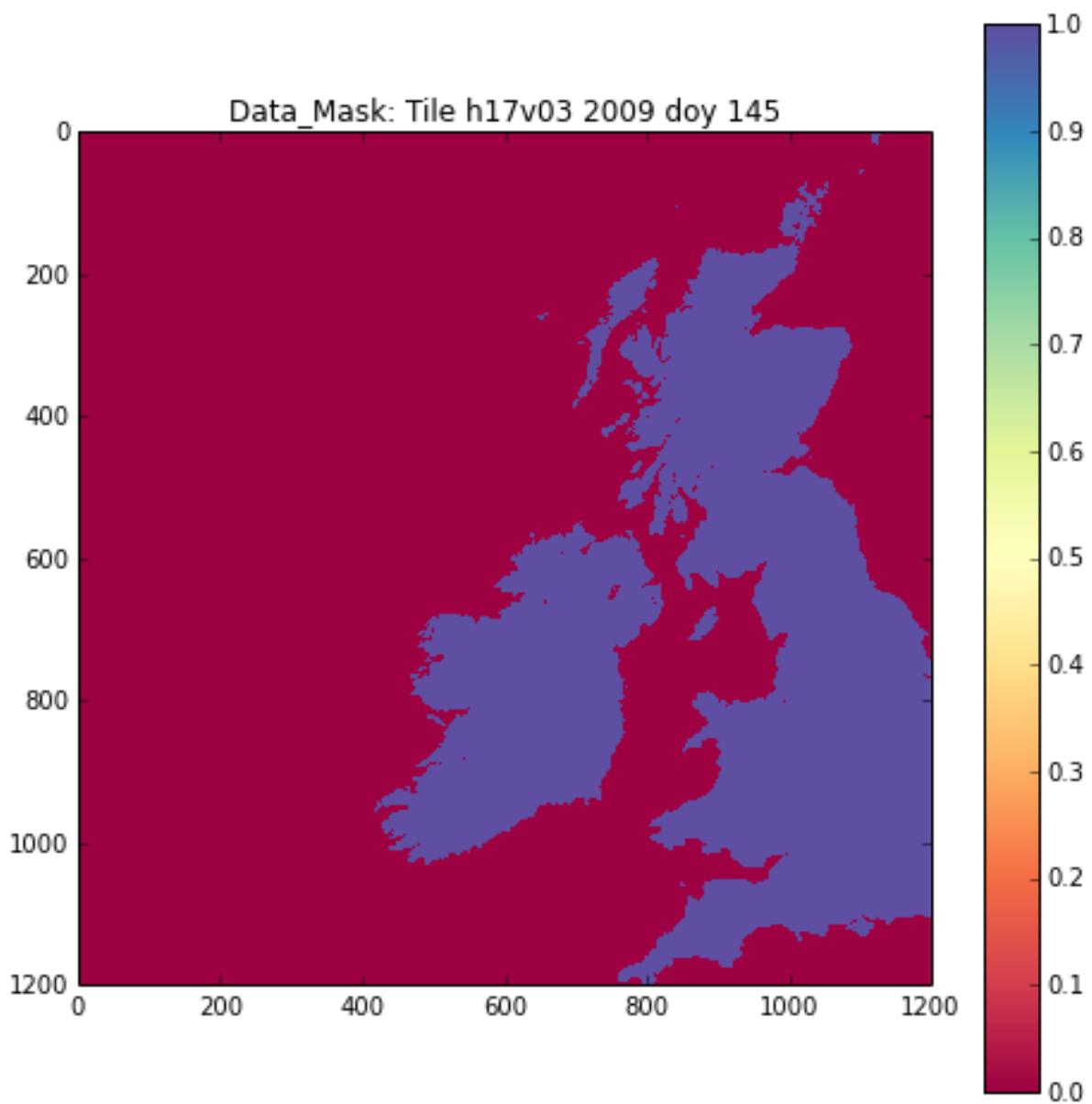
Apart from a few minor outliers, these data look fine.

Let's try developing a mask from Data_Mask:

```
mask = np.array(nc.variables['Data_Mask']).astype(bool)

# plot it
plt.figure(figsize=(8,8))
plt.title('Data_Mask: Tile %s %d doy %03d' % (tile,year,doy))
cmap = plt.get_cmap('Spectral')
plt.imshow(mask, interpolation='none', cmap=cmap, vmin=0., vmax=1.)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x1068ab710>
```



The mask is True where there are valid (land) data.

In a masked array, we want the opposite of this.

We can't directly use `not`, but we can use the bitwise operator `~`:

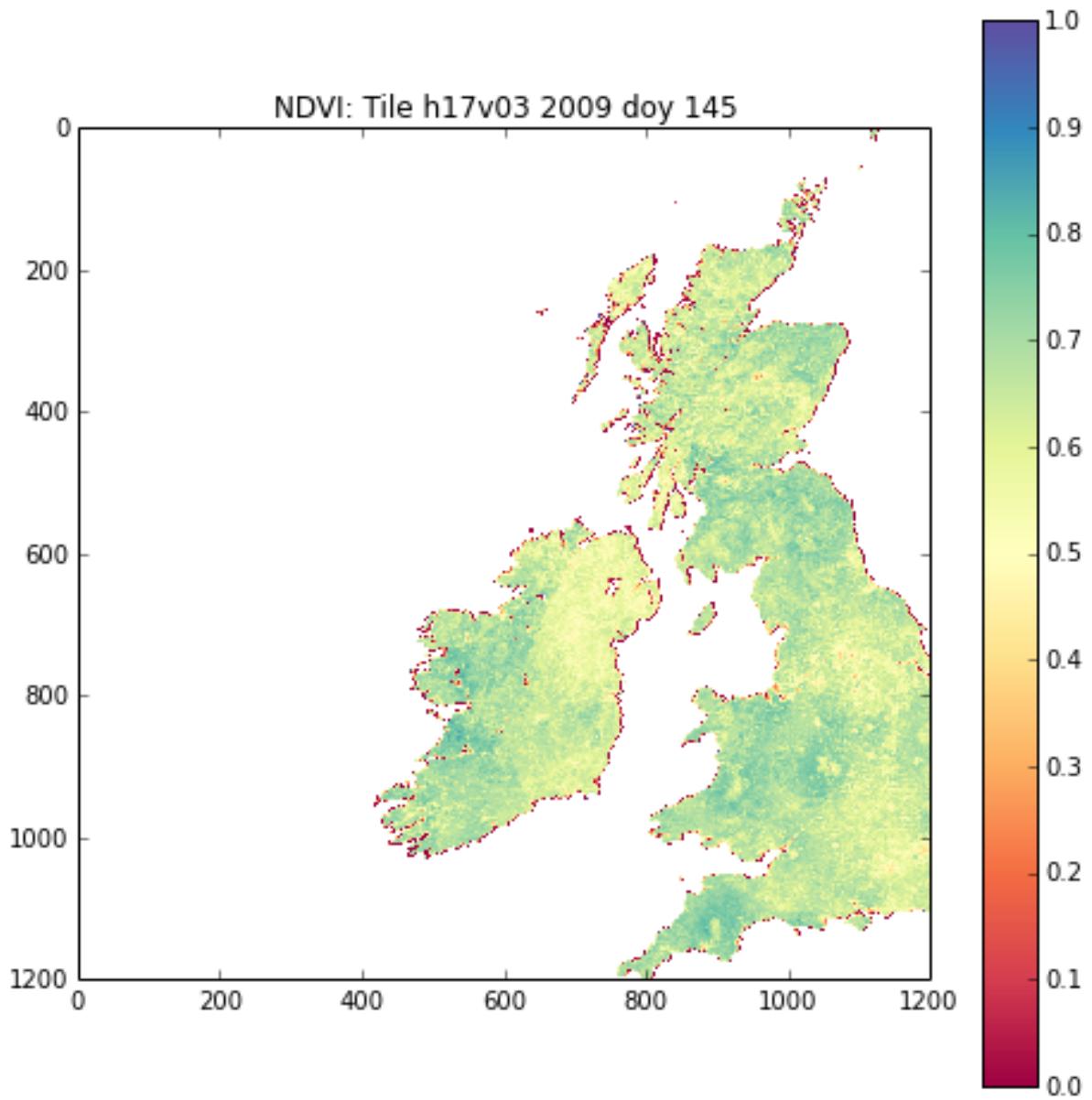
Use the mask in a masked array:

```
import numpy.ma as ma

vis = ma.array(vis,mask=~mask)
nir = ma.array(nir,mask=~mask)
ndvi = (nir - vis)/(nir + vis)

plt.figure(figsize=(8,8))
plt.title('NDVI: Tile %s %d doy %03d'%(tile,year,doy))
cmap = plt.get_cmap('Spectral')
plt.imshow(ndvi,interpolation='none',cmap=cmap,vmin=0.,vmax=1.)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x1072aa290>
```



The data mask hasn't solved the problem for NDVI then.

A problem might arise from a small number of negative reflectance values in the dataset.

We can create masks for these:

```
mask1 = vis < 0.
mask2 = nir < 0
print 'number of -ve VIS pixels', np.sum(mask1)
print 'number of -ve NIR pixels', np.sum(mask2)

number of -ve VIS pixels 317
number of -ve NIR pixels 934
```

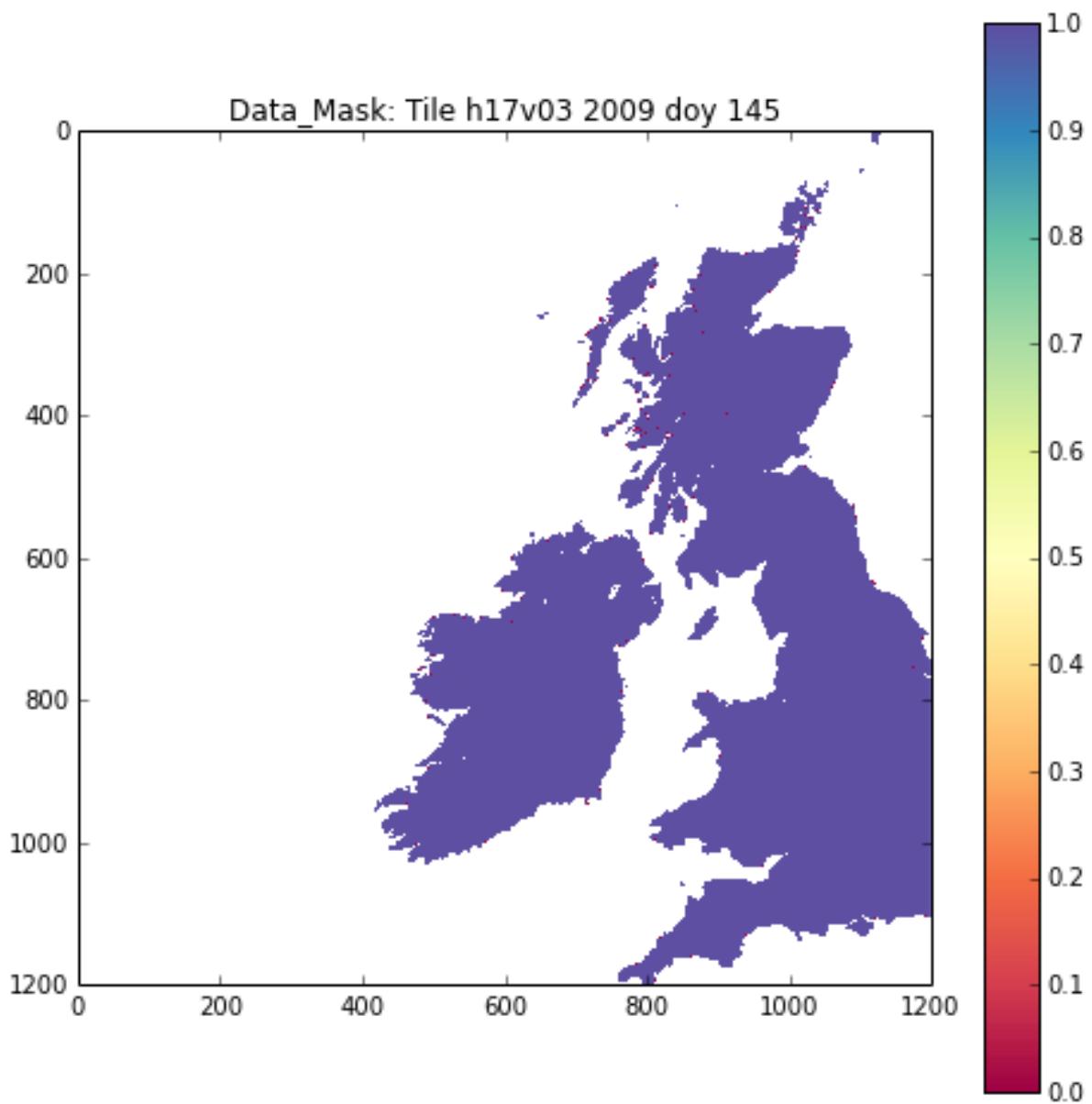
and we can combine them with a bitwise operator, | (or) or & (and) in this case (reversing the conditions):

```
mask = np.array(nc.variables['Data_Mask']).astype(bool) & (vis > 0) & (nir > 0)

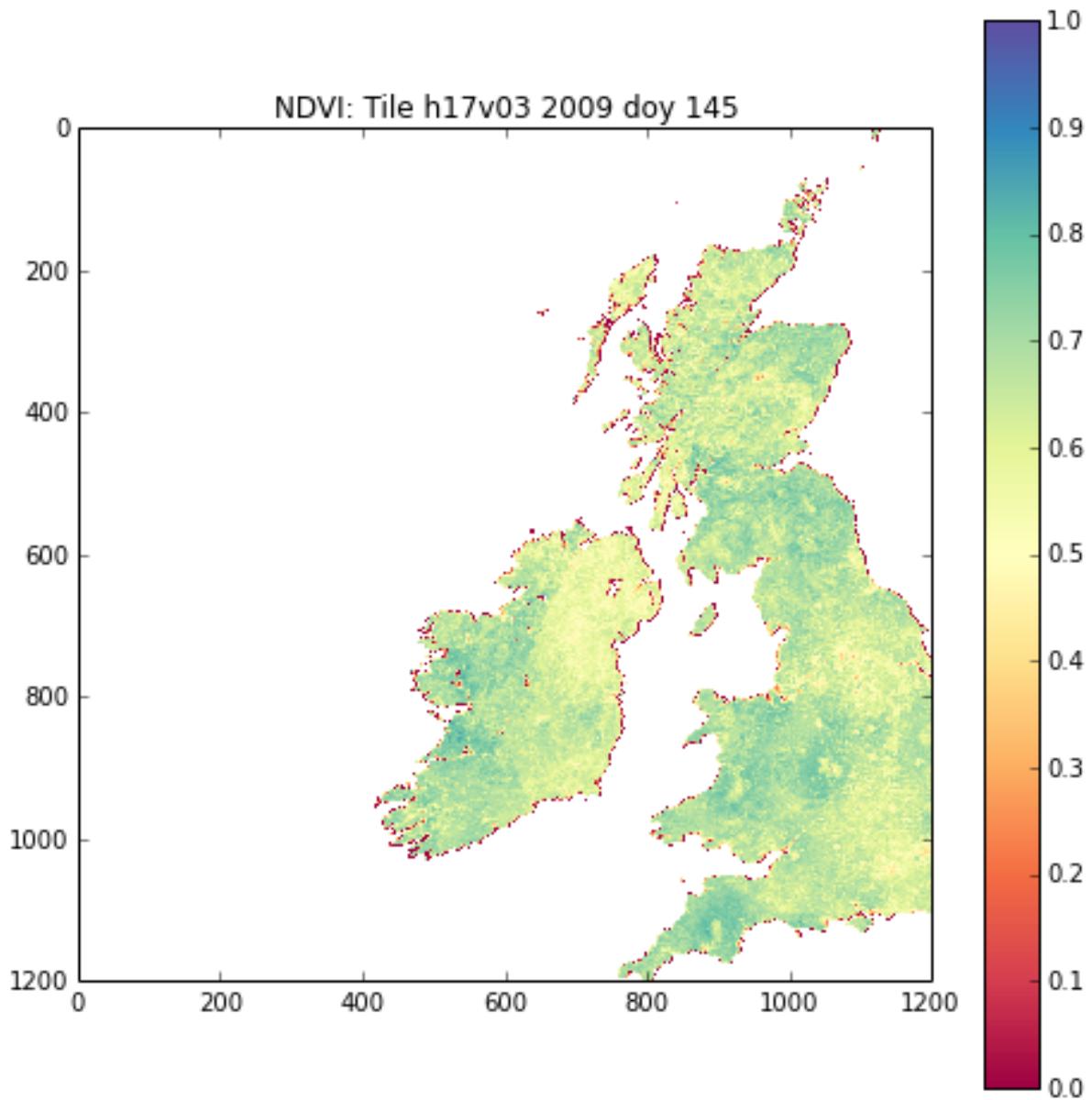
# plot it
plt.figure(figsize=(8,8))
plt.title('Data_Mask: Tile %s %d doy %03d'%(tile,year,doy))
cmap = plt.get_cmap('Spectral')
```

```
plt.imshow(mask, interpolation='none', cmap=cmap, vmin=0., vmax=1.)  
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar instance at 0x107c865f0>
```



```
vis = ma.array(vis,mask=~mask)  
nir = ma.array(nir,mask=~mask)  
ndvi = (nir - vis)/(nir + vis)  
  
plt.figure(figsize=(8, 8))  
plt.title('NDVI: Tile %s %d doy %03d' %(tile,year,doy))  
cmap = plt.get_cmap('Spectral')  
plt.imshow(ndvi, interpolation='none', cmap=cmap, vmin=0., vmax=1.)  
plt.colorbar()  
  
<matplotlib.colorbar.Colorbar instance at 0x107320830>
```



This hasn't entirely sorted it either.

Next have a look at a few more fields before going further:

```
# demonstration of multiple subplots

datasets = np.array([['DHR_VIS', 'DHR_NIR'], \
                     ['DHR_sigmaVIS', 'DHR_sigmaNIR'], \
                     ['Data_Mask', 'Weighted_Number_of_Samples']])

# load up all datasets in dict data
data = {}

dlist = datasets.copy().flatten()

for d in dlist:
    data[d] = np.array(nc.variables[d])

mask = data['Data_Mask'].astype(bool) & ( \
    (data['DHR_VIS'] > 0.) | \
    (data['DHR_NIR'] > 0.))
```

```

s = datasets.shape

# how big for each subplot ?
big = 5

# set the figure size
plt.figure(figsize=(s[1]*big,s[0]*big))

# colorbars for subplots are a bit tricky
# here's one way of sorting this
# using dataset shapes
from matplotlib import gridspec
gs = gridspec.GridSpec(s[0],s[1])

# colour map
cmap = plt.get_cmap('Spectral')

for i,d0 in enumerate(datasets):
    for j,d in enumerate(d0):

        data[d] = ma.array(data[d],mask=~mask)

        axes = plt.subplot(gs[i,j])
        axes.set_title(d)
        # no axis ticks
        axes.set_xticks([])
        axes.set_yticks([])

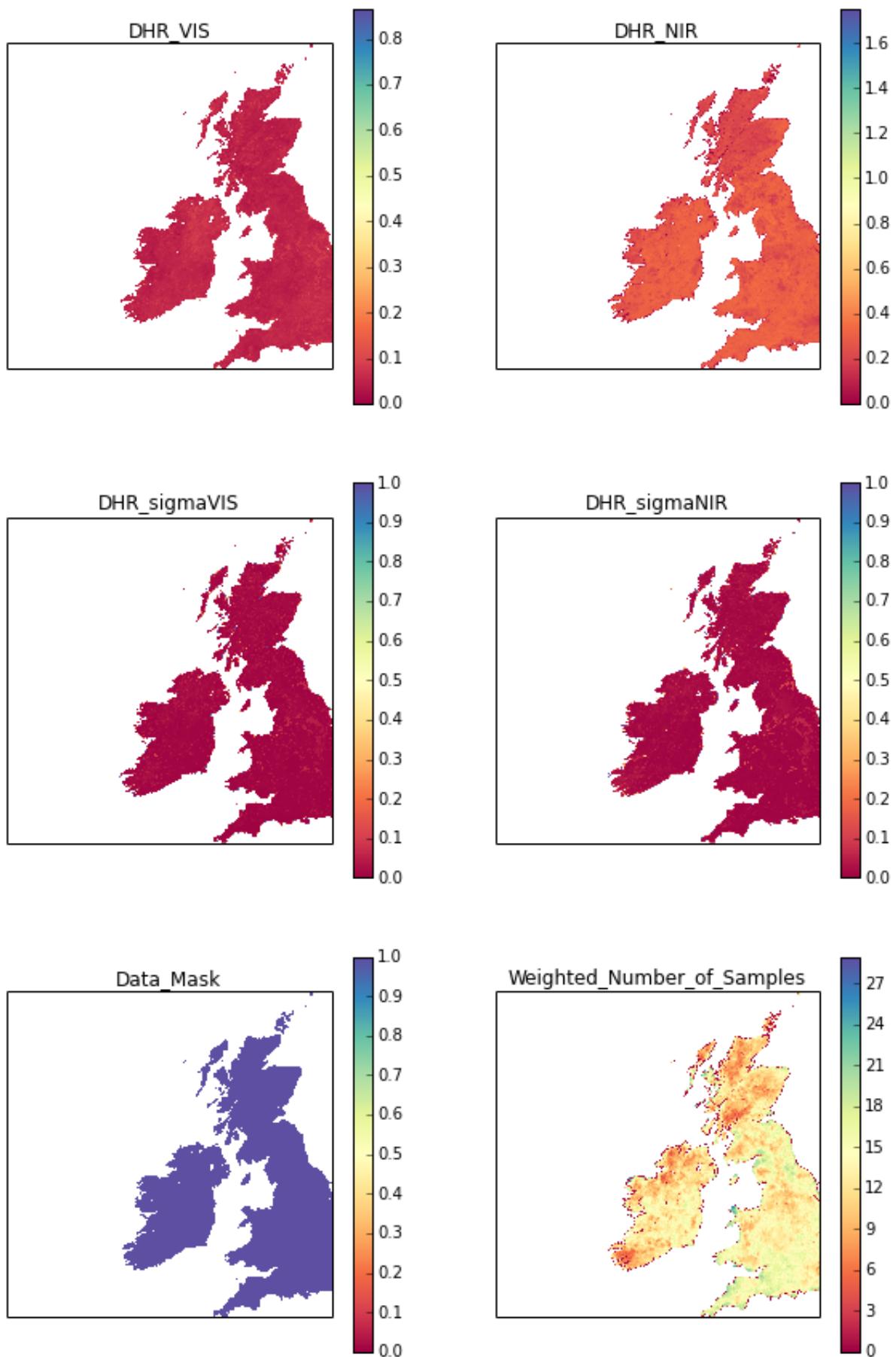
        im = axes.imshow(data[d],cmap=cmap,interpolation='none',vmin=0.)
        plt.colorbar(im)

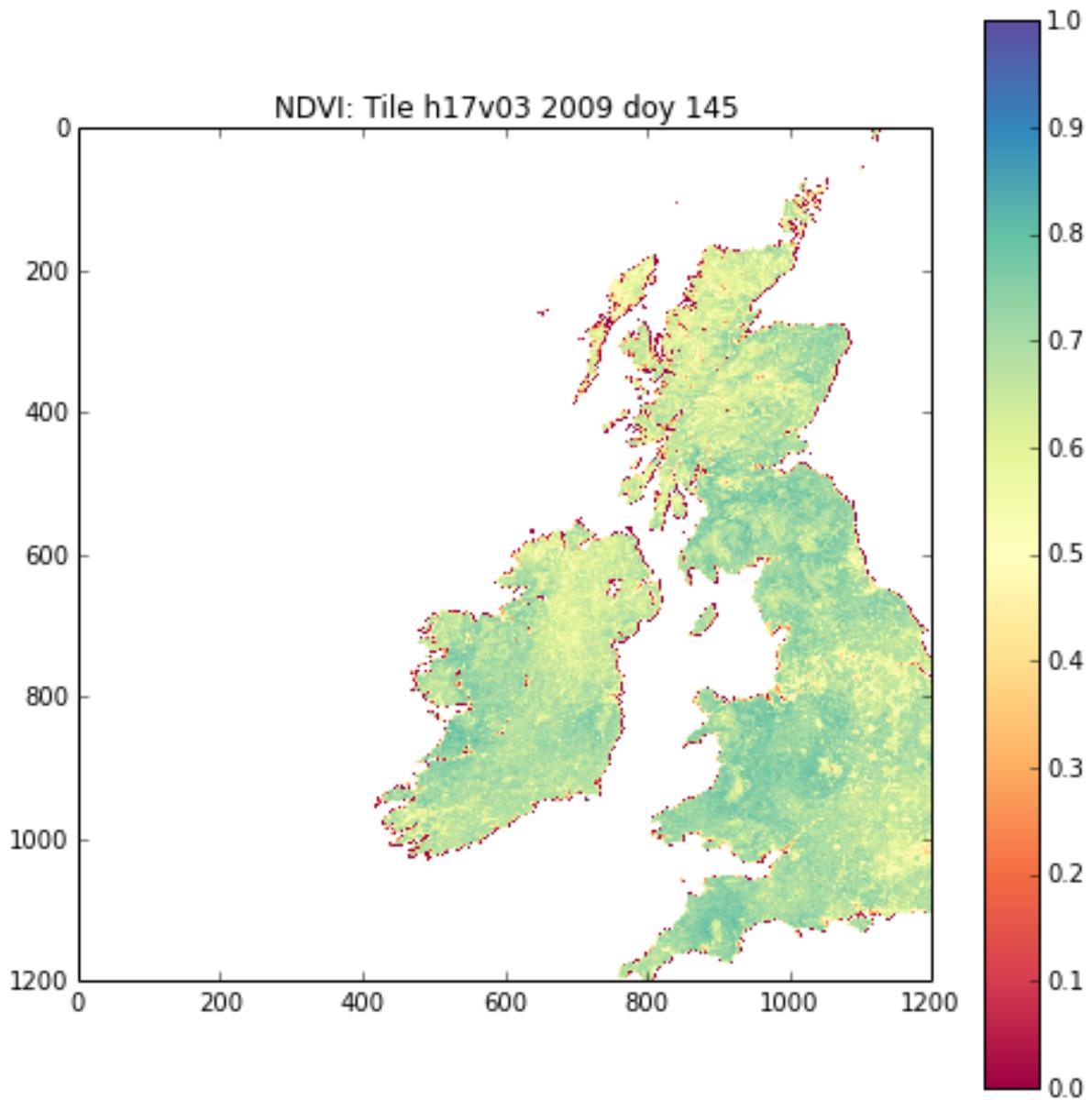
ndvi = (data['DHR_NIR'] - data['DHR_VIS'])/(data['DHR_NIR'] + data['DHR_VIS'])

plt.figure(figsize=(8,8))
plt.title('NDVI: Tile %s %d doy %03d'%(tile,year,doy))
cmap = plt.get_cmap('Spectral')
plt.imshow(ndvi,interpolation='none',cmap=cmap,vmin=0.,vmax=1.)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x116dd8ab8>

```





So it looks as though we need to filter on `'Weighted_Number_of_Samples'` as well, and perhaps on uncertainty:

```
# demonstration of multiple subplots

datasets = np.array([['DHR_VIS', 'DHR_NIR'], \
                     ['DHR_sigmaVIS', 'DHR_sigmaNIR'], \
                     ['Data_Mask', 'Weighted_Number_of_Samples']])

# load up all datasets in dict data
data = {}

dlist = datasets.copy().flatten()

for d in dlist:
    data[d] = np.array(nc.variables[d])

mask = data['Data_Mask'].astype(bool) & \
       (data['Weighted_Number_of_Samples'] > 0.5) & \
       (data['DHR_sigmaVIS'] <= 0.8) & \
       (data['DHR_sigmaNIR'] <= 0.8) &
```

```
(data['DHR_VIS'] >= 0.) & \
(data['DHR_NIR'] >= 0.)

s = datasets.shape

# how big for each subplot ?
big = 5

# set the figure size
plt.figure(figsize=(s[1]*big,s[0]*big))

# colorbars for subplots are a bit tricky
# here's one way of sorting this
# using dataset shapes
from matplotlib import gridspec
gs = gridspec.GridSpec(s[0],s[1])

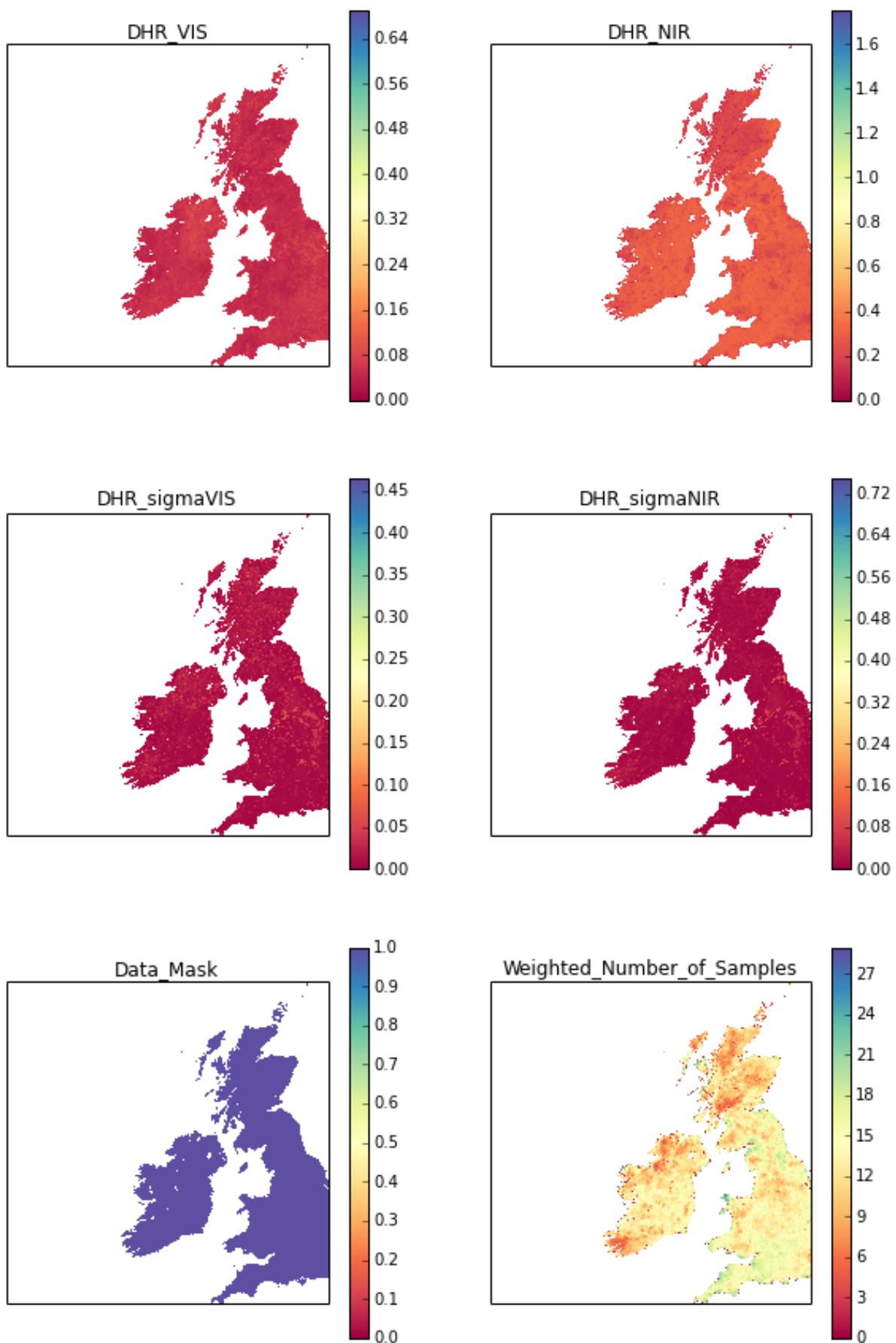
# colour map
cmap = plt.get_cmap('Spectral')

for i,d0 in enumerate(datasets):
    for j,d in enumerate(d0):

        data[d] = ma.array(data[d],mask=~mask)

        axes = plt.subplot(gs[i,j])
        axes.set_title(d)
        # no axis ticks
        axes.set_xticks([])
        axes.set_yticks([])

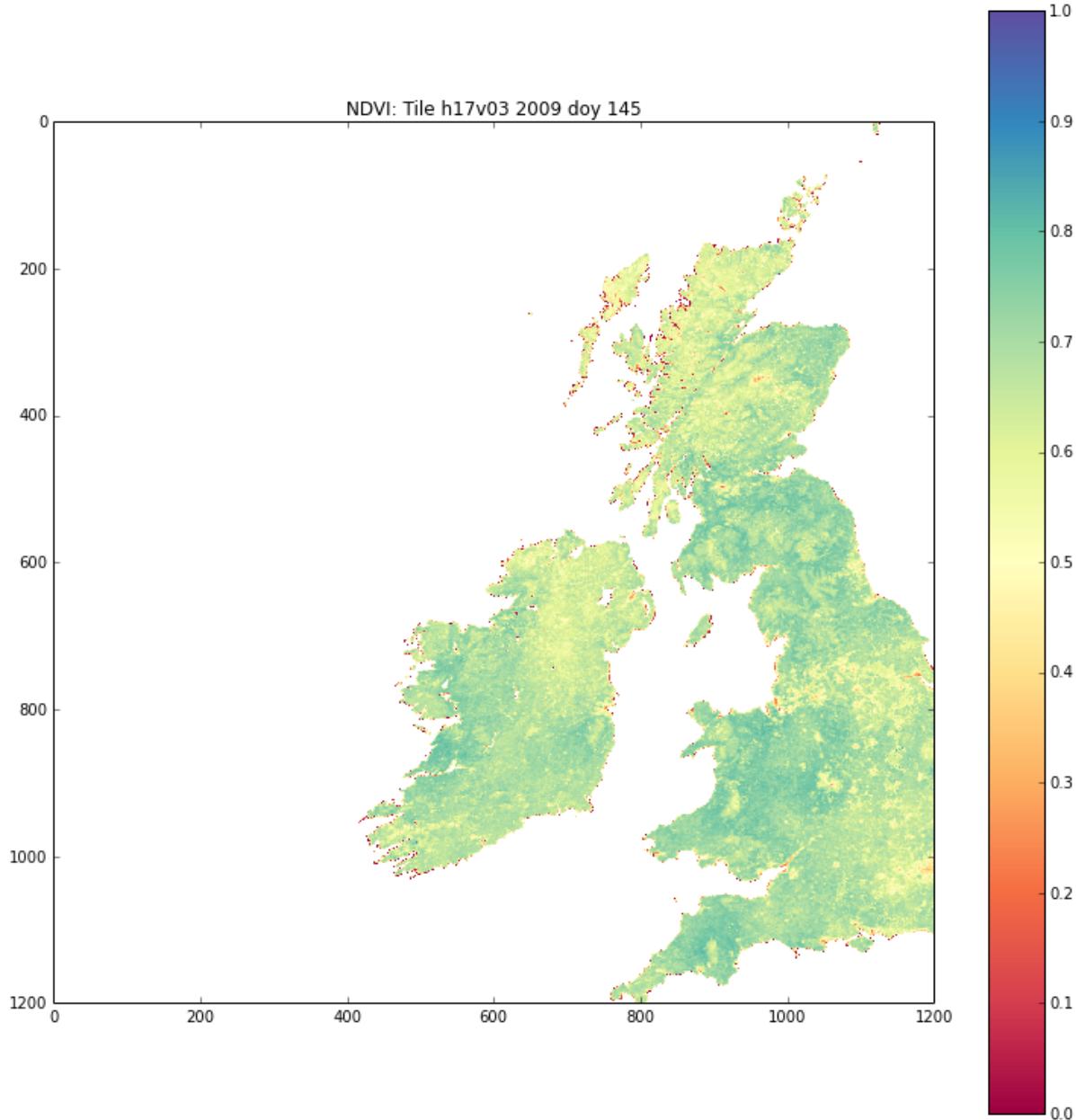
        im = axes.imshow(data[d],cmap=cmap,interpolation='none',vmin=0.)
        plt.colorbar(im)
```



```
ndvi = (data['DHR_NIR'] - data['DHR_VIS'])/(data['DHR_NIR'] + data['DHR_VIS'])

plt.figure(figsize=(13,13))
plt.title('NDVI: Tile %s %d doy %03d' %(tile,year,doy))
cmap = plt.get_cmap('Spectral')
plt.imshow(ndvi, interpolation='none', cmap=cmap, vmin=0., vmax=1.)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x115bb1cb0>
```



Thats quite a bit better, but still not perfect.

Experiment with the conditions of the masking to see how you can get rid of the odd pixels (the ‘red’ ones in the above). Do *not* filter on “ndvi” itself, as we *might* be interested in negative “ndvi” values in some cases.

Once you think you have some useful filtering conditions, try it out on some different dates and tiles.

Some other things to try:

- Write parts of the code as functions.

- Put the code developed into a file and run it from the unix command line.

27.3 A3.3 Solar Radiation model

If you followed the advanced material for the previous chapter, you will have noted the use of `pyephem` as a module that we can use for calculating the solar zenith angle.

There is a similar package `pysolar` that is a little easier to use for solar radiation calculations.

We will install a package `pysolar` into your user area:

at a unix prompt, type:

```
!easy_install --user pysolar

Searching for pysolar
Best match: Pysolar 0.5
Processing Pysolar-0.5-py2.7.egg
Pysolar 0.5 is already the active version in easy-install.pth

Using /Users/plewis/.local/lib/python2.7/site-packages/Pysolar-0.5-py2.7.egg
Processing dependencies for pysolar
Finished processing dependencies for pysolar
```

If all goes well, the text that comes up at the terminal should tell you that this has installed (e.g. in `/home/plewis/.local/lib/python2.7/site-packages/Pysolar-0.5-py2.7.egg`).

We can test to see if we can load and run this package:

```
# from https://github.com/pingswept/pysolar/wiki/examples
import Pysolar
from datetime import datetime

# UCL lat/lon
lat = 51.5248
lon = -0.1336

hour = 12
minute = 0
second = 0
month = 10 # ie October
day = 13
year = 2013

d = datetime(year, month, day, hour, minute, second)
altitude_deg = Pysolar.GetAltitude(lat, lon, d)
zenith = 90. - altitude_deg
# W m^-2
solar = Pysolar.solar.radiation.GetRadiationDirect(d, altitude_deg)
print zenith,solar

59.5052193764 834.866993323

def solar(year, month, day, hour, lat_deg, lon_deg, minute=0, second=0):
    '''Return solar zenith and clear sky radiation
       for given lat, lon and time/date
    '''
    from datetime import datetime
    import Pysolar

    d = datetime(year, month, day, hour, minute, second)
    altitude_deg = Pysolar.GetAltitude(lat_deg, lon_deg, d)
    # W m^-2
```

```
solar_rad = Pysolar.solar.radiation.GetRadiationDirect(d, altitude_deg)
return 90. - altitude_deg,solar_rad

# or import from local module

import sys,os
# put local directory into the path
sys.path.insert(0,os.path.abspath('files%spython'%os.sep))

from solar import solar

import numpy as np

# UCL lat/lon
lat = 51.5248
lon = -0.1336

second = 0
month = 10 # ie October
day = 13
year = 2013

radiation_fields = '#hour zenith solar_rad month day lat lon'

radiation = []
for hour in xrange(24):
    for minute in xrange(60):
        thr = hour + minute/60.
        # append data line as tuple
        radiation.append((thr,) + \
                          solar(year, month, day, hour, lat, lon, minute=minute) +\
                          (month, day, lat, lon))
# convert to numpy array
# transpose so access eg zenith as
# radiation[0]
radiation = np.array(radiation).T

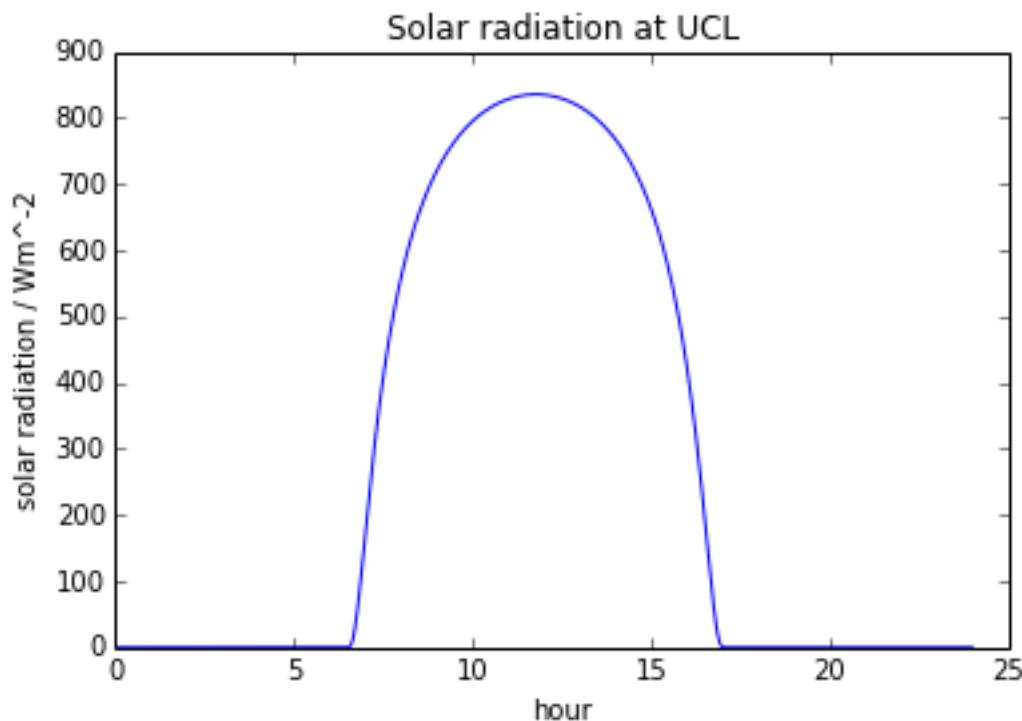
# so we have radiation as
print radiation.shape
print radiation.ndim
print radiation

(7, 1440)
2
[[ 0.00000000e+00  1.66666667e-02  3.33333333e-02 ... ,  2.39500000e+01
  2.39666667e+01  2.39833333e+01]
 [ 1.36158787e+02  1.36145826e+02  1.36131901e+02 ... ,  1.36561804e+02
  1.36551449e+02  1.36540121e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 ... ,  0.00000000e+00
  0.00000000e+00]
 ... ,
 [ 1.30000000e+01  1.30000000e+01  1.30000000e+01 ... ,  1.30000000e+01
  1.30000000e+01  1.30000000e+01]
 [ 5.15248000e+01  5.15248000e+01  5.15248000e+01 ... ,  5.15248000e+01
  5.15248000e+01  5.15248000e+01]
 [-1.33600000e-01 -1.33600000e-01 -1.33600000e-01 ... , -1.33600000e-01
  -1.33600000e-01 -1.33600000e-01]]]

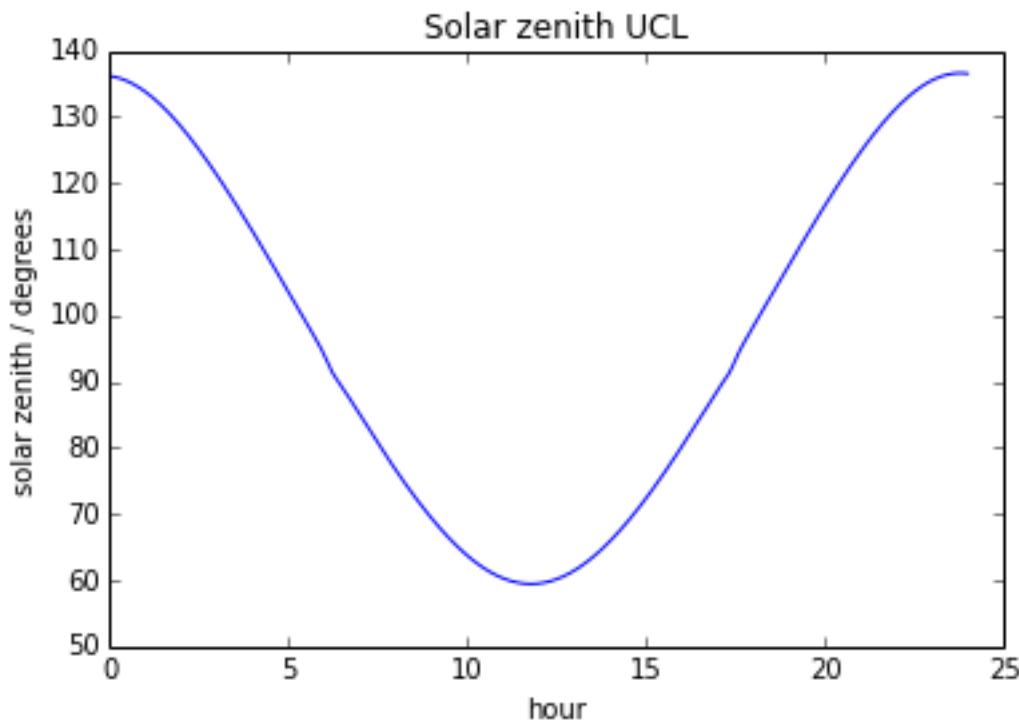
import pylab as plt

plt.title('Solar radiation at UCL')
plt.xlabel('hour')
plt.ylabel('solar radiation / Wm^-2')
```

```
plt.plot(radiation[0],radiation[2])  
[<matplotlib.lines.Line2D at 0x10681fed0>]
```



```
import pylab as plt  
  
plt.title('Solar zenith UCL')  
plt.xlabel('hour')  
plt.ylabel('solar zenith / degrees')  
plt.plot(radiation[0],radiation[1])  
  
[<matplotlib.lines.Line2D at 0x1068bde50>]
```



E3.3 Exercise: Solar radiation modelling

This is a better modelling of solar radiation that we did in the main part of the class today.

There are several things we could do with this.

For example, if we know the albedo, we can calculate the absorbed radiation as previously done (if we assume for the moment albedo is constant with solar zenith angle ... which it isn't, generally), but can now extend over the whole day and integrate to get the total energy per metre squared.

From above, we have `power` per unit area, in Watts per metre squared.

This is the same as `energy` per unit area per second (i.e. the same as $J/(m\ s)$).

So if we sum up the solar radiation from above over the day and multiply by the time interval in seconds (time interval above is 1 minute, so 60 seconds), we get:

```
power_density = radiation[2].sum() * 60
print 'power per unit area = %.3f MJ / m^2' % (power_density/10**6)
```

```
power per unit area = 23.687 MJ / m^2
```

and we could now look at e.g. variations in this over the year (NB this will take some time to calculate if you step every day and minute, so we step every 30 minutes here):

```
import numpy as np

def radiation(year, month, day, lat, lon, minute_step=30):
    rad = []
    for hour in xrange(24):
        for minute in xrange(0, 60, minute_step):
            thr = hour + minute/60.
            # append data line as tuple
            rad.append((thr,) + \
                      solar(year, month, day, hour, lat, lon, minute=minute) +\
                      (month, day, lat, lon))
    # convert to numpy array
    # transpose so access eg zenith as
    # rad[0]
```

```

rad = np.array(rad).T
return rad

def days_in_month(month,year=2013):
    ''' number of days in month'''
    import calendar
    return calendar.monthrange(year,month)[1]

# UCL lat/lon
lat = 51.5248
lon = -0.1336

year = 2013

minute_step = 30

pd = []
for month in xrange(12):
    ndays = days_in_month(month+1,year=year)
    print month,ndays
    for day in xrange(ndays):
        rad = radiation(year, month+1, day+1, lat, lon,minute_step=minute_step)
        pd.append([month+day/float(ndays),rad[2].sum() * 60 * minute_step])
pd = np.array(pd).T

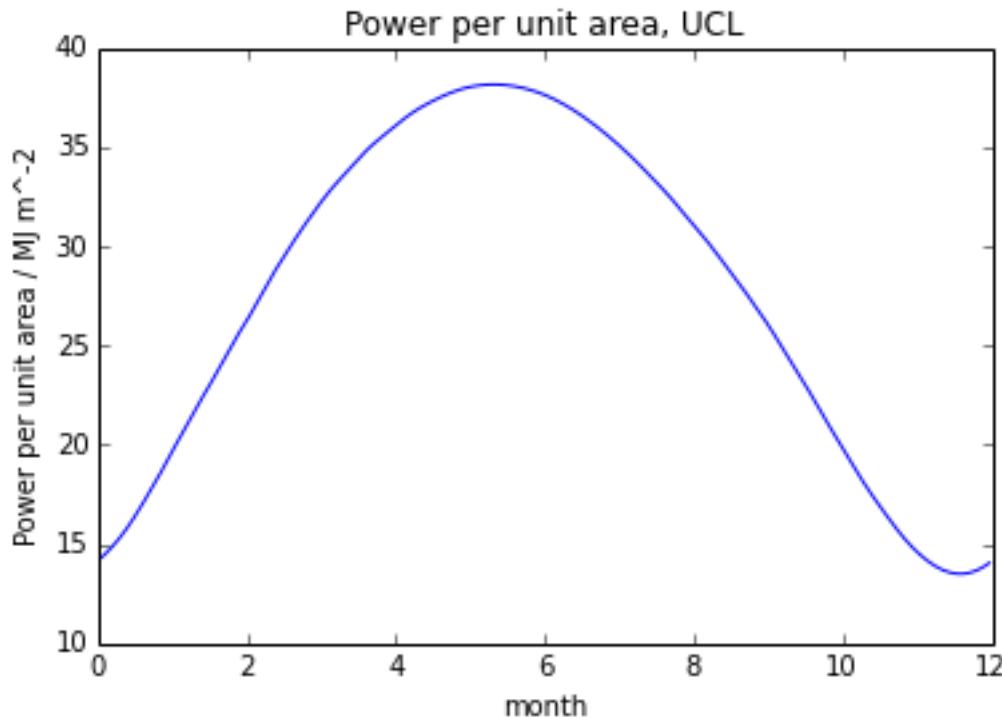
0 31
1 28
2 31
3 30
4 31
5 30
6 31
7 31
8 30
9 31
10 30
11 31

import pylab as plt

plt.title('Power per unit area, UCL')
plt.xlabel('month')
plt.ylabel('Power per unit area / MJ m^-2')
plt.plot(pd[0],pd[1]/10**6)

[<matplotlib.lines.Line2D at 0x106abdc50>]

```



or, if we want to sum over a month:

```
# UCL lat/lon
lat = 51.5248
lon = -0.1336

year = 2013

pd = []

for month in xrange(12):
    pd_month = []
    ndays = days_in_month(month+1,year=year)
    print month,ndays
    for day in xrange(ndays):
        rad = radiation(year, month+1, day+1, lat, lon)
        pd_month.append([rad[2].sum() * 60 * 30])
    pd_month = np.array(pd_month).T
    pd.append([month,pd_month.sum()])
pd = np.array(pd).T

0 31
1 28
2 31
3 30
4 31
5 30
6 31
7 31
8 30
9 31
10 30
11 31

import pylab as plt

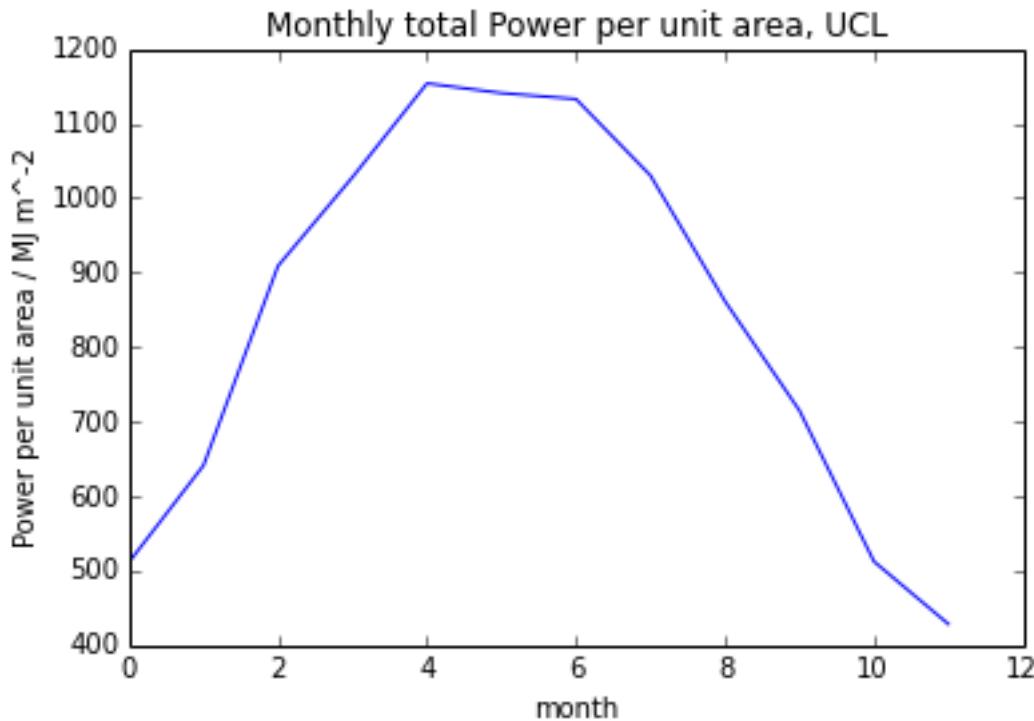
plt.title('Monthly total Power per unit area, UCL')
```

```

plt.xlabel('month')
plt.ylabel('Power per unit area / MJ m^-2')
plt.plot(pd[0],pd[1]/10**6)

[<matplotlib.lines.Line2D at 0x107234d10>]

```



27.4 E3.3 Improved Solar Radiation Modelling

Using the material above and the global albedo datasets from the main class material, **calculate an improved estimate of the total absorbed power per unit area per month (MJ per m² per month) for the Earth land surface.**

You should do this with a function that will take as input the year and returns the monthly total absorbed power density (MJ m⁻² per month) and the monthly total power density (MJ m⁻² per month).

You might have an optional argument `minute_step` to control the resolution of the calculation as above.

You could then use this to derive latitudinal variations in annual and latitudinal total absorbed power per unit area.

CHAPTER
TWENTYEIGHT

E3.3: EXERCISE: IMPROVED SOLAR RADIATION MODELLING

Using the material above and the global albedo datasets from the main class material, **calculate an improved estimate of the total absorbed power per unit area per month (MJ per m² per month) for the Earth land surface.**

You should do this with a function that will take as input the year and returns the monthly total absorbed power density (MJ m⁻² per month) and the monthly total power density (MJ m⁻² per month).

You might have an optional argument `minute_step` to control the resolution of the calculation as above.

You could then use this to derive latitudinal variations in annual and latitudinal total absorbed power per unit area.

AE3.3 ANSWER: IMPROVED SOLAR RADIATION MODELLING

The function definition should look something like:

```
def absorbed_power_density(year,minute_step=60):
    '''Function to calculate the monthly total absorbed solar radiation
    power density in MJ / m^2 for a given year
    given an input dataset albedo and associated lat and lon
    information (in degrees).

    The shape of the albedo dataset is: (12,nlat,nlon)

    Aguments:
        year : integer of the year

    Options:
        minute_step : integer: resolution of steps in minutes. Must be
                      a divisor of 60 (e.g. 10, 15, 30, 60)

    '''

The first thing to do is sort out the lat and lon arrays to be the same shape as albedo.
```

Before that, we'd better read in the albedo, lat and lon datasets here though:

```
import sys
sys.path.insert(0,'files/python')
# modification of masked.py
from masked2 import masked
import numpy as np

year = 2010

data = masked(dataset=['lat','lon','BHR_SW'],year=year)

# take the first lat and lon only as they are all the same
lat = data['lat'][0]
lon = data['lon'][0]
albedo = data['BHR_SW']
s = albedo.shape
print s

(12, 360, 720)

# pad out the lat and lon arrays
# this is a bit tricky so test this in parts
# we dont need this repeated for each month though
lat2 = np.array([lat] * s[2]).T
lon2 = np.array([lon] * s[1])
```

```
print lat2.shape
print lon2.shape

(360, 720)
(360, 720)
```

Now put together the code for solar radiation modelling for a given lat/lon

```
import Pysolar
from datetime import datetime

def solar(year, month, day, hour, lat_deg, lon_deg, minute=0, second=0):
    '''Return solar zenith and clear sky radiation
       for given lat, lon and time/date
    '''
    from datetime import datetime
    import Pysolar

    d = datetime(year, month, day, hour, minute, second)
    altitude_deg = Pysolar.GetAltitude(lat_deg, lon_deg, d)
    # W m^-2
    solar_rad = Pysolar.solar.radiation.GetRadiationDirect(d, altitude_deg)
    return 90. - altitude_deg,solar_rad

def radiation(year, month, day, lat, lon,minute_step=30):
    rad = []
    for hour in xrange(24):
        for minute in xrange(0,60,minute_step):
            thr = hour + minute/60.
            # append data line as tuple
            rad.append((thr,) + \
                       solar(year, month, day, hour, lat, lon, minute=minute) +\
                       (month, day, lat, lon))
    # convert to numpy array
    # transpose so access eg zenith as
    # rad[0]
    rad = np.array(rad).T
    return rad

def days_in_month(month,year=2013):
    ''' number of days in month'''
    import calendar
    return calendar.monthrange(year,month)[1]

# UCL lat/lon
lat = 51.5248
lon = -0.1336

year = 2013

pd = []
for month in xrange(12):
    pd_month = []
    ndays = days_in_month(month+1,year=year)
    print month,ndays
    for day in xrange(ndays):
        rad = radiation(year, month+1, day+1, lat, lon)
        pd_month.append([rad[2].sum() * 60 * 30])
    pd_month = np.array(pd_month).T
    pd.append([month,pd_month.sum()])
pd = np.array(pd).T
```

```

0 31
1 28
2 31
3 30
4 31
5 30
6 31
7 31
8 30
9 31
10 30
11 31

```

Putting that all together now:

```

# functions so far

import sys
sys.path.insert(0,'files/python')
# modification of masked.py
from masked2 import masked
import numpy as np
import Pysolar
from datetime import datetime

def solar(year, month, day, hour, lat_deg, lon_deg, minute=0, second=0):
    '''Return solar zenith and clear sky radiation
       for given lat, lon and time/date
    '''
    from datetime import datetime
    import Pysolar

    d = datetime(year, month, day, hour, minute, second)
    altitude_deg = Pysolar.GetAltitude(lat_deg, lon_deg, d)
    # W m^-2
    solar_rad = Pysolar.solar.radiation.GetRadiationDirect(d, altitude_deg)
    return 90. - altitude_deg,solar_rad

def radiation(year, month, day, lat, lon,minute_step=30):
    rad = []
    for hour in xrange(24):
        for minute in xrange(0,60,minute_step):
            thr = hour + minute/60.
            # append data line as tuple
            rad.append((thr,) + \
                       solar(year, month, day, hour, lat, lon, minute=minute) +\
                       (month, day, lat, lon))
    # convert to numpy array
    # transpose so access eg zenith as
    # rad[0]
    rad = np.array(rad).T
    return rad

def days_in_month(month,year=2013):
    ''' number of days in month'''
    import calendar
    return calendar.monthrange(year,month)[1]

```

Before proceeding, let us examine the variation in total monthly power density with latitude.

With the model developed in the main notes, we would expect no variation with longitude (it was a function of latitude only).

In this more precise model, we would expect only small variation, but we should examine this:

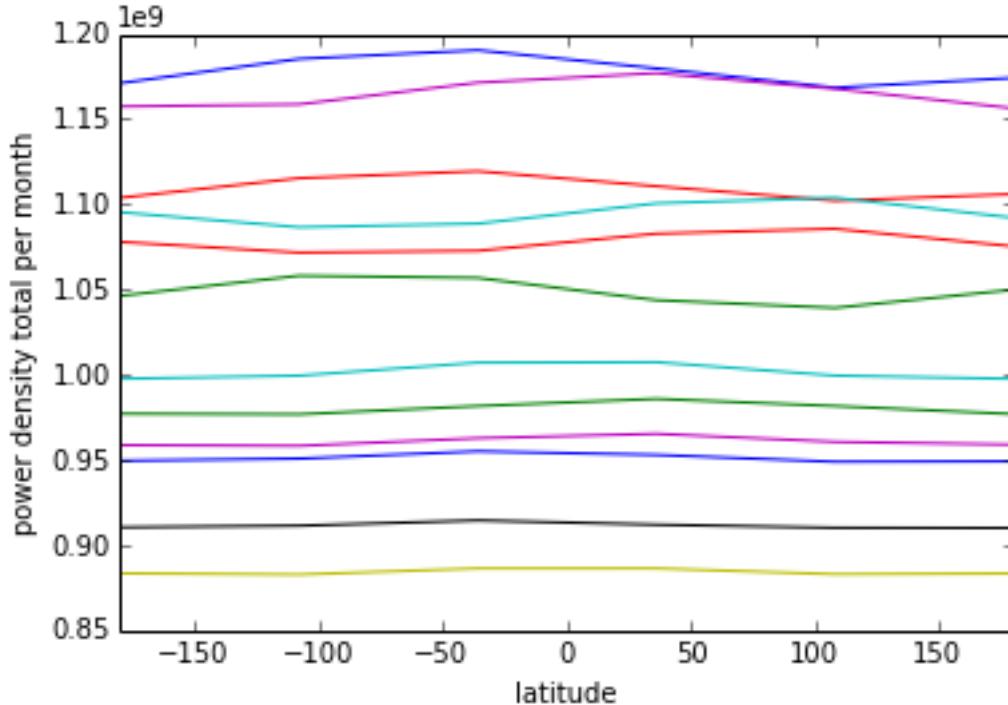
```
'''  
Processing -- test for several lat/long  
import numpy.ma as ma  
  
year = 2010  
  
minute_step = 60  
  
data = masked(dataset=['lat','lon','BHR_SW'],year=year)  
  
lat = data['lat'][0]  
lon = data['lon'][0]  
albedo = data['BHR_SW']  
s = albedo.shape  
lat = ma.array( [lat] *s[2]).T  
lon = ma.array([lon] * s[1])  
  
# array for output  
absorbed = ma.array(np.zeros_like(albedo),mask=albedo.mask)  
#  
  
#for i in xrange(lat.shape[0]):  
#    for j in xrange(lat.shape[1]):  
  
# try it out for 6 samples in longitude, including first and last  
lons = range(0,lon.shape[1],lon.shape[1]/5) + [lon.shape[1]-1]  
  
# an array to hold the sample pd for the sample  
# longitudes  
all_pd = []  
  
i = lon.shape[0]/2  
for j in lons:  
    pd = []  
    # loop over month  
    for month in xrange(12):  
        pd_month = []  
        ndays = days_in_month(month+1,year=year)  
        print i,j,month,ndays  
        # loop over days  
        for day in xrange(ndays):  
            # solar radiation for that day  
            rad = radiation(year, month+1, day+1, \  
                            lat[i,j], lon[i,j], minute_step=minute_step)  
            pd_month.append([rad[2].sum() * 60 * minute_step])  
        pd_month = np.array(pd_month).T  
        pd.append([pd_month.sum()])  
    # pd is the power density for each month  
    all_pd.append(pd)  
  
# MJ per m2  
all_pd = np.array(all_pd).squeeze()  
  
180 0 0 31  
180 0 1 28  
180 0 2 31  
180 0 3 30  
180 0 4 31  
180 0 5 30  
180 0 6 31  
180 0 7 31
```

180 0 8 30
180 0 9 31
180 0 10 30
180 0 11 31
180 144 0 31
180 144 1 28
180 144 2 31
180 144 3 30
180 144 4 31
180 144 5 30
180 144 6 31
180 144 7 31
180 144 8 30
180 144 9 31
180 144 10 30
180 144 11 31
180 288 0 31
180 288 1 28
180 288 2 31
180 288 3 30
180 288 4 31
180 288 5 30
180 288 6 31
180 288 7 31
180 288 8 30
180 288 9 31
180 288 10 30
180 288 11 31
180 432 0 31
180 432 1 28
180 432 2 31
180 432 3 30
180 432 4 31
180 432 5 30
180 432 6 31
180 432 7 31
180 432 8 30
180 432 9 31
180 432 10 30
180 432 11 31
180 576 0 31
180 576 1 28
180 576 2 31
180 576 3 30
180 576 4 31
180 576 5 30
180 576 6 31
180 576 7 31
180 576 8 30
180 576 9 31
180 576 10 30
180 576 11 31
180 719 0 31
180 719 1 28
180 719 2 31
180 719 3 30
180 719 4 31
180 719 5 30
180 719 6 31
180 719 7 31
180 719 8 30
180 719 9 31
180 719 10 30

```
180 719 11 31
```

```
import pylab as plt
plt.plot(lon[0],lons,all_pd)
plt.xlim(-180.,180.)
plt.xlabel('latitude')
plt.ylabel('power density total per month')
```

```
<matplotlib.text.Text at 0x108fc52d0>
```



So, with this in mind, let's just take a single value per latitude as otherwise the processing cost will be rather high.

Putting this into a function (in this case, reading the albedo, lat and long internally):

```
def absorbed_power_density(year,minute_step=60):
    '''Function to calculate the monthly total absorbed solar radiation
    power density in MJ / m^2 for a given year
    given an input dataset albedo and associated lat and lon
    information (in degrees).

    The shape of the albedo dataset is: (12,nlat,nlon)

    Aguments:
        year : integer of the year

    Options:
        minute_step : integer: resolution of steps in minutes. Must be
        a divisor of 60 (e.g. 10, 15, 30, 60)

    '''

    data = masked(dataset=['lat','lon','BHR_SW'],year=year)

    lat = data['lat'][0]
    lon = data['lon'][0]
    albedo = data['BHR_SW']
```

```

s = albedo.shape
lat = ma.array([lat] * s[2]).T
lon = ma.array([lon] * s[1])

# array for output
absorbed = ma.array(np.zeros_like(albedo), mask=albedo.mask)
power_density = ma.array(np.zeros_like(albedo), mask=albedo.mask)
#

for i in xrange(0, lat.shape[0]):
    print 'lat', lat[i,0]
    # single longitude
    for j in [lat.shape[1]/2]:
        pd = []
        # loop over month
        for month in xrange(12):
            pd_month = []
            ndays = days_in_month(month+1, year=year)
            # loop over days
            for day in xrange(ndays):
                # solar radiation for that day
                rad = radiation(year, month+1, day+1, \
                                lat[i,j], lon[i,j], minute_step=minute_step)
                # get rid of -ves
                rad[rad<0] = 0
                pd_month.append([rad[2].sum() * 60 * minute_step])
            pd_month = np.array(pd_month).T
            pd.append([pd_month.sum()])
        pd = np.array(pd).squeeze().T

        # pd is the power density for each month
        # load this into all longitudes
        for k in xrange(power_density.shape[0]):
            power_density[k,i,:] = pd[k]
power_density /= 10.*6 # MJ per m2
absorbed = power_density * (1 - albedo)
return absorbed, power_density

# this will still take some minutes to calculate
absorbed, power_density = absorbed_power_density(year, minute_step=60)

lat 89.6957
lat 89.196
lat 88.6963
lat 88.1966
lat 87.6969
lat 87.1972
lat 86.6975
lat 86.1978
lat 85.6981
lat 85.1984
lat 84.6987
lat 84.1991
lat 83.6994
lat 83.1997
lat 82.7
lat 82.2003
lat 81.7006
lat 81.2009
lat 80.7012
lat 80.2015
lat 79.7018
lat 79.2021

```

```
lat 78.7024
lat 78.2027
lat 77.703
lat 77.2033
lat 76.7036
lat 76.2039
lat 75.7042
lat 75.2045
lat 74.7048
lat 74.2051
lat 73.7054
lat 73.2057
lat 72.706
lat 72.2063
lat 71.7066
lat 71.2069
lat 70.7072
lat 70.2075
lat 69.7078
lat 69.2081
lat 68.7084
lat 68.2087
lat 67.709
lat 67.2093
lat 66.7096
lat 66.2099
lat 65.7102
lat 65.2105
lat 64.7108
lat 64.2111
lat 63.7114
lat 63.2118
lat 62.7121
lat 62.2124
lat 61.7127
lat 61.213
lat 60.7133
lat 60.2136
lat 59.7139
lat 59.2142
lat 58.7145
lat 58.2148
lat 57.7151
lat 57.2154
lat 56.7157
lat 56.216
lat 55.7163
lat 55.2166
lat 54.7169
lat 54.2172
lat 53.7175
lat 53.2178
lat 52.7181
lat 52.2184
lat 51.7187
lat 51.219
lat 50.7193
lat 50.2196
lat 49.7199
lat 49.2202
lat 48.7205
lat 48.2208
lat 47.7211
```

lat 47.2214
lat 46.7217
lat 46.222
lat 45.7223
lat 45.2226
lat 44.7229
lat 44.2232
lat 43.7235
lat 43.2238
lat 42.7241
lat 42.2245
lat 41.7248
lat 41.2251
lat 40.7254
lat 40.2257
lat 39.726
lat 39.2263
lat 38.7266
lat 38.2269
lat 37.7272
lat 37.2275
lat 36.7278
lat 36.2281
lat 35.7284
lat 35.2287
lat 34.729
lat 34.2293
lat 33.7296
lat 33.2299
lat 32.7302
lat 32.2305
lat 31.7308
lat 31.2311
lat 30.7314
lat 30.2317
lat 29.732
lat 29.2323
lat 28.7326
lat 28.2329
lat 27.7332
lat 27.2335
lat 26.7338
lat 26.2341
lat 25.7344
lat 25.2347
lat 24.735
lat 24.2353
lat 23.7356
lat 23.2359
lat 22.7362
lat 22.2365
lat 21.7369
lat 21.2372
lat 20.7375
lat 20.2378
lat 19.7381
lat 19.2384
lat 18.7387
lat 18.239
lat 17.7393
lat 17.2396
lat 16.7399
lat 16.2402

```
lat 15.7405
lat 15.2408
lat 14.7411
lat 14.2414
lat 13.7417
lat 13.242
lat 12.7423
lat 12.2426
lat 11.7429
lat 11.2432
lat 10.7435
lat 10.2438
lat 9.74411
lat 9.24441
lat 8.74471
lat 8.24501
lat 7.74532
lat 7.24562
lat 6.74592
lat 6.24622
lat 5.74653
lat 5.24683
lat 4.74713
lat 4.24743
lat 3.74774
lat 3.24804
lat 2.74834
lat 2.24864
lat 1.74895
lat 1.24925
lat 0.74955
lat 0.249853
lat -0.249845
lat -0.749542
lat -1.24924
lat -1.74894
lat -2.24864
lat -2.74833
lat -3.24803
lat -3.74773
lat -4.24743
lat -4.74712
lat -5.24682
lat -5.74652
lat -6.24622
lat -6.74591
lat -7.24561
lat -7.74531
lat -8.24501
lat -8.7447
lat -9.2444
lat -9.7441
lat -10.2438
lat -10.7435
lat -11.2432
lat -11.7429
lat -12.2426
lat -12.7423
lat -13.242
lat -13.7417
lat -14.2414
lat -14.7411
lat -15.2408
```

lat -15.7405
lat -16.2402
lat -16.7399
lat -17.2396
lat -17.7393
lat -18.239
lat -18.7387
lat -19.2384
lat -19.7381
lat -20.2377
lat -20.7374
lat -21.2371
lat -21.7368
lat -22.2365
lat -22.7362
lat -23.2359
lat -23.7356
lat -24.2353
lat -24.735
lat -25.2347
lat -25.7344
lat -26.2341
lat -26.7338
lat -27.2335
lat -27.7332
lat -28.2329
lat -28.7326
lat -29.2323
lat -29.732
lat -30.2317
lat -30.7314
lat -31.2311
lat -31.7308
lat -32.2305
lat -32.7302
lat -33.2299
lat -33.7296
lat -34.2293
lat -34.729
lat -35.2287
lat -35.7284
lat -36.2281
lat -36.7278
lat -37.2275
lat -37.7272
lat -38.2269
lat -38.7266
lat -39.2263
lat -39.726
lat -40.2257
lat -40.7253
lat -41.225
lat -41.7247
lat -42.2244
lat -42.7241
lat -43.2238
lat -43.7235
lat -44.2232
lat -44.7229
lat -45.2226
lat -45.7223
lat -46.222
lat -46.7217

```
lat -47.2214
lat -47.7211
lat -48.2208
lat -48.7205
lat -49.2202
lat -49.7199
lat -50.2196
lat -50.7193
lat -51.219
lat -51.7187
lat -52.2184
lat -52.7181
lat -53.2178
lat -53.7175
lat -54.2172
lat -54.7169
lat -55.2166
lat -55.7163
lat -56.216
lat -56.7157
lat -57.2154
lat -57.7151
lat -58.2148
lat -58.7145
lat -59.2142
lat -59.7139
lat -60.2136
lat -60.7133
lat -61.213
lat -61.7127
lat -62.2123
lat -62.712
lat -63.2117
lat -63.7114
lat -64.2111
lat -64.7108
lat -65.2105
lat -65.7102
lat -66.2099
lat -66.7096
lat -67.2093
lat -67.709
lat -68.2087
lat -68.7084
lat -69.2081
lat -69.7078
lat -70.2075
lat -70.7072
lat -71.2069
lat -71.7066
lat -72.2063
lat -72.706
lat -73.2057
lat -73.7054
lat -74.2051
lat -74.7048
lat -75.2045
lat -75.7042
lat -76.2039
lat -76.7036
lat -77.2033
lat -77.703
lat -78.2027
```

```
lat -78.7024
lat -79.2021
lat -79.7018
lat -80.2015
lat -80.7012
lat -81.2009
lat -81.7006
lat -82.2002
lat -82.7
lat -83.1996
lat -83.6993
lat -84.199
lat -84.6987
lat -85.1984
lat -85.6981
lat -86.1978
lat -86.6975
lat -87.1972
lat -87.6969
lat -88.1966
lat -88.6963
lat -89.196
lat -89.6957

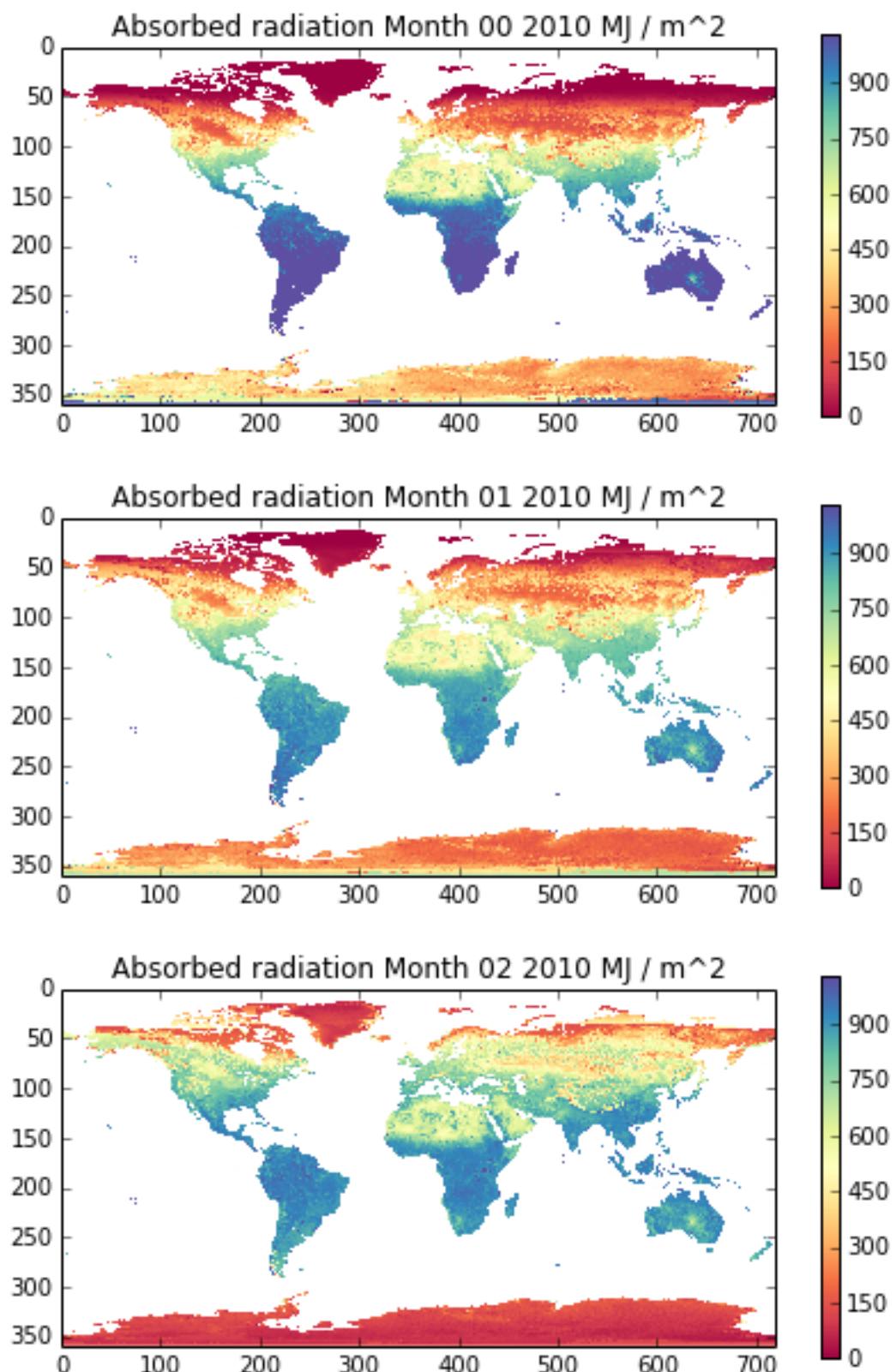
# this is how you can save a numpy array
np.savez('files/data/absorbed.npz',absorbed=np.array(absorbed), \
         power_density=np.array(power_density),mask=absorbed.mask)

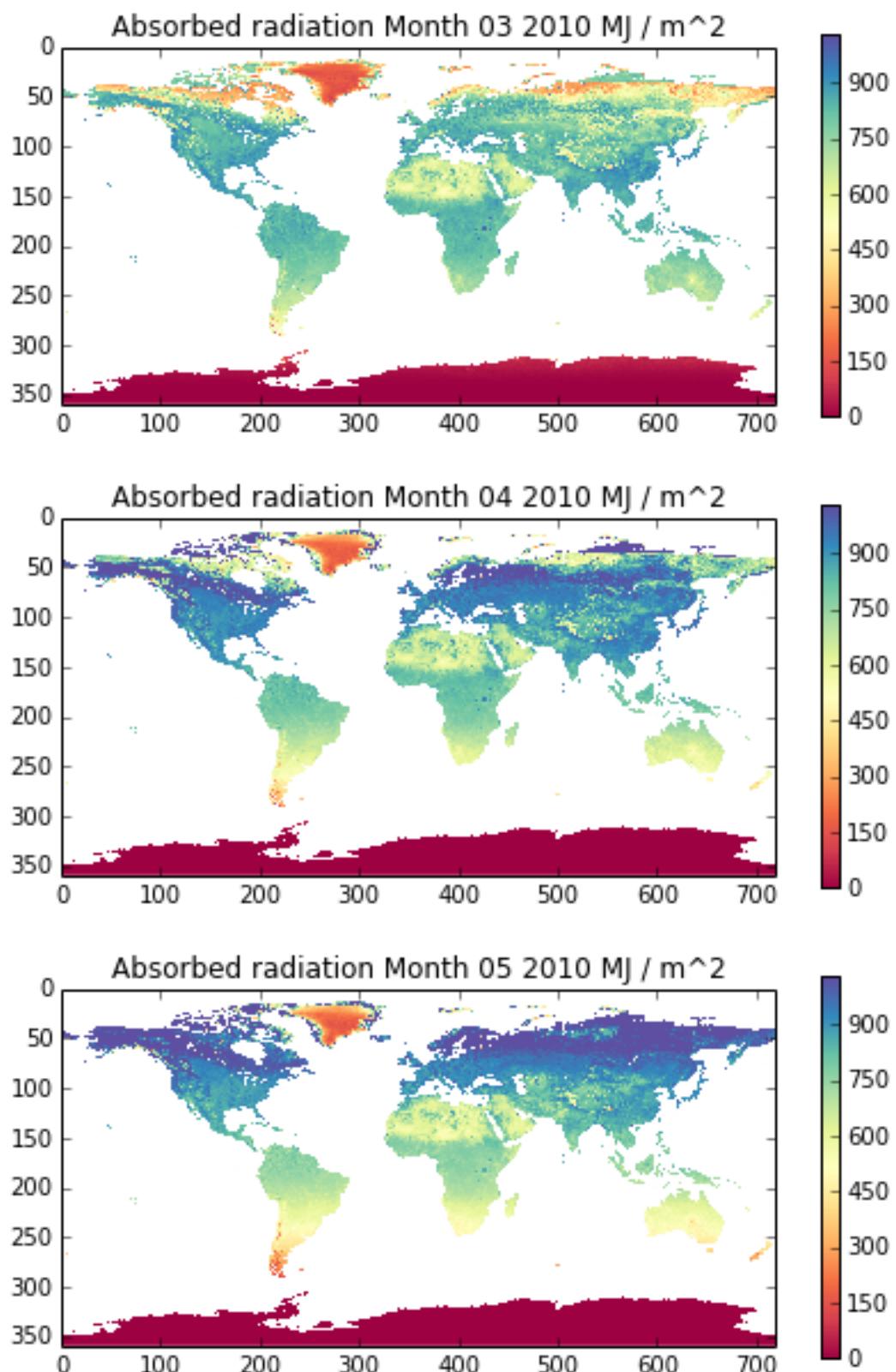
# now next time we should be able to load it
try:
    f = np.load('files/data/absorbed.npz')
    mask = f['mask']
    absorbed,power_density = ma.array(f['absorbed'],mask=mask), \
                             ma.array(f['power_density'],mask=mask)
except:
    absorbed,power_density = absorbed_power_density(year,minute_step=60)

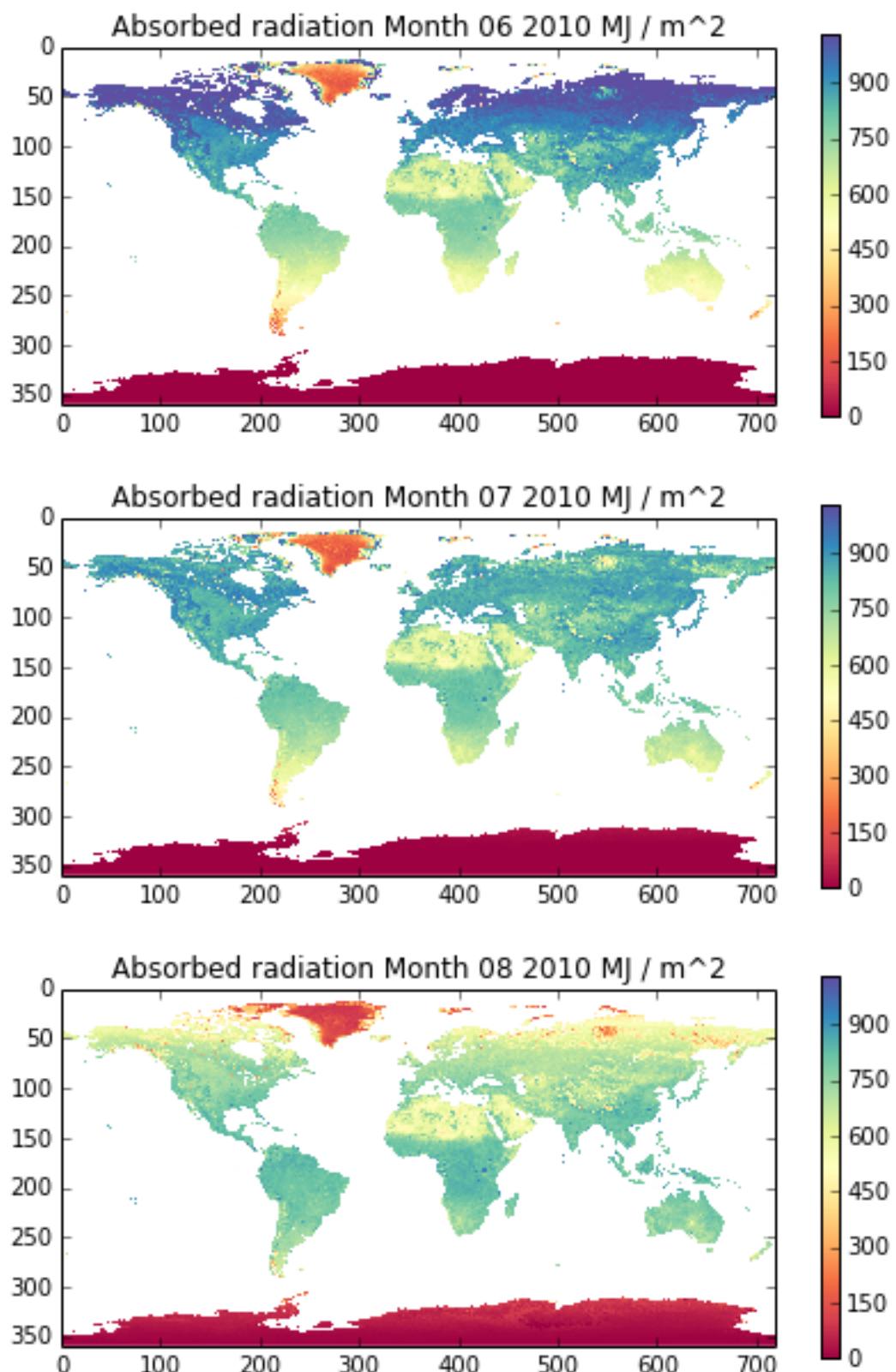
import pylab as plt

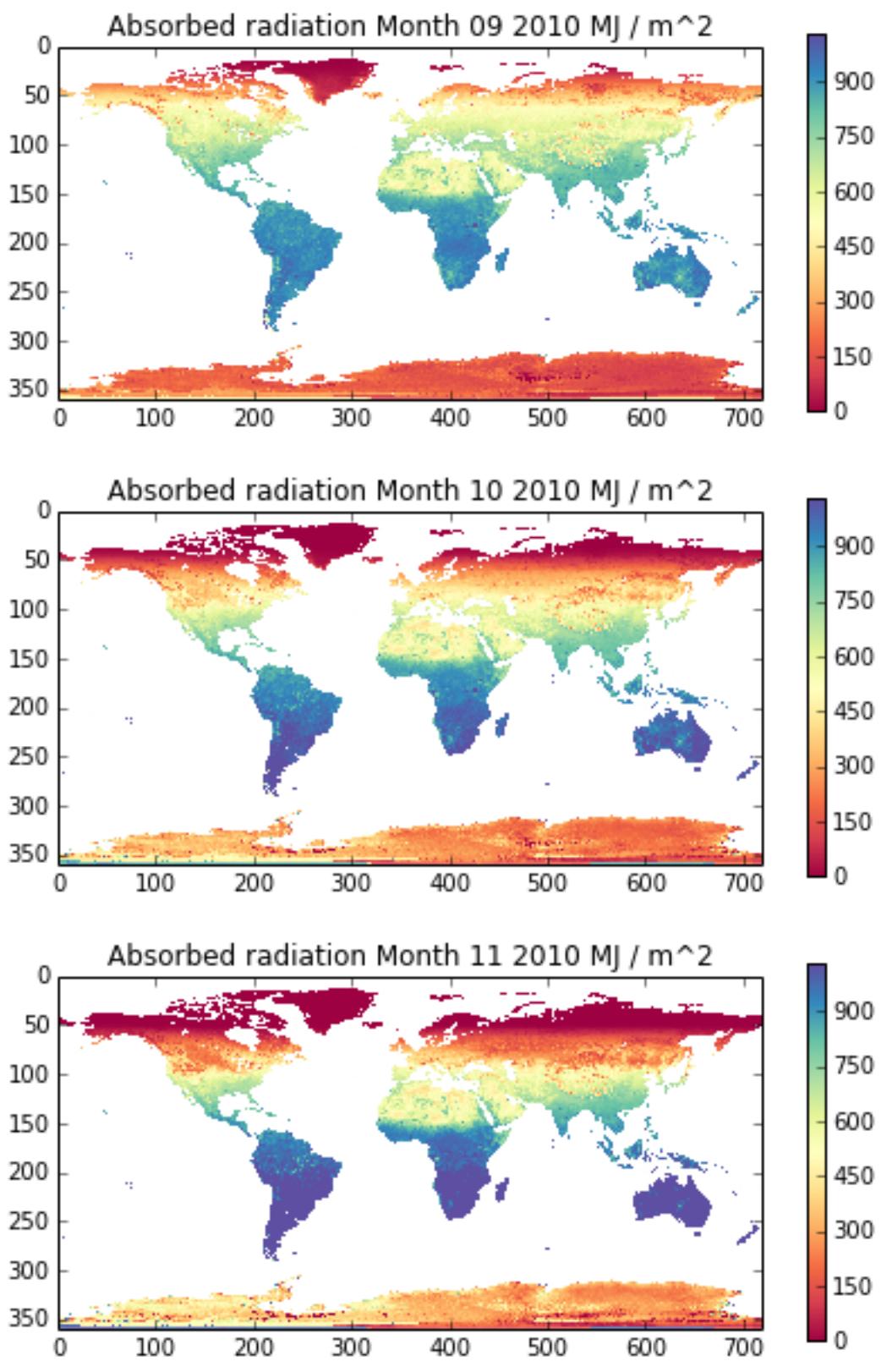
cmap = plt.get_cmap('Spectral')
vmax = absorbed.max()/2
vmin = 0.

for m in xrange(12):
    plt.figure(figsize=(7,3))
    plt.title('Absorbed radiation Month %02d %d MJ / m^2' %(m,year))
    plt.imshow(absorbed[m],cmap=cmap,interpolation='none',vmin=vmin,vmax=vmax)
    plt.colorbar()
    plt.savefig('files/data/absorbed%02d.jpg'%m)
```







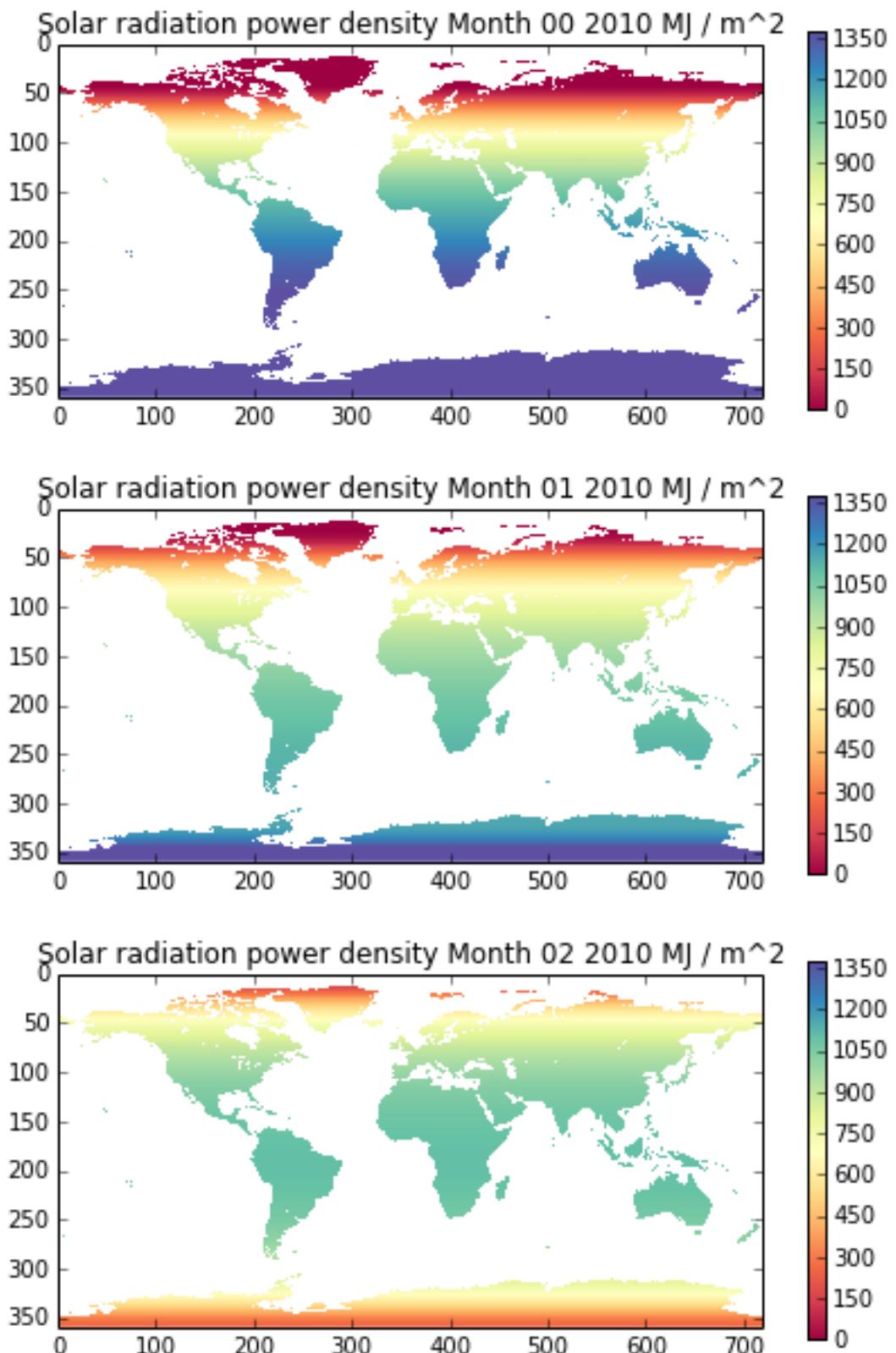


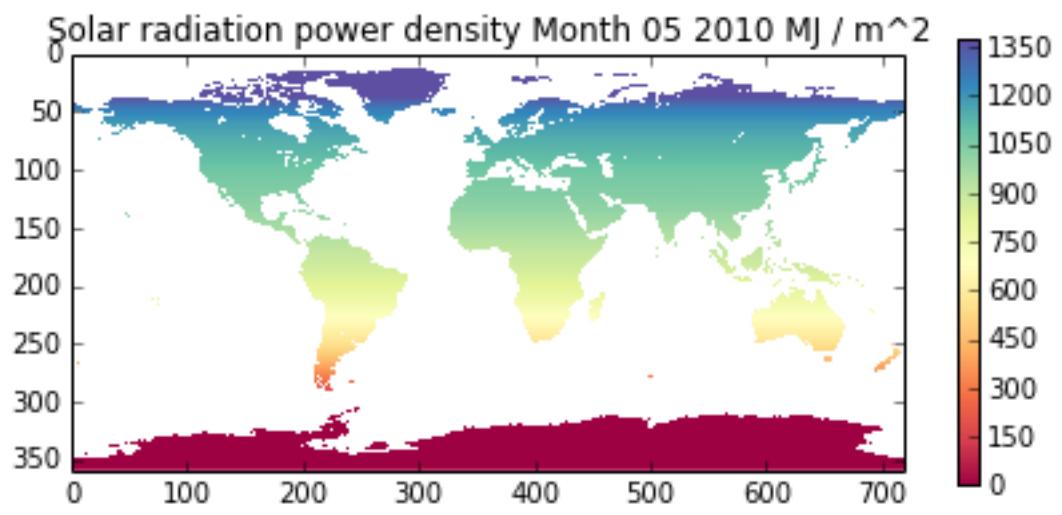
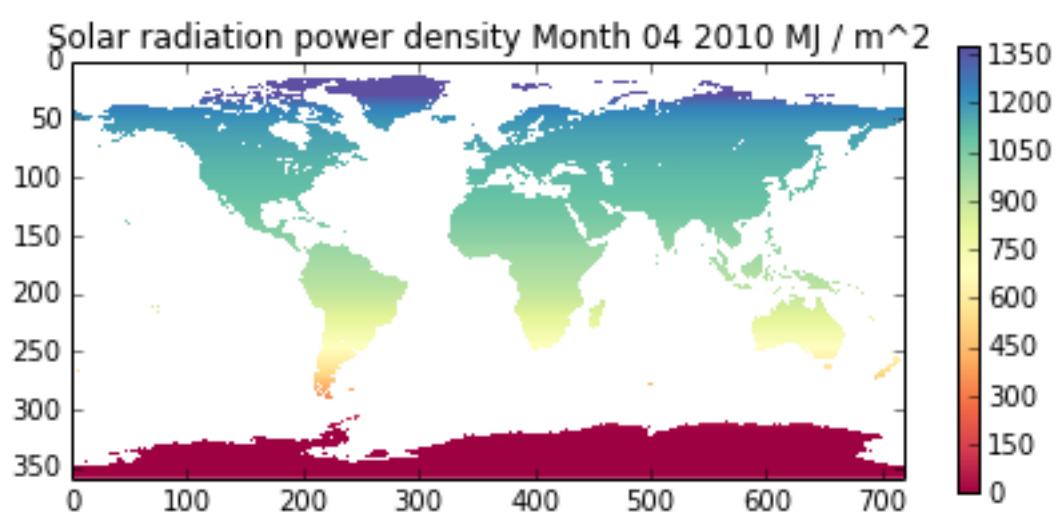
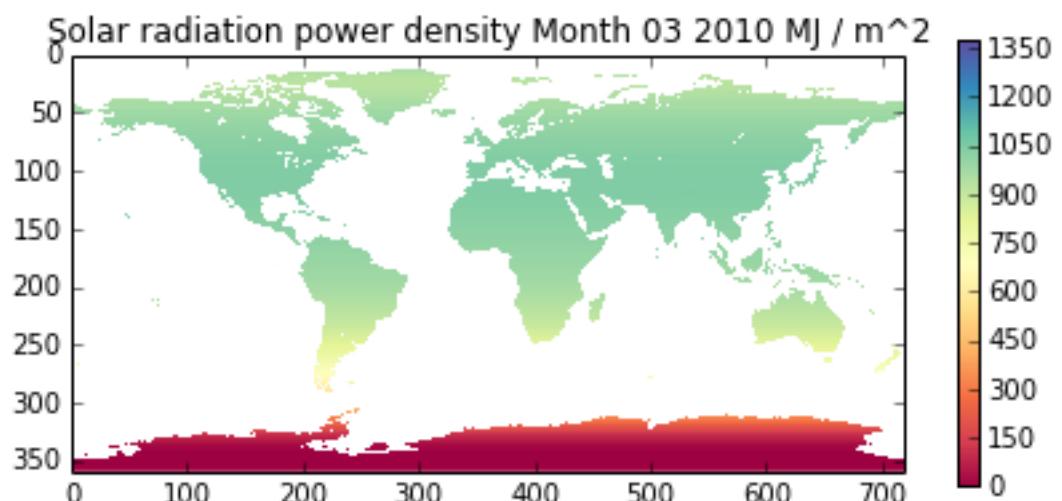
```
import pylab as plt

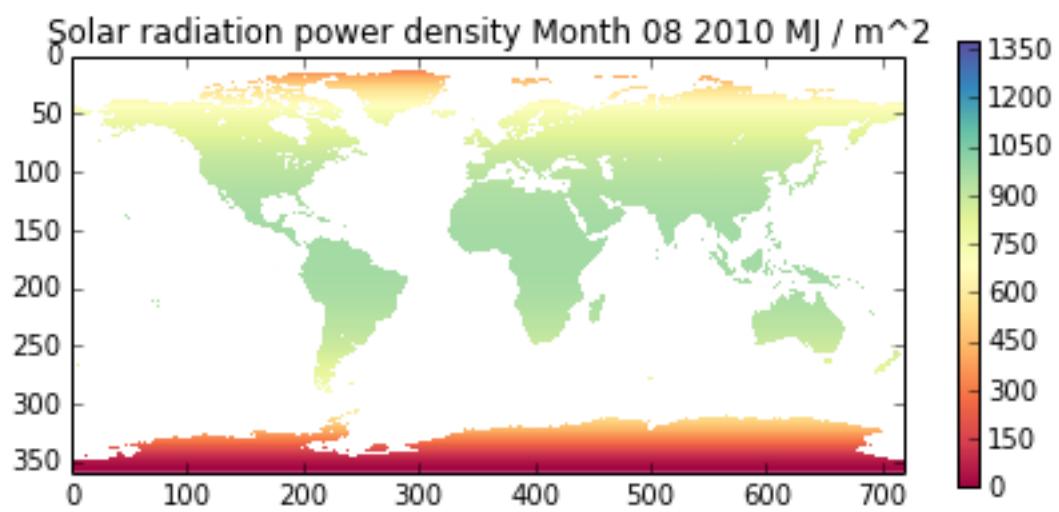
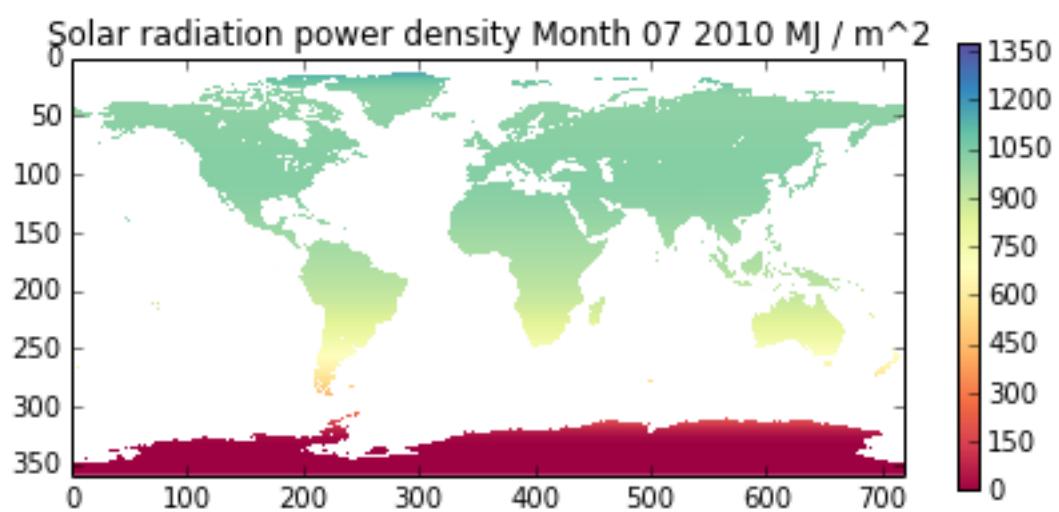
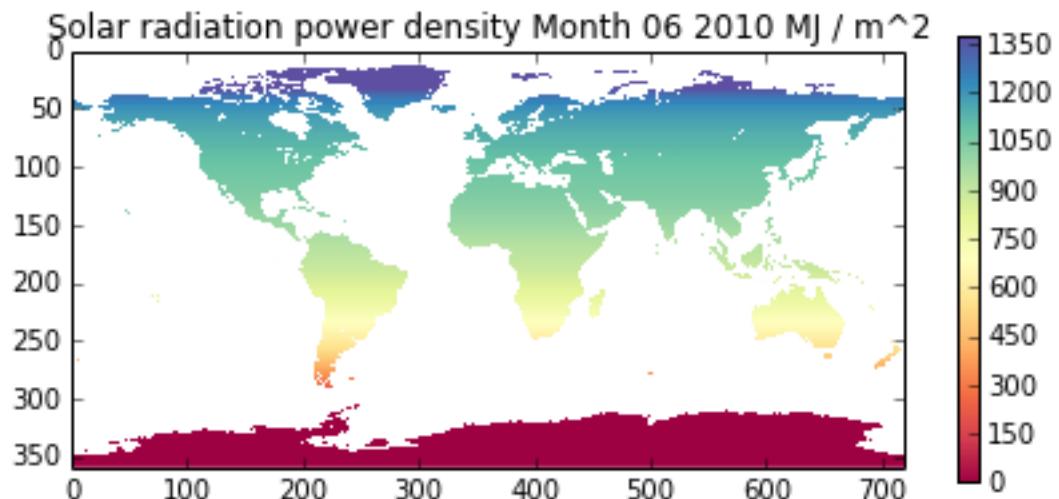
cmap = plt.get_cmap('Spectral')
vmax = absorbed.max() * 2./3.
vmin = 0.

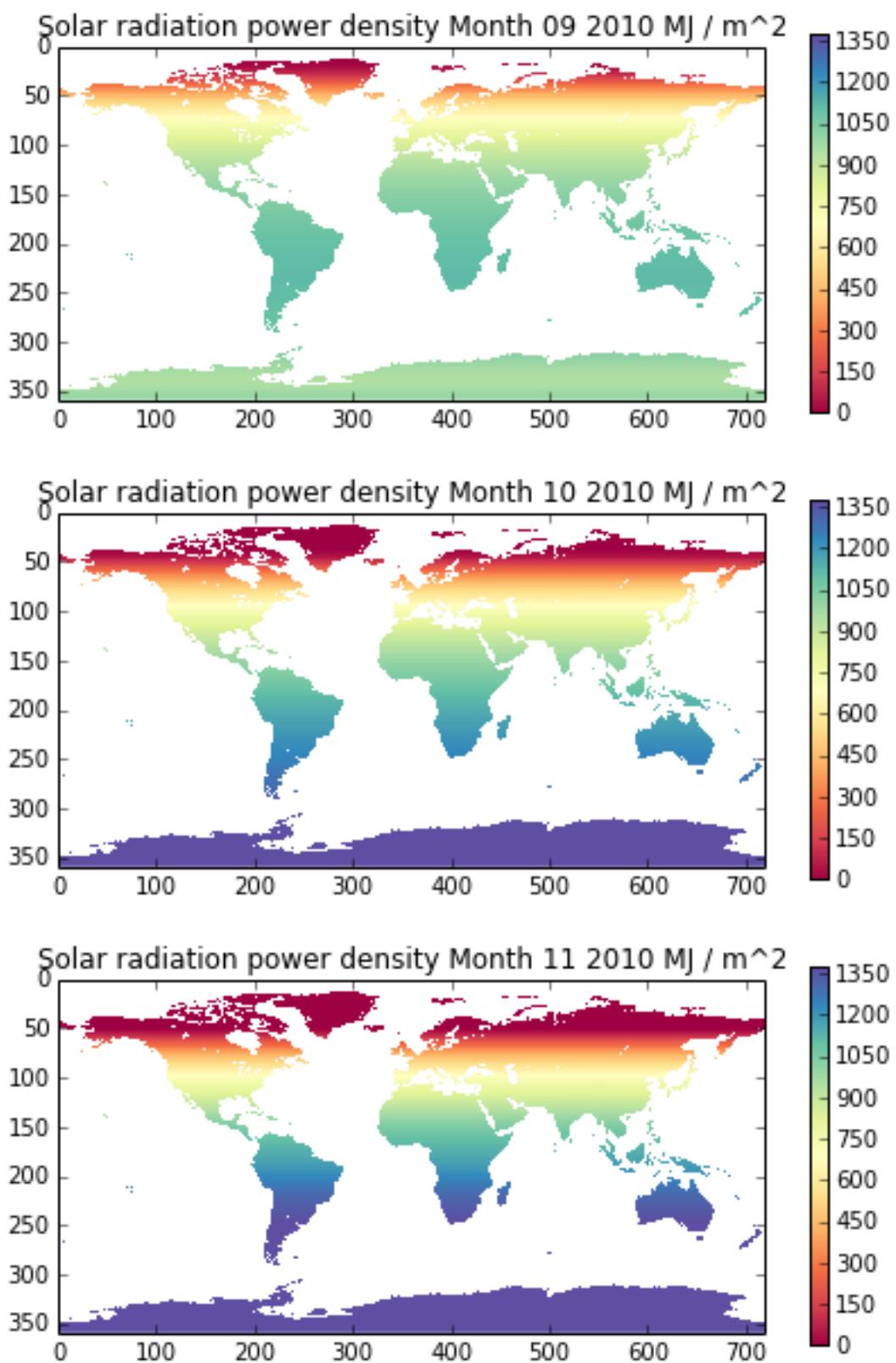
for m in xrange(12):
```

```
plt.figure(figsize=(7,3))
plt.title('Solar radiation power density Month %02d %d MJ / m^2' %(m,year))
plt.imshow(power_density[m],cmap=cmap,interpolation='none',vmin=vmin,vmax=vmax)
plt.colorbar()
plt.savefig('files/data/power%02d.jpg'%m)
```









```
# make some movies!???
import os

for t in ['power', 'absorbed']:
    for m in xrange(12):
        cmd = 'convert files/data/%s%02d.jpg files/data/%s%02d.gif'%(t,m,t,m)
        print cmd
```

```
os.system(cmd)

cmd = "convert -delay 100 -loop 0 \
       files/data/%s???.gif files/data/%s_movie.gif"%(t,t)
os.system(cmd)

convert files/data/power00.jpg files/data/power00.gif
convert files/data/power01.jpg files/data/power01.gif
convert files/data/power02.jpg files/data/power02.gif
convert files/data/power03.jpg files/data/power03.gif
convert files/data/power04.jpg files/data/power04.gif
convert files/data/power05.jpg files/data/power05.gif
convert files/data/power06.jpg files/data/power06.gif
convert files/data/power07.jpg files/data/power07.gif
convert files/data/power08.jpg files/data/power08.gif
convert files/data/power09.jpg files/data/power09.gif
convert files/data/power10.jpg files/data/power10.gif
convert files/data/power11.jpg files/data/power11.gif
convert files/data/absorbed00.jpg files/data/absorbed00.gif
convert files/data/absorbed01.jpg files/data/absorbed01.gif
convert files/data/absorbed02.jpg files/data/absorbed02.gif
convert files/data/absorbed03.jpg files/data/absorbed03.gif
convert files/data/absorbed04.jpg files/data/absorbed04.gif
convert files/data/absorbed05.jpg files/data/absorbed05.gif
convert files/data/absorbed06.jpg files/data/absorbed06.gif
convert files/data/absorbed07.jpg files/data/absorbed07.gif
convert files/data/absorbed08.jpg files/data/absorbed08.gif
convert files/data/absorbed09.jpg files/data/absorbed09.gif
convert files/data/absorbed10.jpg files/data/absorbed10.gif
convert files/data/absorbed11.jpg files/data/absorbed11.gif
```

This code is still quite slow (even for a single longitude).

You should look it over and see how you might make it more efficient.

Some ideas:

- interpolate the radiation field (over latitude?) and possibly improve by sub sampling over longitude and interpolating that as well
- when calculating daily integrals, don't start the calculations until the sun is above the horizon (and the same for sunset).

The exercise asks you to perform summations over latitude. This should be simple enough, being of the form:

```
absorbed.sum(axis=(0,2))
```

which will sum over axes 0 (month) and 2 (longitude).

Total absorbed involves working out the area for each sample pixel and including that in the calculation. That is left as an exercise here.

E3 ANSWERS TO EXERCISE

30.1 E3.2 Exercise: listing

Using Python, produce a listing of the files in the subdirectory “data“ of “geogg122/Chapter3_Scientific_Numerical_Python“ that end with “.nc“ and put this listing in a file called “files/data/data.dat“ with each entry on a different line

30.2 A3.2 Answer: listing

Hopefully, you should already be in the directory geogg122/Chapter3_Scientific_Numerical_Python, if not, you may like to go there before starting this exercise.

If you were to do this from unix, you would get the listing with:

```
!ls -l data/*.nc
```

```
-rw-r--r-- 1 plewis staff 115202868 11 Oct 09:53 data/GlobAlbedo.1998145.h17v03.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:07 data/GlobAlbedo.200901.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:07 data/GlobAlbedo.200902.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200903.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200904.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200905.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200906.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18758652 11 Oct 20:08 data/GlobAlbedo.200907.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200908.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200909.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200910.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200911.mosaic.5.nc
-rw-r--r-- 1 plewis staff 18669672 11 Oct 20:08 data/GlobAlbedo.200912.mosaic.5.nc
```

or similar.

To do this in Python, you should use glob, and the same ‘pattern’:

```
import glob

files = glob.glob('data/*.nc')
print files

['data/GlobAlbedo.1998145.h17v03.nc', 'data/GlobAlbedo.200901.mosaic.5.nc', 'data/GlobAlbedo.200902.mosaic.5.nc', 'data/GlobAlbedo.200903.mosaic.5.nc', 'data/GlobAlbedo.200904.mosaic.5.nc', 'data/GlobAlbedo.200905.mosaic.5.nc', 'data/GlobAlbedo.200906.mosaic.5.nc', 'data/GlobAlbedo.200907.mosaic.5.nc', 'data/GlobAlbedo.200908.mosaic.5.nc', 'data/GlobAlbedo.200909.mosaic.5.nc', 'data/GlobAlbedo.200910.mosaic.5.nc', 'data/GlobAlbedo.200911.mosaic.5.nc', 'data/GlobAlbedo.200912.mosaic.5.nc']
```

This is a list. We want to write this to a file called files/data/data.dat.

First we open it, then simply use writelines to write the list of strings, then close the file.

```
filename = 'files/data/data.dat'

# open in write mode
```

```
fp = open(filename, 'w')
fp.writelines(files)
fp.close()
```

Hopefully, that all worked well, but just to check from unix:

```
!cat files/data/data.dat
```

```
data/GlobAlbedo.1998145.h17v03.ncdata/GlobAlbedo.200901.mosaic.5.ncdata/GlobAlbedo.200902.mosaic.5.nc
```

which isn't quite what we wanted: we need to insert a newline character at the end of each string before writing.

There are several ways to do this, e.g.:

```
files = glob.glob('data/*.nc')

for i,file in enumerate(files):
    files[i] = file + '\n'
print files

['data/GlobAlbedo.1998145.h17v03.ncn', `data/GlobAlbedo.200901.mosaic.5.ncn`, `data/GlobAlbedo.200902.mosaic.5.ncn']

files = glob.glob('data/*.nc')

# or:
files = [file + '\n' for file in files]

print files

['data/GlobAlbedo.1998145.h17v03.ncn', `data/GlobAlbedo.200901.mosaic.5.ncn`, `data/GlobAlbedo.200902.mosaic.5.ncn']

# or all at once if you like:

files = [file + '\n' for file in glob.glob('data/*.nc')]

print files

['data/GlobAlbedo.1998145.h17v03.ncn', `data/GlobAlbedo.200901.mosaic.5.ncn`, `data/GlobAlbedo.200902.mosaic.5.ncn']
```

or several other ways ...

Putting this together:

```
import glob

files = [file + '\n' for file in glob.glob('data/*.nc')]

filename = 'files/data/data.dat'

# open in write mode
fp = open(filename, 'w')
fp.writelines(files)
fp.close()
```

then checking:

```
!cat files/data/data.dat
```

```
data/GlobAlbedo.1998145.h17v03.nc
data/GlobAlbedo.200901.mosaic.5.nc
data/GlobAlbedo.200902.mosaic.5.nc
data/GlobAlbedo.200903.mosaic.5.nc
data/GlobAlbedo.200904.mosaic.5.nc
data/GlobAlbedo.200905.mosaic.5.nc
data/GlobAlbedo.200906.mosaic.5.nc
data/GlobAlbedo.200907.mosaic.5.nc
```

```
data/GlobAlbedo.200908.mosaic.5.nc
data/GlobAlbedo.200909.mosaic.5.nc
data/GlobAlbedo.200910.mosaic.5.nc
data/GlobAlbedo.200911.mosaic.5.nc
data/GlobAlbedo.200912.mosaic.5.nc
```

which *is* what we wanted.

30.3 E3.2 Exercise: Making Movies

30.3.1 E3.2.1 Software

You can *sort of* make movies in pylab, but you generally have to make a system call to unix at some point, so it's probably easier to do this all in unix with the utility 'convert <<http://www.imagemagick.org/script/convert.php>>'__.

At the unix prompt, check that you have access to convert:

```
berlin% which convert
/usr/bin/convert
```

If this doesn't come up with anything useful, there is probably a version in /usr/bin/convert or /usr/local/bin/convert (If you don't have it on your local machine, install 'ImageMagick <<http://www.imagemagick.org/script/index.php>>'__ which contains the command line tool convert).

To use this, e.g.:

from the unix command line:

```
berlin% cd ~/Data/geogg122/Chapter3_Scientific_Numerical_Python
berlin% convert files/data/albedo.jpg files/data/albedo.gif
```

or from within a notebook:

```
!convert files/data/albedo.jpg files/data/albedo.gif
```

Or, more practically here, you can run a unix command directly from Python:

```
import os
cmd = 'convert files/data/albedo.jpg files/data/albedo.gif'
os.system(cmd)

0
```

This will convert the file files/data/albedo.jpg (in jpeg format) to files/data/albedo.gif (in gif format).

Figure 30.1: albedo

We can also use convert to make animated gifs, which is one way of making a movie.

30.3.2 E3.2.2 Looping over a set of images

You have all of the code you need above to be able to read a GlobAlbedo file for a given month and waveband in Python and save a picture in jpeg format, but to recap for BHR_VIS:

```
from netCDF4 import Dataset
import pylab as plt
import os
```

```
root = 'files/data/'

month_list = ['January', 'February', 'March', 'April', 'May', 'June', \
              'July', 'August', 'September', 'October', 'November', 'December']
# make a dictionary from 2 lists
month_dict = dict(zip(range(1,13),month_list))

# example filename : use formatting string:
# %d%02d
year = 2009

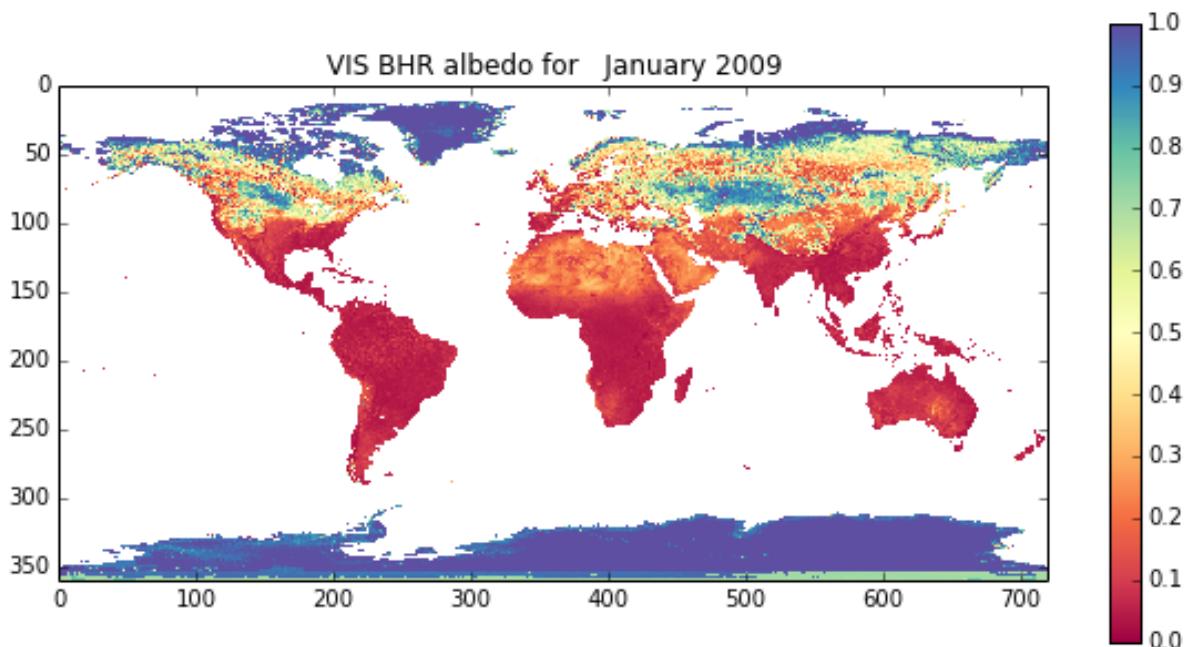
# set the month
month = 1

''' Read the data '''
local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)
# load the netCDF data from the file f.filename
nc = Dataset(local_file,'r')
band = nc.variables['DHR_VIS']

''' Plot the data and save as picture jpeg format '''
# make a string with the output file name
out_file = root + 'GlobAlbedo.%d%02d.jpg'%(year,month)
# plot
plt.figure(figsize=(10, 5))
plt.clf()
# %9s forces the string to be 9 characters long
plt.title('VIS BHR albedo for %s %d'%(month_dict[month],year))
# use nearest neighbour interpolation
plt.imshow(band,interpolation='nearest',cmap=plt.get_cmap('Spectral'),vmin=0.0,vmax=1.0)
# show a colour bar
plt.colorbar()
plt.savefig(out_file)

''' Convert the file to gif '''
# set up the unix command which is of the form
# convert input output
# Here input will be out_file
# and output we can get with out_file.replace('.jpg','.gif')
# i.e. replacing where it says .jpg with .gif
cmd = 'convert %s %s'%(out_file,out_file.replace('.jpg','.gif'))
os.system(cmd)
```

0



Modify the code above to loop over each month, so that it generates a set of gif format files for the TOTAL SHORTWAVE ALBEDO

You should confirm that these exist, and that the file modification time is when you ran it (not when I generated the files for these notes, which is Oct 10 2013).

```
ls -l files/data/GlobAlbedo*.gif
```

```
-rw-r--r-- 1 plewis staff 340658 11 Oct 20:00 files/data/GlobAlbedo.2009.SW.1.gif
-rw-r--r-- 1 plewis staff 340658 11 Oct 19:59 files/data/GlobAlbedo.2009.SW.gif
-rw-r--r-- 1 plewis staff 55299 11 Oct 20:21 files/data/GlobAlbedo.200901.gif
-rw-r--r-- 1 plewis staff 28139 11 Oct 19:59 files/data/GlobAlbedo.200902.gif
-rw-r--r-- 1 plewis staff 28259 11 Oct 19:59 files/data/GlobAlbedo.200903.gif
-rw-r--r-- 1 plewis staff 28249 11 Oct 19:59 files/data/GlobAlbedo.200904.gif
-rw-r--r-- 1 plewis staff 28468 11 Oct 19:59 files/data/GlobAlbedo.200905.gif
-rw-r--r-- 1 plewis staff 28672 11 Oct 19:59 files/data/GlobAlbedo.200906.gif
-rw-r--r-- 1 plewis staff 28656 11 Oct 19:59 files/data/GlobAlbedo.200907.gif
-rw-r--r-- 1 plewis staff 28275 11 Oct 19:59 files/data/GlobAlbedo.200908.gif
-rw-r--r-- 1 plewis staff 28952 11 Oct 19:59 files/data/GlobAlbedo.200909.gif
-rw-r--r-- 1 plewis staff 28450 11 Oct 19:59 files/data/GlobAlbedo.200910.gif
-rw-r--r-- 1 plewis staff 28570 11 Oct 19:59 files/data/GlobAlbedo.200911.gif
-rw-r--r-- 1 plewis staff 28438 11 Oct 19:59 files/data/GlobAlbedo.200912.gif
```

30.3.3 A3.2.2 Answer: Looping over a set of images

Really all you need to do here is to make `month` appear in a loop, e.g. using:

```
for month in range(1,13):
```

and then make sure that all of the code below is in that loop (i.e. indented) as below.

One additional thing is to make sure you select the waveband you were supposed to (shortwave albedo)

```
band = nc.variables['DHR_SW']
```

and finally, make sure you change the title:

You should *also* however, go through the code above line by line, making sure you appreciate what is going on at each stage and why we have done these things (in this order).

```
from netCDF4 import Dataset
import pylab as plt
import os

root = 'files/data/'

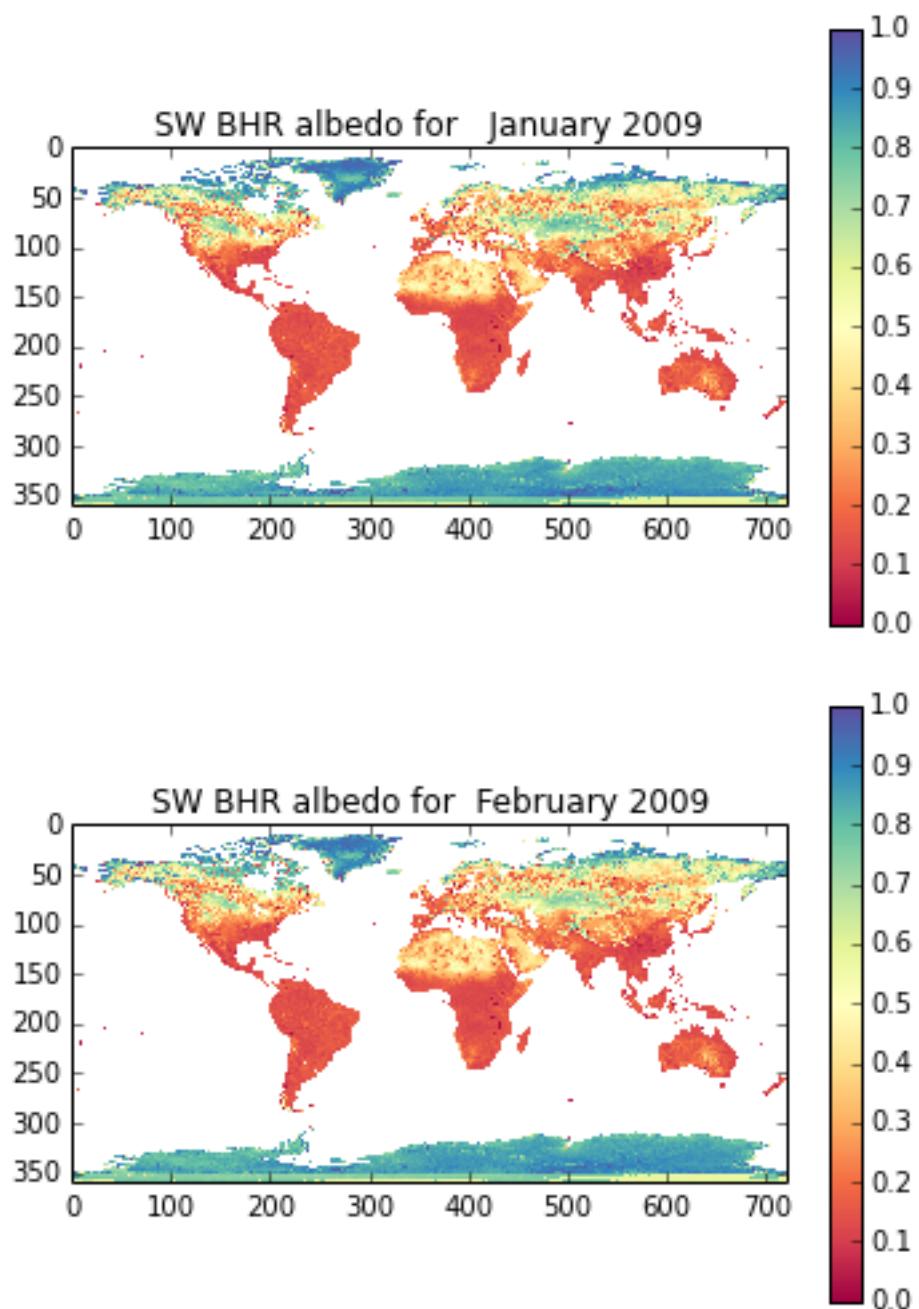
month_list = ['January','February','March','April','May','June',\
              'July','August','September','October','November','December']
# make a dictionary from 2 lists
month_dict = dict(zip(range(1,13),month_list))

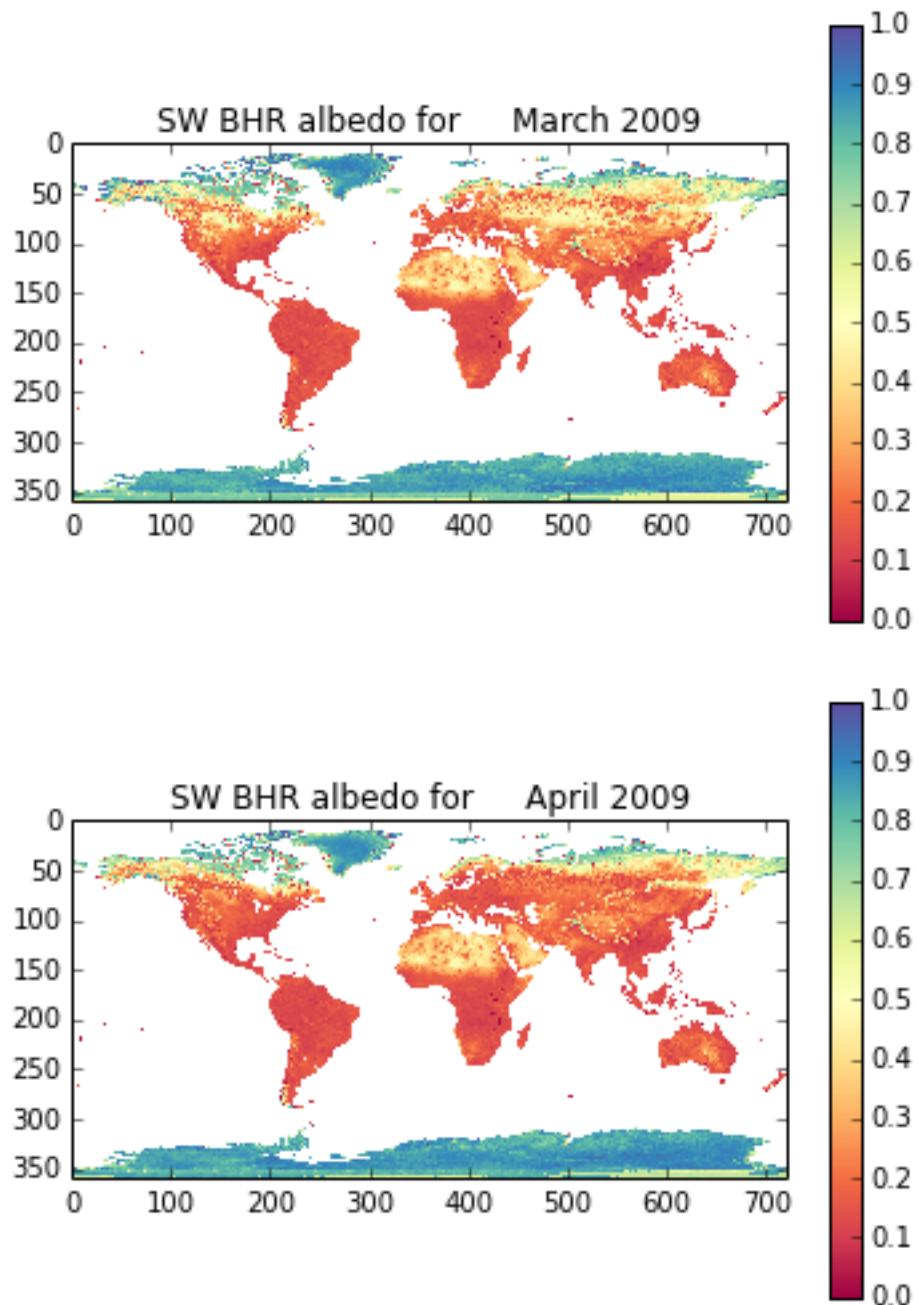
# example filename : use formatting string:
# %d%02d
year = 2009

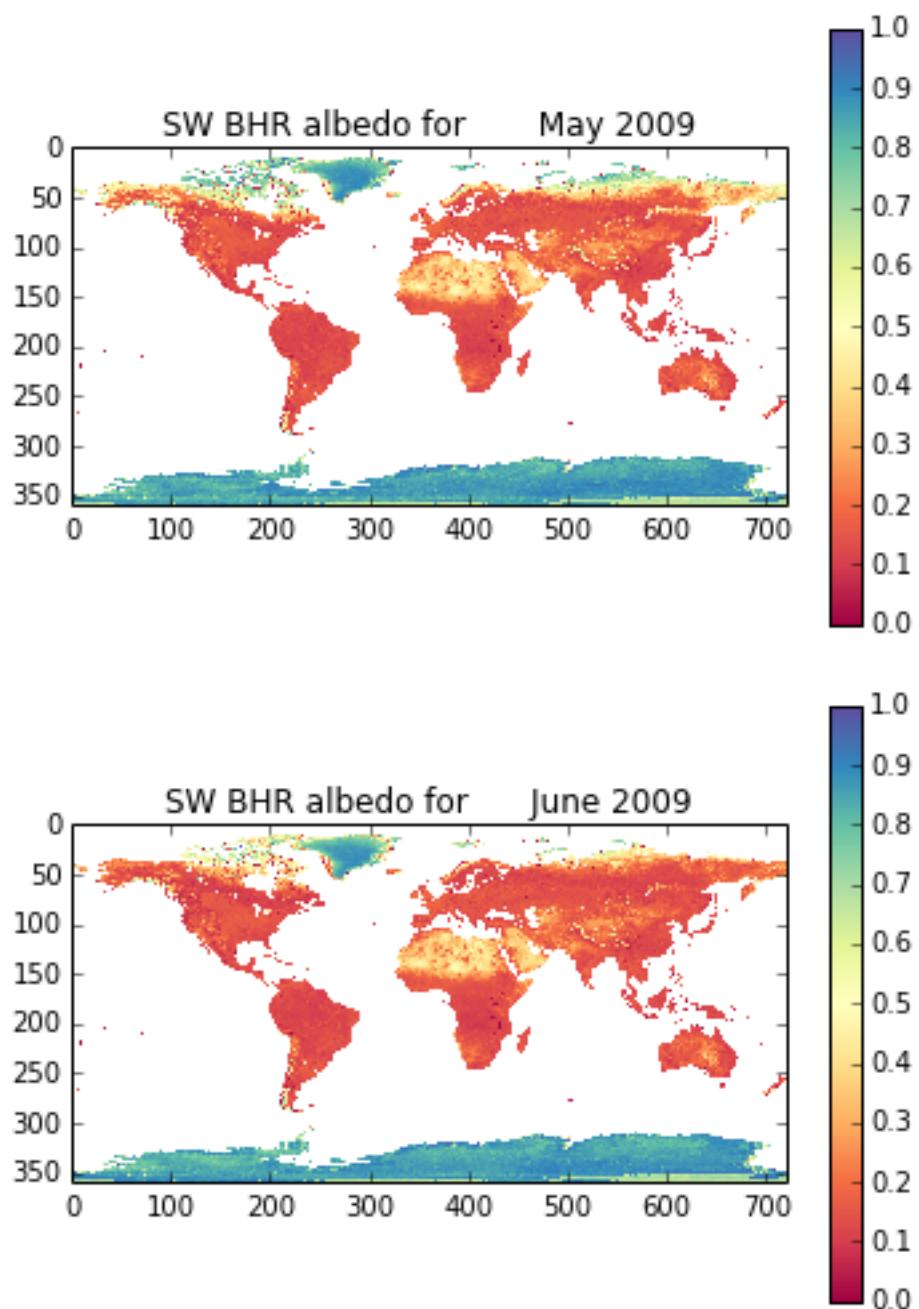
# set the month
for month in range(1,13):
    ''' Read the data '''
    local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)
    # load the netCDF data from the file f.filename
    nc = Dataset(local_file,'r')
    # select which band we want
    band = nc.variables['DHR_SW']

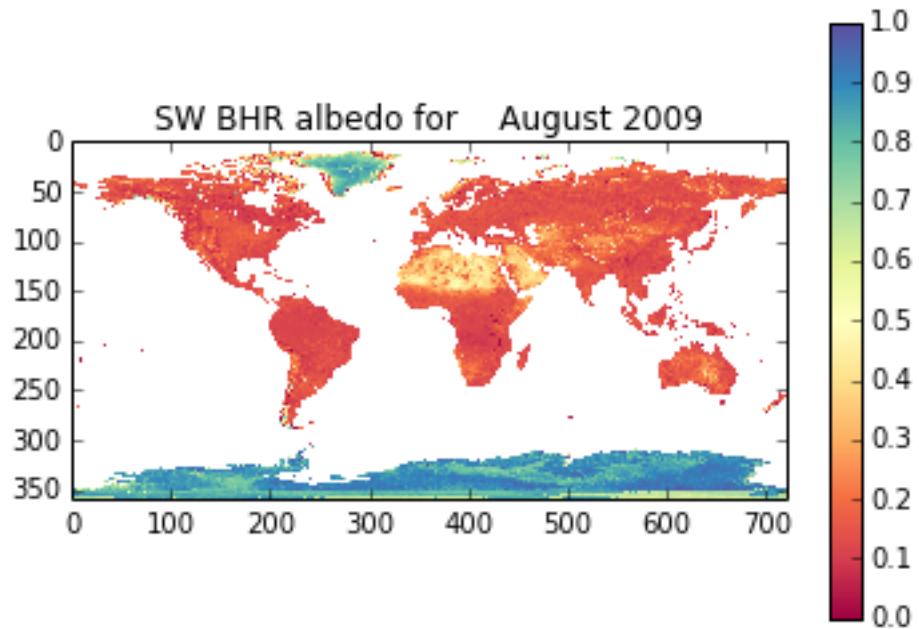
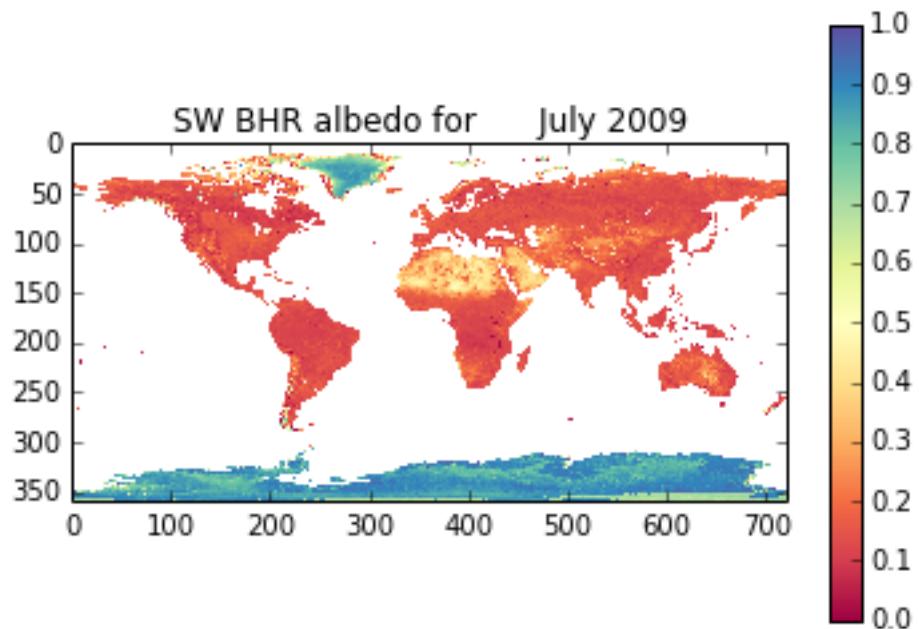
    ''' Plot the data and save as picture jpeg format '''
    # make a string with the output file name
    out_file = root + 'GlobAlbedo.%d%02d.jpg'%(year,month)
    # plot
    plt.figure()
    plt.clf()
    # %9s forces the string to be 9 characters long
    plt.title('SW BHR albedo for %9s %d'%(month_dict[month],year))
    # use nearest neighbour interpolation
    plt.imshow(band,interpolation='nearest',cmap=plt.get_cmap('Spectral'),vmin=0.0,vmax=1.0)
    # show a colour bar
    plt.colorbar()
    plt.savefig(out_file)

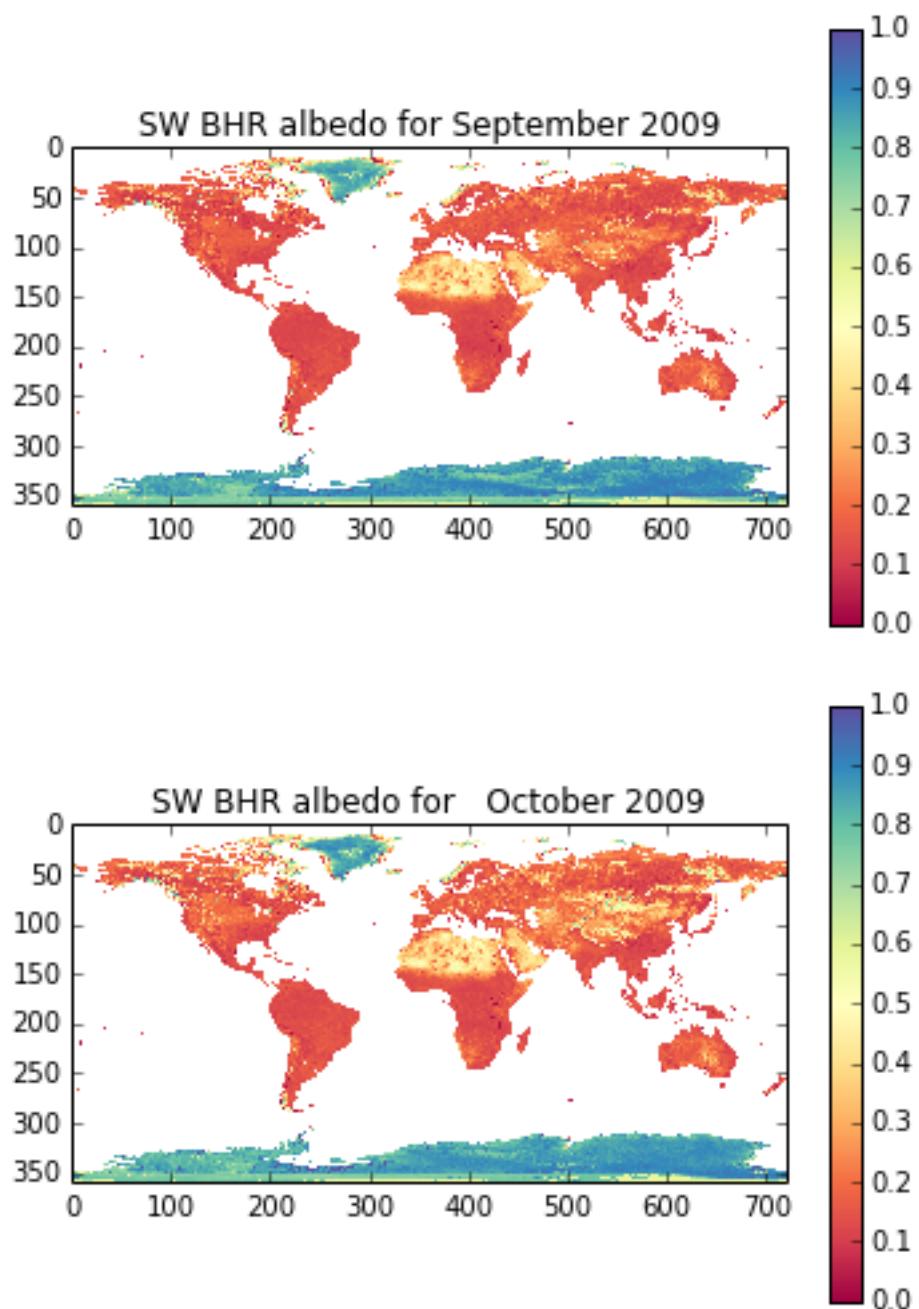
    ''' Convert the file to gif '''
    # set up the unix command which is of the form
    # convert input output
    # Here input will be out_file
    # and output we can get with out_file.replace('.jpg','.gif')
    # i.e. replacing where it says .jpg with .gif
    cmd = 'convert %s %s'%(out_file,out_file.replace('.jpg','.gif'))
    os.system(cmd)
```

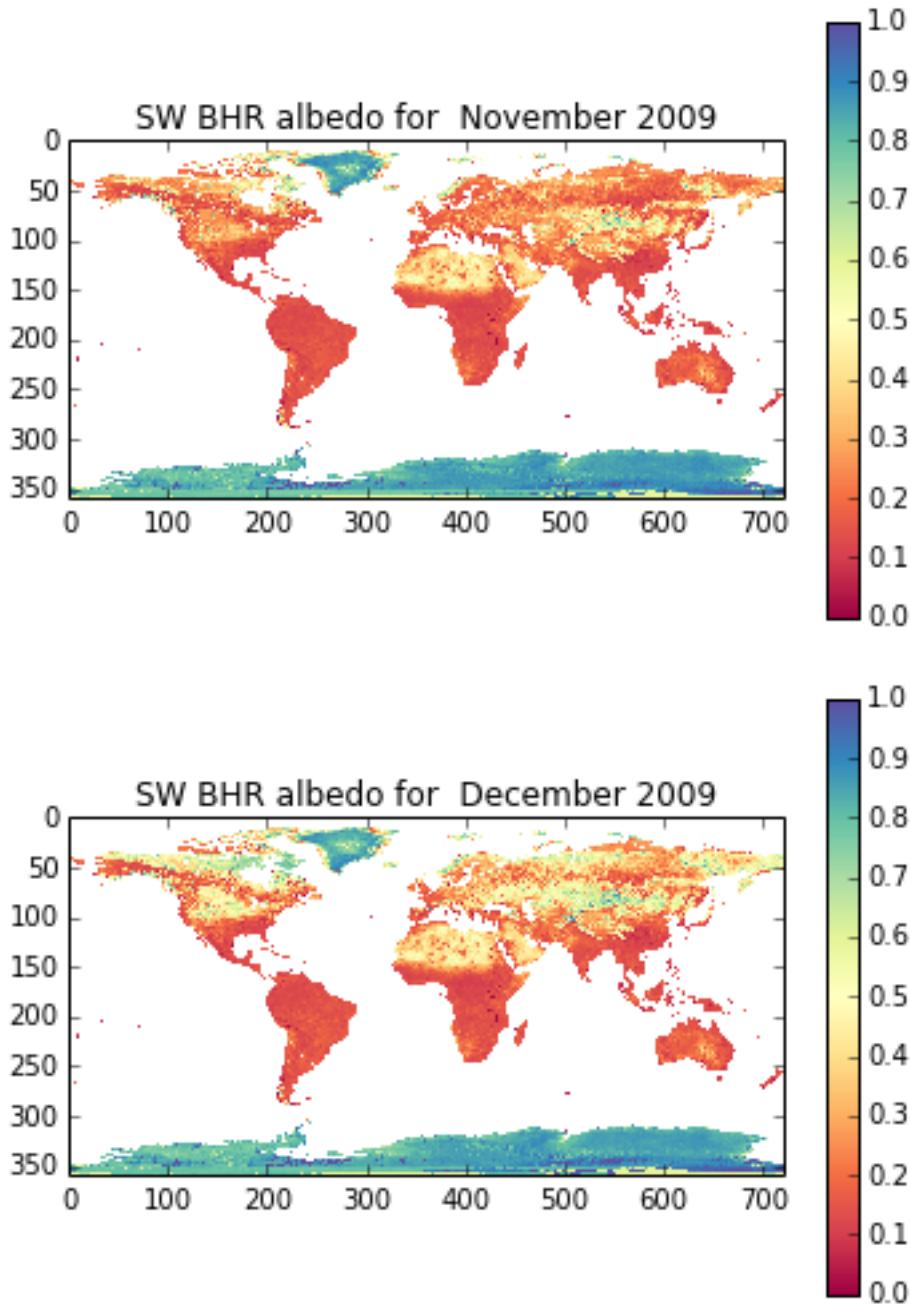












30.3.4 E3.2.3 Make the movie

The unix command to convert these files to an animated gif is:

```
convert -delay 100 -loop 0 files/data/GlobAlbedo.2009???.gif files/data/GlobAlbedo.2009.SW.gif
```

Run this (ideally, from within Python) to create the animated gif GlobAlbedo.2009.SW.gif

Again, confirm that *you* created this file (and it is not just a version you downloaded):

```
ls -l files/data/GlobAlbedo.2009.SW.gif
```

```
-rw-r--r-- 1 plewis staff 340658 11 Oct 19:59 files/data/GlobAlbedo.2009.SW.gif
```

30.3.5 A3.2.3 Answer: Make the movie

You could just type the command at the unix prompt ... but to do it using a `system` call from within Python, e.g.:

```
import os

# this is quite a neat way of generating the string for the input files
out_file = 'files/data/GlobAlbedo.%d.SW.1.gif'%year
in_files = out_file.replace('.SW.1.gif','???.gif')

cmd = 'convert -delay 100 -loop 0 %s %s'%(in_files,out_file)
# check the cmd is ok
print cmd

convert -delay 100 -loop 0 files/data/GlobAlbedo.2009???.gif files/data/GlobAlbedo.2009.SW.1.gif

# good ... so now run it
os.system(cmd)

0
```

To view the animated gif you have generated, open it in a browser.

30.4 E3.3 Exercise: 3D Masked Array

```
from netCDF4 import Dataset
import numpy as np

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# empty list
data = []

# loop over month
# use enumerate so we have an index counter
for i,month in enumerate(months):
    # this then is the file we want
    local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

    # load the netCDF data from the file local_file
    nc = Dataset(local_file,'r')
    # append what we read to the list called data
    data.append(np.array(nc.variables['DHR_SW']))

# convert data to a numpy array (its a list of arrays at the moment)
data = np.array(data)
```

N.B. Do this exercise before proceeding to the next section.

Taking the code above as a starting point, generate a masked array of the GlobAlbedo dataset for the year 2009.

30.5 A3.3 Answer: 3D Masked Array

To recap, to make a masked array, we use some code such as:

```
import numpy.ma as ma

band = np.array(nc.variables['DHR_SW'])

masked_band = ma.array(band,mask=np.isnan(band) )
print masked_band.mask

[[ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 [ True  True  True ...,  True  True  True]
 ...
 [False False False ..., False False False]
 [False False False ..., False False False]
 [False False False ..., False False False]]
```

So we can do this for each band as we loop over each month:

```
from netCDF4 import Dataset
import numpy as np
import numpy.ma as ma

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# empty list
data = []

# loop over month
# use enumerate so we have an index counter
for i,month in enumerate(months):
    # this then is the file we want
    local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

    # load the netCDF data from the file local_file
    nc = Dataset(local_file,'r')
    # load into the variable 'band'
    band = np.array(nc.variables['DHR_SW'])
    # convert to a masked array
    masked_band = ma.array(band,mask=np.isnan(band) )
    # append what we read to the list called data
    data.append(masked_band)

# convert data to a numpy array (its a list of arrays at the moment)
data = np.array(data)
```

That's a good start, but we used:

```
data = np.array(data)
```

at the end, which means that we have a 3D numpy array ... rather than a masked array:

```
print type(data)
print data.shape
print data.ndim
```

```
<type 'numpy.ndarray'>
(12, 360, 720)
3
```

so we need to replace this with a function to make it into a masked array:

```
from netCDF4 import Dataset
import numpy as np
import numpy.ma as ma

root = 'files/data/'
year = 2009

# which months?
months = xrange(1,13)

# empty list
data = []

# loop over month
# use enumerate so we have an index counter
for i,month in enumerate(months):
    # this then is the file we want
    local_file = root + 'GlobAlbedo.%d%02d.mosaic.5.nc'%(year,month)

    # load the netCDF data from the file local_file
    nc = Dataset(local_file,'r')
    # load into the variable 'band'
    band = np.array(nc.variables['DHR_SW'])
    # convert to a masked array
    masked_band = ma.array(band,mask=np.isnan(band) )
    # append what we read to the list called data
    data.append(masked_band)

# convert data to a numpy array (its a list of arrays at the moment)
data = ma.array(data)

print type(data)
print data.shape
print data.ndim

<class 'numpy.ma.core.MaskedArray'>
(12, 360, 720)
3
```

we might notice that now, the data mask (`data.mask`) is a 3D mask:

```
print data.mask.shape
print data.mask.ndim

(12, 360, 720)
3
```

and we might just check that the mask is the same for all months:

First, lets sum the masks over axis 0 (i.e. over all months):

```
sum = (data.mask).sum(axis=0)
print sum.shape

(360, 720)

print sum
```

```
[[12 12 12 ... , 12 12 12]
 [12 12 12 ... , 12 12 12]
 [12 12 12 ... , 12 12 12]
 ...
 [ 0  0  0 ... ,  0  0  0]
 [ 0  0  0 ... ,  0  0  0]
 [ 0  0  0 ... ,  0  0  0]]
```

This should be 12 (where mask is True) or 0 (where mask is False). How can we check that quickly?

Fortunately, there is a convenient numpy function `np.unique` that will give you the unique values in an array:

```
np.unique(sum)

array([ 0, 12])
```

The only values in here are 12 and 0, so the masks must be consistent!

```
%pylab inline
%config InlineBackend.figure_format = 'svg'
```

Populating the interactive namespace from numpy and matplotlib

CHAPTER
THIRTYONE

INCOMING SOLAR RADIATION

Incoming solar radiation, A , ignoring the effect of the atmosphere, clouds, etc is given by

$$A = E_0 \sin \theta$$

where E_0 is the solar constant (given as $1360 W m^{-2}$), and θ is the solar elevation angle. The solar elevation angle is approximately given by

$$\sin \theta = \cos h \cos \delta \cos \varphi + \sin \delta \sin \varphi$$

where h is the hour angle, δ is the solar declination angle and φ is the latitude. The solar declination can be approximated by

$$\delta = -\arcsin [0.39779 \cos (0.98565 (N + 10) + 1.914 \sin (0.98565 (N - 2)))]$$

where N is the day of year beginning with $N=0$ at 00:00:00 UTC on January 1

```
e0 = 1360.
latitude = np.deg2rad ( 45. )
N = np.arange ( 0.5, 366, 1 )
h = -22.5
t0 = np.deg2rad ( 0.98565*(N-2) )
t1 = 0.39779*np.cos( np.deg2rad ( 0.98565*(N+10) + 1.914*np.sin ( t0 ) ) )
t2 = -np.arcsin ( t1 )

sin_theta = np.cos ( np.deg2rad(h) )*np.cos (t2)*np.cos(latitude) + np.sin ( t2)*np.sin(latitude)
incoming_rad = e0*sin_theta
plt.plot ( N, incoming_rad , label="10:30AM")
h = 0
t0 = np.deg2rad ( 0.98565*(N-2) )
t1 = 0.39779*np.cos( np.deg2rad ( 0.98565*(N+10) + 1.914*np.sin ( t0 ) ) )
t2 = -np.arcsin ( t1 )

sin_theta = np.cos ( np.deg2rad(h) )*np.cos (t2)*np.cos(latitude) + np.sin ( t2)*np.sin(latitude)
incoming_rad = e0*sin_theta

plt.plot ( N, incoming_rad , label="12:00PM")

plt.xlabel("Day of Year")
plt.ylabel('Incoming solar radiation $[W\\cdot m^{-2}]$')

<matplotlib.text.Text at 0x4543590>
```


CHAPTER 4. GEOSPATIAL DATA

In this session, we will introduced the `gdal` geospatial module which can read a wide range of scientific data formats. You will find that using it to read data is quite similar to the work we did last week on netCDF datasets.

The main challenges are also much the same: very often, you need to be able to read data from a ‘stack’ of image files and generate a useful 3D (space and time) dataset from these. Once you have the data in such a form, there are *many* things we can do with it, and very many of these are convenient to do using array-based expressions such as in `numpy` (consider the simplicity of the expression `absorbed = rad * (1 - albedo)` from last week’s exercise).

That said, it can sometimes be quite an effort to prepare datasets in this form. Last week, we developed a ‘valid data’ mask from the GlobAlbedo dataset, as invalid data were stored as `nan`. Very often though, scientific datasets have more complex ‘Quality Control’ (QC) information, that gives per-pixel information describing the quality of the product at that location (e.g. it was very cloudy so the results are not so good).

To explore this, we will first consider the [MODIS Leaf Area Index \(LAI\)](#) product taht is mapped at 1 km resolution, every 8 days from the year 2000.

We will learn how to read in these data (in `hdf` format) using `gdal`, and how to interpret the QC information in such products to produce valid data masks. As an exercise, you will wrap some code around that to form a 3D masked array of the dataset.

Next, we will consider how to download such data. This should be a reinforcement of material from last week, but it is useful to know how to conveniently access NASA data products. A challenge in the exercise then is to download a different dataset (MODIS snow cover) for the UK, and form a masked 3D dataset from this.

Finally, we will introduce vector datasets and show you python tools that allow you (among many other things) to build a mask in the projection and sampling of your spatial dataset (MODIS LAI in this case).

There are many features and as many complexities to the Python tools we will deal with today, but in this material, we cover some very typical tasks you will want to do. They all revolve around generating masked 3D datasets from NASA MODIS datasets, which is a very useful form of global biophysical information over the last decade+. We also provide much material for further reading and use when you are more confident in your programming.

A final point here is that the material we cover today is very closely related to what you will need to do in the first section of your assessed practical that we will introduce next week, so you really need to get to grips with this now.

There is not as much ‘new’ material as in previous weeks now, but we assume that you have understood, and can make use of, material from those lectures.

First, we will examine data from a NASA MODIS product on Leaf Area Index (LAI).

- 4.1 MODIS LAI product
- 4.2 Downloading data
- 4.3 Vector masking

32.1 4.1 MODIS LAI product

GDAL covers a much wider set of file formats and methods than the netCDF library that we previously used.

Basic operation involves:

- load gdal
- open a spatial dataset (an hdf format file here)
- specify which subsets you want.

We can explore the subsets in the file with `GetSubDatasets()`:

```
# how to find out which datasets are in the file

import gdal # Import GDAL library bindings

# The file that we shall be using
# Needs to be on current directory
filename = 'files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'

g = gdal.Open(filename)
# g should now be a GDAL dataset, but if the file isn't found
# g will be none. Let's test this:
if g is None:
    print "Problem opening file %s!" % filename
else:
    print "File %s opened fine" % filename

subdatasets = g.GetSubDatasets()
for fname, name in subdatasets:
    print name
    print "\t", fname

File files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf opened fine
[1200x1200] Fpar_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] Lai_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparLai_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparExtra_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] LaiStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
```

In the previous code snippet we have done a number of different things:

1. Import the GDAL library
2. Open a file with GDAL, storing a handler to the file in `g`
3. Test that `g` is not `None` (as this indicates failure opening the file. Try changing `filename` above to something else)
4. We then use the `GetSubDatasets()` method to read out information on the different subdatasets available from this file (compare to the output of `gdalinfo` on the shelf earlier)
5. Loop over the retrieved subdatasets to print the name (human-readable information) and the GDAL filename. This last item is the filename that you need to use to tell GDAL to open a particular data layer of the 6 layers present in this example

Let's say that we want to access the LAI information. By contrasting the output of the above code (or `gdalinfo`) to the contents of the [LAI/fAPAR product information page](#), we find out that we want the layers for `Lai_1km`, `FparLai_QC`, `FparExtra_QC` and `LaiStdDev_1km`.

To read these individual datasets, we need to open each of them individually using GDAL, and the GDAL filenames used above:

```
# How to access specific datasets in gdal

# Let's create a list with the selected layer names
selected_layers = [ "Lai_1km", "FparLai_QC", "LaiStdDev_1km" ]

# We will store the data in a dictionary
# Initialise an empty dictionary
data = {}

# for convenience, we will use string substitution to create a
# template for GDAL filenames, which we'll substitute on the fly:
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
# This has two substitutions (the %s parts) which will refer to:
# - the filename
# - the data layer

for i, layer in enumerate ( selected_layers ):
    this_file = file_template % ( filename, layer )
    print "Opening Layer %d: %s" % (i+1, this_file )
    g = gdal.Open ( this_file )

    if g is None:
        raise IOError
    data[layer] = g.ReadAsArray()
    print "\t>>> Read %s!" % layer

Opening Layer 1: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.2011213154534.hdf":MOD_
    >>> Read Lai_1km!
Opening Layer 2: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.2011213154534.hdf":MOD_
    >>> Read FparLai_QC!
Opening Layer 3: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.2011213154534.hdf":MOD_
    >>> Read LaiStdDev_1km!
```

In the previous code, we have seen a way of neatly creating the filenames required by GDAL to access the independent datasets: a template string that gets substituted with the filename and the layer name. Note that the presence of double quotes in the template requires us to use single quotes around it. The data is now stored in a dictionary, and can be accessed as e.g. `data['Lai_1km']` which is a numpy array:

```
type(data['Lai_1km'])

numpy.ndarray

print data['Lai_1km']

[[ 3  3  2 ...,  6  8 21]
 [ 4  3  6 ...,  8 18 14]
 [ 3 12 11 ..., 12  8  8]
 ...,
 [ 2  3  2 ..., 18 11 17]
 [ 2  3  3 ..., 16 19 15]
 [ 3  2  2 ..., 15 16 15]]
```

Now we have to translate the LAI values into meaningful quantities. According to the [LAI](#) webpage, there is a scale factor of 0.1 involved for LAI and SD LAI:

```
lai = data['Lai_1km'] * 0.1
lai_sd = data['LaiStdDev_1km'] * 0.1

print "LAI"
print lai
print "SD"
print lai_sd

LAI
[[ 0.3  0.3  0.2 ... ,  0.6  0.8  2.1]
 [ 0.4  0.3  0.6 ... ,  0.8  1.8  1.4]
 [ 0.3  1.2  1.1 ... ,  1.2  0.8  0.8]
 ...
 [ 0.2  0.3  0.2 ... ,  1.8  1.1  1.7]
 [ 0.2  0.3  0.3 ... ,  1.6  1.9  1.5]
 [ 0.3  0.2  0.2 ... ,  1.5  1.6  1.5]]
SD
[[ 0.2  0.2  0.1 ... ,  0.2  0.1  0.3]
 [ 0.2  0.2  0.2 ... ,  0.2  0.3  0.2]
 [ 0.  0.1  0.2 ... ,  0.1  0.2  0.2]
 ...
 [ 0.1  0.1  0.1 ... ,  0.3  0.  0.1]
 [ 0.1  0.1  0.1 ... ,  0.2  0.2  0.1]
 [ 0.1  0.1  0.1 ... ,  0.1  0.2  0.1]]

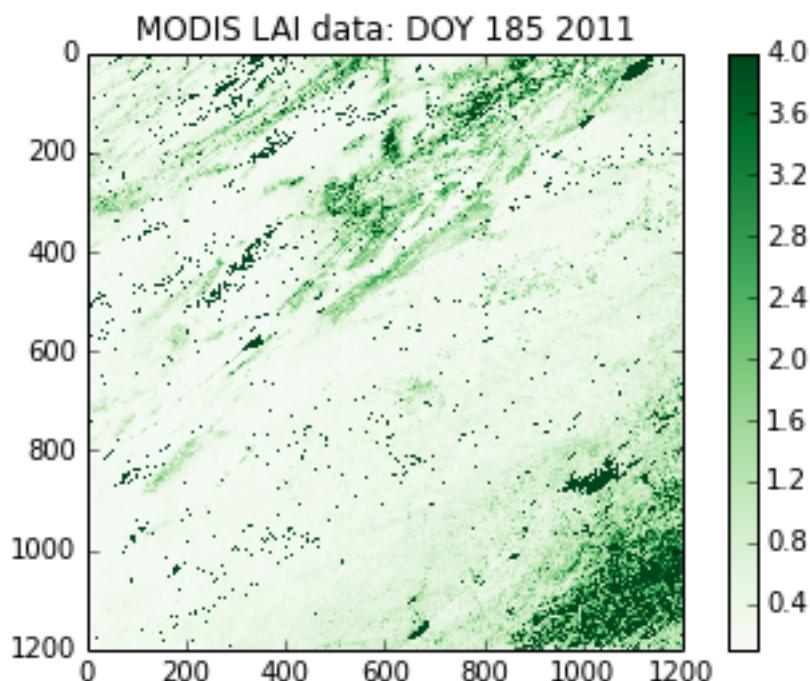
# plot the LAI

import pylab as plt

# colormap
cmap = plt.cm.Greens

plt.imshow(lai,interpolation='none',vmin=0.1,vmax=4.,cmap=cmap)
plt.title('MODIS LAI data: DOY 185 2011')
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc680e1830>
```



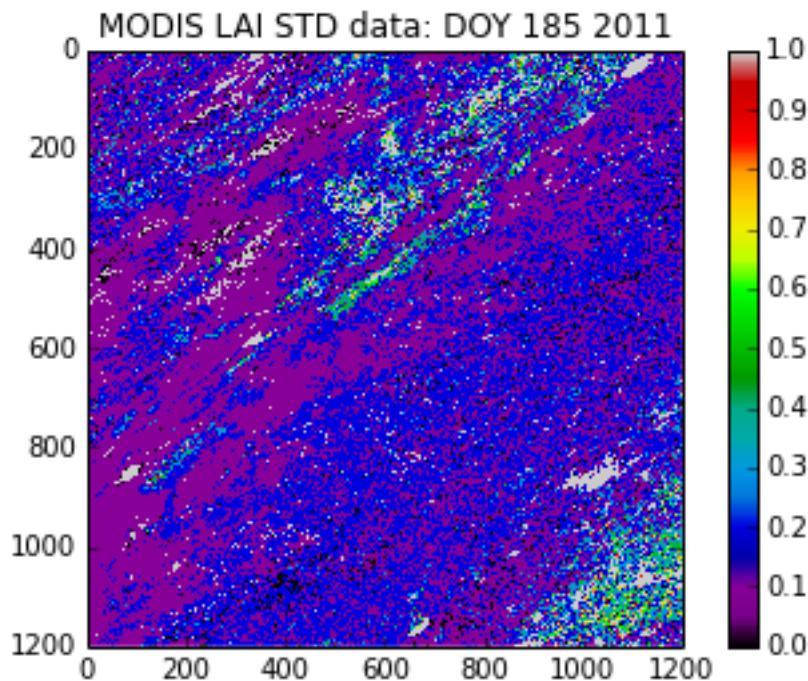
```
# plot the LAI std

import pylab as plt

# colormap
cmap = plt.cm.spectral
# this sets the no data colour. 'k' is black

plt.imshow(lai_sd, interpolation='none', vmax=1., cmap=cmap)
plt.title('MODIS LAI STD data: DOY 185 2011')
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc6954c098>
```



It is not possible to produce LAI estimates if it is persistently cloudy, so the dataset may contain some gaps.

These are identified in the dataset using the QC (Quality Control) information.

We should then examine this.

The codes for this are also given on the LAI product page. They are described as bit combinations:

Bit No.

Parameter Name	Bit Combination	Explanation
----------------	-----------------	-------------

0

MODLAND_QC bits

0

Good quality (main algorithm with or without saturation)

1

Other Quality (back-up algorithm or fill values)

1

Sensor

0</td><td> TERRA</td>

1

AQUA

2

DeadDetector

0

Detectors apparently fine for up to 50% of channels 1 2

1

Dead detectors caused >50% adjacent detector retrieval

3-4

CloudState

00

Significant clouds NOT present (clear)

01

Significant clouds WERE present

10

Mixed clouds present on pixel

11

Cloud state not defined assumed clear

5-7

CF_QC

000

Main (RT) method used best result possible (no saturation) </td>

001

Main (RT) method used with saturation. Good very usable

010

Main (RT) method failed due to bad geometry empirical algorithm used

011

Main (RT) method failed due to problems other than geometry empirical algorithm used

100

Pixel not produced at all value couldn't be retrieved (possible reasons: bad L1B data unusable MODAGAGG data)

In using this information, it is up to the user which data he/she wants to pass through for any further processing. There are clearly trade-offs: if you look for only the highest quality data, then the number of samples is likely to be lower than if you were more tolerant. But if you are too tolerant, you will get spurious results. You may find useful information on how to convert from actual QA flags to diagnostics in this page (they focus on NDVI/EVI, but the theory is the same).

But let's just say that we want to use only the highest quality data.

This means we want bit 0 to be 0 ...

Let's have a look at the QC data:

```
qc = data['FparLai_QC'] # Get the QC data which is an unsigned 8 bit byte
print qc , qc.dtype

[[2 2 0 ..., 0 2 2]
 [2 2 0 ..., 2 0 2]
 [0 2 0 ..., 0 0 0]
 ...,
 [0 0 2 ..., 0 8 0]
 [0 0 0 ..., 0 0 2]
 [0 2 0 ..., 2 2 2]] uint8
```

We see various byte values:

```
np.unique(qc)

array([ 0,   2,   8,  10,  16,  18,  32,  34,  40,  42,  48,  50,  97,
       99, 105, 107, 113, 115, 157], dtype=uint8)

# translated into binary using bin()
for i in np.unique(qc):
    print i,bin(i)

0 0b0
2 0b10
8 0b1000
10 0b1010
16 0b10000
18 0b10010
32 0b100000
34 0b100010
40 0b101000
42 0b101010
48 0b110000
50 0b110010
97 0b1100001
99 0b1100011
105 0b1101001
107 0b1101011
113 0b1110001
115 0b1110011
157 0b10011101
```

We could try to come up with an interpretation of each of these ... or we could try to mask the qc bytes to see bit 0 only if that's what we are interested in. This is quite possibly a new concept for most of you, but it is very common that when interpreting QC data in data products, you need to think about bit masking. You will find more details on this in the advanced section of Chapter 1, but we will consider the minimum we need right now.

Byte data are formed of 8 bits, e.g.:

```
105 == (1 * 2**6) + (1 * 2**5) + (0 * 2**4) + (1 * 2**3) + (0 * 2**2) + (0
* 2**1) + (1 * 2**0)
```

So, in binary, we represent the decimal number 105 by 1101001 as we saw above.

The QC values are to be interpreted in this manner.

If we want *only* bit 1, we can perform a *bitwise* operation with the byte data.

In this case, it would be an ‘and’ operation (&) with the value 1:

```
# suppose we consider the value 105
# which from above, we know to have
# bit 0 set as 1
test = 105
```

```
bit_zero = test & 1
print bit_zero

1

# suppose we consider the value 104
# which we could work out has bit 1 as 0
test = 104
bit_zero = test & 1
print bit_zero

0

# other bit fields are a 'little' more complicated
tests = np.unique(qc)

for t in tests:
    # if we want bit field 5-7
    # we form a binary mask
    mask57 = 0b11100000
    # but 0
    mask0 = 0b00000001
    # and use & as before and right shift 5 (>> 5)
    qa57 = (t & mask57) >> 5
    qa0 = (t & mask0) >> 0
    print t,qa57,qa0,bin(t)

0 0 0 0b0
2 0 0 0b10
8 0 0 0b1000
10 0 0 0b1010
16 0 0 0b10000
18 0 0 0b10010
32 1 0 0b100000
34 1 0 0b100010
40 1 0 0b101000
42 1 0 0b101010
48 1 0 0b110000
50 1 0 0b110010
97 3 1 0b1100001
99 3 1 0b1100011
105 3 1 0b1101001
107 3 1 0b1101011
113 3 1 0b1110001
115 3 1 0b1110011
157 4 1 0b10011101
```

So, for example (examining the table above) 105 is interpreted at 0b011 in fields 5 to 7 (which is 3 in decimal). This indicates that ‘Main (RT) method failed due to problems other than geometry empirical algorithm used’. Here, bit zero is set to 1, so this is a ‘bad’ pixel.

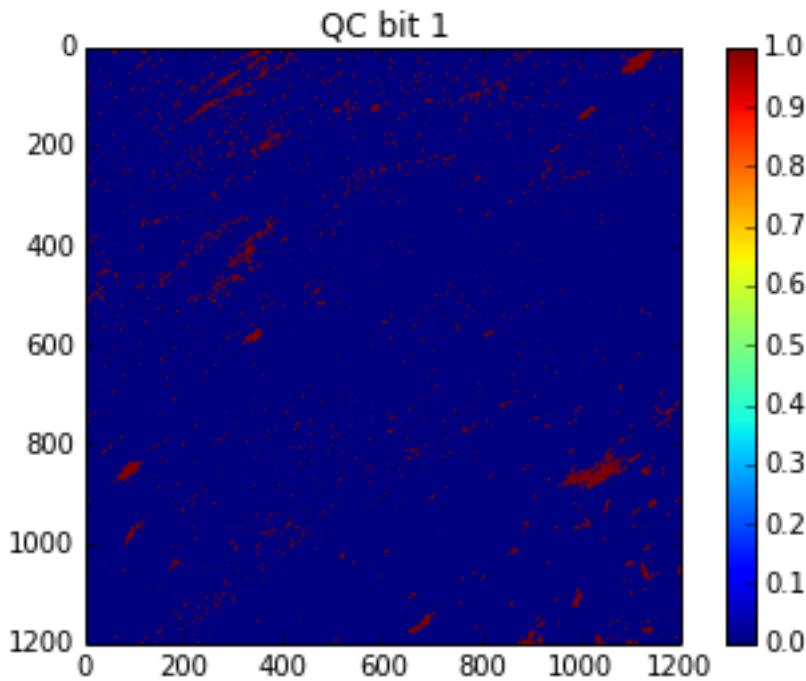
In this case, we are only interested in bit 0, which is an easier task than interpreting all of the bits.

```
# the good data are where qc bit 1 is 0

qc = data['FparLai_QC'] # Get the QC data
# find bit 0
qc = qc & 1

plt.imshow(qc)
plt.title('QC bit 1')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar instance at 0xc6b71b8>
```



We can use this mask to generate a masked array. Masked arrays, as we have seen before, are like normal arrays, but they have an associated mask.

Remember that the mask in a masked array should be `False` for good data, so we can directly use `qc` as defined above.

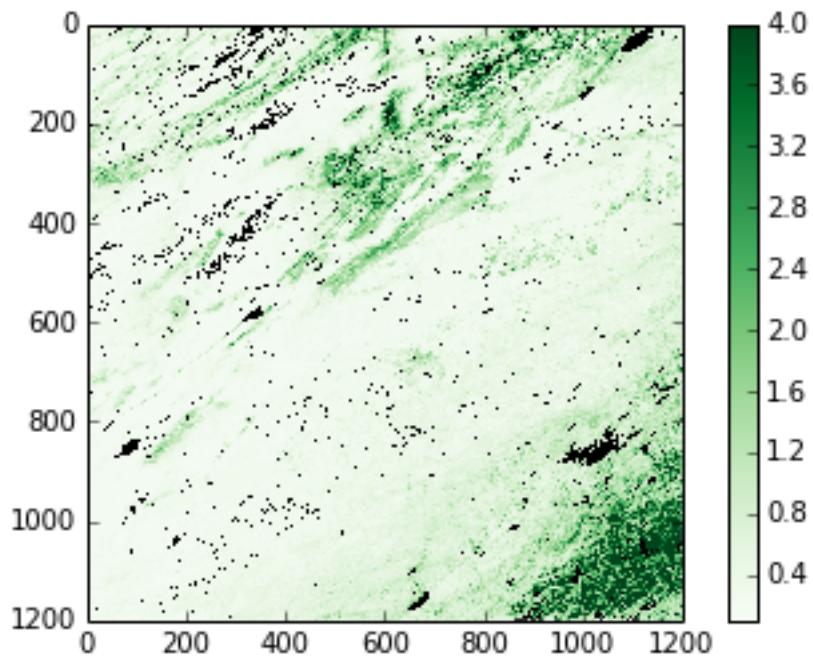
We shall also choose another colormap (there are [lots to choose from](#)), and set values outside the 0.1 and 4 to be shown as black pixels.

```
# colormap
cmap = plt.cm.Greens
cmap.set_bad( 'k' )
# this sets the no data colour. 'k' is black

# generate the masked array
laim = np.ma.array( lai, mask=qc )

# and plot it
plt.imshow( laim, cmap=cmap, interpolation='none', vmin=0.1, vmax=4)
plt.colorbar()
```

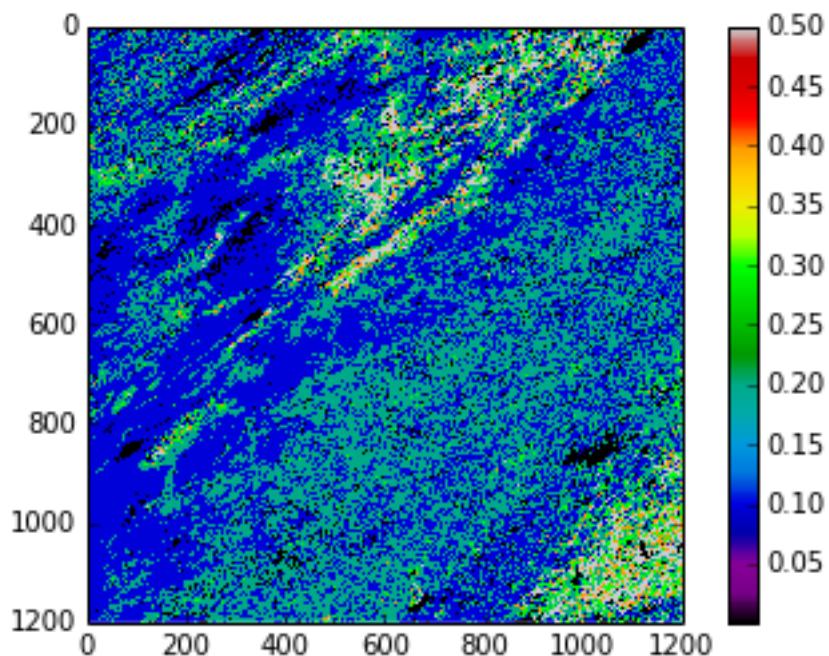
```
<matplotlib.colorbar.Colorbar instance at 0x2adc6b8dd5a8>
```



Similarly, we can do a similar thing for Standard Deviation

```
cmap = plt.cm.spectral
cmap.set_bad( 'k' )
stdm = np.ma.array( lai_sd, mask=qc )
plt.imshow( stdm, cmap=cmap, interpolation='none', vmin=0.001, vmax=0.5)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc70725878>
```



For convenience, we might wrap all of this up into a function:

```
import gdal
import numpy as np
import numpy.ma as ma
```

```

def getLAI(filename, \
    qc_layer = 'FparLai_QC', \
    scale = [0.1, 0.1], \
    selected_layers = ["Lai_1km", "LaiStdDev_1km"]):

    # get the QC layer too
    selected_layers.append(qc_layer)
    scale.append(1)
    # We will store the data in a dictionary
    # Initialise an empty dictionary
    data = {}
    # for convenience, we will use string substitution to create a
    # template for GDAL filenames, which we'll substitute on the fly:
    file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
    # This has two substitutions (the %s parts) which will refer to:
    # - the filename
    # - the data layer
    for i,layer in enumerate(selected_layers):
        this_file = file_template % (filename, layer)
        g = gdal.Open (this_file)

        if g is None:
            raise IOError

        data[layer] = g.ReadAsArray() * scale[i]

    qc = data[qc_layer] # Get the QC data
    # find bit 0
    qc = qc & 1

    odata = {}
    for layer in selected_layers[:-1]:
        odata[layer] = ma.array(data[layer],mask=qc)

    return odata

filename = 'files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'

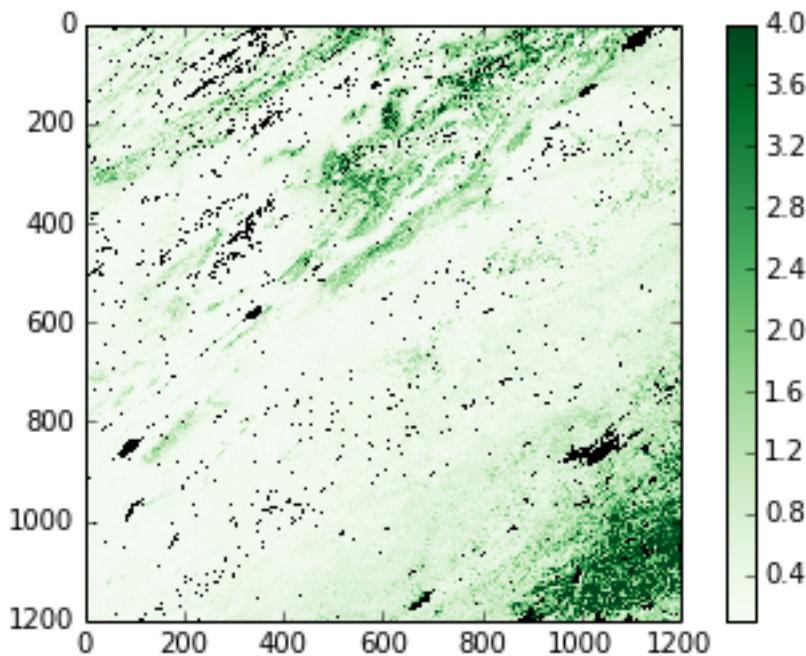
lai_data = getLAI(filename)

# colormap
cmap = plt.cm.Greens
cmap.set_bad ('k')
# this sets the no data colour. 'k' is black

# and plot it
plt.imshow (lai_data['Lai_1km'], cmap=cmap, interpolation='nearest', vmin=0.1, vmax=4)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2adc709ffe18>

```



32.2 Exercise 4.1

You are given the MODIS LAI data files for the year 2012 in the directory `files/data` for the UK (MODIS tile h17v03).

Read these LAI datasets into a masked array, using QA bit 0 to mask the data (i.e. good quality data only) and generate a movie of LAI.

You should end up with something like:

32.3 4.2 Downloading data

For the exercise and notes above, you were supplied with several datasets that had been previously downloaded.

32.3.1 4.2.1 Reverb

These NASA data can be accessed in several ways (except on Wednesdays when they go down for maintainance (or when there is a Government shutdown ...)). The most direct way is to use [Reverb](#) to explore and access data. If you do this, you can, for example [search for MODIS snow cover MOD10 datasets covering the UK](#) for some given time period. If you follow this through, e.g. selecting [MODIS/Terra Snow Cover Daily L3 Global 500m SIN Grid V005](#) and then search for ‘granules’, you should get access to the datasets you want (select e.g. one of the files for gid h17v03 and save to cart). You then view the items in your cart, click ‘download’ and then ‘save’.

This should give you a text file with some urls in it, e.g.:

```
ftp://n4ft101u.ecs.nasa.gov/DP0/MOST/MOD10A1.005/2013.02.21/MOD10A1.A2013052.h17v03.005.  
ftp://n4ft101u.ecs.nasa.gov/DP0/MOST/MOD10A1.005/2013.02.21/MOD10A1.A2013052.h17v03.005.  
ftp://n4ft101u.ecs.nasa.gov/DP0/BRWS/Browse.001/2013.02.23/BROWSE.MOD10A1.A2013052.h17v03.005.  
http://browse.echo.nasa.gov/NSIDC_ECS/2013/02/23/:BR:Browse.001:46841269:1.BINARY
```

Here, we can note that one of them is a jpeg file:

which is the quicklook (BROWSE file) for that tile / date.

The hdf dataset is, in this case `ftp://n4ftl01u.ecs.nasa.gov/DP0/MOST/MOD10A1.005/2013.02.21/MOD10A1...`

From this, we see that the data server is `n4ftl01u.ecs.nasa.gov` and that the MODIS snow products for the MODIS Terra instrument are in the directory:

`ftp://n4ftl01u.ecs.nasa.gov/DP0/MOST.`

If we explored that, we would find the datasets from the MODIS Aqua platform were in:

`ftp://n4ftl01u.ecs.nasa.gov/DP0/MOSA.`

The directories below that give the date and filename.

32.3.2 4.2.2 FTP access

Now we have discovered something about the directory structure on the server, we could explore this to get the datasets we want (rather than having to go through Reverb).

Some of the datasets on Reverb are accessible only through `http`, but the snow products (at present) are available by `ftp`.

In either case, if we want to run some ‘batch’ process to download many files (e.g. all files for a year for some tile), the first thing we need is the set of urls for the files we want.

We won’t go into detail here about how to get this, but it is covered in the advanced section (at the very least, you could always get the set of urls from Reverb).

So, let’s suppose now that we have a file containing some urls that we want to pull:

```
!head -10 < files/data/robot_snow.2012.txt
```

```
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h00v08.005.2012006235251
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h00v09.005.2012006235244
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h00v10.005.2012006234603
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v08.005.2012006234607
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v09.005.2012006234607
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v10.005.2012006235245
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h01v11.005.2012006234656
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h02v06.005.2012006235323
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h02v08.005.2012006234419
ftp://n4ftl01u.ecs.nasa.gov/MOSA/MYD10A1.005/2012.01.01/MYD10A1.A2012001.h02v09.005.2012006234419
```

This is a file containing the filenames of *all* MOD10A1 (daily snow cover) files for a given year, pulled from the `ftp` server. It is possible to do this in Python (see advanced section), but actually must easier and faster with an `ftp` script ‘`files/python/zat-snow`’. You don’t need to run this, as it has already been run for you and the results put in the files:

```
ls -l files/data/robot_snow*.txt
```

```
[0m-rw-rw-r-- 1 plewis plewis 9034752 Oct 22 09:25 [0mfiles/data/robot_snow.2000.txt[0m
-rw-rw-r-- 1 plewis plewis 10820568 Oct 22 09:25 [0mfiles/data/robot_snow.2001.txt[0m
-rw-rw-r-- 1 plewis plewis 16321938 Oct 22 09:25 [0mfiles/data/robot_snow.2002.txt[0m
-rw-rw-r-- 1 plewis plewis 22423068 Oct 22 09:25 [0mfiles/data/robot_snow.2003.txt[0m
-rw-rw-r-- 1 plewis plewis 7633374 Oct 22 09:25 [0mfiles/data/robot_snow.2004.txt[0m
-rw-rw-r-- 1 plewis plewis 18872448 Oct 22 09:25 [0mfiles/data/robot_snow.2005.txt[0m
-rw-rw-r-- 1 plewis plewis 11433078 Oct 22 09:25 [0mfiles/data/robot_snow.2006.txt[0m
-rw-rw-r-- 1 plewis plewis 22663686 Oct 22 09:25 [0mfiles/data/robot_snow.2007.txt[0m
-rw-rw-r-- 1 plewis plewis 22668990 Oct 22 09:25 [0mfiles/data/robot_snow.2008.txt[0m
-rw-rw-r-- 1 plewis plewis 22705317 Oct 22 09:25 [0mfiles/data/robot_snow.2009.txt[0m
-rw-rw-r-- 1 plewis plewis 22712370 Oct 22 09:25 [0mfiles/data/robot_snow.2010.txt[0m
-rw-rw-r-- 1 plewis plewis 17129166 Oct 22 09:25 [0mfiles/data/robot_snow.2011.txt[0m
-rw-rw-r-- 1 plewis plewis 22756824 Oct 22 09:25 [0mfiles/data/robot_snow.2012.txt[0m
```

```
-rw-rw-r-- 1 plewis plewis 18104898 Oct 22 09:25 [0mfiles/data/robot_snow.2013.txt [0m
[m
```

We can do similar things for http, but that is a little more complicated (again, see advanced notes or `files/zat > files/data/robot.txt <files/zat>`__ for the MODIS LAI product.

```
!head -10 < files/data/robot.txt
```

```
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h00v08.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h00v09.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h00v10.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v07.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v08.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v09.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v10.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h01v11.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h02v06.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2002.07.04/MCD15A2.A2002185.h02v08.00
```

The file files/data/robot_snow.2012.txt contains the names for all tiles and all files.

So if we want just e.g. tile h17v03 and sensor (MOST), we can most easily filter this in unix:

```
tile=h17v03
year=2012
type=MOST

file=files/data/robot_snow.${year}_${type}_${tile}.txt

grep $tile < files/data/robot_snow.$year.txt | grep $type > $file

# how many files?
wc -l < $file

# look at the first 10 ...
head -10 < $file

366
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.01/MOD10A1.A2012001.h17v03.005.2012003054416
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.02/MOD10A1.A2012002.h17v03.005.2012004061011
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.03/MOD10A1.A2012003.h17v03.005.2012005061244
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.04/MOD10A1.A2012004.h17v03.005.2012006054639
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.05/MOD10A1.A2012005.h17v03.005.2012007052708
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.06/MOD10A1.A2012006.h17v03.005.2012008070328
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.07/MOD10A1.A2012007.h17v03.005.2012011144012
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.08/MOD10A1.A2012008.h17v03.005.2012011154609
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.09/MOD10A1.A2012009.h17v03.005.2012011222125
ftp://n4ftl01u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.10/MOD10A1.A2012010.h17v03.005.2012012062821
```

With a more sensible set of urls now (only a few hundred), we can consider how to download them.

We can (of course) do this in Python (see advanced notes), but we can also use unix tools such as curl or wget, which may be easier.

For example, in bash:

```
tile=h17v03
year=2012
type=MOST

file=snow_list_${tile}_${year}_${type}.txt

# cd temporarily to the local directory
pushd files/data
# -nc : no clobber : dont download if its there already
```

```
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
popd

or in tcsh:

set tile=h17v03
set year=2012
set type=MOST

file=snow_list_${tile}_${year}_${type}.txt

# cd temporarily to the local directory
pushd files/data
# -nc : no clobber : dont download if its there already
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
popd
```

For the moment, let's just pull only some of these by filtering the month as well:

```
tile=h17v03
year=2012
type=MOST
month=01

file=files/data/robot_snow.${year}_${type}_${tile}_${month}.txt

# the dot in the year / month grep need to be escaped
# because dot means something special to grep
grep $tile < files/data/robot_snow.$year.txt | grep $type | grep "${year}\.${month}" > $file

# how many files?
wc -l < $file

# look at the first 10 ...
head -10 < $file

31
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.01/MOD10A1.A2012001.h17v03.005.2012003054416
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.02/MOD10A1.A2012002.h17v03.005.2012004061011
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.03/MOD10A1.A2012003.h17v03.005.2012005061244
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.04/MOD10A1.A2012004.h17v03.005.2012006054639
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.05/MOD10A1.A2012005.h17v03.005.2012007052708
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.06/MOD10A1.A2012006.h17v03.005.2012008070328
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.07/MOD10A1.A2012007.h17v03.005.2012011144012
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.08/MOD10A1.A2012008.h17v03.005.2012011154609
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.09/MOD10A1.A2012009.h17v03.005.2012011222125
ftp://n4ft101u.ecs.nasa.gov/MOST/MOD10A1.005/2012.01.10/MOD10A1.A2012010.h17v03.005.2012012062821

tile=h17v03
year=2012
type=MOST
month=01

file=robot_snow.${year}_${type}_${tile}_${month}.txt

# cd temporarily to the local directory
pushd files/data
# -nc : no clobber : dont download if its there already
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
# cd back again
```

```
popd  
Process is terminated.
```

32.4 Exercise 4.2 A Different Dataset

We have now downloaded a different dataset, the [MOD10A product](#), which is the 500 m MODIS daily snow cover product, over the UK.

This is a good opportunity to see if you can apply what was learned above about interpreting QC information and using `gdal` to examine a dataset.

If you examine the [data description page](#), you will see that the data are in HDF EOS format (the same as the LAI product).

32.4.1 E4.2.1 Download

Download the MODIS Terra daily snow product for the UK for the year 2012 for the month of February using the urls in `files/data/robot_snow.2012.txt` and put them in the directory `files/data`.

32.4.2 E4.2.1 Explore

Show all of the subset data layers in this dataset.

32.4.3 E4.3.3 Read a dataset

Suppose we are interested in the dataset `Fractional_Snow_Cover` over the land surface.

Read this dataset for one of the files into a numpy array and show a plot of the dataset.

32.4.4 E4.3.4 Water mask

The [data description page](#) tells us that values of 239 will indicate whether the data is ocean. You can use this information to build the water mask.

Demonstrate how to build a water mask from one of these files, setting the mask `False` for land and `True` for water.

Produce a plot of this.

32.4.5 E3.4.5 Valid pixel mask

As well as having a land/water mask, we should generate a mask for valid pixels. For the snow dataset, values between 0 and 100 (inclusive) represent valid snow cover data values. Other values are not valid for some reason. Set the mask to `False` for valid pixels and `True` for others. Produce a plot of the mask.

32.4.6 E4.3.6 3D dataset

Generate a 3D masked numpy array using the valid pixel mask for masking, of `Fractional_Snow_Cover` for each day of February 2012.

You might like to produce a movie of the result.

Hint: you will need a list of filenames for this. You can either use `glob` as in previous exercises, or you might notice that you have the file `files/data/robot_snow.2012_MOST_h17v03_02.txt` with the urls, from which you should be able to derive the file names. However you get your list of filenames, you should probably apply a `sort()` to the result to make sure they are in the correct order.

32.5 4.3 Vector masking

In this section, we will use a pre-defined function to generate a mask from some vector boundary data.

In this case, we will generate a mask for Ireland, projected into the coordinate system of the MODIS LAI dataset, and use that to generate a new LAI data only for Ireland.

Sometimes, geospatial data is acquired and recorded for particular geometric objects such as polygons or lines. An example is a road layout, where each road is represented as a geometric object (a line, with points given in a geographical projection), with a number of added *features* associated with it, such as the road name, whether it is a toll road, or whether it is dual-carriageway, etc. This data is quite different to a raster, where the entire scene is tessellated into pixels, and each pixel holds a value (or an array of value in the case of multiband rasterfiles).

If you are familiar with databases, vector files are effectively a database, where one of the fields is a geometry object (a line in our previous road example, or a polygon if you consider a cadastral system). We can thus select different records by writing queries on the features. Some of these queries might be spatial (e.g. check whether a point is inside a particular country polygon).

The most common format for vector data is the **ESRI Shapfile**, which is a multifile format (i.e., several files are needed in order to access the data). We'll start by getting hold of a shapefile that contains the countries of the world as polygons, together with information on country name, capital name, population, etc. The file is available here.

Figure 32.1: World

We will download the file with `wget` (or `curl` if you want to), and uncompress it using `unzip` in the shell:

```
# Downloads the data using wget
!wget -nc http://aprsworld.net/gisdata/world/world.zip -O files/data/world.zip
# or if you want to use curl...
#! curl http://aprsworld.net/gisdata/world/world.zip -o world.zip
!pushd files/data;unzip -o -x world.zip;popd

--2013-10-22 15:32:06-- http://aprsworld.net/gisdata/world/world.zip
Resolving aprsworld.net... 72.251.203.219
Connecting to aprsworld.net|72.251.203.219|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3436277 (3.3M) [application/zip]
Saving to: `files/data/world.zip'

100%[=====] 3,436,277 3.38M/s in 1.0s

2013-10-22 15:32:07 (3.38 MB/s) - `files/data/world.zip' saved [3436277/3436277]

/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter4_GDAL/files/data /archive/
Archive: world.zip
  inflating: world.dbf
  inflating: world.shp
  inflating: world.shx
/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter4_GDAL
```

We need to import `ogr`, and then open the file. As with GDAL, we get a handler to the file, (`g` in this case). OGR files can have different layers, although Shapefiles only have one. We need to select the layer using `GetLayer(0)` (selecting the first layer).

```
from osgeo import ogr

g = ogr.Open( "files/data/world.shp" )
layer = g.GetLayer( 0 )
```

In order to see a field (the field NAME) we can loop over the features in the layer, and use the GetField('NAME') method. We'll only do ten features here:

```
n_feat = 0
for feat in layer:

    print feat.GetField('NAME')

    n_feat += 1
    if n_feat == 10:
        break

GUATEMALA
BOLIVIA
PARAGUAY
URUGUAY
SURINAME
FRENCH GUIANA
WESTERN SAHARA
GAMBIA
MOROCCO
MALI
```

If you wanted to see the different layers, we could do this using:

```
layerDefinition = layer.GetLayerDefn()

for i in range(layerDefinition.GetFieldCount()):
    print "Field %d: %s" % ( i+1, layerDefinition.GetFieldDefn(i).GetName() )

Field 1: NAME
Field 2: CAPITAL
Field 3: APPROX
Field 4: AREA
Field 5: SOURCETHM
```

There is much more information on using `ogr` on the associated notebook OGR_Python that you should explore at some point.

One thing we may often wish to do with such vector datasets is produce a mask, e.g. for national boundaries. One of the complexities of this is changing the projection that the vector data come in to that of the raster dataset.

This is too involved to go over in this session, so we will simply present you with a function to achieve this.

This is available as `files/python/raster_mask.py`.

Most of the code below should be familiar from above (we make use of the `getLAI()` function we developed).

```
import sys
sys.path.insert(0,'files/python')
from raster_mask import raster_mask, getLAI

# test this on an LAI file

# the data file name
filename = 'files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf'

# a layer (doesn't matter so much which: use for geometry info)
layer = 'Lai_1km'
```

```

# the full dataset specification
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%(filename,layer)

# make a raster mask
# from the layer IRELAND in world.shp
mask = raster_mask(file_spec,\n
                    target_vector_file = "files/data/world.shp",\n
                    attribute_filter = "NAME = 'IRELAND'")

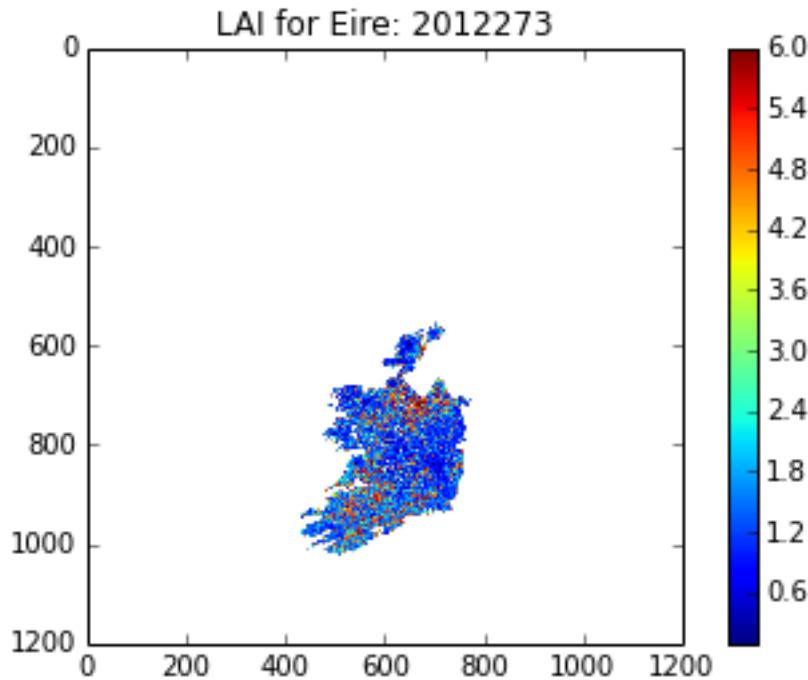
# get the LAI data
data = getLAI(filename)

# reset the data mask
# 'mask' is True for Ireland
# so take the opposite
data['Lai_1km'] = ma.array(data['Lai_1km'],mask=mask)
data['LaiStdDev_1km'] = ma.array(data['Lai_1km'],mask=mask)

plt.title('LAI for Eire: 2012273')
plt.imshow(data['Lai_1km'],vmax=6)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x7153878>

```



32.6 Exercise 4.3

Apply the concepts above to generate a 3D masked numpy data array of LAI and std LAI for Eire for the year 2012.

Plot your results and make a move of LAI.

Plot average LAI for Eire as a function of day of year for 2012.

CHAPTER
THIRTYTHREE

SUMMARY

In this session, we have learned to use some geospatial tools using GDAL in Python. A good set of [working notes on how to use GDAL](#) has been developed that you will find useful for further reading, as well as looking at the advanced section.

We have also very briefly introduced dealing with vector datasets in `ogr`, but this was mainly through the use of a pre-defined function that will take an ESRI shapefile (vector dataset), warp this to the projection of a raster dataset, and produce a mask for a given layer in the vector file.

If there is time in the class, we will develop some exercises to examine the datasets we have generated and/or to explore some different datasets or different locations.

GDAL AND OGR LIBRARIES

In the previous session, we used the GDAL library to open HDF files. GDAL is not limited to a single file format, but can actually cope with many different raster file formats seamlessly. For *vector* data (i.e., data that is stored by features, each being made up of fields containing different types of information, one of them being a *geometry*, such as polygon, line or point), GDAL has a sister library called OGR. The usefulness of these two libraries is that they allow the user to deal with many of the different file formats in a consistent way.

It is important to note that both GDAL and OGR come with a suite of command line tools that let you do many complex tasks on the command line directly. A listing of the GDAL command line tools is available [here](#), but bear in mind that many blogs etc. carry out examples of using GDAL tools in practice. For OGR, most of the library can be accessed using [ogr2ogr](#), but as usual, you might find more useful information on [blogs](#) etc.

34.1 GDAL data type

GDAL provides a very handy way of dealing with raster data in many different formats, not only by making access to the data easy, but also abstracting the nuances and complications of different file formats. In GDAL, a raster file is made up of the actual raster data (i.e., the values of each pixel of LAI that we saw before), and of some *metadata*. Metadata is data that describes something, and in this case it could be the projection, the location of corners and pixel spacing, etc.

34.1.1 The GeoTransform

GDAL stores information about the location of each pixel using the GeoTransform. The GeoTransform contains the coordinates (in some projection) of the upper left (UL) corner of the image (taken to be the **borders of the pixel** in the UL corner, not the center), the pixel spacing and an additional rotation. By knowing these figures, we can calculate the location of each pixel in the image easily. Let's see how this works with an easy example. We have prepared a GeoTIFF file (GeoTIFF is the more ubiquitous file format for EO data) of the MODIS landcover product for the UK. The data has been extracted from the HDF-EOS files that are similar to the LAI product that we have seen before, and converted. The file is 'lc_h17v03.tif <https://raw.github.com/jgomezdans/geogg122/master/ChapterX_GDAL/lc_h17v03.tif>'. We will open the file in Python, and have a look at finding a particular location.

Assume we are interested in locating Kinder Scout, a moorland in the Peak District National Park. Its coordinates are 1.871417W, 53.384726N. In the MODIS integerised sinusoidal projection, the coordinates are (-124114.3, 5936117.4) (you can use the [MODLAND tile calculator website](#) to do this calculation yourself).

```
import gdal # Import GDAL library
g = gdal.Open ( "lc_h17v03.tif" ) # Open the file
if g is None:
    print "Could not open the file!"
geo_transform = g.GetGeoTransform ()
print geo_transform
print g.RasterXSize, g.RasterYSize
```

```
(-1111950.519667, 463.3127165279167, 0.0, 6671703.118, 0.0, -463.3127165279165)
2400 2400
```

In the previous code, we open the file (we just use the filename), and then query the object for its GeoTransform, which we then print out. The 6-element tuple comprises

1. The Upper Left *easting* coordinate (i.e., *horizontal*)
2. The E-W pixel spacing
3. The rotation (0 degrees if image is “North Up”)
4. The Upper left *northing* coordinate (i.e., *vertical*)
5. The rotation (0 degrees)
6. The N-S pixel spacing, negative as we will be counting from the UL corner

We have also seen that the dataset is of size 2400x2400, using RasterXSize and RasterYSize. The goal is to find the pixel number (i, j) , $0 \leq i, j < 2400$ that corresponds to Kinder Scout. To do this, we use the following calculations:

```
pixel_x = (-124114.3 - geo_transform[0])/geo_transform[1] \
    # The difference in distance between the UL corner (geot[0] \
    #and point of interest. Scaled by geot[1] to get pixel number

pixel_y = (5936117.4 - geo_transform[3])/(geo_transform[5]) # Like for pixel_x, \
    #but in vertical direction. Note the different elements of geot \
    #being used

print pixel_x, pixel_y

2132.11549009 1587.66572071
```

So the pixel number is a floating point number, which we might need to round off as an integer. Let's plot the entire raster map (with minimum value 0 to ignore the ocean) using `plt.imshow` and plot the location of Kinder Scout using `plt.plot`. We will also use `plt.annotate` to add a label with an arrow:

```
lc = g.ReadAsArray() # Read raster data
# Now plot the raster data using gist_earth palette
plt.imshow ( lc, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
# Plot location of our area of interest as a red dot ('ro')
plt.plot ( pixel_x, pixel_y, 'ro' )
# Annotate
plt.annotate('Kinder Scout', xy=(pixel_x, pixel_y), \
    xycoords='data', xytext=(-150, -60), \
    textcoords='offset points', size=12, \
    bbox=dict(boxstyle="round4,pad=.5", fc="0.8"), \
    arrowprops=dict(arrowstyle="->", \
    connectionstyle="angle,angleA=0,angleB=-90,rad=10", \
    color='w'), )
# Remove vertical and horizontal ticks
plt.xticks([])
plt.yticks([])

([], <a list of 0 Text yticklabel objects>)
```

Try it out in some other places!

Find the longitude and latitude of some places of interest in the British isles (West of Greenwich!) and using the MODLAND MODIS tile calculator and the geotransform, repeat the above experiment. Note that the MODIS

calculator calculates both the projected coordinates in the MODIS sinusoidal projection, as well as the pixel number, so it is a helpful way to check whether you got the right result.

Park name

Longitude [deg]

Latitude [deg]

Dartmoor national park

-3.904

50.58

New forest national park

-1.595

50.86

Exmoor national park

-3.651

51.14

Pembrokeshire coast national park

-4.694

51.64

Brecon beacons national park

-3.432

51.88</td>

Pembrokeshire coast national park

-4.79

51.99

Norfolk and suffolk broads

1.569

52.62

Snowdonia national park

-3.898 </td><td>52.9</td>

Peak district national park

-1.802

53.3

Yorkshire dales national park

-2.157

54.23

North yorkshire moors national park

-0.8855

54.37

Lake district national park

-3.084

54.47
Galloway forest park
-4.171
54.87
Northumberland national park
-2.228
55.28
Loch lomond and the trossachs national park
-4.593
56.24
Tay forest park
-4.025
56.59</td>
Cairngorms national park
-3.545
57.08

34.2 The projection

Projections in GDAL objects are stored can be accessed by querying the dataset using the `GetProjection()` method. If we do that on the currently opened dataset (stored in variable `g`), we get:

```
print g.GetProjection()
```

```
PROJCS["unnamed",GEOGCS["Unknown datum based upon the custom spheroid",DATUM["Not_specified_based_
```

The above is the description of the projection (in this case, MODIS sinusoidal) in WKT (well-known text) format. There are a number of different ways of specifying projections, the most important being

- WKT
- Proj4
- EPSG codes

The site spatialreference.org allows you to search a large collection of projections, and get the representation that you want to use.

34.3 Saving files

So far, we have read data from files, but lets see how we can save raster data **to** a new file. We will use the previous landcover map as an example. We will write a method to save the data in a format provided by the user. The procedure is fairly straightforward: we get a handler to a driver (e.g. a GeoTIFF or Erdas Imagine format), we create the output file (giving a filename, number of rows, columns, bands, the data type), and then add the relevant metadata (projection, geotransform, ...). We then select a band from the output and copy the array that we want to write to that band.

```

g = gdal.Open ( "lc_h17v03.tif" ) # Open original file
# Get the x, y and number of bands from the original file
x_size, y_size, n_bands = g.RasterXSize, g.RasterYSize, g.RasterCount
data = g.ReadAsArray ()
driver = gdal.GetDriverByName ( "HFA" ) # Get a handler to a driver
# Maybe try "GeoTIFF" here
# Next line creates the output dataset with
# 1. The filename ("test_lc_h17v03.img")
# 2. The raster size (x_size, y_size)
# 3. The number of bands
# 4. The data type (in this case, Byte.
#     Other typical values might be gdal.GDT_Int16
#     or gdal.GDT_Float32)

dataset_out = driver.Create ( "test_lc_h17v03.img", x_size, y_size, n_bands, \
                             gdal.GDT_Byte )
# Set the output geotransform by reading the input one
dataset_out.SetGeoTransform ( g.GetGeoTransform() )
# Set the output projection by reading the input one
dataset_out.SetProjection ( g.GetProjectionRef() )
# Now, get band # 1, and write our data array.
# Note that the data array needs to have the same type
# as the one specified for dataset_out
dataset_out.GetRasterBand ( 1 ).WriteArray ( data )
# This bit forces GDAL to close the file and write to it
dataset_out = None

```

The output file should hopefully exist in this directory. Let's use `gdalinfo` <<http://www.gdal.org/gdalinfo.html>> to find out about it

```
!gdalinfo test_lc_h17v03.img
```

```

Driver: HFA/Erdas Imagine Images (.img)
Files: test_lc_h17v03.img
Size is 2400, 2400
Coordinate System is:
PROJCS["Sinusoidal",
    GEOGCS["GCS_Unknown datum based upon the custom spheroid",
        DATUM["Not_specified_based_on_custom_spheroid",
            SPHEROID["Custom_spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,6671703.117999999783933)
Pixel Size = (463.312716527916677,-463.312716527916507)
Corner Coordinates:
Upper Left  (-1111950.520, 6671703.118) ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-1111950.520, 5559752.598) ( 15d33'26.06"W, 50d 0' 0.00"N)
Upper Right  (      0.000, 6671703.118) ( 0d 0' 0.01"E, 60d 0' 0.00"N)
Lower Right  (      0.000, 5559752.598) ( 0d 0' 0.01"E, 50d 0' 0.00"N)
Center       (-555975.260, 6115727.858) ( 8d43' 2.04"W, 55d 0' 0.00"N)
Band 1 Block=64x64 Type=Byte, ColorInterp=Undefined
    Description = Layer_1
    Metadata:
        LAYER_TYPE=athematic

```

So the previous code works. Since this is something we typically do (read some data from one or more files, manipulate it and save the result in output files), it makes a lot of sense to try to put this code in a method that is more or less generic, that we can test and then re-use. Here's a first attempt at it:

```
def save_raster ( output_name, raster_data, dataset, driver="GTiff" ):
    """
    A function to save a 1-band raster using GDAL to the file indicated
    by ``output_name``. It requires a GDAL-accessible dataset to collect
    the projection and geotransform.

    Parameters
    -----
    output_name: str
        The output filename, with full path and extension if required
    raster_data: array
        The array that we want to save
    dataset: str
        Filename of a GDAL-friendly dataset that we want to use to
        read geotransform & projection information
    driver: str
        A GDAL driver string, like GTiff or HFA.
    """

    # Open the reference dataset
    g = gdal.Open ( dataset )
    # Get the Geotransform vector
    geo_transform = g.GetGeoTransform ()
    x_size = g.RasterXSize # Raster xsize
    y_size = g.RasterYSize # Raster ysize
    srs = g.GetProjectionRef () # Projection
    # Need a driver object. By default, we use GeoTIFF
    driver = gdal.GetDriverByName ( driver )
    dataset_out = driver.Create ( output_name, x_size, y_size, 1, \
                                  gdal.GDT_Float32 )
    dataset_out.SetGeoTransform ( geo_transform )
    dataset_out.SetProjection ( srs )
    dataset_out.GetRasterBand ( 1 ).WriteArray ( \
        raster_data.astype(np.float32) )
    dataset_out = None
```

Now try modifying that method so that you can

1. Select the output data type different to Float32
2. Provide a given projection and geotransform (e.g. if you don't have a GDAL filename)

34.4 Reprojecting things

Previously, we have used the [MODLAND](#) grid converter site to go from latitude/longitude pairs to MODIS projection. However, in practice, we might want to use a range of different projections, and convert many points at the same time, so how do we go about that?

In GDAL/OGR, most projection-related tools are in the `osr` package, which needs to be imported like e.g. `gdal` itself. The main tools are the `osr.SpatialReference` object, which defines a projection object (with no projection to start with), and the `osr.CoordinateTransformation` object.

Once you instantiate `osr.SpatialReference`, it holds no projection information. You need to use methods to set it up, using EPSG codes, Proj4 strings, or whatever. These methods typically start by `ImportFrom` (e.g. `ImportFromEPSG`, `ImportFromProj4`, ...).

The `CoordinateTransformation` requires a source and destination spatial references that have been configured. Once this is done, it exposes the method `TransformPoint` to convert coordinates from the source to the destination projection.

Let's see how this works by converting some latitude/longitude pairs to the Ordnance Survey's [National Grid](#) projection. The projection is also available in [spatialreference.org](#), where we can gleam its EPSG code (27700).

The EPSG code for longitude latitude is 4326. Let's see this in practice:

```
from osgeo import osr

# Define the source projection, WGS84 lat/lon.
wgs84 = osr.SpatialReference() # Define a SpatialReference object
wgs84.ImportFromEPSG( 4326 ) # And set it to WGS84 using the EPSG code

# Now for the target projection, Ordnance Survey's British National Grid
osng = osr.SpatialReference() # define the SpatialReference object
# In this case, we get the projection from a Proj4 string
osng.ImportFromEPSG( 27700 )
# or, if using the proj4 representation
osng.ImportFromProj4 ( "+proj=tmerc +lat_0=49 +lon_0=-2 " + \
                      "+k=0.9996012717 +x_0=400000 +y_0=-100000 " + \
                      "+ellps=airy +datum=OSGB36 +units=m +no_defs" )

# Now, we define a coordinate transformation object, *from* wgs84 *to* OSNG
tx = osr.CoordinateTransformation( wgs84, osng )
# We loop over the lines of park_data,
#           using the split method to split by newline characters
park_name, lon, lat = "Snowdonia national park", -3.898,      52.9

# Actually do the transformation using the TransformPoint method
osng_x, osng_y, osng_z = tx.TransformPoint ( lon, lat )
# Print out
print park_name, lon, lat, osng_x, osng_y

Snowdonia national park -3.898 52.9 272430.180112 335304.936823
```

You can test the result is reasonable by feeding the data for `osng_x` and `osng_y` in the OS own coordinate conversion website and making sure that the calculated longitude latitude pair is the same as the one you started with.

34.5 Reprojecting whole rasters

34.5.1 Using command line tools

The easiest way to reproject a raster file is to use GDAL's '`gdalwarp <http://www.gdal.org/gdalwarp.html>`' tool. As an example, let's say we want to reproject the landcover file from earlier on into latitude/longitude (WGS84):

```
!gdalwarp -of GTiff -t_srs "EPSG:4326" -ts 2400 2400 test_lc_h17v03.img lc_h17v03_wgs84.tif

Output dataset lc_h17v03_wgs84.tif exists,
but some commandline options were provided indicating a new dataset
should be created. Please delete existing dataset and run again.
```

We see here that the command takes a number of arguments:

1. `-of GTiff` is the output format (in this case GeoTIFF)
2. `-t_srs "EPSG:4326"` is the **to** projection (the **from** projection is already specified in the source dataset), in this case, lat/long WGS84, known by its **EPSG code**
3. `-ts 2400 2400` instructs `gdalwarp` to use an output of size 2400*2400.
4. `test_lc_h17v03.img` is the **input dataset**
5. `lc_h17v03_wgs84.tif` is the **output dataset**

Note that `gdalwarp` will reproject the data, and decide on the pixel size based on some considerations. This can result in the size of the raster changing. If you wanted to still keep the same raster size, we use the `-ts 2400 2400` option, or select an appropriate pixel size using `-tr xres yres` (note it has to be in the target projection, so degrees in this case). We can use `gdalinfo` to see what we've done.

```
!gdalinfo test_lc_h17v03.img
```

```
Driver: HFA/Erdas Imagine Images (.img)
Files: test_lc_h17v03.img
Size is 2400, 2400
Coordinate System is:
PROJCS["Sinusoidal",
    GEOGCS["GCS_Unknown datum based upon the custom spheroid",
        DATUM["Not_specified_based_on_custom_spheroid",
            SPHEROID["Custom_spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,6671703.117999999783933)
Pixel Size = (463.312716527916677,-463.312716527916507)
Corner Coordinates:
Upper Left  (-1111950.520, 6671703.118)  ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-1111950.520, 5559752.598)  ( 15d33'26.06"W, 50d 0' 0.00"N)
Upper Right  (      0.000, 6671703.118)  ( 0d 0' 0.01"E, 60d 0' 0.00"N)
Lower Right  (      0.000, 5559752.598)  ( 0d 0' 0.01"E, 50d 0' 0.00"N)
Center       ( -555975.260, 6115727.858)  ( 8d43' 2.04"W, 55d 0' 0.00"N)
Band 1 Block=64x64 Type=Byte, ColorInterp=Undefined
    Description = Layer_1
    Metadata:
        LAYER_TYPE=athematic
```

```
!gdalinfo lc_h17v03_wgs84.tif
```

```
Driver: GTiff/GeoTIFF
Files: lc_h17v03_wgs84.tif
Size is 2400, 2400
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (-19.99999994952233,59.99999994611805)
Pixel Size = (0.008333919248404,-0.004166959624202)
Metadata:
    AREA_OR_POINT=Area
Image Structure Metadata:
    INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (-20.0000000, 60.0000000)  ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-20.0000000, 49.9992969)  ( 20d 0' 0.00"W, 49d59'57.47"N)
Upper Right  ( 0.0014062, 60.0000000)  ( 0d 0' 5.06"E, 60d 0' 0.00"N)
Lower Right  ( 0.0014062, 49.9992969)  ( 0d 0' 5.06"E, 49d59'57.47"N)
Center       ( -9.9992969, 54.9996484)  ( 9d59'57.47"W, 54d59'58.73"N)
Band 1 Block=2400x3 Type=Byte, ColorInterp=Gray
    Description = Layer_1
```

```
Metadata:
  LAYER_TYPE=athematic
```

Let's see how different these two datasets are:

```
g = gdal.Open( "lc_h17v03_wgs84.tif" )
wgs84 = g.ReadAsArray()
g = gdal.Open("test_lc_h17v03.img")
modis = g.ReadAsArray()
plt.subplot( 1, 2, 1 )
plt.imshow( modis, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.subplot( 1, 2, 2 )
plt.imshow( wgs84, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )

<matplotlib.image.AxesImage at 0xe566950>
```

34.5.2 Reprojecting using the Python bindings

The previous section demonstrated how you can reproject raster files using command line tools. Sometimes, you might want to do this from inside a Python script. Ideally, you would have a python method that would perform the projection for you. GDAL allows this by defining in-memory raster files. These are normal GDAL datasets, but that don't exist on the filesystem, only in the computer's memory. They are a convenient "scratchpad" for quick intermediate calculations. GDAL also makes available a function, `gdal.ReprojectImage` that exposes most of the abilities of `gdalwarp`. We shall combine these two tricks to carry out the reprojection. As an example, we shall look at the case where the landcover data for the British Isles mentioned in the previous section needs to be reprojected to the Ordnance Survey National Grid, an appropriate projection for the UK.

The main complication comes from the need of `gdal.ReprojectImage` to operate on GDAL datasets. In the previous section, we saved the data to a GeoTIFF file, so this gives us a starting dataset. We still need to create the output dataset. This means that we need to define the geotransform and size of the output dataset before the projection is made. This entails gathering information on the extent of the original dataset, projecting it to the destination projection, and calculating the number of pixels and geotransform parameters from there. This is a (heavily commented) function that performs just that task:

```
def reproject_dataset( dataset, \
                      pixel_spacing=463., epsg_from=4326, epsg_to=27700 ):
    """
    A sample function to reproject and resample a GDAL dataset from within
    Python. The idea here is to reproject from one system to another, as well
    as to change the pixel size. The procedure is slightly long-winded, but
    goes like this:

    1. Set up the two Spatial Reference systems.
    2. Open the original dataset, and get the geotransform
    3. Calculate bounds of new geotransform by projecting the UL corners
    4. Calculate the number of pixels with the new projection & spacing
    5. Create an in-memory raster dataset
    6. Perform the projection
    """

    # Define the UK OSNG, see <http://spatialreference.org/ref/epsg/27700/>
    osng = osr.SpatialReference()
    osng.ImportFromEPSG(epsg_to)
    wgs84 = osr.SpatialReference()
    wgs84.ImportFromEPSG(epsg_from)
    tx = osr.CoordinateTransformation(wgs84, osng)
    # Up to here, all the projection have been defined, as well as a
    # transformation from the from to the to :
    # We now open the dataset
    g = gdal.Open(dataset)

    # Get the Geotransform vector
```

```
geo_t = g.GetGeoTransform ()
x_size = g.RasterXSize # Raster xsize
y_size = g.RasterYSize # Raster ysize

# Work out the boundaries of the new dataset in the target projection
(ulx, uly, ulz) = tx.TransformPoint( geo_t[0], geo_t[3])
(lrx, lry, lrz) = tx.TransformPoint( geo_t[0] + geo_t[1]*x_size, \
                                     geo_t[3] + geo_t[5]*y_size )
print ulx, uly, ulz
print lrx, lry, lrz
# See how using 27700 and WGS84 introduces a z-value!
# Now, we create an in-memory raster
mem_drv = gdal.GetDriverByName( 'MEM' )
# The size of the raster is given the new projection and pixel spacing
# Using the values we calculated above. Also, setting it to store one band
# and to use Float32 data type.
dest = mem_drv.Create('', int((lrx - ulx)/pixel_spacing), \
                      int((uly - lry)/pixel_spacing), 1, gdal.GDT_Float32)
# Calculate the new geotransform
new_geo = ( ulx, pixel_spacing, geo_t[2], \
            uly, geo_t[4], -pixel_spacing )
# Set the geotransform
dest.SetGeoTransform( new_geo )
dest.SetProjection( osng.ExportToWkt() )
# Perform the projection/resampling
res = gdal.ReprojectImage( g, dest, \
                           wgs84.ExportToWkt(), osng.ExportToWkt(), \
                           gdal.GRA_Bilinear )
return dest
```

The function returns a GDAL in-memory file object, where you can `ReadAsArray` etc. As it stands, `reproject_dataset` does not write to disk. However, we can save the in-memory raster to any format supported by GDAL very conveniently by making a copy of the dataset. This takes a few lines of code:

```
# Do in memory reprojection
reprojected_dataset = reproject_dataset ( "lc_h17v03_wgs84.tif" )
# Output driver, as before
driver = gdal.GetDriverByName ( "GTiff" )
# Create a copy of the in memory dataset `reprojected_dataset`, and save it
dst_ds = driver.CreateCopy( "test_lc_h17v03_OSNG.tif", reprojected_dataset, 0 )
dst_ds = None # Flush the dataset to disk

-595472.202548 1261034.77555 -55.2326775854
543532.18509 12933.1712342 -43.993001678
```

Let's see how the different projections look like by plotting them side by side

```
plt.subplot ( 1, 3, 1 )
plt.title ( "MODIS sinusoidal" )
plt.imshow ( modis, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.subplot ( 1, 3, 2 )
plt.imshow ( wgs84, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
g = gdal.Open("test_lc_h17v03_OSNG.tif" )
osng = g.ReadAsArray()
plt.title ( "WGS84, Lat/Long" )
plt.subplot ( 1, 3, 3 )
plt.imshow ( osng, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.title("OSNG")

<matplotlib.text.Text at 0x2b9eac0b4ed0>
```

GDAL AND RELATED CODES AND LIBRARIES

GDAL does not always come with your Python distribution.

It can sometimes be a little tricky to install.

You do not need it for this class as you can always run things on the UCL Geography system via ssh, but might like to install it, so we'll try to keep some notes up to date here on how to do that (and typical problems you encounter).

Since it relies on libraries not written in Python, you need to get these libraries installed. You may also want to install the GDAL tools.

There are many ways of doing this that you can find by searching on the internet.

A good place to start is trac.osgeo.org.

Here, we try to give some practical guidance. (Note: I've not been able to test this under windows, so [give me some feedback](#) about what works or doesn't).

35.1 OS X

35.1.1 Pre-compiled GDAL utilities

There are several places you can download pre-compiled versions of the GDAL utilities from.

One example is <http://www.kyngchaos.com/software/frameworks>. If you install GDAL complete from there, you should find the GDAL utilities in /Library/Frameworks/GDAL.framework/Programs:

```
!ls /Library/Frameworks/GDAL.framework/Programs
```

```
[31mepsg_tr.py [m[m [31mgdal_fillnodata.py [m[m [31mgdalchksum.py [m[m [31mgdalindex [m[m
[31mesri2wkt.py [m[m [31mgdal_grid[m[m [31mgdalem[m[m [31mgdaltransform[m[m
[31mgcps2vec.py [m[m [31mgdal_merge.py [m[m [31mgdalenhance[m[m [31mgdaltransformer[m[m
[31mgcps2wld.py [m[m [31mgdal_polygonize.py [m[m [31mgdalident.py [m[m [31mgdalwarp[m[m
[31mgdal-config[m[m [31mgdal_proximity.py [m[m [31mgdalimport.py [m[m [31mmkgraticule[m[m
[31mgdal2tiles.py [m[m [31mgdal_rasterize[m[m [31mgdalinfo[m[m [31mnearblack[m[m
[31mgdal2xyz.py [m[m [31mgdal_retile.py [m[m [31mgdallocationinfo[m[m [31mogr2ogr[m[m
[31mgdal_auth.py [m[m [31mgdal_sieve.py [m[m [31mgdalmanage[m[m [31mogrinfo[m[m
[31mgdal_calc.py [m[m [31mgdal_translate[m[m [31mgdalmove.py [m[m [31mogrindex[m[m
[31mgdal_contour[m[m [31mgdaladdo[m[m [31mgdalserver[m[m [31mrct2rgb.py [m[m
[31mgdal_edit.py [m[m [31mgdalbuildvrt[m[m [31mgdalsrsinfo[m[m [31mrgb2pct.py [m[m
[31mgdalindex[m[m [31mgdaltransformer[m[m [31mgdaltransformer[m[m [31mtestepsg[m[m
```

If you want this in your path, put the following line at the bottom of the file `~/.bashrc` (if using bash):

```
export PATH=/Library/Frameworks/GDAL.framework/Programs:$PATH
```

or, if using tcsh or csh, put this at the end of your file `~/.cshrc`:

```
setenv PATH "/Library/Frameworks/GDAL.framework/Programs:${PATH}"
```

or

```
set path = (/Library/Frameworks/GDAL.framework/Programs $path)
```

and then type:

```
source ~/.bashrc
```

or

```
source ~/.cshrc
```

as appropriate (or open new shells).

You can test that you can access the gdal commands with e.g.:

```
!which gdalinfo
```

```
!gdalinfo --version
```

```
/Library/Frameworks/GDAL.framework/Programs/gdalinfo
```

```
GDAL 1.10.1, released 2013/08/26
```

35.1.2 Compiling source (use homebrew)

If you want to compile GDAL yourself for OS X, it is probably easiest to install the [Homebrew](#) software, type:

```
rehash
```

at the unix prompt (to update your path) and follow any instructions it gives you to sort out any conflicts you might have, then simply:

```
brew install gdal
```

WORKING WITH VECTOR DATA: OGR

36.1 Vector data

Sometimes, geospatial data is acquired and recorded for particular geometric objects such as polygons or lines. An example is a road layout, where each road is represented as a geometric object (a line, with points given in a geographical projection), with a number of added *features* associated with it, such as the road name, whether it is a toll road, or whether it is dual-carriageway, etc. This data is quite different to a raster, where the entire scene is tessellated into pixels, and each pixel holds a value (or an array of value in the case of multiband rasterfiles).

If you are familiar with databases, vector files are effectively a database, where one of the fields is a geometry object (a line in our previous road example, or a polygon if you consider a cadastral system). We can thus select different records by writing queries on the features. Some of these queries might be spatial (e.g. check whether a point is inside a particular country polygon).

The most common format for vector data is the **ESRI Shapfile**, which is a multifile format (i.e., several files are needed in order to access the data). We'll start by getting hold of a shapefile that contains the countries of the world as polygons, together with information on country name, capital name, population, etc. The file is available here.

Figure 36.1: World

We will download the file with `wget` (or `curl` if you want to), and uncompress it using `unzip` in the shell:

```
# Downloads the data using wget
!wget http://aprsworld.net/gisdata/world/world.zip
# or if you want to use curl...
#! curl http://aprsworld.net/gisdata/world/world.zip -o world.zip
!unzip -o -x world.zip

--2013-10-22 15:47:43-- http://aprsworld.net/gisdata/world/world.zip
Resolving aprsworld.net... 72.251.203.219
Connecting to aprsworld.net|72.251.203.219|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3436277 (3.3M) [application/zip]
Saving to: `world.zip.1'

100%[=====] 3,436,277 3.28M/s in 1.0s

2013-10-22 15:47:45 (3.28 MB/s) - `world.zip.1' saved [3436277/3436277]

Archive: world.zip
  inflating: world.dbf
  inflating: world.shp
  inflating: world.shx
```

We need to import `ogr`, and then open the file. As with GDAL, we get a handler to the file, (`g` in this case). OGR files can have different layers, although Shapefiles only have one. We need to select the layer using `GetLayer(0)` (selecting the first layer).

```
from osgeo import ogr

g = ogr.Open( "world.shp" )
layer = g.GetLayer( 0 )
```

In order to see a field (the field NAME) we can loop over the features in the layer, and use the GetField('NAME') method. We'll only do ten features here:

```
n_feat = 0
for feat in layer:

    print feat.GetField('NAME')

    n_feat += 1
    if n_feat == 10:
        break
```

```
GUATEMALA
BOLIVIA
PARAGUAY
URUGUAY
SURINAME
FRENCH GUIANA
WESTERN SAHARA
GAMBIA
MOROCCO
MALI
```

If you wanted to see the different layers, we could do this using:

```
layerDefinition = layer.GetLayerDefn()
```

```
for i in range(layerDefinition.GetFieldCount()):
    print "Field %d: %s" % ( i+1, layerDefinition.GetFieldDefn(i).GetName() )

Field 1: NAME
Field 2: CAPITAL
Field 3: APPROX
Field 4: AREA
Field 5: SOURCETHM
```

Each feature, in addition to the fields shown above, will have a Geometry field. We get a handle to this using the GetGeometryRef() method. Geometries have many methods, such as ExportToKML() to export to KML (Google Maps/Earth format):

```
the_geometry = feat.GetGeometryRef()
the_geometry.ExportToKML()
```

```
'<Polygon><outerBoundaryIs><LinearRing><coordinates>-12.0443,14.669667 -11.87845,14.8252 -11.7645
```

Many of the methods that don't start with `_` are interesting. Let's see what these are. typically, the interesting methods start with an upper case letter, so we'll only show those:

```
for m in dir( the_geometry ):
    if m[0].isupper():
        print m
```

```
AddGeometry
AddGeometryDirectly
AddPoint
AddPoint_2D
Area
AssignSpatialReference
```

Boundary
Buffer
Centroid
Clone
CloseRings
Contains
ConvexHull
Crosses
Destroy
Difference
Disjoint
Distance
Empty
Equal
Equals
ExportToGML
ExportToJson
ExportToKML
ExportToWkb
ExportToWkt
FlattenTo2D
GetArea
GetBoundary
GetCoordinateDimension
GetDimension
GetEnvelope
GetEnvelope3D
GetGeometryCount
GetGeometryName
GetGeometryRef
GetGeometryType
GetPoint
GetPointCount
GetPoint_2D
GetPoints
GetSpatialReference
GetX
GetY
GetZ
Intersect
Intersection
Intersects
IsEmpty
IsRing
IsSimple
IsValid
Length
Overlaps
PointOnSurface
Segmentize
SetCoordinateDimension
SetPoint
SetPoint_2D
Simplify
SimplifyPreserveTopology
SymDifference
SymmetricDifference
Touches
Transform
TransformTo
Union
UnionCascaded
Within

WkbSize

You'll notice that many of these mechanisms e.g. Overlaps or Touches are effectively geoprocessing operations (they operate on geometries and return True if one geometry overlaps or touches, respectively, the other). Other operations, such as Buffer return a buffered version of the same geometry. This allows you to actually do fairly complicated geoprocessing operations with OGR. However, if you want to do geoprocessing in earnest, you should really be using [Shapely](#).

A particularly useful webpage for this section is [available in the OGR cookbook](#). Have a look through that if you want more in depth information.

36.2 Selecting attributes and/or data extents

OGR provides an easy way to select attributes on a given layer. This is done using a SQL-like syntax (you can read more on [OGR's SQL subset here](#)). The main point is that the *attribute filter* is applied to a complete layer. For example, let's say that we want to select only countries with a population (field APPROX) larger than 90 000 000 inhabitants:

```
g = ogr.Open ( "world.shp" )
lyr = g.GetLayer( 0 )
lyr.SetAttributeFilter ( "APPROX > 90000000" )
for feat in lyr:
    print feat.GetFieldAsString ( "NAME" ) + " has %d inhabitants" % \
        feat.GetFieldAsInteger("APPROX")

PAKISTAN has 123490000 inhabitants
JAPAN has 124710000 inhabitants
RUSSIAN FEDERATION has 150500000 inhabitants
INDIA has 873850000 inhabitants
BANGLADESH has 120850000 inhabitants
BRAZIL has 159630000 inhabitants
NIGERIA has 91700000 inhabitants
CHINA has 1179030000 inhabitants
INDONESIA has 186180000 inhabitants
JOHNSTON ATOLL has 256420000 inhabitants
KINGMAN REEF - PALMYRA ATOLL has 256420000 inhabitants
UNITED STATES has 256420000 inhabitants
```

So we get a list of populous countries (note that Johnston Atoll and Palmyra are part of the US, and report the sample population as the US!)

An additional way to filter the data is by geographical extent. Let's say we wanted a list of all the countries in (broadly speaking) Europe, *i.e.* a geographical extent in longitude from 14W to 37E, and in latitude from 72N to 38N. We can use SetSpatialFilterRect to do this:

```
g = ogr.Open ( "world.shp" )
lyr = g.GetLayer( 0 )
lyr.SetSpatialFilterRect ( -14, 37, 38, 72 )
for feat in lyr:
    print feat.GetFieldAsString ( "NAME" ) + " ---- " + feat.GetFieldAsString ( "CAPITAL" )

ALGERIA ---- ALGIERS
BELGIUM ---- BRUSSELS
LUXEMBOURG ---- LUXEMBOURG
SAN MARINO ---- SAN MARINO
AUSTRIA ---- VIENNA
CZECH REPUBLIC ---- PRAGUE
SLOVENIA ---- LJUBLJANA
HUNGARY ---- BUDAPEST
SLOVAKIA ---- BRATISLAVA
YUGOSLAVIA ---- BELGRADE [BEOGRADE]
```

BOSNIA AND HERZEGOVINA ---- SARAJEVO
 ALBANIA ---- TIRANE
 MACEDONIA, THE FORMER YUGOSLAV REPUBLIC ---- SKOPJE
 LITHUANIA ---- VILNIUS
 LATVIA ---- RIGA
 BULGARIA ---- SOFIA
 BELARUS ---- MINSK
 MOLDOVA, REPUBLIC OF ---- KISHINEV
 IRELAND ---- DUBLIN
 ICELAND ---- REYKJAVIK
 SPAIN ---- MADRID
 SWEDEN ---- STOCKHOLM
 FINLAND ---- HELSINKI
 TURKEY ---- ANKARA
 RUSSIAN FEDERATION ---- MOSCOW
 GREECE ---- ATHENS
 PORTUGAL ---- LISBON
 POLAND ---- WARSAW
 NORWAY ---- OSLO
 GERMANY ---- BERLIN
 ESTONIA ---- TALLINN
 TUNISIA ---- TUNIS
 CROATIA ---- ZAGREB
 ROMANIA ---- BUCURESTI
 UKRAINE ---- KIEV
 NETHERLANDS ---- AMSTERDAM
 JERSEY ---- SAINT HELIER
 GUERNSEY ---- SAINT PETER PORT
 FAROE ISLANDS ---- TORSHAVN
 DENMARK ---- COPENHAGEN
 MONACO ---- MONACO
 ANDORRA ---- ANDORRA LA VELLA
 LIECHTENSTEIN ---- VADUZ
 SWITZERLAND ---- BERN
 ISLE OF MAN ---- DOUGLAS
 UNITED KINGDOM ---- LONDON
 FRANCE ---- PARIS
 VATICAN CITY (HOLY SEE) ---- VATICAN CITY
 ITALY ---- ROME

36.3 Saving a vector file

Saving a vector file using OGR requires a number of steps:

1. Definition of the format
2. Definition of the layer projection and geometry type (e.g. lines, polygons...)
3. Definition of the data type of the different fields
4. Creation of a feature, population of the different fields, and setting a geometry
5. Addition of the feature to the layer
6. Destruction of the feature

This appears quite involved, but let's see how this works. Note that when you generate a new vector file, OGR will fail if the file already exists. You might want to use `os.remove()` to get rid of the file if it exists.

Let's see how this is done with an example which is a snippet that creates a GeoJSON file with the location of the different national parks. GeoJSON is a nice geographic format, and [github allows you to display it easily as a map](#).

```
# National park information, separated by TABs
import os
from osgeo import ogr,osr

parks = """Dartmoor national park\t-3.904\t50.58
New forest national park\t-1.595\t50.86
Exmoor national park\t-3.651\t51.14
Pembrokeshire coast national park\t-4.694\t51.64
Brecon beacons national park\t-3.432\t51.88
Pembrokeshire coast national park\t-4.79\t51.99
Norfolk and suffolk broads\t1.569\t52.62
Snowdonia national park\t-3.898\t52.9
Peak district national park\t-1.802\t53.3
Yorkshire dales national park\t-2.157\t54.23
North yorkshire moors national park\t-0.8855\t54.37
Lake district national park\t-3.084\t54.47
Galloway forest park\t-4.171\t54.87
Northumberland national park\t-2.228\t55.28
Loch lomond and the trossachs national park\t-4.593\t56.24
Tay forest park\t-4.025\t56.59
Cairngorms national park\t-3.545\t57.08"""

# See if the file exists from a previous run of this snippet
if os.path.exists ( "parks.json" ):
    # It does exist, so remove it
    os.remove ( "parks.json" )

# We need the output projection to be set to Lat/Long
latlong = osr.SpatialReference()
latlong.ImportFromEPSG( 4326 )

# Invoke the GeoJSON driver
drv = ogr.GetDriverByName( 'GeoJSON' )
# This is the output filename
dst_ds = drv.CreateDataSource( 'parks.json' )
# This is a single layer dataset. The layer needs to be of points
# and needs to have the WGS84 projection, which we defined above
dst_layer = dst_ds.CreateLayer('', srs =latlong , \
                                geom_type=ogr.wkbPoint )

# We just need a field with the Park's name, and its type is a String
field_defn=ogr.FieldDefn( 'name', ogr.OFTString )
dst_layer.CreateField( field_defn )

# Algorithm is as follows:
# 1. Loop over lines
# 2. Split line into park name, longitude, latitude
# 3. Create WKT of the point
# 4. Set the attribute name to name of park
# 5. Clean up

for park_id, line in enumerate( parks.split( "\n" ) ):
    # Get the relevant information
    park_name, lon, lat = line.split("\t")
    # Create a geopraphical representation of the current park
    wkt = "POINT ( %f %f )" % ( float(lon), float(lat) )
    # Create a feature, using the attributes/fields that are
    # required for this layer
    feat = ogr.Feature(feature_def=dst_layer.GetLayerDefn())
    # Feed the WKT into a geometry
    p = ogr.CreateGeometryFromWkt( wkt )
```

```

# Feed the geometry into a WKT
feat.SetGeometryDirectly( p )
# Set the name field to its value
feat.SetField( "name", park_name )
# Attach the feature to the layer
dst_layer.CreateFeature( feat )
# Clean up
feat.Destroy()

# Close file
dst_ds = None

```

You can see the result of this on [github](#).

Additionally, note that if we had defined a coordinate transformation as in the raster session, we could apply this transformation to an OGR geometry entity (in the snippet above, `p` would be such), and it would be reprojected.

Exercise Modify the above snippet to output a GeoJSON file for the Peak District National Park, whose UTM30N (EPSG code: 32630) co-ordinates are 577659, 5911841.

36.4 Rasterising

A very frequent problem one finds is how to mask out an area in a raster file that is defined as polygon in a shapefile. For example, if you have a raster of the world's population density, and you want to extract all the pixels that belong to one particular country, how do you go about that? One way around this is to *rasterise* the polygon(s), which translates into “burning” pixels that fall within the polygon with a number, resulting in a mask.

The way to do this is to use GDAL’s `RasterizeLayer` method. The method takes a handle to a GDAL dataset (one that you create yourself, with the right projection and geotransform, as you’ve seen above), and a OGR layer. The syntax for `RasterizeLayer` is

```
err = gdal.RasterizeLayer ( raster_ds, [raster_band_no], ogr_layer, burn_values=[burn_val] )
```

where `raster_ds` is the GDAL raster datasource (note that it needs to be georeferenced, *i.e.* it requires projection and geotransform), `raster_band_no` is the band of the GDAL dataset where we want to burn pixels, `ogr_layer` is the vector layer object, and `burn_val` is the value that we want to burn.

Let’s use `gdal.RasterizeLayer` in conjunction with all that we have covered above. Say we want to create a mask that only selects the UK or Ireland in `world.shp`, and we want to apply this mask to the MODIS landcover product that we used in the GDAL session (`h17v03.tif`), file `lc_h17v03.tif`. We find that in this case, `world.shp` is in longitude/latitude, and the MODIS data is in the MODIS projection, so we will reproject the vector data to match the MODIS data (so the latter is not interpolated and artifacts introduced). To make this efficient and avoid saving to disk, we shall use *in-memory vector and rasters*, and we will output a numpy array as our mask. Note then the steps:

1. Create the projection conversion object (as for GDAL before)
2. Create an in memory **raster** dataset to store the mask, using `lc_h17v03.tif` as a reference for geotransforms, array size and projection.
3. Create an in memory **vector** dataset to hold the features that will be reprojected
4. Open `world.shp` and apply an `AttributeFilter` to select a country
5. Select a geometry from `world.shp`, project it and store it in the destination in memory vector layer
6. Once this is done, use `gdal.RasterizeLayer` with both in-memory raster and vector datasets
7. Read the in memory raster into an array

This is a particularly good exercise that will stress all that we have learned so far.

```
from osgeo import ogr,osr
import gdal

reference_filename = "lc_h17v03.tif"
target_vector_file = "world.shp"
attribute_filter = "NAME = 'IRELAND'"
burn_value = 1

# First, open the file that we'll be taking as a reference
# We will need to gleam the size in pixels, as well as projection
# and geotransform.

g = gdal.Open( reference_filename )

# We now create an in-memory raster, with the appropriate dimensions
drv = gdal.GetDriverByName('MEM')
target_ds = drv.Create('', g.RasterXSize, g.RasterYSize, 1, gdal.GDT_Byte)
target_ds.SetGeoTransform( g.GetGeoTransform() )

# We set up a transform object as we saw in the previous notebook.
# This goes from WGS84 to the projection in the reference datasets

wgs84 = osr.SpatialReference() # Define a SpatialReference object
wgs84.ImportFromEPSG( 4326 ) # And set it to WGS84 using the EPSG code

# Now for the target projection, Ordnance Survey's British National Grid
to_proj = osr.SpatialReference() # define the SpatialReference object
# In this case, we get the projection from a Proj4 string

# or, if using the proj4 representation
to_proj.ImportFromWkt( g.GetProjectionRef() )
target_ds.SetProjection ( to_proj.ExportToWkt() )
# Now, we define a coordinate transformation object, *from* wgs84 *to* OSNG
tx = osr.CoordinateTransformation( wgs84, to_proj )

# We define an output in-memory OGR dataset
# You could also do select a driver for an eg "ESRI Shapefile" here
# and give it a sexier name than out!

drv = ogr.GetDriverByName( 'Memory' )
dst_ds = drv.CreateDataSource( 'out' )
# This is a single layer dataset. The layer needs to be of polygons
# and needs to have the target files' projection
dst_layer = dst_ds.CreateLayer('', srs = to_proj, geom_type=ogr.wkbPolygon )

# Open the original shapefile, get the first layer, and filter by attribute
vector_ds = ogr.Open( target_vector_file )
lyr = vector_ds.GetLayer( 0 )
lyr.SetAttributeFilter( attribute_filter )

# Get a field definition from the original vector file.
# We don't need much more detail here
feature = lyr.GetFeature(0)
field = feature.GetFieldDefnRef( 0 )
# Apply the field definition from the original to the output
dst_layer.CreateField( field )
feature_defn = dst_layer.GetLayerDefn()
# Reset the original layer so we can read all features
lyr.ResetReading()
for feat in lyr:
    # For each feature, get the geometry
    geom = feat.GetGeometryRef()
```

```

# transform it to the reference projection
geom.Transform ( tx )
# Create an output feature
out_geom = ogr.Feature ( feature_defn )
# Set the geometry to be the reprojected/transformed geometry
out_geom.SetGeometry ( geom )
# Add the feature with its geometry to the output yaer
dst_layer.CreateFeature(out_geom)
# Clear things up
out_geom.Destroy
geom.Destroy
# Done adding geometries
# Reset the output layer to the 0th geometry
dst_layer.ResetReading()

# Now, we rastertize the output vector in-memory file
# into the in-memory output raster file

err = gdal.RasterizeLayer(target_ds, [1], dst_layer,
                           burn_values=[burn_value])
if err != 0:
    print("error:", err)

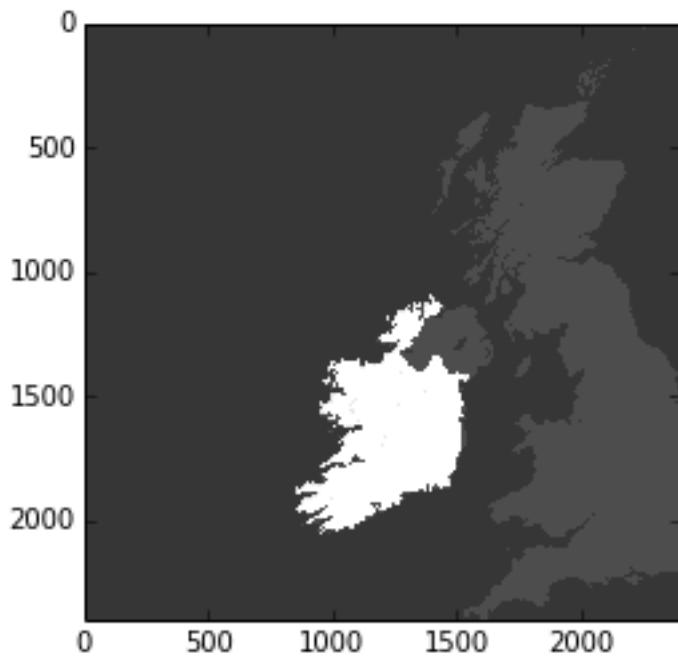
# Read the data from the raster, this is your mask
data = target_ds.ReadAsArray()

# Plotting to see whether this makes sense.

ndata = g.ReadAsArray()
plt.imshow ( ndata, interpolation='nearest', cmap=plt.cm.gray, vmin=0, vmax=1, alpha=0.3 )
plt.hold ( True )

plt.imshow ( data, interpolation='nearest', cmap=plt.cm.gray, alpha=0.7 )
plt.grid ( False )
plt.show()

```



36.5 Using matplotlib to plot geometries

Using matplotlib to plot geometries from OGR can be quite tedious. Here's an example of plotting a map of Angola from the `world.shp`. In the same vein of recommending Shapely and Fiona above for serious geoprocessing of vector data, you are encouraged to use `descartes` for plotting vector data!

```
import matplotlib.path as mpath
import matplotlib.patches as mpatches

# Extract first layer of features from shapefile using OGR
ds = ogr.Open('world.shp')
lyr = ds.GetLayer(0)

# Prepare figure
plt.ioff()
plt.subplot(1,1,1)
ax = plt.gca()

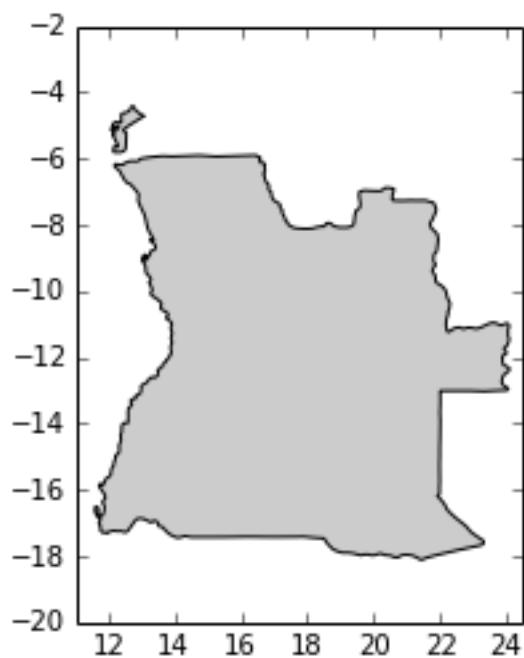
paths = []
lyr.ResetReading()

lyr.SetAttributeFilter ( " NAME = 'ANGOLA' " )
ax.set_xlim(11, 24.5)
ax.set_ylim(-20, -2)
# Read all features in layer and store as paths

for feat in lyr:

    for geom in feat.GetGeometryRef():
        envelope = np.array(geom.GetEnvelope())
        # check if geom is polygon
        if geom.GetGeometryType() == ogr.wkbPolygon:
            codes = []
            all_x = []
            all_y = []
            for i in range(geom.GetGeometryCount()):
                # Read ring geometry and create path
                r = geom.GetGeometryRef(i)
                x = [r.GetX(j) for j in range(r.GetPointCount())]
                y = [r.GetY(j) for j in range(r.GetPointCount())]
                # skip boundary between individual rings
                codes += [mpath.Path.MOVETO] + \
                          (len(x)-1)*[mpath.Path.LINETO]
                all_x += x
                all_y += y
            path = mpath.Path(np.column_stack((all_x,all_y)), codes)
            paths.append(path)
    # Add paths as patches to axes
    for path in paths:
        patch = mpatches.PathPatch(path, \
                                    facecolor='0.8', edgecolor='black')
        ax.add_patch(patch)

ax.set_aspect(1.0)
plt.show()
```



A4.1. GETTING MODIS URLs

Access to MODIS data is now through `http`, which means that previous methods using `ftp` no longer operate. In some ways, this complicates automatic download (also, download seems now to be throttled, which means it takes longer to access the data).

That said, you can of course still easily order data through NASA tools such as `reverb`.

Some tools have been developed to allow automated access to MODIS products from Python, such as `get_modis`, but here, we will demonstrate how you can do it yourself, semi-automatically.

We will see that a large part of the overhead and complexity is negotiating the directory structure.

We have provided a shell programme '`zat <files/python/zat>`' that will produce a list of urls of the MODIS products on the USGS server (use `zat > 'urls.txt <files/data/robot.txt>`'), which you would probably find more convenient than this section.

So, only go through section A1 if you are particularly interested in trawling directories with `http` ...

Once we have a full list of the urls of the hdf files that we want, life is much simpler. Such a list of urls is *exactly* what `reverb` supplies you with.

37.1 A4.1.1 Identify the server and directory structure

First, you need to identify which datasets you want. You should explore the data products e.g. through `reverb` to do this.

If you go through the ordering system for one tile of these products, you can get the information you need for further data download. When you come to order the data, it will give you a download file.

As an example:

- MODIS LAI/fAPAR for Terra and Aqua 8 day composite for 17 Jan 2013 for tile h18v03

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2013.01.17/MCD15A2.A2013017.h18v03.005.201302606505

<http://e4ftl01.cr.usgs.gov/WORKING/BRWS/Browse.001/2013.01.26/BROWSE.MCD15A2.A2013017.h18v03.005.201302606505>

<http://e4ftl01.cr.usgs.gov/WORKING/BRWS/Browse.001/2013.01.26/BROWSE.MCD15A2.A2013017.h18v03.005.201302606505>

37.2 A4.1.2 Identify the available dates

From this, we see that the server is `e4ftl01.cr.usgs.gov`, that the `hdf` data (the spatial dataset we want) for the product `MCD15A2` version `005` is in the directory `MODIS_Composites/MOTA/MCD15A2.005`.

Below that, we have the date and then the filename.

Let's use `urllib2` to explore this:

```
import urllib2
url_base = 'http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005'
response = urllib2.urlopen(url_base)
html = response.read()

# print the first 30 lines
html.split('\n')[:30]
```

This is an html directory listing. We can identify the directories as lines that contain [DIR].

We can use `find` to identify lines that have this field:

```
dirs = []
for line in html.split('\n'):
    if line.find('[DIRS]'):
        dirs.append(line)

# or more succinctly
dirs = [line for line in html.split('\n') if line.find('[DIR]') != -1]

dirs[:3]
```

We notice that the first such line is the directory listing information, so, what we really want is:

```
dirs = [line for line in html.split('\n') if line.find('[DIR]') != -1][1:]

dirs[:3]
```

The subdirectory name is just after the field `href=""`:

```
print dirs[1]

print dirs[1].split('href="')[1]

print dirs[1].split('href="')[1].split('/>')[0]
```

So, in this case, we can get the subdirectory names with:

```
dirs = [line.split('href="')[1].split('/>')[0] for line in html.split('\n') if line.find('[DIR]') != -1][1:]

# print the first 10
dirs[:10]
```

The pattern is YYYY.MM.DD. So we could split these as we go along. It would be convenient to have this as a numpy array:

```
dirs = np.array([line.split('href="')[1].split('/>')[0].split('.') \
                for line in html.split('\n') if line.find('[DIR]') != -1][1:])

dirs[:10]

all_years = np.sort(np.unique(dirs[:,0]))
all_months = np.sort(np.unique(dirs[:,1]))
all_doy = np.sort(np.unique(dirs[:,2]))
```

years,months,doy

37.3 A4.1.3 Identify the datasets

We know the full url is of the form:

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2013.01.17/MCD15A2.A2013017

Simplifying what we did above:

```
import urllib2
url_base = 'http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005'
response = urllib2.urlopen(url_base)
dirs = np.array([line.split('href="')[1].split('/>')[0] for line in html.split('\n') if line.find('a href="') != -1])

years = np.array([i.split('.')[0] for i in dirs])
# year mask
year = '2012'
mask = (year == years)
sub_dirs = dirs[mask]
print sub_dirs

# test with first one
this_date = sub_dirs[0]

url_date = url_base + '/' + this_date
print url_date
response1 = urllib2.urlopen(url_date)
html1 = response1.read()

# print the first 21 lines
html1.split('\n')[:21]
```

We note that the directory contains data for all tiles.

Lets filter only lines that have the tile we want in:

```
tile = 'h18v03'
lines = [line for line in html1.split('\n') if line.find(tile) != -1]

lines
```

We want the .hdf file, so refine the filter:

```
tile = 'h18v03'
hdf_lines = [i for i in [line for line in html1.split('\n') \
                        if line.find(tile) != -1] if i.find('.hdf') != -1]

hdf_lines
```

Now split this to get the filename we want:

```
hdf_lines[0].split('<a href="')[1]

hdf_lines[0].split('<a href="')[1].split('">')[0]
```

So, putting all of that together:

```
tile = 'h18v03'
hdf_lines = [i for i in [line for line in html1.split('\n') \
                        if line.find(tile) != -1] if i.find('.hdf') != -1]
hdf_file = hdf_lines[0].split('<a href="')[1].split('">')[0]
```

37.4 A4.1.4 Some code for MODIS LAI filenames for a year

The http access is quite slow, so this may take some minutes to run.

```
year = '2012'
tile = 'h17v03'
```

```
hdf_files = []

import urllib2

# base URL for the product
url_base = 'http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005'

response = urllib2.urlopen(url_base)
html = response.read()

dirs = np.array([line.split('href="')[1].split('/>')[0] for line in html.split('\n') if line.find('a href="') != -1])

# identify years
years = np.array([i.split('.')[0] for i in dirs])
# year mask
mask = (year == years)
sub_dirs = dirs[mask]

for this_date in sub_dirs:
    url_date = url_base + '/' + this_date
    print url_date
    response1 = urllib2.urlopen(url_date)
    html1 = response1.read()
    hdf_lines = [i for i in [line for line in html1.split('\n') \
                             if line.find(tile) != -1] if i.find('.hdf') != -1]
    hdf_file = url_date + '/' + hdf_lines[0].split('<a href="')[1].split('>')[0]
    hdf_files.append(hdf_file+'\n')

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.01
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.09
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.17
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.25
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.02
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.10
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.18
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.26
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.05
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.13
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.21
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.29
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.06
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.14
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.22
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.30
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.05.08
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.05.16
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.05.24
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.01
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.09
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.17
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.25
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.03
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.11
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.19
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.27
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.04
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.12
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.20
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.28
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.05
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.13
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.21
```

```
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.29
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.07
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.15
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.23
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.31
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.11.08
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.11.16
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.11.24
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.02
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.10
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.18
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.26
```

In case download fails later, lets save this list hdf_files.

```
f = open('files/data/lai_list.txt', 'w')
f.writelines(hdf_files)
f.close()
```

37.5 A4.1.5 Pull Data from url

This part is actually faster than doing all of that messing around with directories.

You don't really want to have to do too much of the directory exploration, so it is *probably* a good idea to just periodically scan the whole structure and store that in a local file. You can then parse the local file much more easily (that is what we do in the main part of the class).

This is achieved for instance with the shell `zat <files/python/zat>` (use zat > `urls.txt <files/data/robot.txt>`) if you want to do an update, or just use the existing url file.

```
import urllib2

f = open('files/data/lai_list.txt', 'r')
hdf_files = f.readlines()
f.close()

for url in hdf_files:
    url = url.strip()
    print url
    response = urllib2.urlopen(url.strip())
    ofile = 'files/data/' + url.split('/')[-1]
    f = open(ofile, 'w')
    f.write(response.read())
    f.close()

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.01/MCD15A2.A2012001.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.09/MCD15A2.A2012009.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.17/MCD15A2.A2012017.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.01.25/MCD15A2.A2012025.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.02/MCD15A2.A2012033.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.10/MCD15A2.A2012041.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.18/MCD15A2.A2012049.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.02.26/MCD15A2.A2012057.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.05/MCD15A2.A2012065.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.13/MCD15A2.A2012073.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.21/MCD15A2.A2012081.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.03.29/MCD15A2.A2012089.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.06/MCD15A2.A2012097.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.14/MCD15A2.A2012105.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.22/MCD15A2.A2012113.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.04.30/MCD15A2.A2012121.h17v03.00
```

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.05.08/MCD15A2.A2012129.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.05.16/MCD15A2.A2012137.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.05.24/MCD15A2.A2012145.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.01/MCD15A2.A2012153.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.09/MCD15A2.A2012161.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.17/MCD15A2.A2012169.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.06.25/MCD15A2.A2012177.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.03/MCD15A2.A2012185.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.11/MCD15A2.A2012193.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.19/MCD15A2.A2012201.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.07.27/MCD15A2.A2012209.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.04/MCD15A2.A2012217.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.12/MCD15A2.A2012225.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.20/MCD15A2.A2012233.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.08.28/MCD15A2.A2012241.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.05/MCD15A2.A2012249.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.13/MCD15A2.A2012257.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.21/MCD15A2.A2012265.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.09.29/MCD15A2.A2012273.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.07/MCD15A2.A2012281.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.15/MCD15A2.A2012289.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.23/MCD15A2.A2012297.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.10.31/MCD15A2.A2012305.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.11.08/MCD15A2.A2012313.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.11.16/MCD15A2.A2012321.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.11.24/MCD15A2.A2012329.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.02/MCD15A2.A2012337.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.10/MCD15A2.A2012345.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.18/MCD15A2.A2012353.h17v03.000
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2012.12.26/MCD15A2.A2012361.h17v03.000

CHAPTER
THIRTYEIGHT

A4.2 GDAL TOOLS AND HDF FORMAT

HDF(Hierarchical Data Format) and HDF-EOS are common formats for EO data so you need to have some idea how to use and manipulate them.

A hierarchical data format is essentially a format that ‘packs’ together various aspects of a dataset (metadata, raster data etc.) into a binary file. There are many tools for manipulating and reading HDF in python, but we will use one of the more generic tools, `gdal` here.

When using HDF files, we need to have some idea of the structure of the contents, although you can clearly explore that yourself in an interactive session. MODIS products have extensive information available to help you interpret the datasets, for example the MODIS LAI/fAPAR product [MOD15A2](#). We will use this as an example to explore a dataset.

You will need access to the file `files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf <files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf>`_, which you might access from the [MODIS Land Products site](#)

Before going into the Python coding for GDAL, it is worthwhile looking over some of the tools that are provided with GDAL and that can be run from the shell. In particular, we can use the `gdalinfo` program, that takes a filename and will output a copious description of the data, including metadata, but also geographic projection, size, number of bands, etc.

Here, we will look at the first 20 lines that come out of `gdalinfo`:

```
!gdalinfo files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf | head -20

Driver: HDF4/Hierarchical Data Format Release 4
Files: files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf
Size is 512, 512
Coordinate System is ''
Metadata:
  ALGORITHMPACKAGEACCEPTANCEDATE=10-01-2004
  ALGORITHMPACKAGEMATURITYCODE=Normal
  ALGORITHMPACKAGENAME=MCDPR_15A2
  ALGORITHMPACKAGEVERSION=5
  ASSOCIATEDINSTRUMENTSHORTNAME=MODIS
  ASSOCIATEDINSTRUMENTSHORTNAME=MODIS
  ASSOCIATEDPLATFORMSHORTNAME=Aqua
  ASSOCIATEDPLATFORMSHORTNAME=Terra
  ASSOCIATEDSENSORSHORTNAME=MODIS
  ASSOCIATEDSENSORSHORTNAME=MODIS
  AUTOMATICQUALITYFLAG=Passed
  AUTOMATICQUALITYFLAGEXPLANATION=No automatic quality assessment is performed in the PGE
  CHARACTERISTICBINANGULARSIZE=30.0
  CHARACTERISTICBINSIZE=926.625433055556
  DATACOLUMNS=1200
```

We can use standard unix filters (e.g. `grep`) to look at particular fields:

```
# Filter lines that do not have BOUNDINGCOORDINATE in them
file=files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf
gdalinfo $file | grep BOUNDINGCOORDINATE

EASTBOUNDINGCOORDINATE=-92.3664205550513
NORTHBOUNDINGCOORDINATE=39.9999999964079
SOUTHBOUNDINGCOORDINATE=29.9999999973059
WESTBOUNDINGCOORDINATE=-117.486656023174
```

We can check this against e.g. the [UNH MODIS tile calculator](#), just to confirm that we have interpreted the coordinates correctly.

We can apply other shell GDAL tools, e.g. to perform a reprojection from the native [MODIS sinusoidal](#) projection, to the [Contiguous United States NAD27 Albers Equal Area](#):

```
# a bash script

# set the variables file to be the filename for convenience
file=files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf

# dselete the output file if it exists
rm -f files/data/output_file.tif

# reproject the data
gdalwarp -of GTiff \
    -t_srs '+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 \
    +y_0=0 +ellps=clrk66 +units=m +no_defs' -tr 1000 1000 \
    'HDF4_EOS:EOS_GRID:${file}':MOD_Grid_MOD15A2:Lai_1km' files/data/output_file.tif

# convert to gif for viewing
gdal_translate -outsize 30% 30% -of gif \
    files/data/output_file.tif files/data/output_file.gif

Creating output file that is 2152P x 1323L.
Processing input file HDF4_EOS:EOS_GRID:files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf:files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf
Using internal nodata values (eg. 255) for image HDF4_EOS:EOS_GRID:files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf
0...10...20...30...40...50...60...70...80...90...100 - done.
Input file size is 2152, 1323
0...10...20...30...40...50...60...70...80...90...100 - done.
```

where `MCD15A2.A2011185.h09v05.005.2011213154534.hdf` is the name of the input HDF file, `MOD_Grid_MOD15A2:Lai_1km` is the data product we want, and the rather menacing string `+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0 +ellps=clrk66 +units=m +no_defs` specifies the projection in Proj4 format. You can typically find the projection you want on [spatialreference.org](#), and just copy and paste the contents of [Proj4 definition](#) (remember to surround it by quotes). The option `-tr xres yres` specifies the desired resolution of the output dataset (1000 by 1000 m in the case above). `-of GTiff` specifies the GeoTiff format to be used as output.

CHAPTER
THIRTYNINE

A4.3 FURTHER GEOSPATIAL NOTES

There are additonal notes that go into the details of `gdal` and vectors processing tools.

Follow these on:

- `gdal`
- `ogr`

There are no explicit advanced exercises this week. Instead, you should eplore these notes and see if you can apply the concepts to your own datasets.

EXERCISE 4.1

You are given the MODIS LAI data files for the year 2012 in the directory `files/data` for the UK (MODIS tile).

Read the LAI datasets into a masked array, using QA bit 0 to mask the data (i.e. good quality data only) and generate a movie of LAI.

40.1 Answer 4.1

You ought be getting familiar with this sort of problem, as it is very common.

Since we have the code in the main notes to read a single data dataset:

```
import gdal # Import GDAL library bindings
import numpy as np

# The file that we shall be using
# Needs to be on current directory
filename = 'files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'

g = gdal.Open(filename)
# g should now be a GDAL dataset, but if the file isn't found
# g will be none. Let's test this:
if g is None:
    print "Problem opening file %s!" % filename
else:
    print "File %s opened fine" % filename

subdatasets = g.GetSubDatasets()
for fname, name in subdatasets:
    print name
    print "\t", fname

# Let's create a list with the selected layer names
selected_layers = [ "Lai_1km", "FparLai_QC" ]
# We will store the data in a dictionary
# Initialise an empty dictionary
data = {}
# for convenience, we will use string substitution to create a
# template for GDAL filenames, which we'll substitute on the fly:
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
for i, layer in enumerate ( selected_layers ):
    this_file = file_template % ( filename, layer )
    print "Opening Layer %d: %s" % ( i+1, this_file )
    g = gdal.Open ( this_file )

    if g is None:
        raise IOError
```

```
data[layer] = g.ReadAsArray()
print "\t>>> Read %s!" % layer

# scale the LAI
lai = data['Lai_1km'] * 0.1

# pull out the QC
qc = data['FparLai_QC']
# find bit 0
qc = qc & 1

# generate the masked array
laim = np.ma.array ( lai, mask=qc )

File files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf opened fine
[1200x1200] Fpar_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] Lai_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparLai_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparExtra_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] FparStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
[1200x1200] LaiStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
Opening Layer 1: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_
    >>> Read Lai_1km!
Opening Layer 2: HDF4_EOS:EOS_GRID:"files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_
    >>> Read FparLai_QC!
```

We can use this as a function that reads a single dataset:

```
def read_modis_lai(filename):
    """ Read MODIS LAI data from filename
        and return masked array
    """

    g = gdal.Open(filename)
    # g should now be a GDAL dataset, but if the file isn't found
    # g will be none. Let's test this:
    if g is None:
        print "Problem opening file %s!" % filename
    else:
        print "File %s opened fine" % filename

    subdatasets = g.GetSubDatasets()
    #for fname, name in subdatasets:
    #    print name
    #    print "\t", fname

    # Let's create a list with the selected layer names
    selected_layers = [ "Lai_1km", "FparLai_QC" ]
    # We will store the data in a dictionary
    # Initialise an empty dictionary
    data = {}
    # for convenience, we will use string substitution to create a
    # template for GDAL filenames, which we'll substitute on the fly:
    file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
    for i, layer in enumerate ( selected_layers ):
        this_file = file_template % ( filename, layer )
```

```

#print "Opening Layer %d: %s" % (i+1, this_file )
g = gdal.Open ( this_file )

if g is None:
    raise IOError
data[layer] = g.ReadAsArray()
#print "\t>>> Read %s!" % layer

# scale the LAI
lai = data['Lai_1km'] * 0.1

# pull out the QC
qc = data['FparLai_QC']
# find bit 0
qc = qc & 1

# generate the masked array
laim = np.ma.array ( lai, mask=qc )

return laim

```

and then loop:

```

import glob

year = 2012
tile = 'h17v03'
files = np.sort(glob.glob('files/data/MCD15A2.A%d*.%s.*.hdf'%(year,tile)))

lai = []
for f in files:
    lai.append(read_modis_lai(f))

# force it to be a masked array
lai = np.ma.array(lai)

File files/data/MCD15A2.A2012017.h17v03.005.2012026072526.hdf opened fine
File files/data/MCD15A2.A2012025.h17v03.005.2012052124839.hdf opened fine
File files/data/MCD15A2.A2012033.h17v03.005.2012042060649.hdf opened fine
File files/data/MCD15A2.A2012041.h17v03.005.2012050092057.hdf opened fine
File files/data/MCD15A2.A2012049.h17v03.005.2012068144447.hdf opened fine
File files/data/MCD15A2.A2012057.h17v03.005.2012068140544.hdf opened fine
File files/data/MCD15A2.A2012065.h17v03.005.2012075021749.hdf opened fine
File files/data/MCD15A2.A2012073.h17v03.005.2012083010304.hdf opened fine
File files/data/MCD15A2.A2012081.h17v03.005.2012090131602.hdf opened fine
File files/data/MCD15A2.A2012089.h17v03.005.2012107201245.hdf opened fine
File files/data/MCD15A2.A2012097.h17v03.005.2012108125047.hdf opened fine
File files/data/MCD15A2.A2012105.h17v03.005.2012116125519.hdf opened fine
File files/data/MCD15A2.A2012113.h17v03.005.2012122072153.hdf opened fine
File files/data/MCD15A2.A2012121.h17v03.005.2012137221611.hdf opened fine
File files/data/MCD15A2.A2012129.h17v03.005.2012142001241.hdf opened fine
File files/data/MCD15A2.A2012137.h17v03.005.2012153021910.hdf opened fine
File files/data/MCD15A2.A2012145.h17v03.005.2012160130927.hdf opened fine
File files/data/MCD15A2.A2012153.h17v03.005.2012166161748.hdf opened fine
File files/data/MCD15A2.A2012161.h17v03.005.2012170080216.hdf opened fine
File files/data/MCD15A2.A2012169.h17v03.005.2012181134242.hdf opened fine
File files/data/MCD15A2.A2012177.h17v03.005.2012188150145.hdf opened fine
File files/data/MCD15A2.A2012185.h17v03.005.2012208181105.hdf opened fine
File files/data/MCD15A2.A2012193.h17v03.005.2012202144013.hdf opened fine
File files/data/MCD15A2.A2012201.h17v03.005.2012215131931.hdf opened fine
File files/data/MCD15A2.A2012209.h17v03.005.2012219144450.hdf opened fine
File files/data/MCD15A2.A2012217.h17v03.005.2012228215213.hdf opened fine
File files/data/MCD15A2.A2012225.h17v03.005.2012234105932.hdf opened fine

```

File files/data/MCD15A2.A2012233.h17v03.005.2012242093511.hdf opened fine
File files/data/MCD15A2.A2012241.h17v03.005.2012250182515.hdf opened fine
File files/data/MCD15A2.A2012249.h17v03.005.2012261231425.hdf opened fine
File files/data/MCD15A2.A2012257.h17v03.005.2012270114223.hdf opened fine
File files/data/MCD15A2.A2012265.h17v03.005.2012276134731.hdf opened fine
File files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf opened fine
File files/data/MCD15A2.A2012281.h17v03.005.2012297135831.hdf opened fine
File files/data/MCD15A2.A2012289.h17v03.005.2012299194634.hdf opened fine
File files/data/MCD15A2.A2012297.h17v03.005.2012306163257.hdf opened fine
File files/data/MCD15A2.A2012305.h17v03.005.2012314140451.hdf opened fine
File files/data/MCD15A2.A2012313.h17v03.005.2012322095802.hdf opened fine
File files/data/MCD15A2.A2012321.h17v03.005.2012335133638.hdf opened fine
File files/data/MCD15A2.A2012329.h17v03.005.2012340181739.hdf opened fine
File files/data/MCD15A2.A2012337.h17v03.005.2012346165133.hdf opened fine
File files/data/MCD15A2.A2012345.h17v03.005.2012356133200.hdf opened fine
File files/data/MCD15A2.A2012353.h17v03.005.2012363125132.hdf opened fine
File files/data/MCD15A2.A2012361.h17v03.005.2013007202756.hdf opened fine
File files/data/MCD15A2.A2012001.h17v03.005.2012017211237.hdf opened fine
File files/data/MCD15A2.A2012009.h17v03.005.2012019044037.hdf opened fine
File files/data/MCD15A2.A2012017.h17v03.005.2012026072526.hdf opened fine
File files/data/MCD15A2.A2012025.h17v03.005.2012052124839.hdf opened fine
File files/data/MCD15A2.A2012033.h17v03.005.2012042060649.hdf opened fine
File files/data/MCD15A2.A2012041.h17v03.005.2012050092057.hdf opened fine
File files/data/MCD15A2.A2012049.h17v03.005.2012068144447.hdf opened fine
File files/data/MCD15A2.A2012057.h17v03.005.2012068140544.hdf opened fine
File files/data/MCD15A2.A2012065.h17v03.005.2012075021749.hdf opened fine
File files/data/MCD15A2.A2012073.h17v03.005.2012083010304.hdf opened fine
File files/data/MCD15A2.A2012081.h17v03.005.2012090131602.hdf opened fine
File files/data/MCD15A2.A2012089.h17v03.005.2012107201245.hdf opened fine
File files/data/MCD15A2.A2012097.h17v03.005.2012108125047.hdf opened fine
File files/data/MCD15A2.A2012105.h17v03.005.2012116125519.hdf opened fine
File files/data/MCD15A2.A2012113.h17v03.005.2012122072153.hdf opened fine
File files/data/MCD15A2.A2012121.h17v03.005.2012137221611.hdf opened fine
File files/data/MCD15A2.A2012129.h17v03.005.2012142001241.hdf opened fine
File files/data/MCD15A2.A2012137.h17v03.005.2012153021910.hdf opened fine
File files/data/MCD15A2.A2012145.h17v03.005.2012160130927.hdf opened fine
File files/data/MCD15A2.A2012153.h17v03.005.2012166161748.hdf opened fine
File files/data/MCD15A2.A2012161.h17v03.005.2012170080216.hdf opened fine
File files/data/MCD15A2.A2012169.h17v03.005.2012181134242.hdf opened fine
File files/data/MCD15A2.A2012177.h17v03.005.2012188150145.hdf opened fine
File files/data/MCD15A2.A2012185.h17v03.005.2012208181105.hdf opened fine
File files/data/MCD15A2.A2012193.h17v03.005.2012202144013.hdf opened fine
File files/data/MCD15A2.A2012201.h17v03.005.2012215131931.hdf opened fine
File files/data/MCD15A2.A2012209.h17v03.005.2012219144450.hdf opened fine
File files/data/MCD15A2.A2012217.h17v03.005.2012228215213.hdf opened fine
File files/data/MCD15A2.A2012225.h17v03.005.2012234105932.hdf opened fine
File files/data/MCD15A2.A2012233.h17v03.005.2012242093511.hdf opened fine
File files/data/MCD15A2.A2012241.h17v03.005.2012250182515.hdf opened fine
File files/data/MCD15A2.A2012249.h17v03.005.2012261231425.hdf opened fine
File files/data/MCD15A2.A2012257.h17v03.005.2012270114223.hdf opened fine
File files/data/MCD15A2.A2012265.h17v03.005.2012276134731.hdf opened fine
File files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf opened fine
File files/data/MCD15A2.A2012281.h17v03.005.2012297135831.hdf opened fine
File files/data/MCD15A2.A2012289.h17v03.005.2012299194634.hdf opened fine
File files/data/MCD15A2.A2012297.h17v03.005.2012306163257.hdf opened fine
File files/data/MCD15A2.A2012305.h17v03.005.2012314140451.hdf opened fine
File files/data/MCD15A2.A2012313.h17v03.005.2012322095802.hdf opened fine
File files/data/MCD15A2.A2012321.h17v03.005.2012335133638.hdf opened fine
File files/data/MCD15A2.A2012329.h17v03.005.2012340181739.hdf opened fine
File files/data/MCD15A2.A2012337.h17v03.005.2012346165133.hdf opened fine
File files/data/MCD15A2.A2012345.h17v03.005.2012356133200.hdf opened fine
File files/data/MCD15A2.A2012353.h17v03.005.2012363125132.hdf opened fine
File files/data/MCD15A2.A2012361.h17v03.005.2013007202756.hdf opened fine

```
# plot the data
import pylab as plt

# work out a consistent scaling
lai_max = np.max(lai)

for i,f in enumerate(files):
    fig = plt.figure(figsize=(7,7))
    plt.imshow(lai[i],interpolation='none',vmin=0.,vmax=lai_max*0.75)
    # remember filenames of the form
    # files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'
    file_id = f.split('/')[-1].split('.')[1:-5][1:]
    print file_id
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('files/images/lai_uk_%s.jpg'%file_id)
    plt.close(fig)

2012001
2012009
2012017
2012025
2012033
2012041
2012049
2012057
2012065
2012073
2012081
2012089
2012097
2012105
2012113
2012121
2012129
2012137
2012145
2012153
2012161
2012169
2012177
2012185
2012193
2012201
2012209
2012217
2012225
2012233
2012241
2012249
2012257
2012265
2012273
2012281
2012289
2012297
2012305
2012313
2012321
2012329
2012337
2012345
```

```
2012353
2012361
2012001
2012009
2012017
2012025
2012033
2012041
2012049
2012057
2012065
2012073
2012081
2012089
2012097
2012105
2012113
2012121
2012129
2012137
2012145
2012153
2012161
2012169
2012177
2012185
2012193
2012201
2012209
2012217
2012225
2012233
2012241
2012249
2012257
2012265
2012273
2012281
2012289
2012297
2012305
2012313
2012321
2012329
2012337
2012345
2012353
2012361

# now make a movie ...

import os

cmd = 'convert -delay 100 -loop 0 files/images/lai_uk_*.jpg files/images/lai_uk2.gif'
os.system(cmd)

0
0
```

EXERCISE 4.2: A DIFFERENT DATASET

We have now downloaded a different dataset, the [MOD10A product](#), which is the 500 m MODIS daily snow cover product, over the UK.

This is a good opportunity to see if you can apply what was learned above about interpreting QC information and using gdal to examine a dataset.

If you examine the [data description page](#), you will see that the data are in HDF EOS format (the same as the LAI product).

41.1 E4.2.1 Download

```
tile=h17v03
year=2012
type=MODIS
month=02

file=robot_snow.${year}_${type}_${tile}_${month}.txt

grep $tile < files/data/robot_snow.$year.txt | grep $type | grep "${year}\.${month}" > files/data/$file

wc -l files/data/$file

# cd temporarily to the local directory
pushd files/data
# -nc : no clobber : dont download if its there already
# -nH --cut-dirs=3 : ignore the directories
wget -nc -i $file -nH --cut-dirs=3
# cd back again
popd
echo $file

29 files/data/robot_snow.2012_MODIS_h17v03_02.txt
~/p/geogg122_local/geogg122/Chapter4_GDAL/files/data ~/p/geogg122_local/geogg122/Chapter4_GDAL
~/p/geogg122_local/geogg122/Chapter4_GDAL
robot_snow.2012_MODIS_h17v03_02.txt

Exception in thread Thread-2:
Traceback (most recent call last):
  File "/opt/anaconda/lib/python2.7/threading.py", line 808, in __bootstrap_inner
    self.run()
  File "/opt/anaconda/lib/python2.7/site-packages/IPython/kernel/zmq/heartbeat.py", line 55, in run
    zmq.device(zmq.FORWARDER, self.socket, self.socket)
  File "device.pyx", line 55, in zmq.core.device.device (zmq/core/device.c:854)
ZMQError: Interrupted system call

File `MOD10A1.A2012032.h17v03.005.2012046141710.hdf' already there; not retrieving.
File `MOD10A1.A2012033.h17v03.005.2012035071148.hdf' already there; not retrieving.
```

```
File `MOD10A1.A2012034.h17v03.005.2012036060335.hdf' already there; not retrieving.  
File `MOD10A1.A2012035.h17v03.005.2012037061127.hdf' already there; not retrieving.  
File `MOD10A1.A2012036.h17v03.005.2012038044224.hdf' already there; not retrieving.  
File `MOD10A1.A2012037.h17v03.005.2012039055929.hdf' already there; not retrieving.  
File `MOD10A1.A2012038.h17v03.005.2012040045622.hdf' already there; not retrieving.  
File `MOD10A1.A2012039.h17v03.005.2012041055153.hdf' already there; not retrieving.  
File `MOD10A1.A2012040.h17v03.005.2012042055706.hdf' already there; not retrieving.  
File `MOD10A1.A2012041.h17v03.005.2012045014038.hdf' already there; not retrieving.  
File `MOD10A1.A2012042.h17v03.005.2012045024036.hdf' already there; not retrieving.  
File `MOD10A1.A2012043.h17v03.005.2012045091534.hdf' already there; not retrieving.  
File `MOD10A1.A2012044.h17v03.005.2012046060303.hdf' already there; not retrieving.  
File `MOD10A1.A2012045.h17v03.005.2012047061553.hdf' already there; not retrieving.  
File `MOD10A1.A2012046.h17v03.005.2012048071318.hdf' already there; not retrieving.  
File `MOD10A1.A2012047.h17v03.005.2012049053946.hdf' already there; not retrieving.  
File `MOD10A1.A2012048.h17v03.005.2012050063219.hdf' already there; not retrieving.  
File `MOD10A1.A2012049.h17v03.005.2012051070019.hdf' already there; not retrieving.  
File `MOD10A1.A2012050.h17v03.005.2012052062212.hdf' already there; not retrieving.  
File `MOD10A1.A2012051.h17v03.005.2012053064001.hdf' already there; not retrieving.  
File `MOD10A1.A2012052.h17v03.005.2012054062116.hdf' already there; not retrieving.  
File `MOD10A1.A2012053.h17v03.005.2012056015333.hdf' already there; not retrieving.  
File `MOD10A1.A2012054.h17v03.005.2012062191100.hdf' already there; not retrieving.  
File `MOD10A1.A2012055.h17v03.005.2012057063901.hdf' already there; not retrieving.  
File `MOD10A1.A2012056.h17v03.005.2012058061115.hdf' already there; not retrieving.  
File `MOD10A1.A2012057.h17v03.005.2012059062619.hdf' already there; not retrieving.  
File `MOD10A1.A2012058.h17v03.005.2012060061357.hdf' already there; not retrieving.  
File `MOD10A1.A2012059.h17v03.005.2012061063515.hdf' already there; not retrieving.  
File `MOD10A1.A2012060.h17v03.005.2012062064750.hdf' already there; not retrieving.
```

```
29 files/data/robot_snow_2012_MOST_h17v03_02.txt  
~/p/geogg122_local/geogg122/Chapter4_GDAL/files/data ~/p/geogg122_local/geogg122/Chapter4_GDAL  
~/p/geogg122_local/geogg122/Chapter4_GDAL  
robot_snow_2012_MOST_h17v03_02.txt
```

```
File `MOD10A1.A2012032.h17v03.005.2012046141710.hdf' already there; not retrieving.  
File `MOD10A1.A2012033.h17v03.005.2012035071148.hdf' already there; not retrieving.  
File `MOD10A1.A2012034.h17v03.005.2012036060335.hdf' already there; not retrieving.  
File `MOD10A1.A2012035.h17v03.005.2012037061127.hdf' already there; not retrieving.  
File `MOD10A1.A2012036.h17v03.005.2012038044224.hdf' already there; not retrieving.  
File `MOD10A1.A2012037.h17v03.005.2012039055929.hdf' already there; not retrieving.  
File `MOD10A1.A2012038.h17v03.005.2012040045622.hdf' already there; not retrieving.  
File `MOD10A1.A2012039.h17v03.005.2012041055153.hdf' already there; not retrieving.  
File `MOD10A1.A2012040.h17v03.005.2012042055706.hdf' already there; not retrieving.  
File `MOD10A1.A2012041.h17v03.005.2012045014038.hdf' already there; not retrieving.  
File `MOD10A1.A2012042.h17v03.005.2012045024036.hdf' already there; not retrieving.  
File `MOD10A1.A2012043.h17v03.005.2012045091534.hdf' already there; not retrieving.  
File `MOD10A1.A2012044.h17v03.005.2012046060303.hdf' already there; not retrieving.  
File `MOD10A1.A2012045.h17v03.005.2012047061553.hdf' already there; not retrieving.  
File `MOD10A1.A2012046.h17v03.005.2012048071318.hdf' already there; not retrieving.  
File `MOD10A1.A2012047.h17v03.005.2012049053946.hdf' already there; not retrieving.  
File `MOD10A1.A2012048.h17v03.005.2012050063219.hdf' already there; not retrieving.  
File `MOD10A1.A2012049.h17v03.005.2012051070019.hdf' already there; not retrieving.  
File `MOD10A1.A2012050.h17v03.005.2012052062212.hdf' already there; not retrieving.  
File `MOD10A1.A2012051.h17v03.005.2012053064001.hdf' already there; not retrieving.  
File `MOD10A1.A2012052.h17v03.005.2012054062116.hdf' already there; not retrieving.  
File `MOD10A1.A2012053.h17v03.005.2012056015333.hdf' already there; not retrieving.  
File `MOD10A1.A2012054.h17v03.005.2012062191100.hdf' already there; not retrieving.  
File `MOD10A1.A2012055.h17v03.005.2012057063901.hdf' already there; not retrieving.  
File `MOD10A1.A2012056.h17v03.005.2012058061115.hdf' already there; not retrieving.  
File `MOD10A1.A2012057.h17v03.005.2012059062619.hdf' already there; not retrieving.  
File `MOD10A1.A2012058.h17v03.005.2012060061357.hdf' already there; not retrieving.  
File `MOD10A1.A2012059.h17v03.005.2012061063515.hdf' already there; not retrieving.  
File `MOD10A1.A2012060.h17v03.005.2012062064750.hdf' already there; not retrieving.
```

41.2 E4.2.2 Explore

We explore the dataset in the same way as above:

```
# how to find out which datasets are in the file

import gdal # Import GDAL library bindings

# The file that we shall be using
# Needs to be on current directory
filename = 'files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf'

g = gdal.Open(filename)
# g should now be a GDAL dataset, but if the file isn't found
# g will be none. Let's test this:
if g is None:
    print "Problem opening file %s!" % filename
else:
    print "File %s opened fine" % filename

subdatasets = g.GetSubDatasets()
for fname, name in subdatasets:
    print name
    print "\t", fname

File files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf opened fine
[2400x2400] Snow_Cover_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Spatial_QA MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Albedo_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
[2400x2400] Fractional_Snow_Cover MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
File files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf opened fine
[2400x2400] Snow_Cover_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Spatial_QA MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Albedo_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
[2400x2400] Fractional_Snow_Cover MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_Grid_Snow_500m
```

41.3 E4.3.3 Read a dataset

This is very nearly the same as the LAI dataset reading.

We need obviously to select a different layer name.

We should also notice that the `file_template` will be slightly different (look at the list of subsets above).

```
# How to access specific datasets in gdal

# Let's create a list with the selected layer names
selected_layers = [ "Fractional_Snow_Cover" ]

# We will store the data in a dictionary
# Initialise an empty dictionary
data = {}
```

```
# for convenience, we will use string substitution to create a
# template for GDAL filenames, which we'll substitute on the fly:
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_Snow_500m:%s'
# This has two substitutions (the %s parts) which will refer to:
# - the filename
# - the data layer

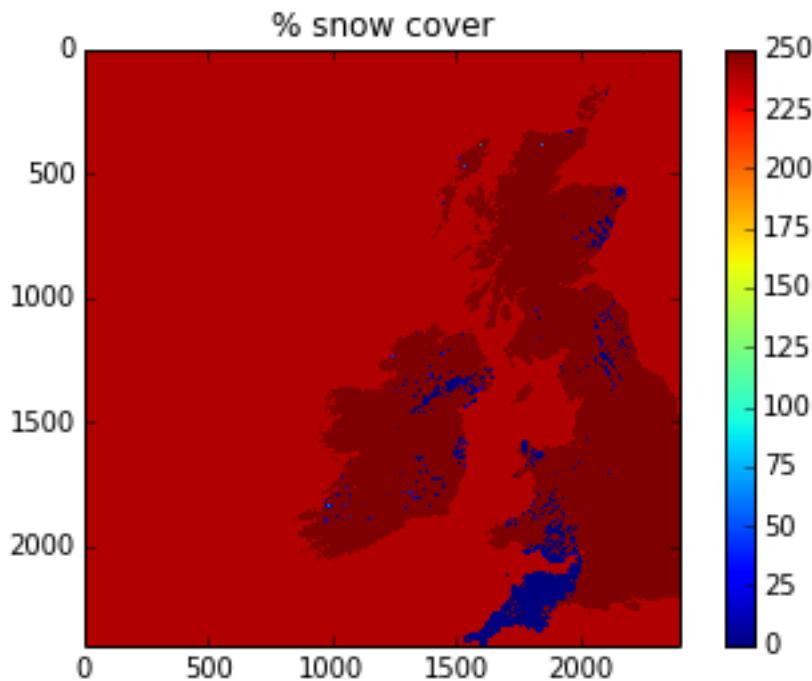
for i, layer in enumerate ( selected_layers ):
    this_file = file_template % ( filename, layer )
    print "Opening Layer %d: %s" % (i+1, this_file )
    g = gdal.Open ( this_file )

    if g is None:
        raise IOError
    data[layer] = g.ReadAsArray()
    print "\t>>> Read %s!" % layer

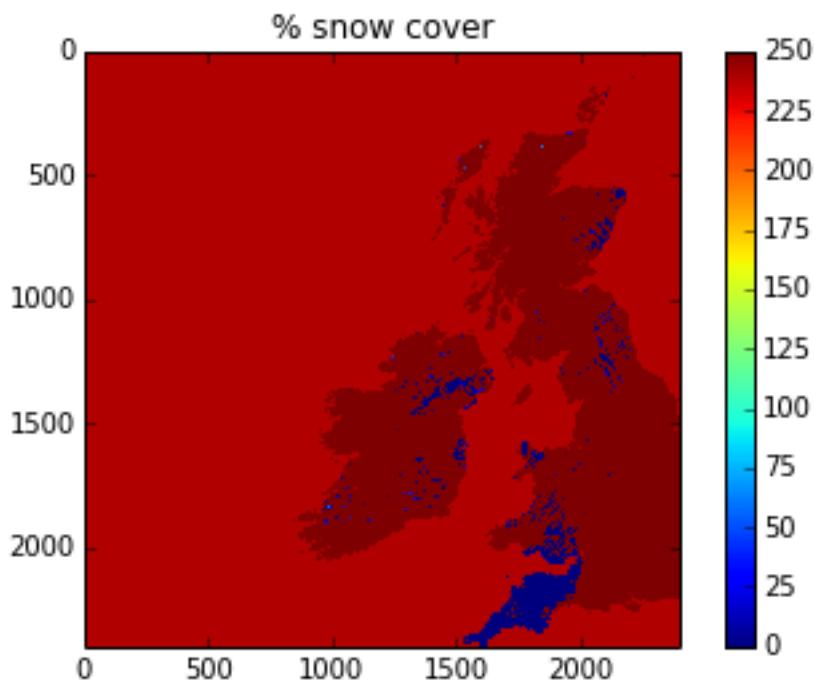
Opening Layer 1: HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_
    >>> Read Fractional_Snow_Cover!
Opening Layer 1: HDF4_EOS:EOS_GRID:"files/data/MOD10A1.A2012060.h17v03.005.2012062064750.hdf":MOD_
    >>> Read Fractional_Snow_Cover!

plt.imshow(data["Fractional_Snow_Cover"])
plt.colorbar()
plt.title('% snow cover')

<matplotlib.text.Text at 0x5419ca90>
```



```
<matplotlib.text.Text at 0x5c29a310>
```



41.4 E4.3.4 Water mask

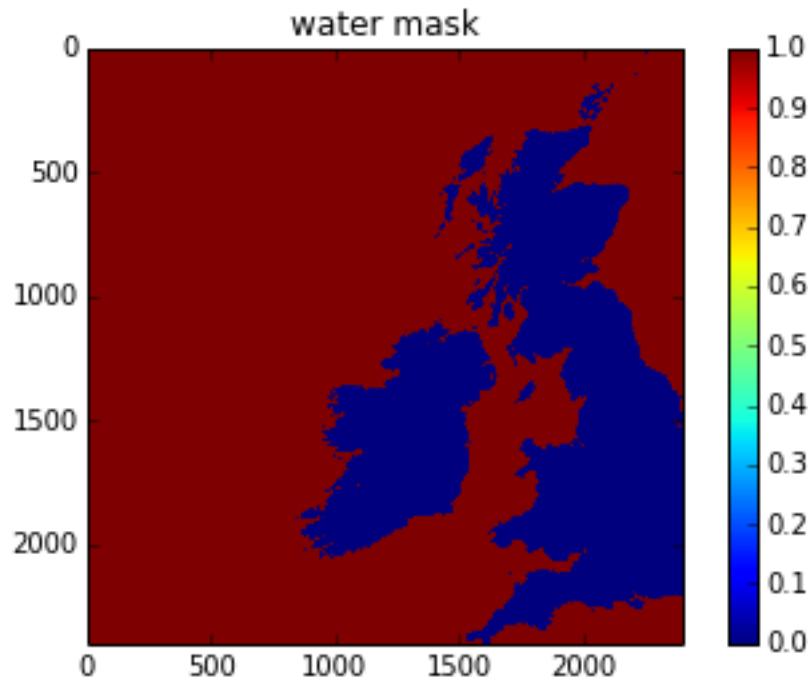
The data description page tells us that values of 237 239 will indicate whether the data are ocean or inland water bodies. These are what you should use to build the water mask.

```
snow = data["Fractional_Snow_Cover"]

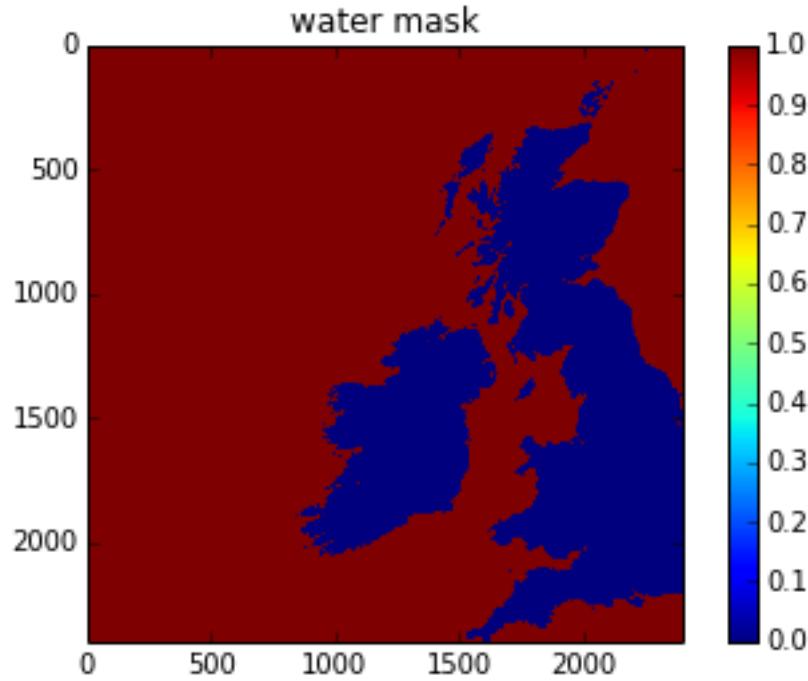
water = (snow == 239)

plt.imshow(water)
plt.colorbar()
plt.title('water mask')

<matplotlib.text.Text at 0x4eb36890>
```



<matplotlib.text.Text at 0x58256990>



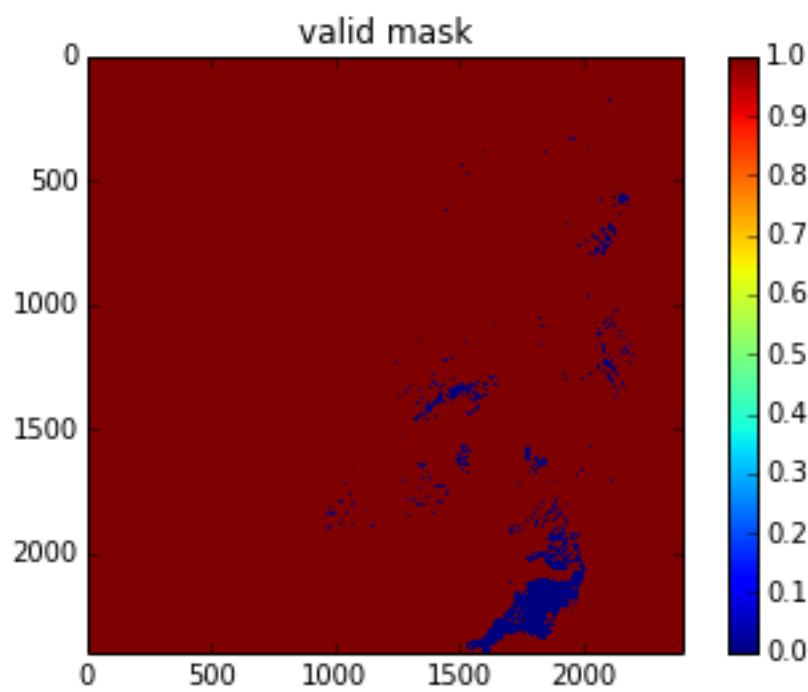
41.5 E4.3.5 Valid pixel mask

```
snow = data["Fractional_Snow_Cover"]

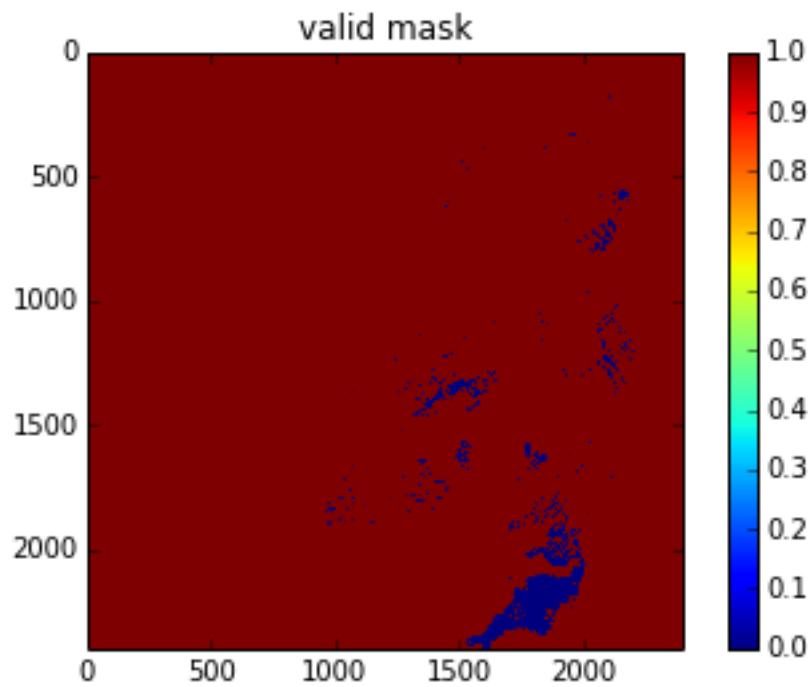
valid_mask = (snow > 100)

plt.imshow(valid_mask)
plt.colorbar()
plt.title('valid mask')
```

```
<matplotlib.text.Text at 0x1c8f0e10>
```



```
<matplotlib.text.Text at 0x5ad9cb50>
```



41.6 E4.3.6 3D dataset

You should be used to this sort of thing by now.

First, lets sort the filenames:

```
# filenames from the file
url_file = 'files/data/robot_snow.2012_MOST_h17v03_02.txt'

# open the file for read
fp = open(url_file, 'r')

# the strip is to get rid of the \n character
filenames = [f.split('/')[-1].strip() for f in fp.readlines()]

# close the file
fp.close

# lets see what we got
print filenames

['MOD10A1.A2012032.h17v03.005.2012046141710.hdf', 'MOD10A1.A2012033.h17v03.005.2012035071148.hdf',
 ['MOD10A1.A2012032.h17v03.005.2012046141710.hdf', 'MOD10A1.A2012033.h17v03.005.2012035071148.hdf']]
```

Using `glob` you might try:

```
# using glob
from glob import glob

filenames = glob('files/data/MOD10A1.A2012*.hdf')
print filenames

['files/data/MOD10A1.A2012301.h17v03.005.2012303060756.hdf', 'files/data/MOD10A1.A201222.h17v03.005.2012303060756.hdf',
 ['files/data/MOD10A1.A2012301.h17v03.005.2012303060756.hdf', 'files/data/MOD10A1.A201222.h17v03.005.2012303060756.hdf']]
```

But that isn't refined enough (i.e. it won't only pull files for February if there are more).

So, we need to pull files from doy 32 to 60 inclusive ... which is a lot more tricky ...

```
# using glob
from glob import glob

filenames = glob('files/data/MOD10A1.A201203[2-9]*.hdf') +\
            glob('files/data/MOD10A1.A201204??.*.hdf') +\
            glob('files/data/MOD10A1.A201205??.*.hdf') +\
            glob('files/data/MOD10A1.A2012060??.*.hdf')
print filenames

['files/data/MOD10A1.A2012035.h17v03.005.2012037061127.hdf', 'files/data/MOD10A1.A2012033.h17v03.005.2012037061127.hdf',
 ['files/data/MOD10A1.A2012035.h17v03.005.2012037061127.hdf', 'files/data/MOD10A1.A2012033.h17v03.005.2012037061127.hdf']]
```

Practically, you might be better off reading an interpreting the url file as above, or alternatively, put all of the February files in a different directory and `glob` that ...

```
# in any case, we should sort the filenames
filenames = sort(filenames)
```

Next, let's define a function to read a single file into a numpy array (and simplify what we have above a bit):

```
import numpy.ma as ma

def read_snow(filename):

    layers = "Fractional_Snow_Cover"

    # for convenience, we will use string substitution to create a
    # template for GDAL filenames, which we'll substitute on the fly:
    file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_Snow_500m:%s'
    # This has two substitutions (the %s parts) which will refer to:
    # - the filename
```

```
# - the data layer

this_file = file_template % ( filename, layer )
g = gdal.Open ( this_file )

if g is None:
    raise IOError
snow = g.ReadAsArray()

# dont use this here, but just in case useful
#water = (snow == 239)

valid_mask = (snow > 100)

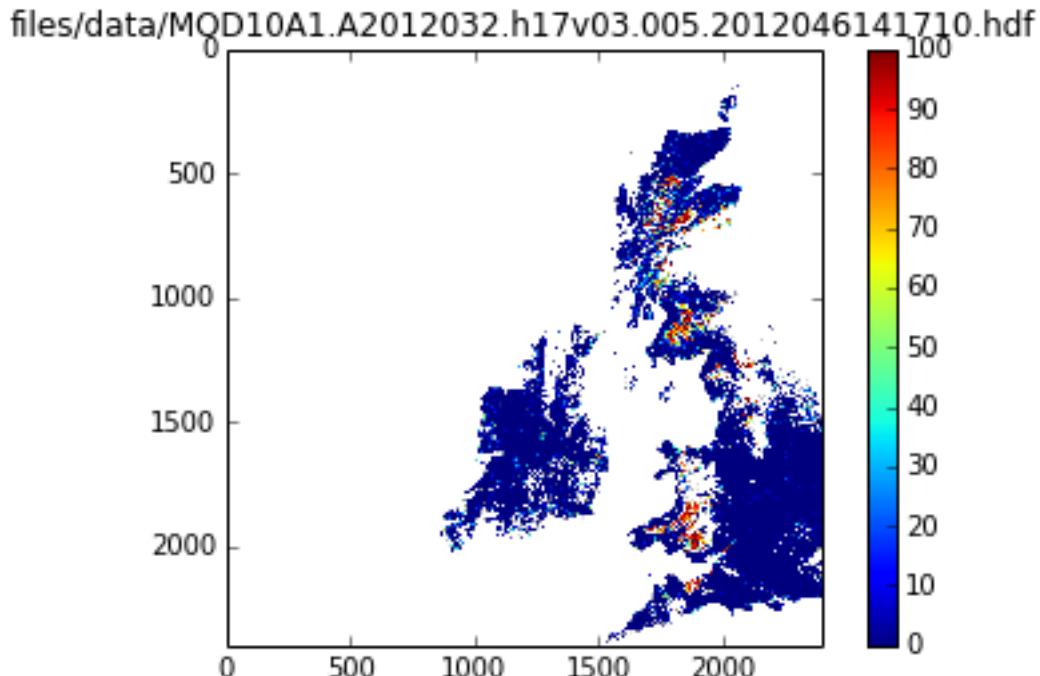
return ma.array(snow,mask=valid_mask)
```

Now, test it:

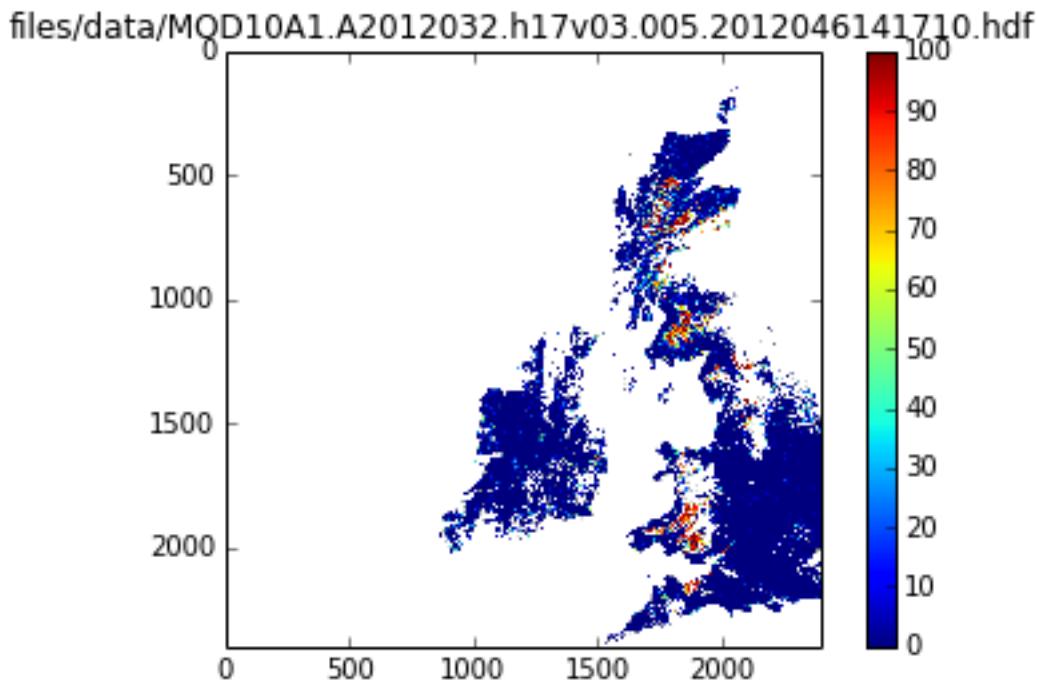
```
snow = read_snow(filenames[0])

plt.imshow(snow,vmax=100)
plt.colorbar()
plt.title(filenames[0])

<matplotlib.text.Text at 0x51650d50>
```



```
<matplotlib.text.Text at 0x55663490>
```



Now its just a loop as in several previous examples:

```
snow = []
for f in filenames:
    snow.append(read_snow(f))

snow = ma.array(snow)

# or neater ...

snow = ma.array([read_snow(f) for f in filenames])

print snow.shape
(29, 2400, 2400)

# now plot and save images
# plot the data
import pylab as plt

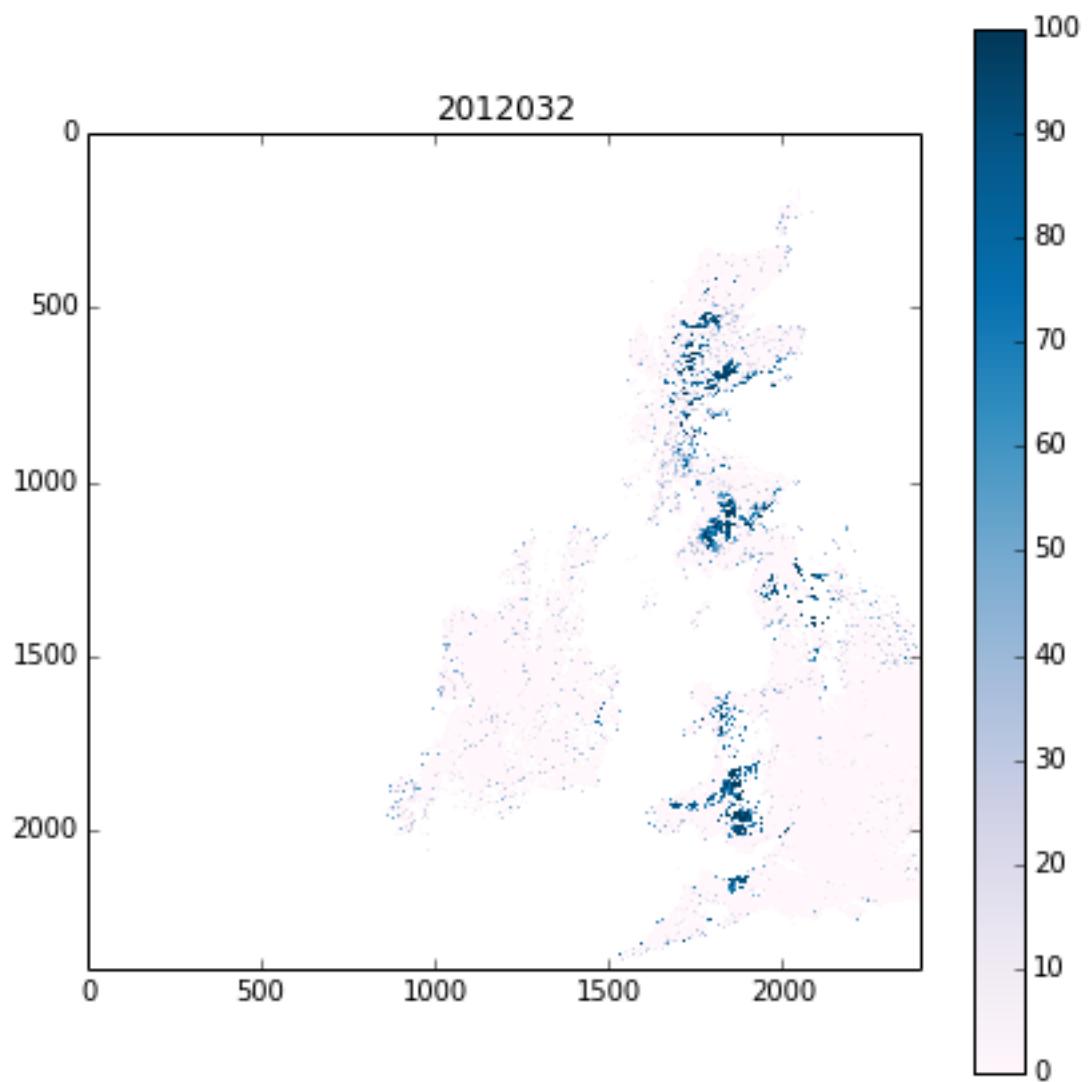
cmap = plt.cm.PuBu

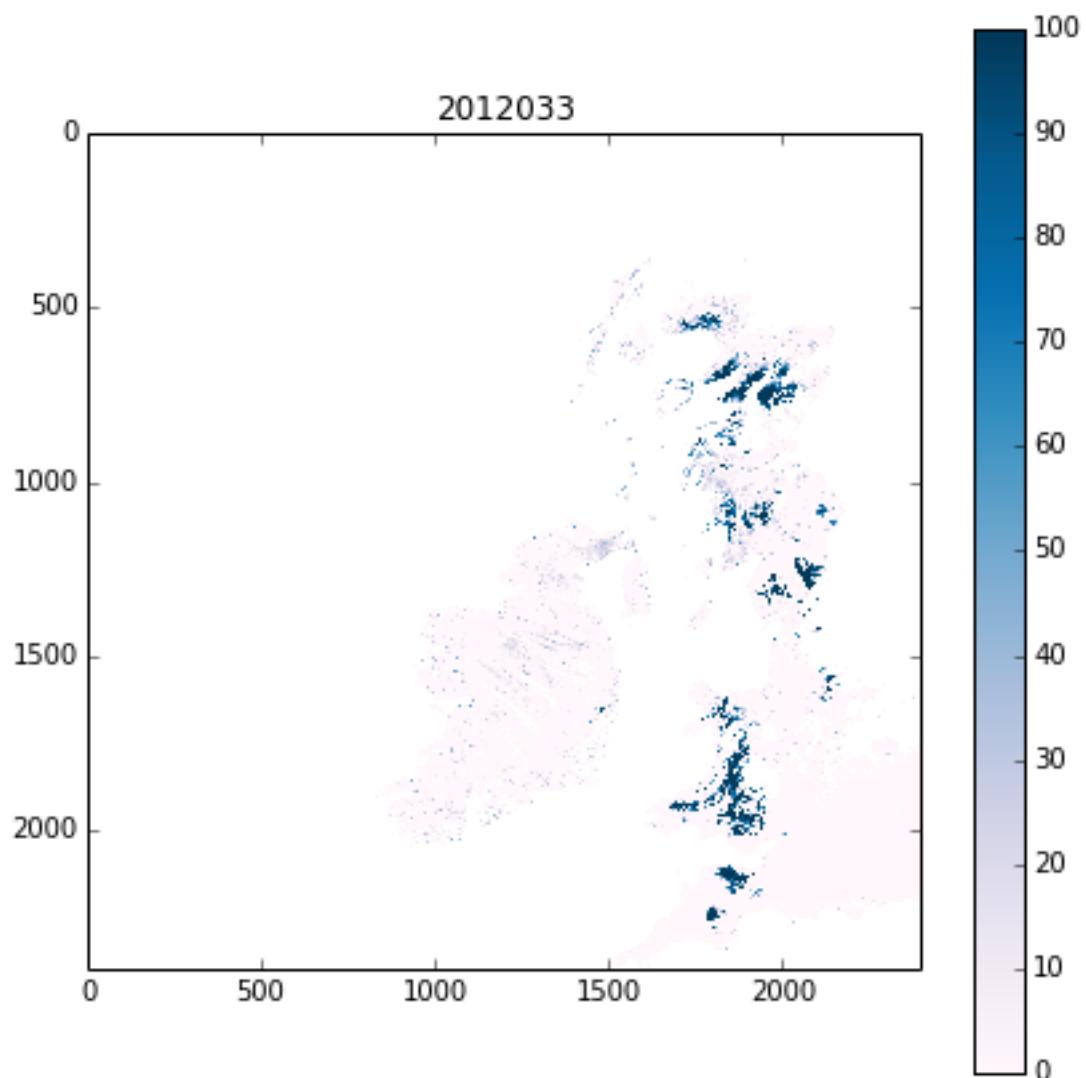
for i,f in enumerate(filenames):
    plt.figure(figsize=(7,7))
    plt.imshow(snow[i],cmap=cmap,interpolation='none',vmin=0.,vmax=100)
    # remember filenames of the form
    # files/data/MCD15A2.A2011185.h09v05.005.2011213154534.hdf'
    file_id = f.split('/')[-1].split('.')[5][1:]
    print file_id
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('files/images/snow_uk_%s.jpg'%file_id)

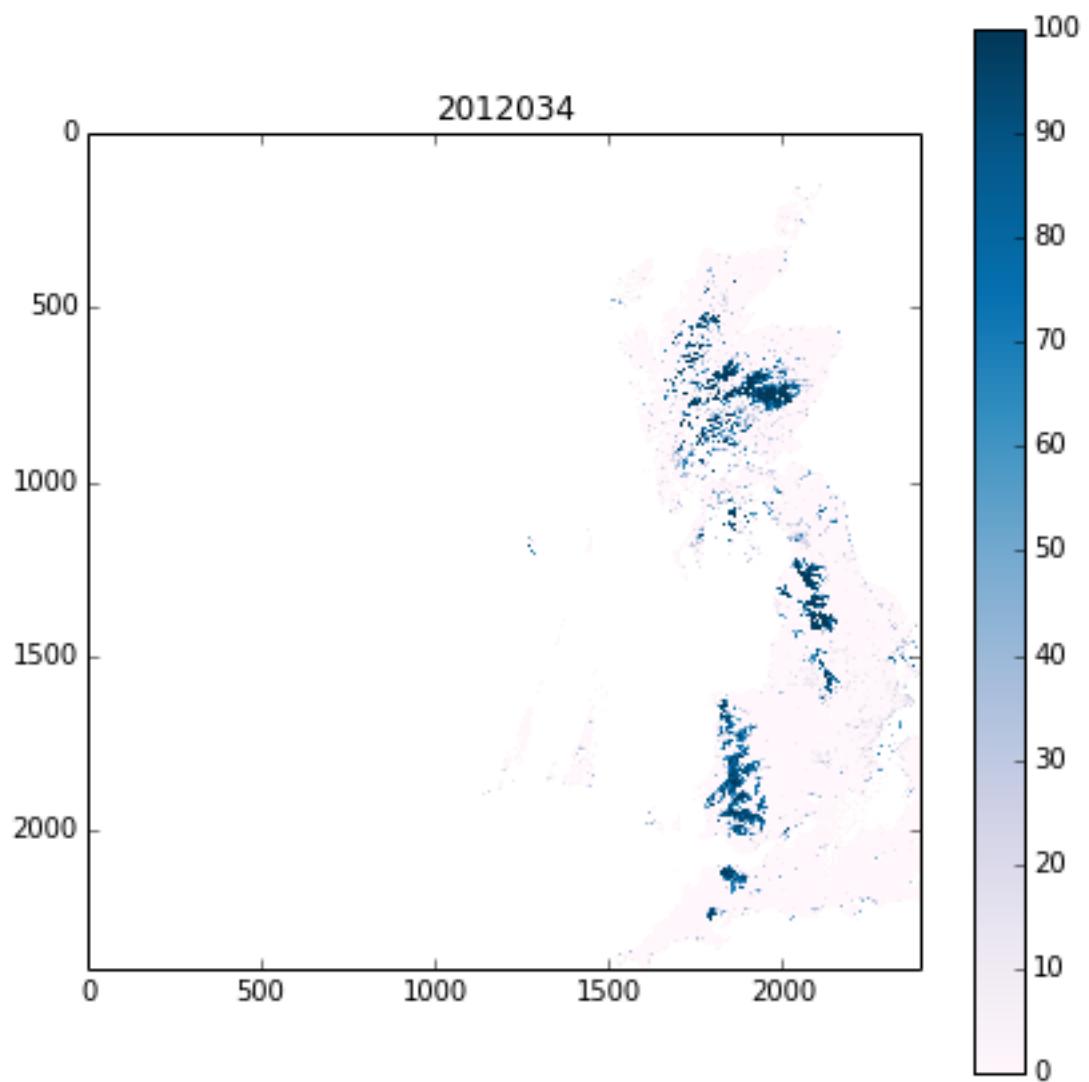
2012032
2012033
2012034
2012035
```

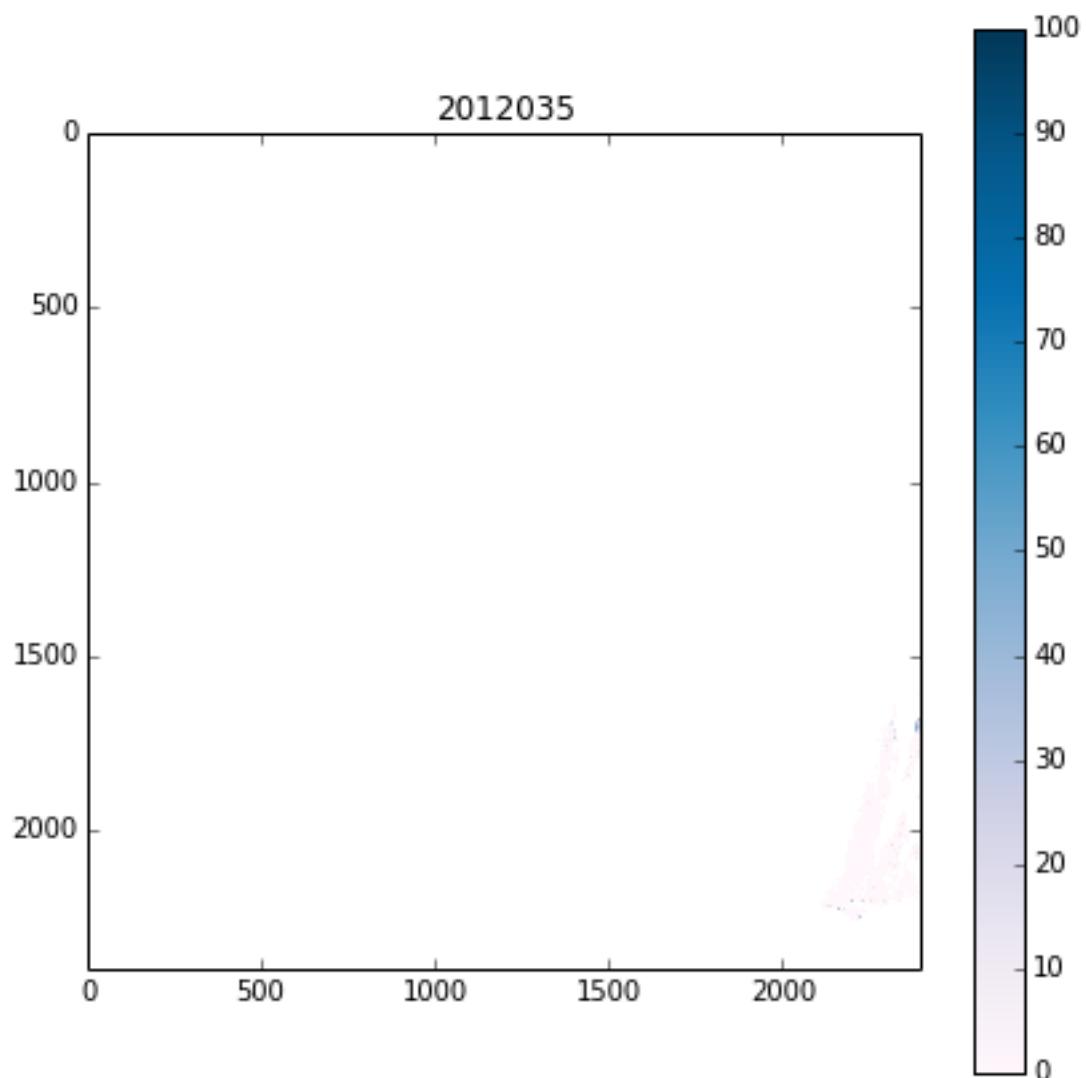
2012036
2012037
2012038
2012039
2012040
2012041
2012042
2012043
2012044
2012045
2012046
2012047
2012048
2012049
2012050
2012051
2012052
2012053
2012054
2012055
2012056
2012057
2012058
2012059
2012060

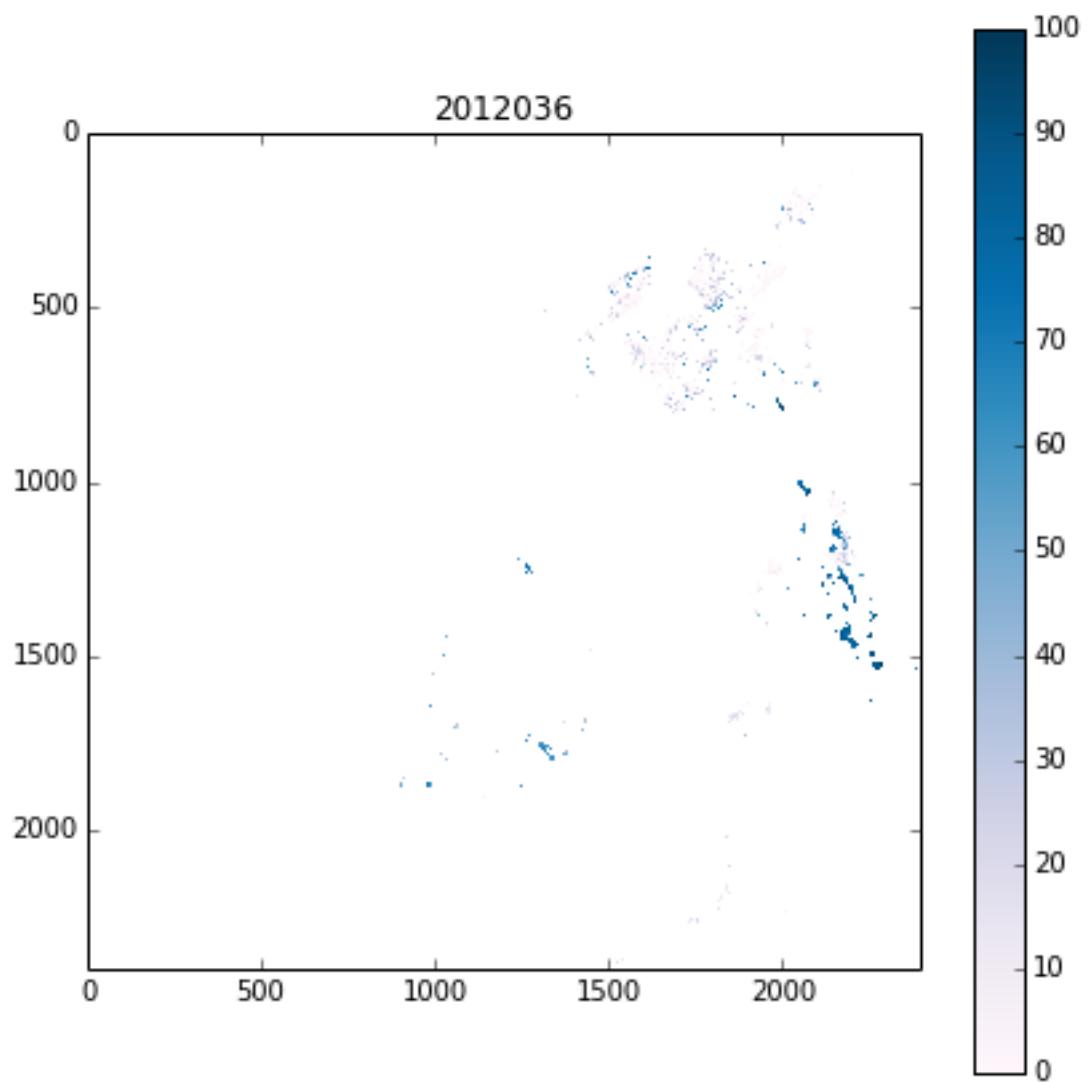
/opt/anaconda/lib/python2.7/site-packages/matplotlib/pyplot.py:412: RuntimeWarning: More than
max_open_warning, RuntimeWarning)

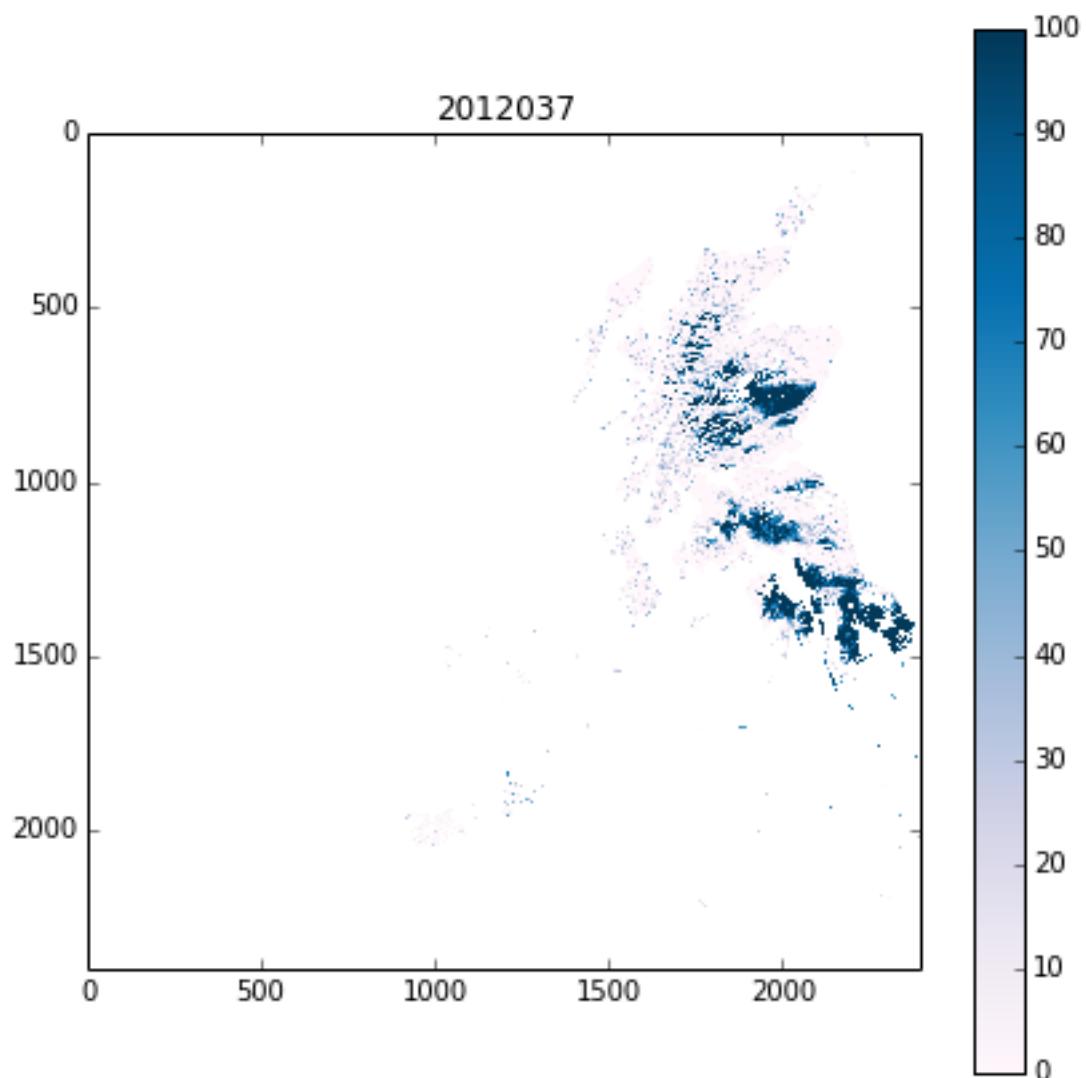


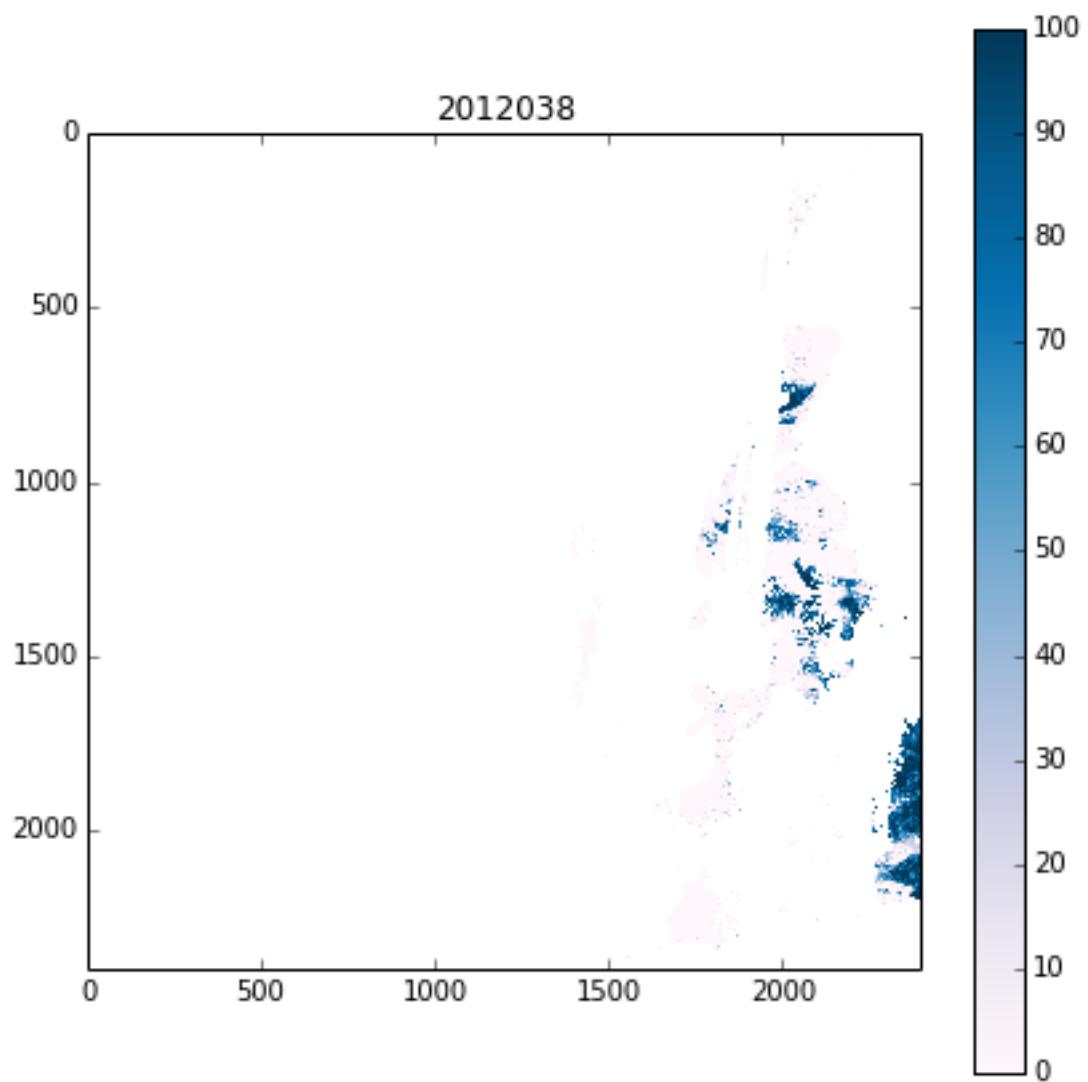


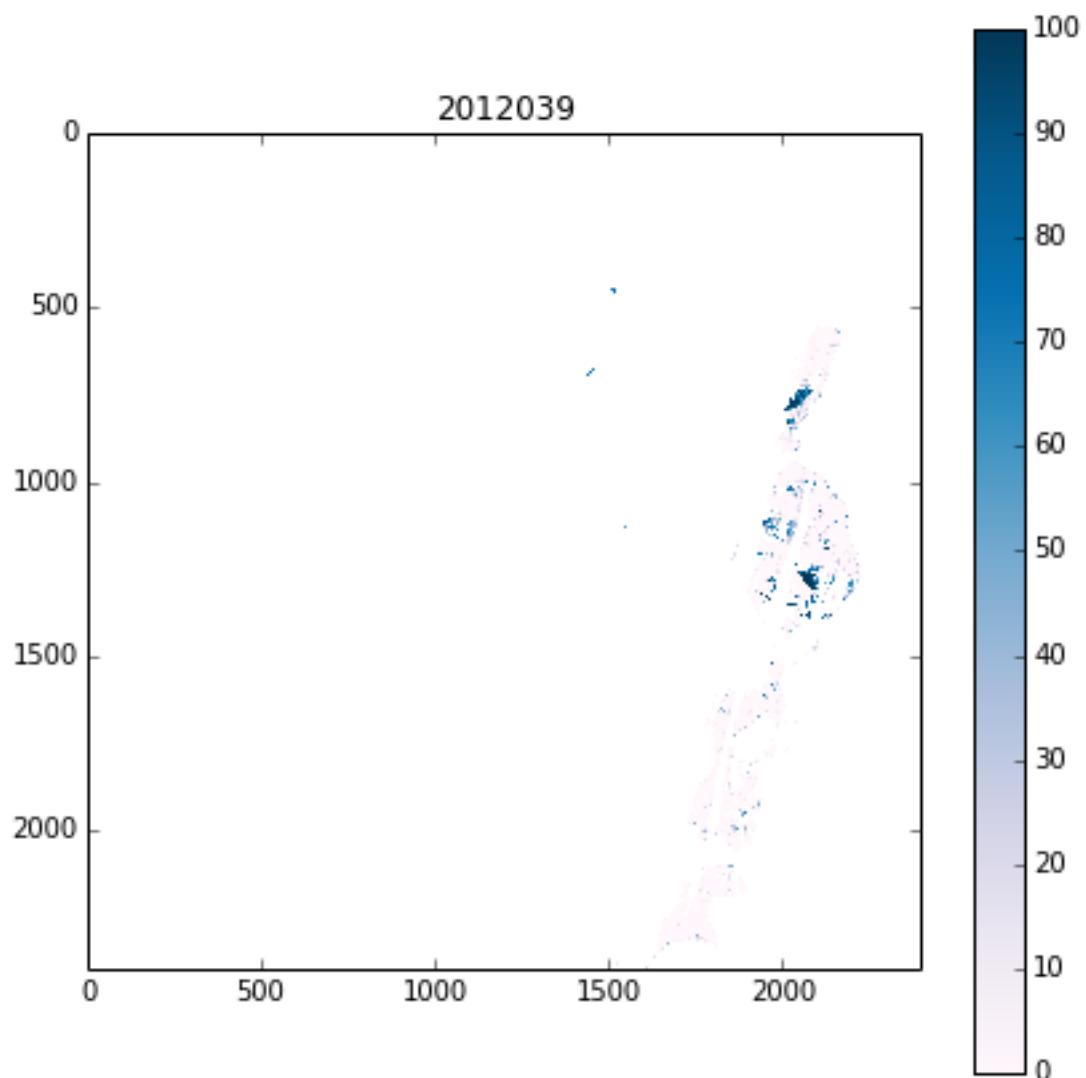


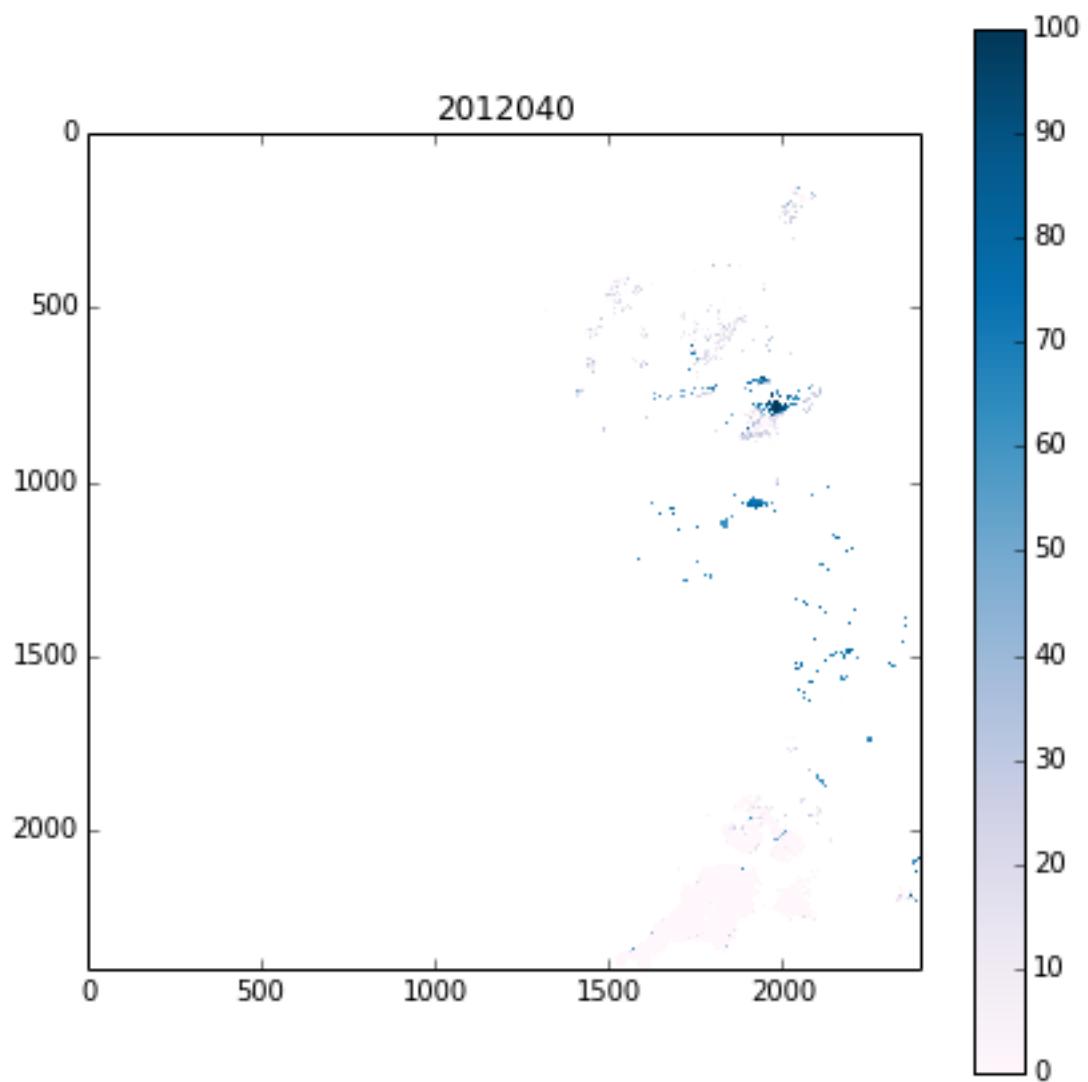


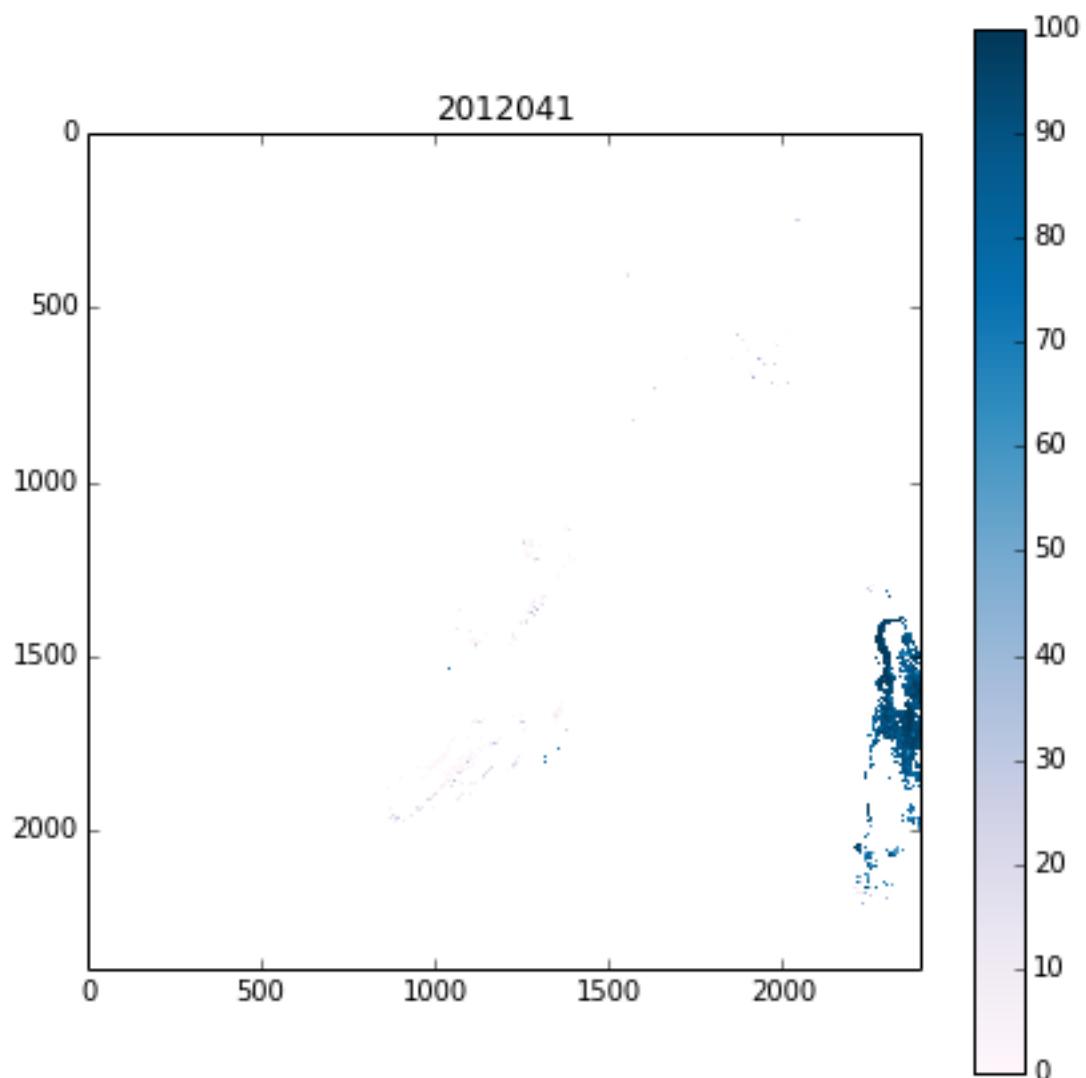


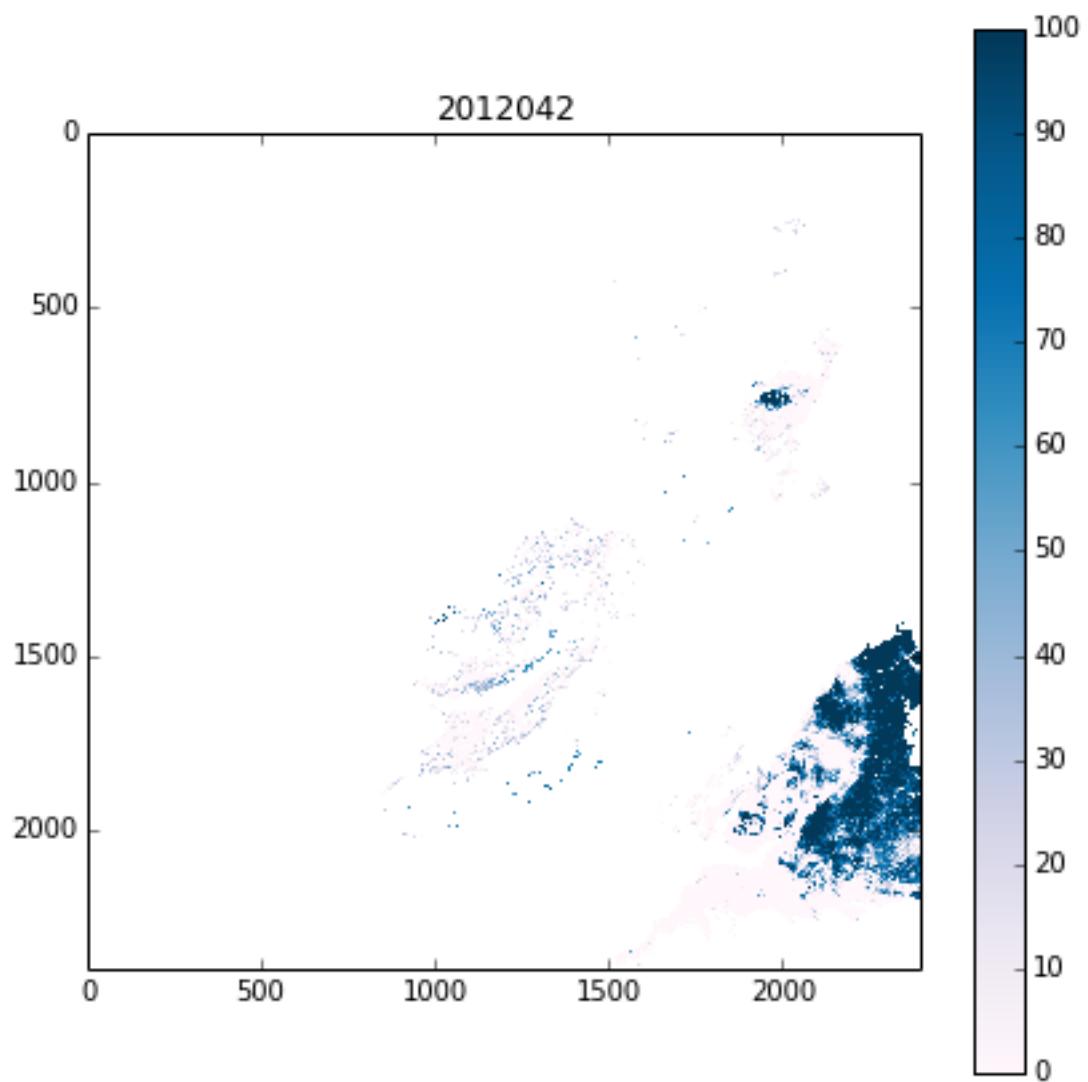


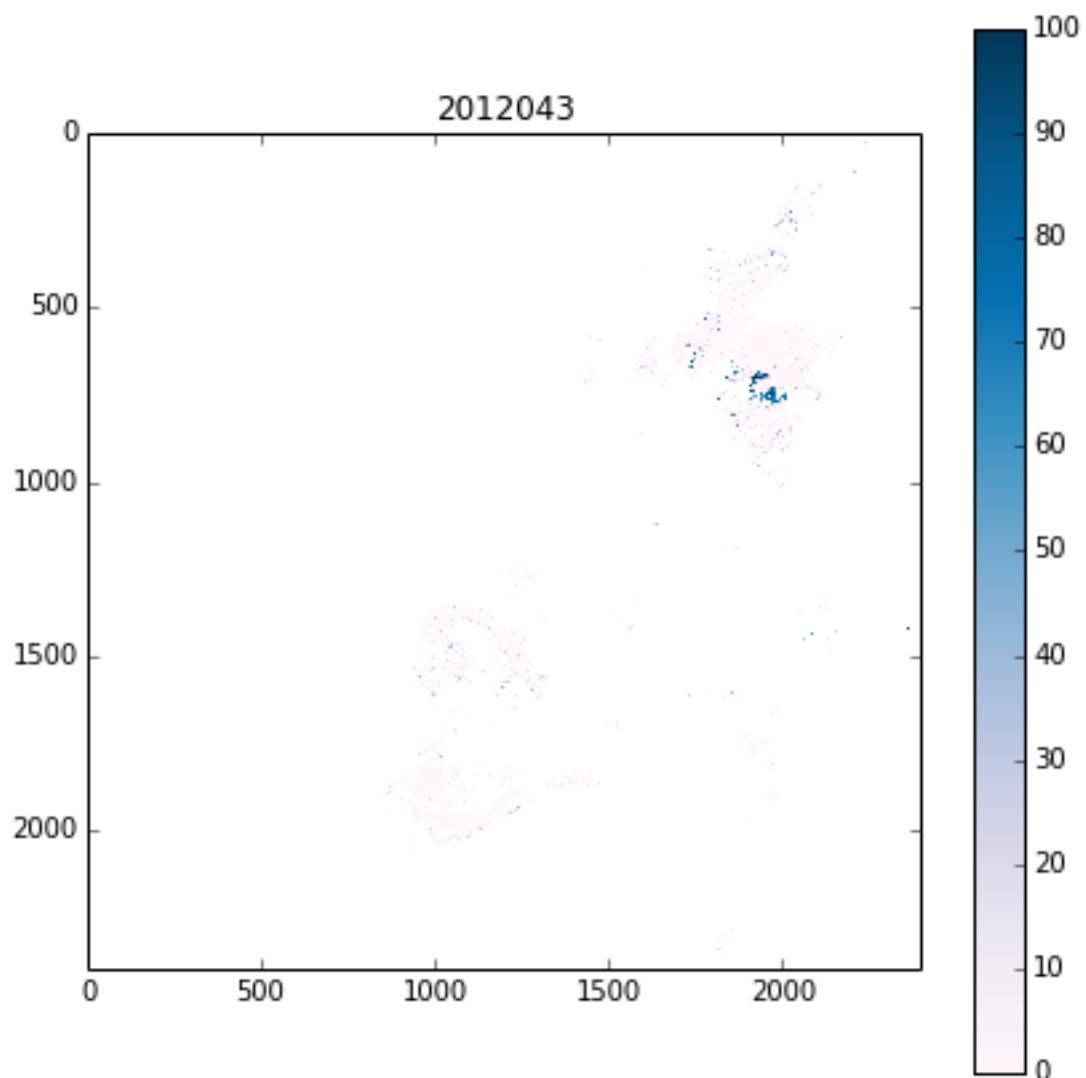


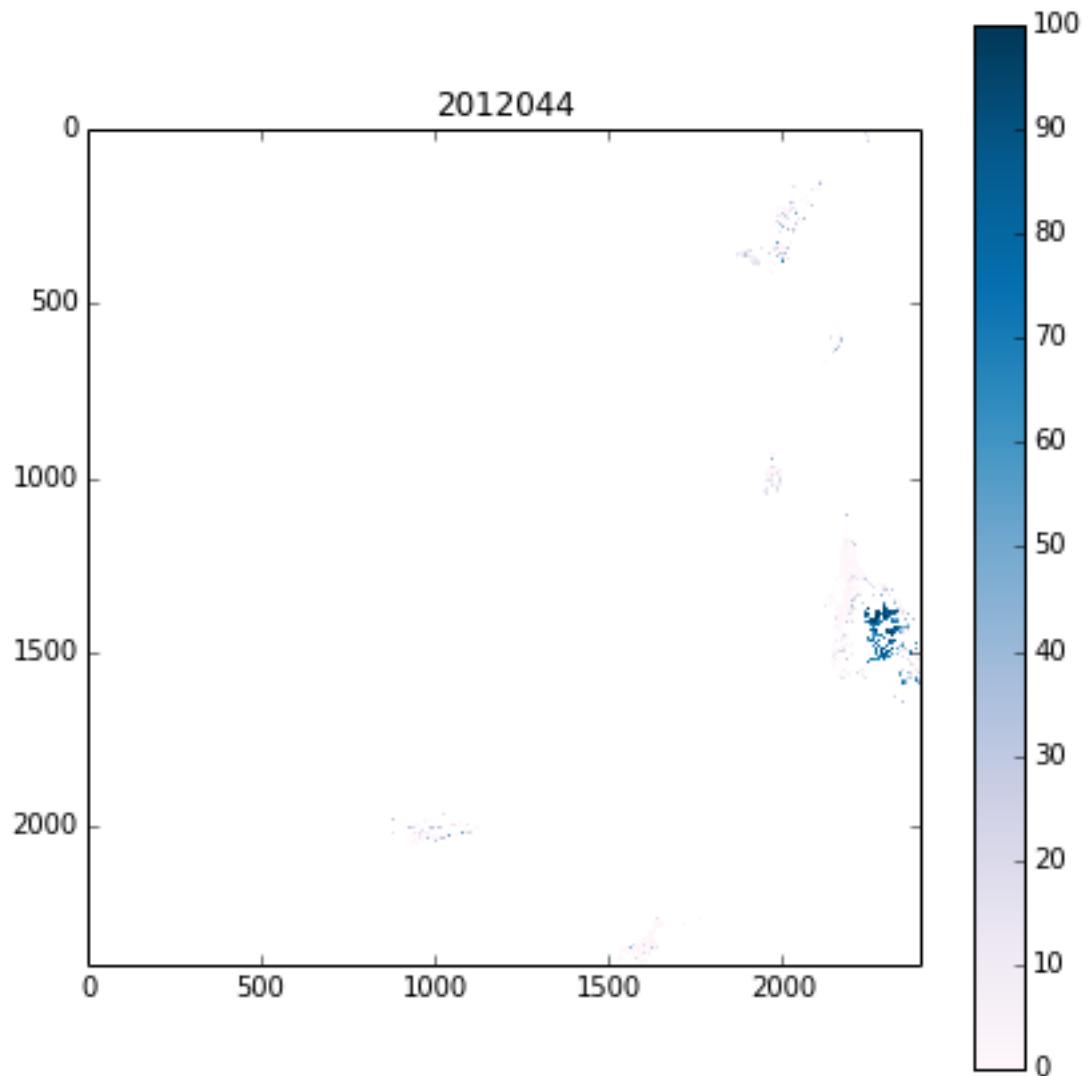


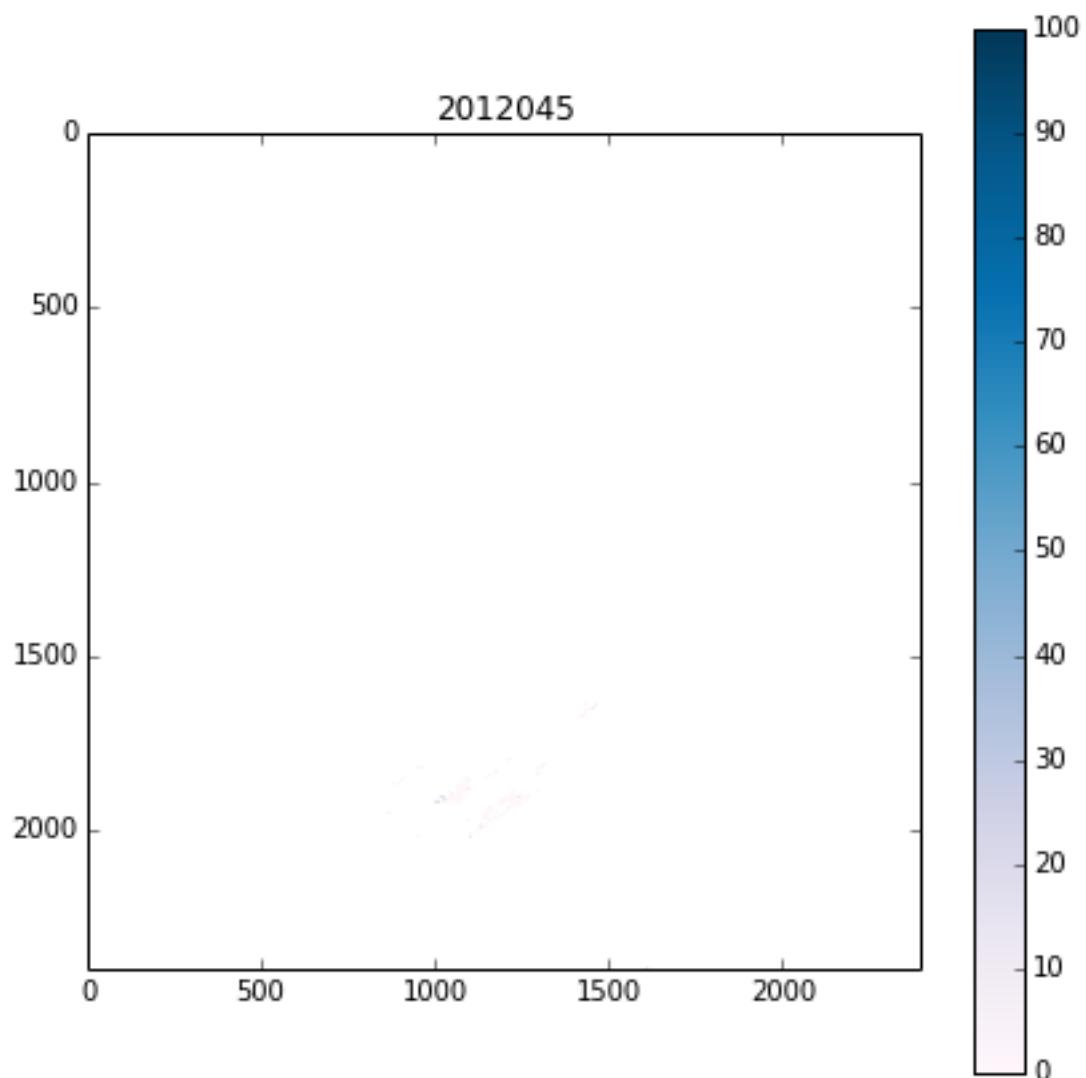


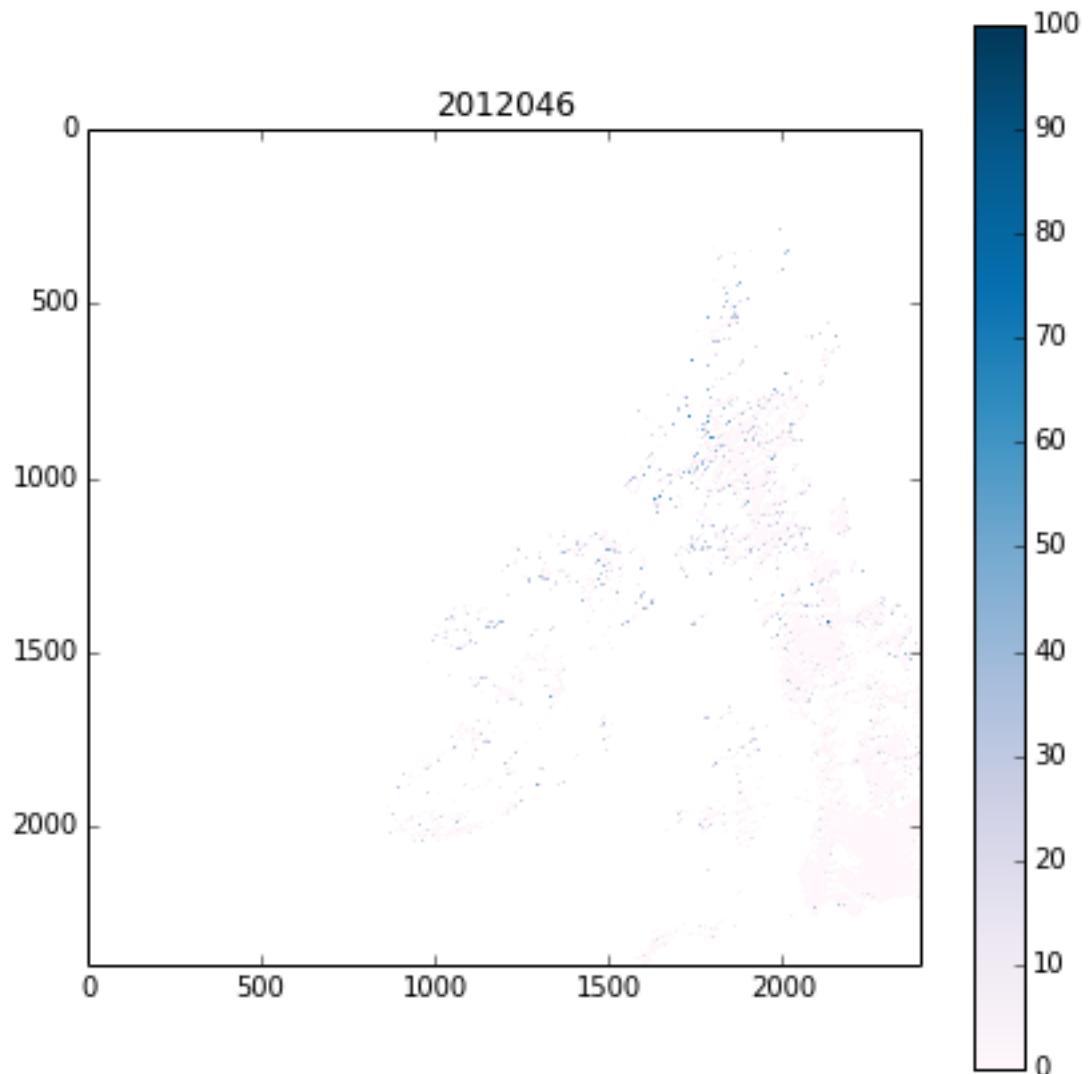


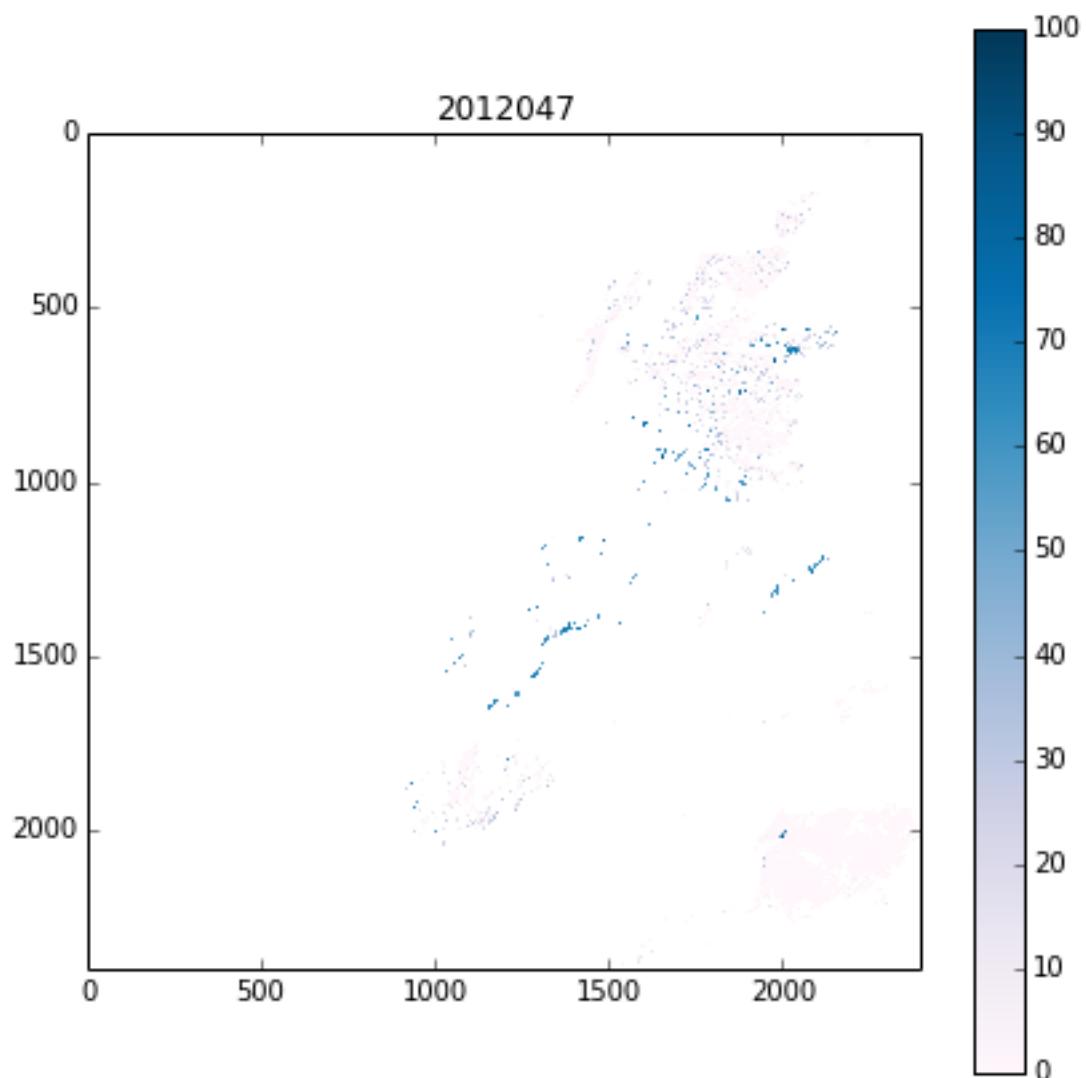


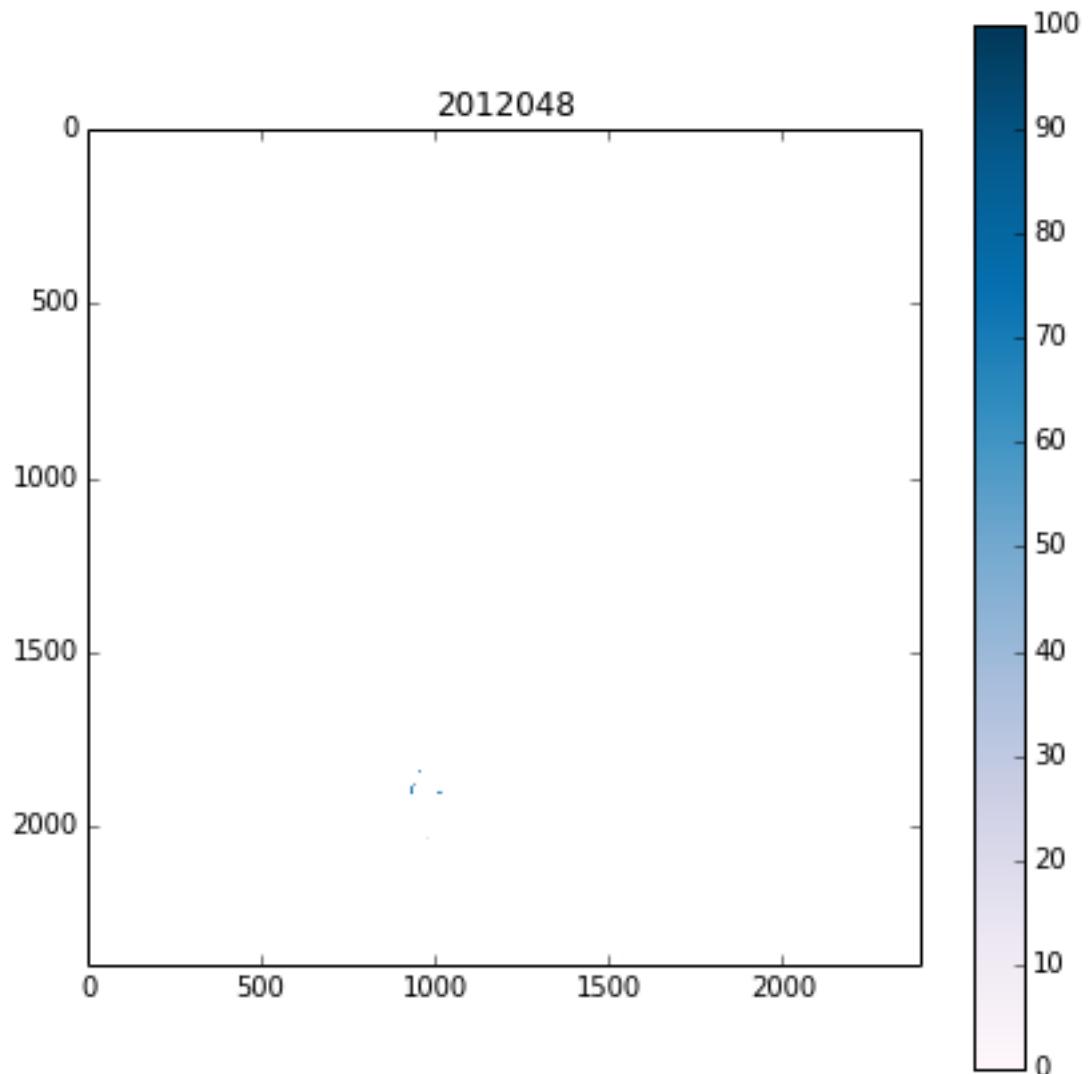


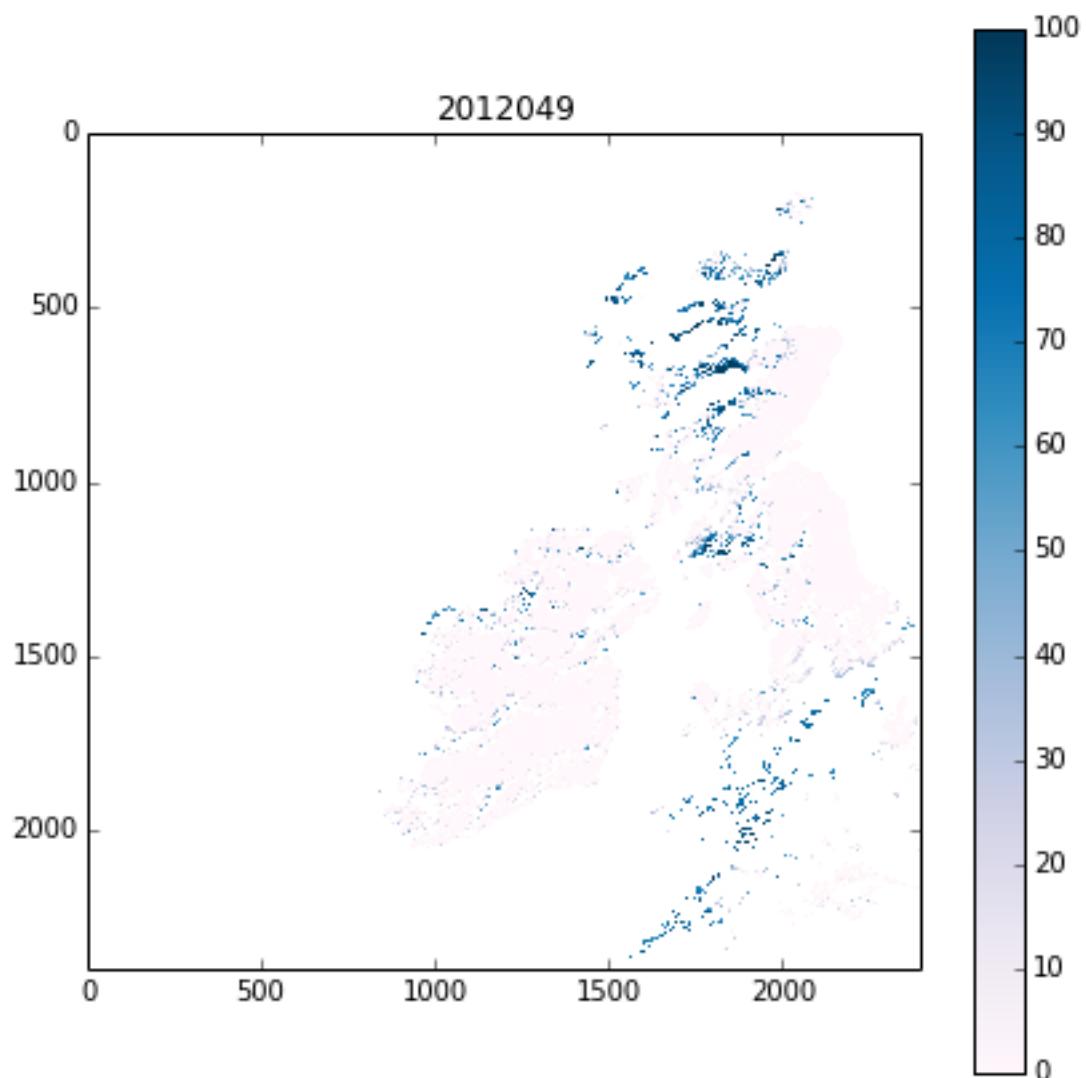


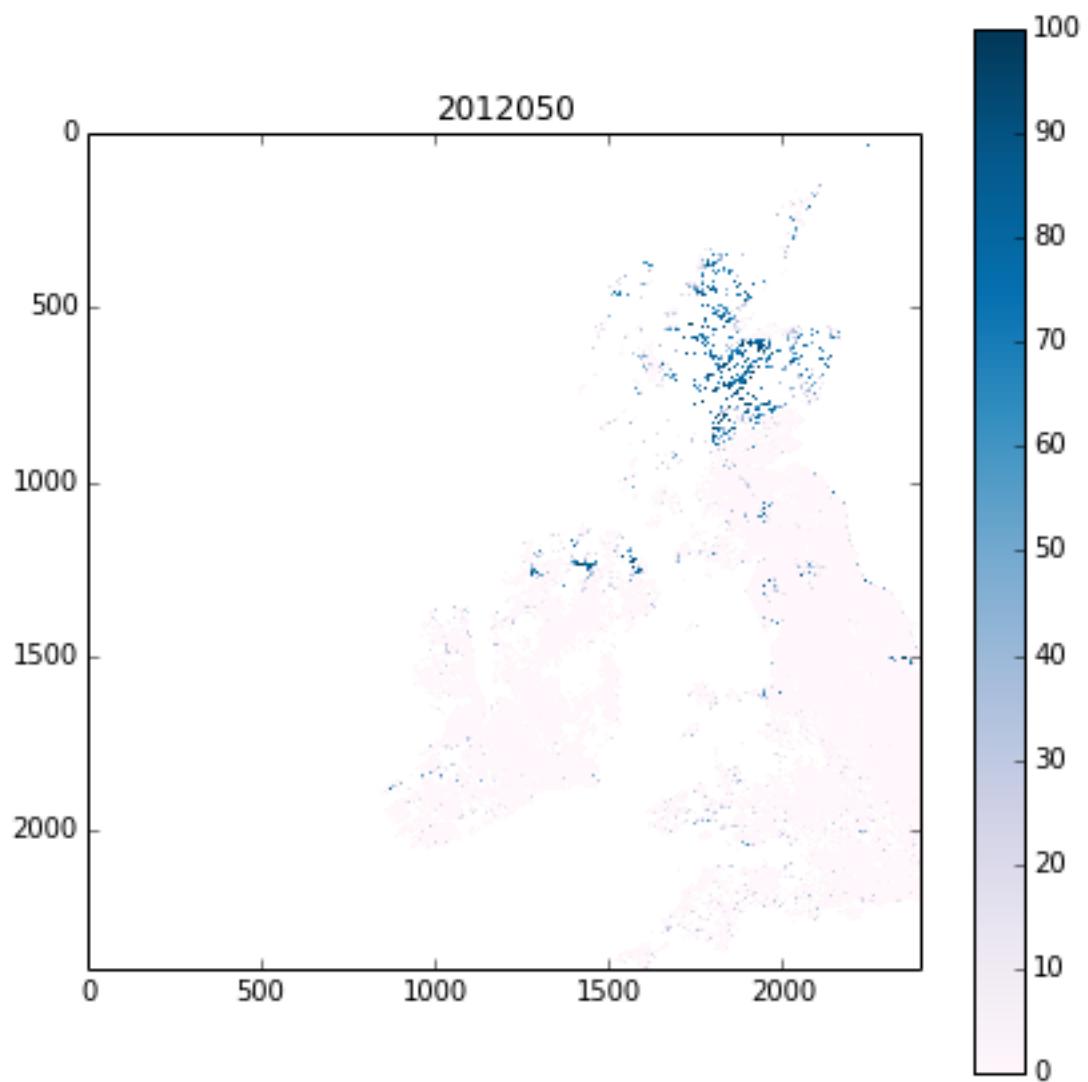


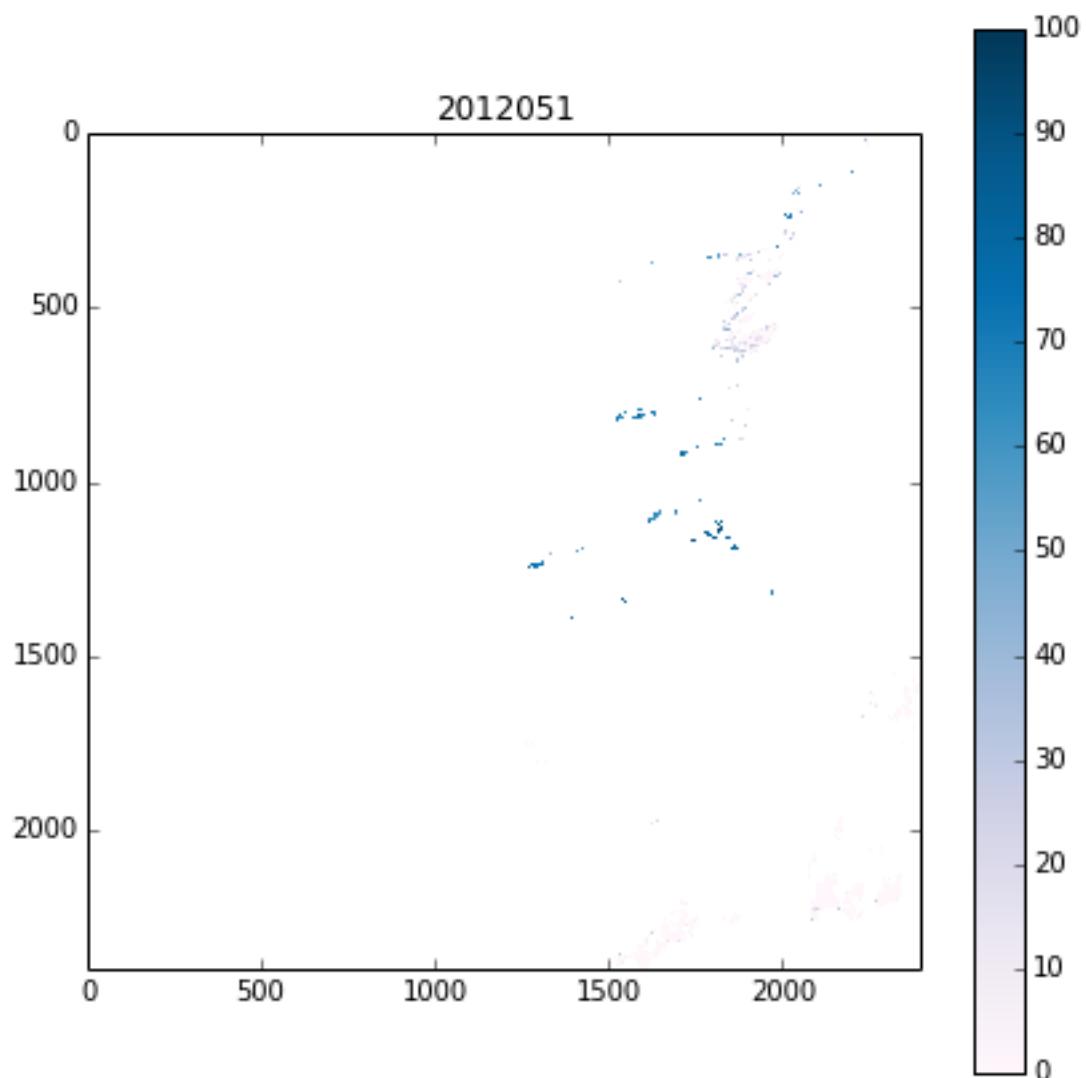


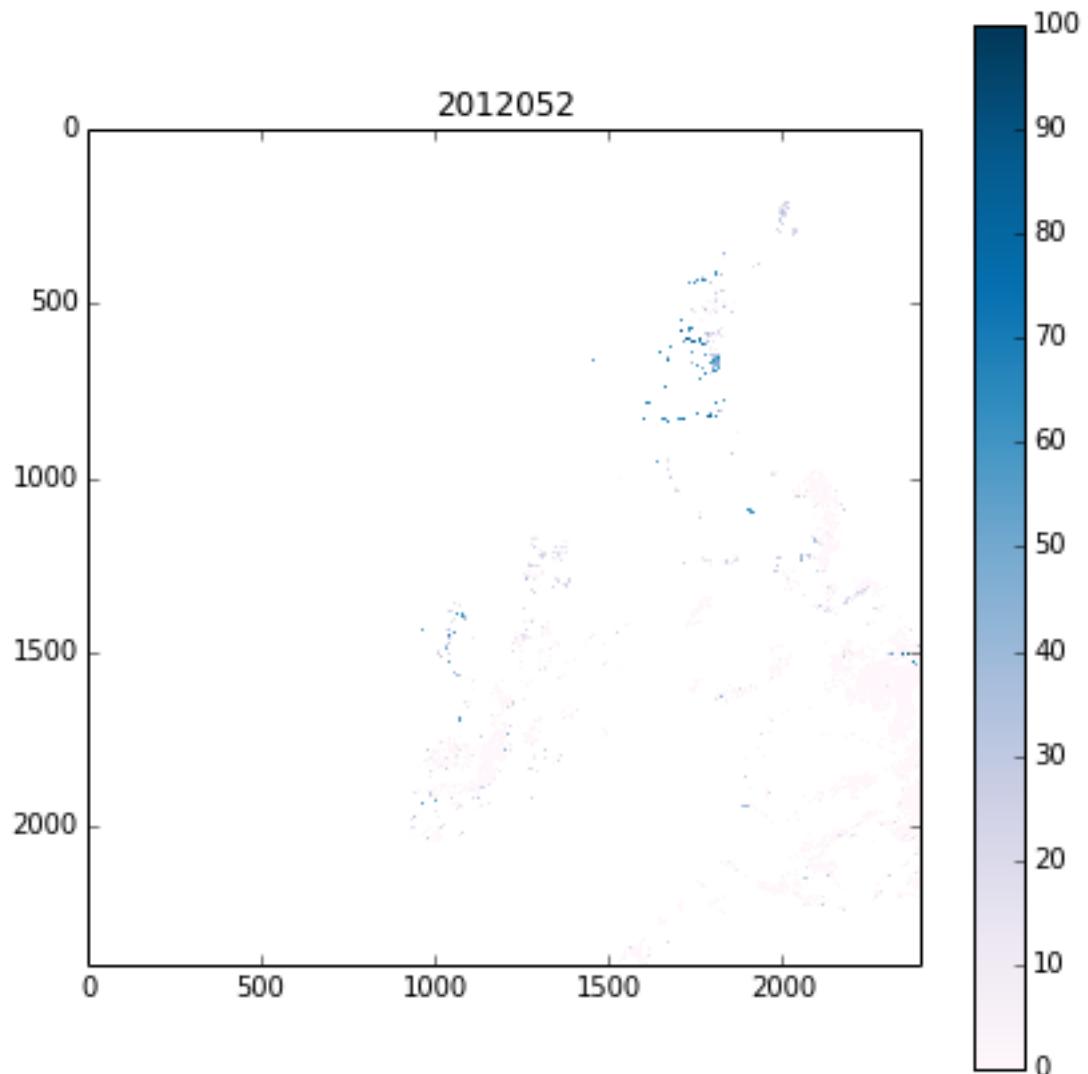


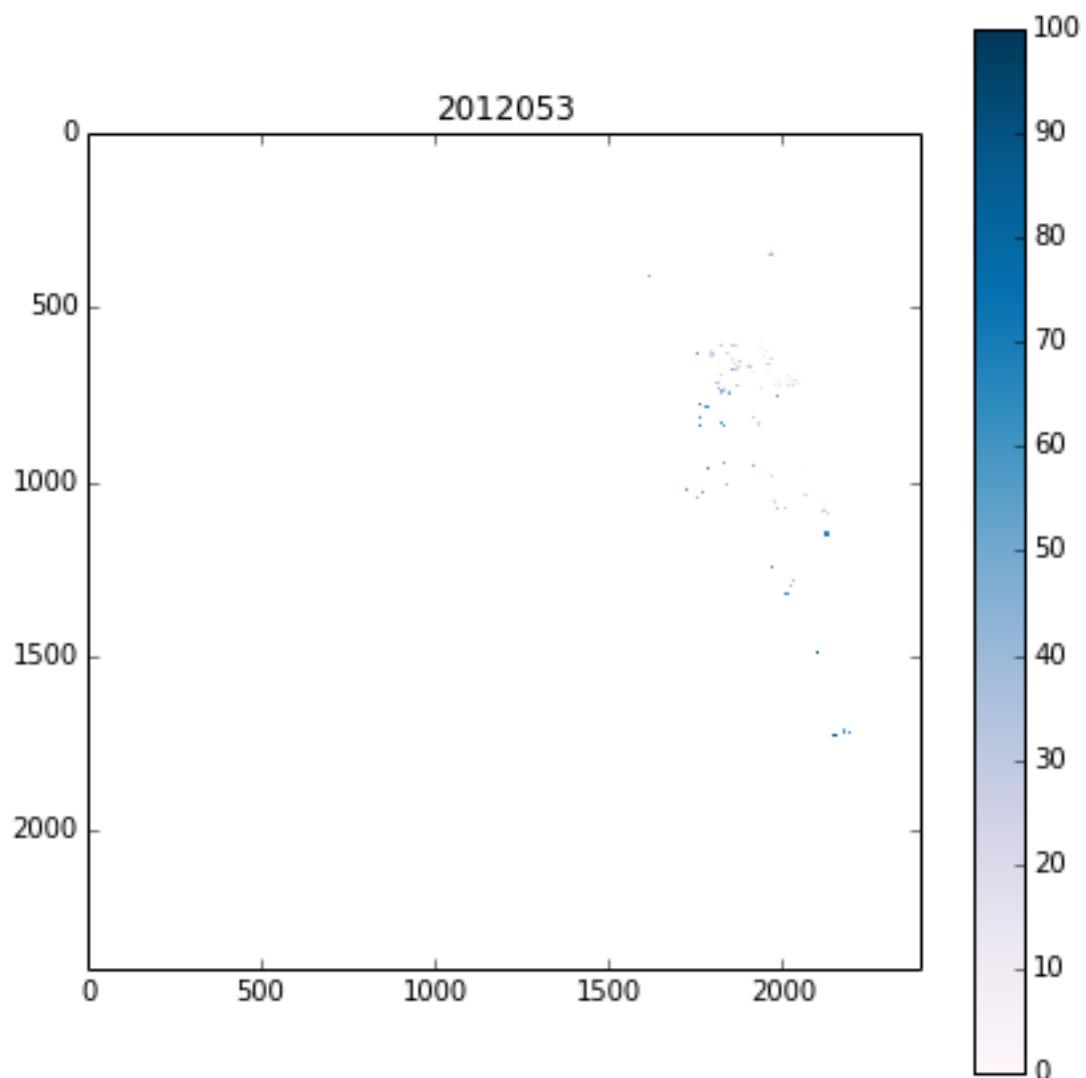


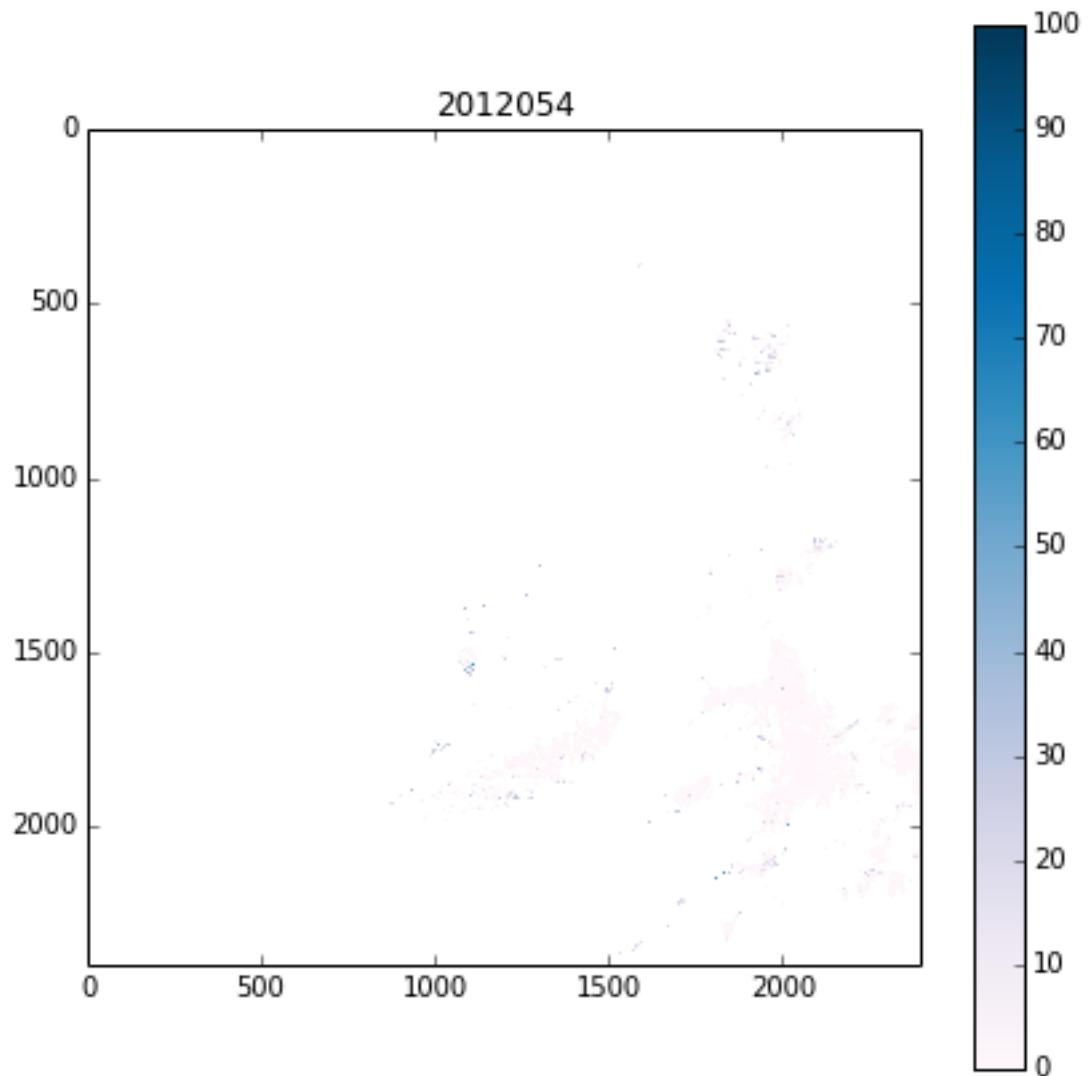


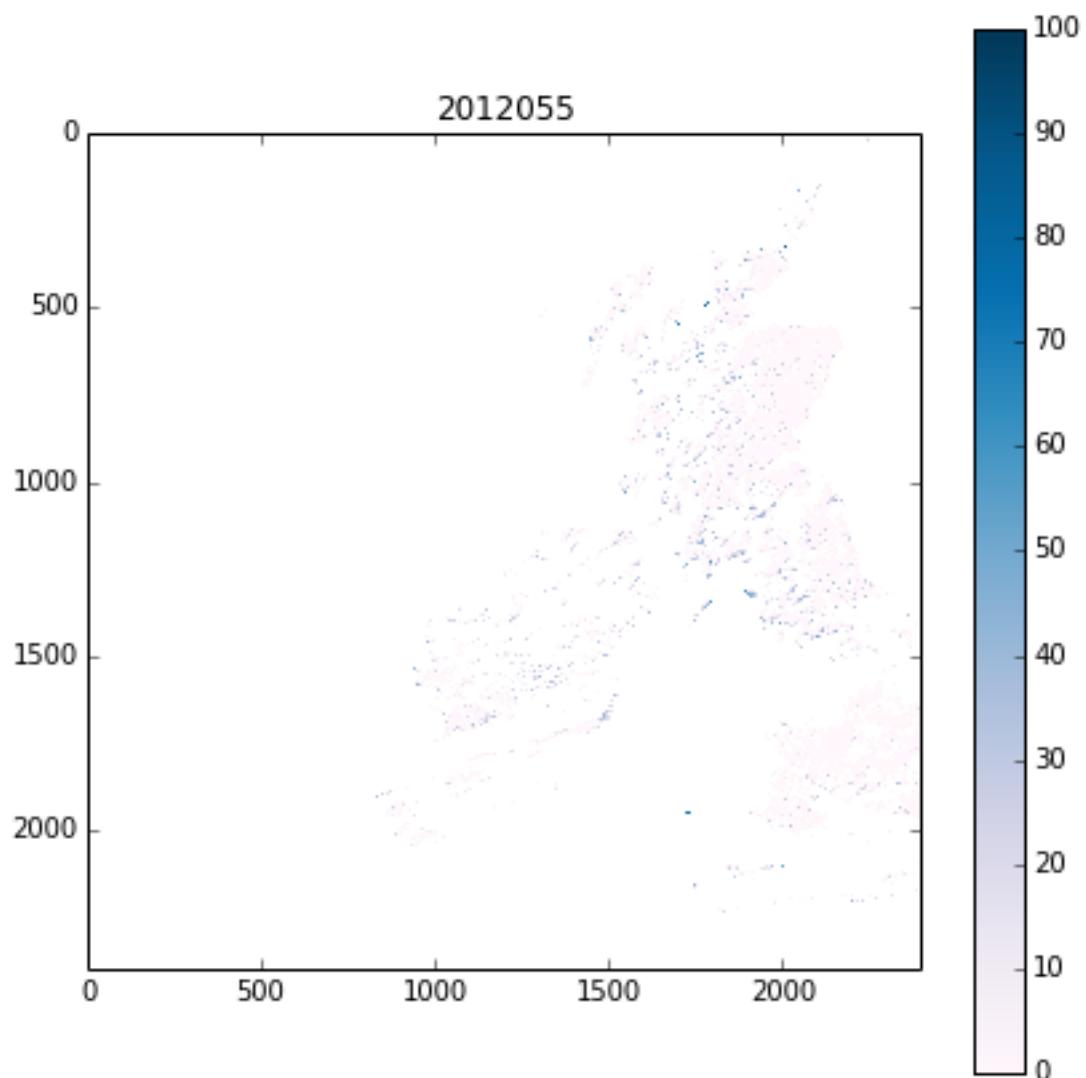


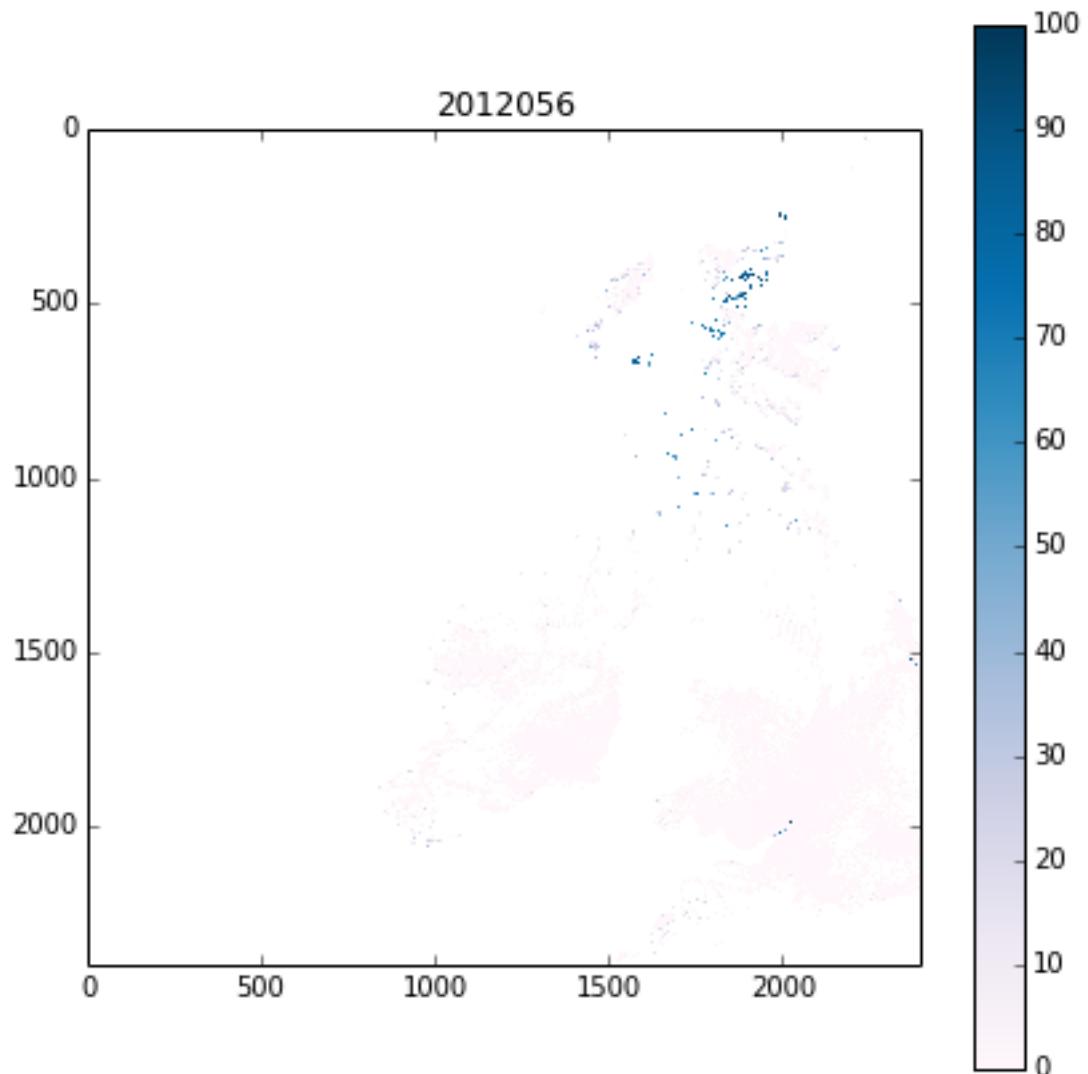


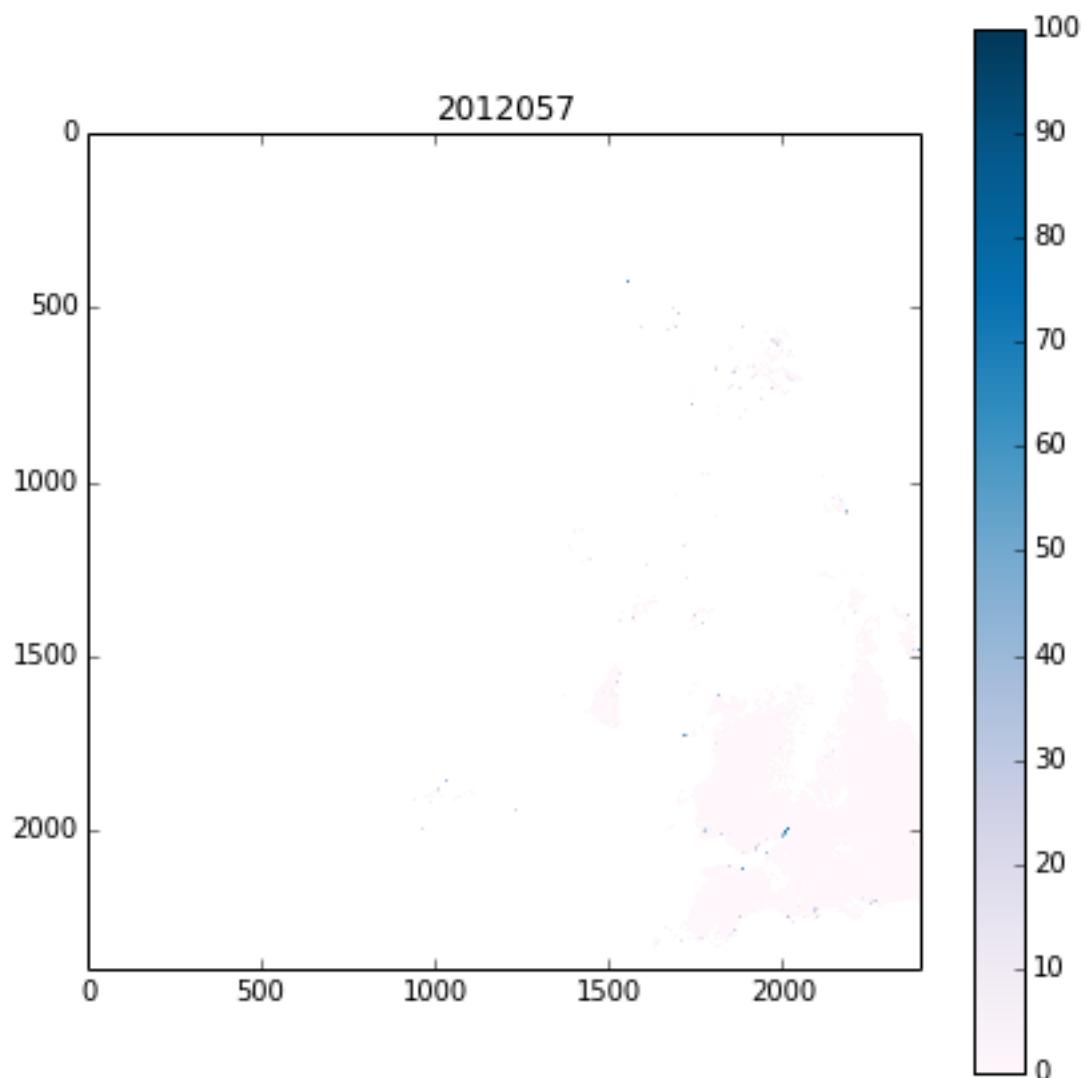


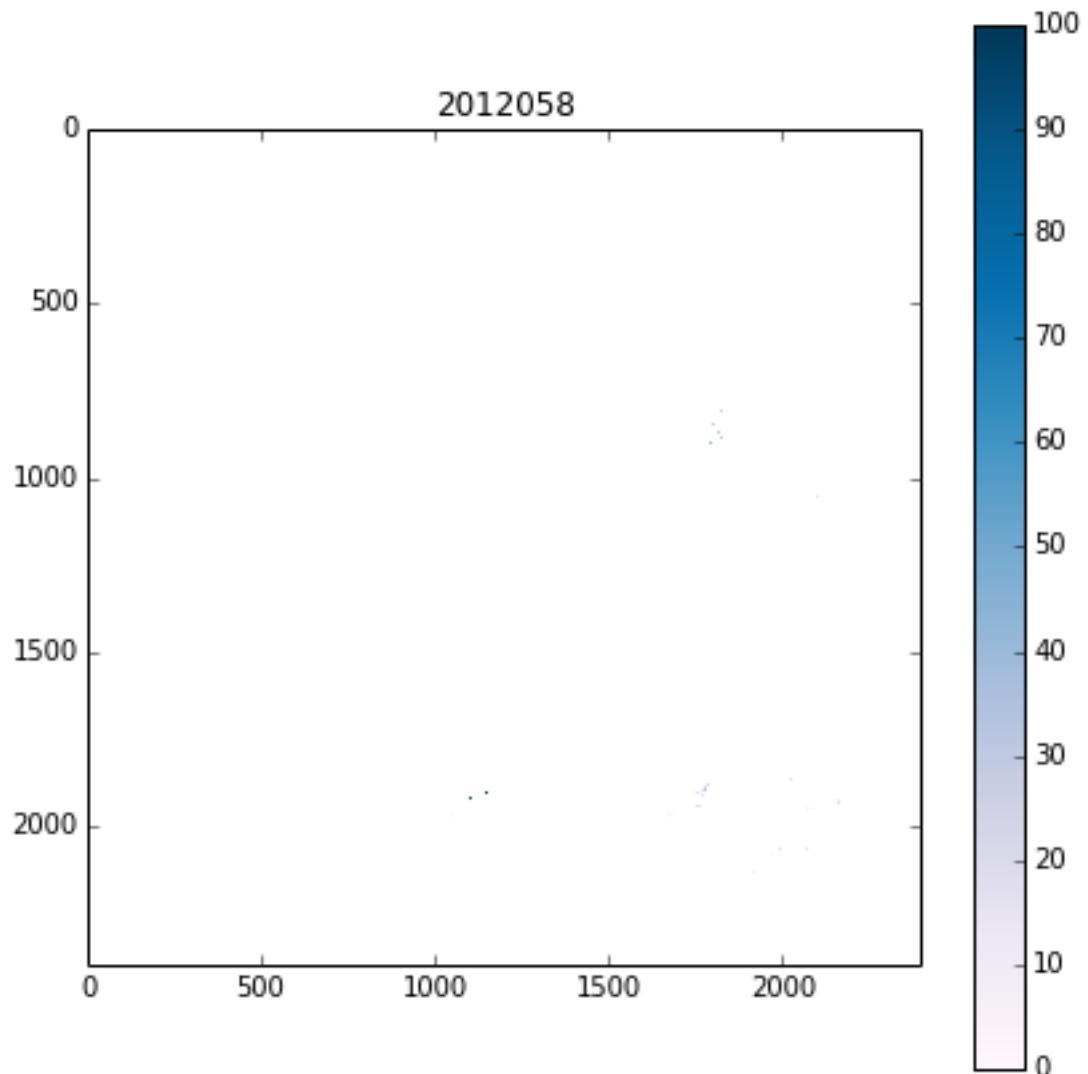


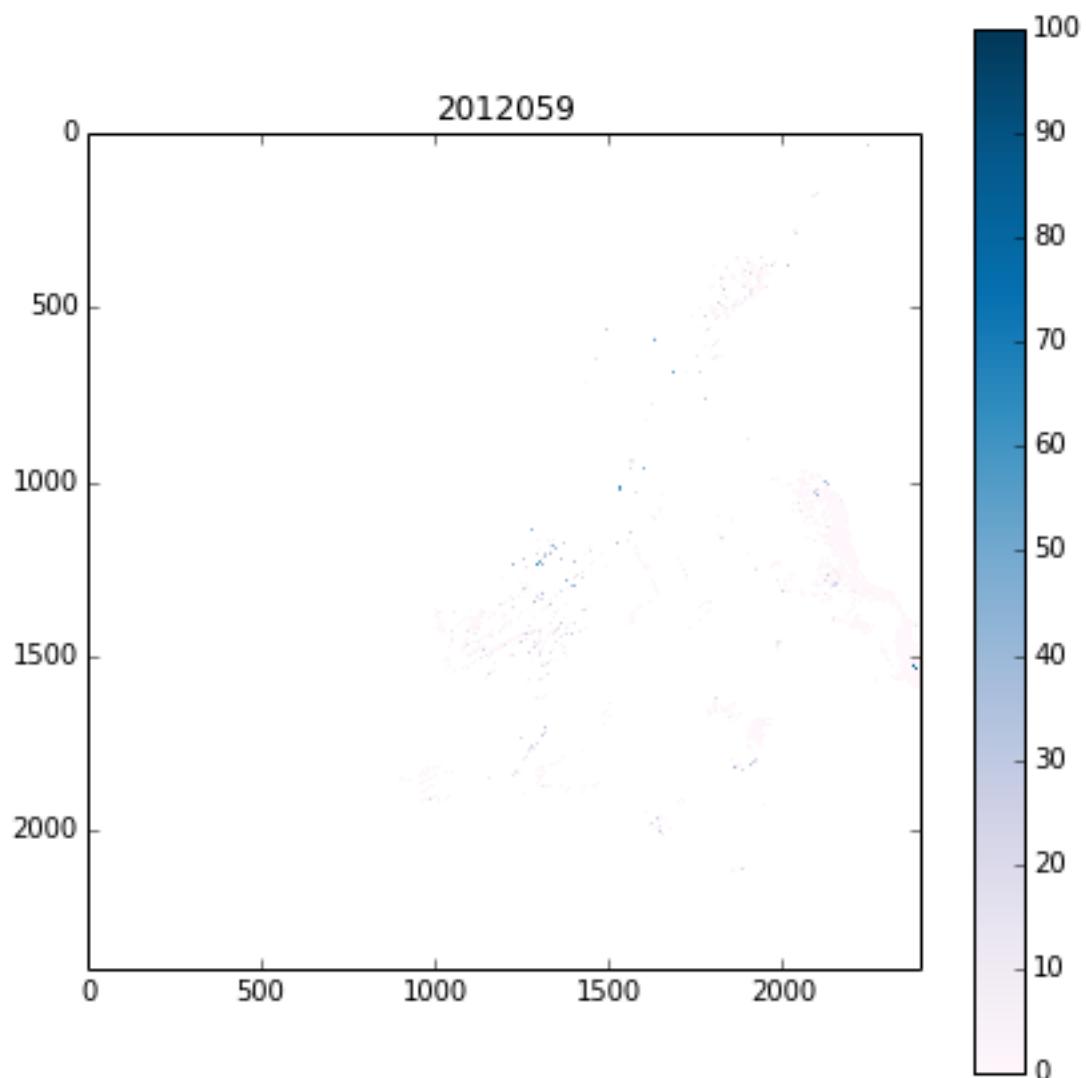


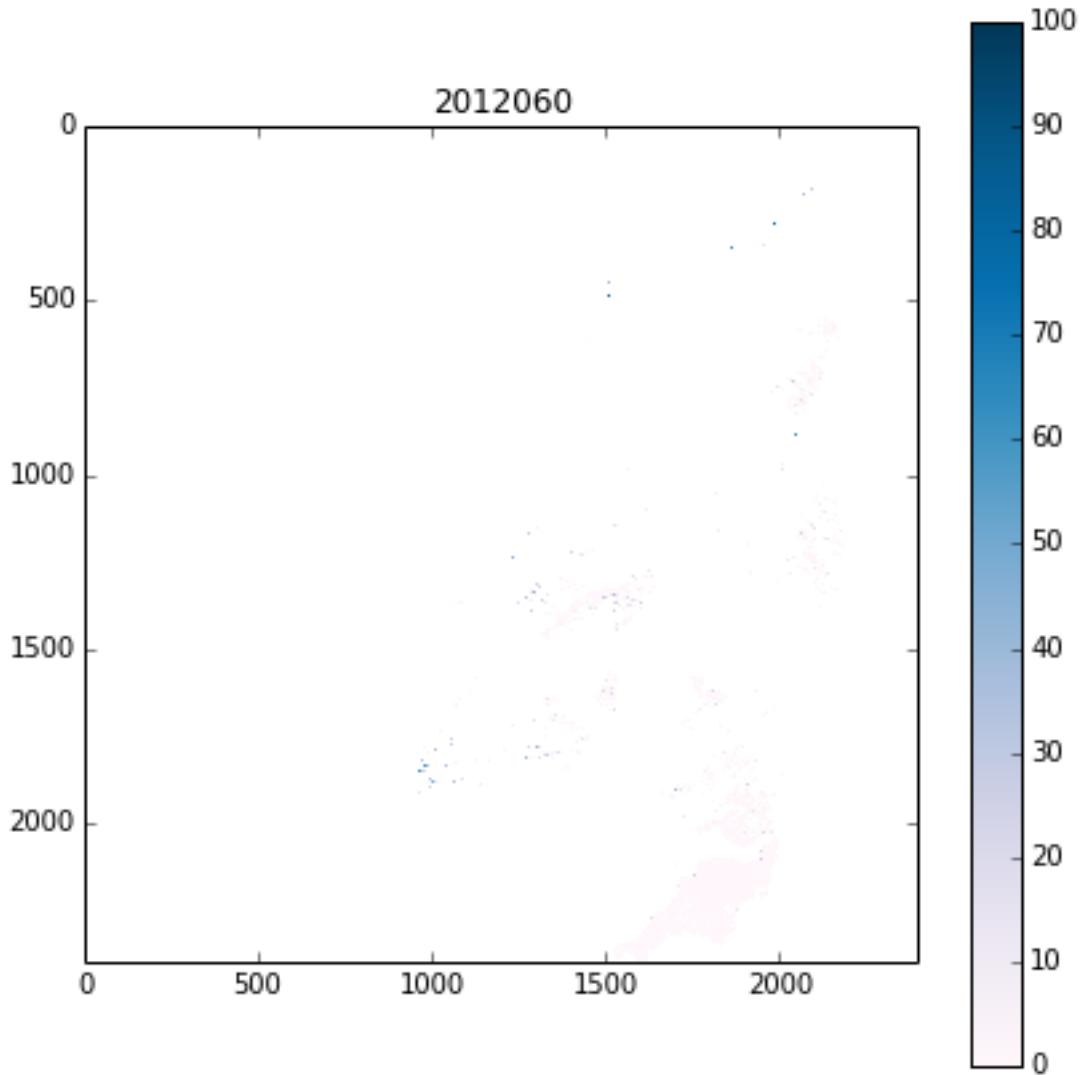












```
# now make a movie ...

import os

cmd = 'convert -delay 100 -loop 0 files/images/snow_uk_*.jpg files/images/snow_uk2.gif'
os.system(cmd)

0
```

41.7 Exercise 4.3

Again, you should be used to this sort of thing by now.

We have the following code to base this on:

```
import sys
sys.path.insert(0,'files/python')
from raster_mask import raster_mask, getLAI

# test this on an LAI file
```

```

# the data file name
filename = 'files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf'

# a layer (doesn't matter so much which: use for geometry info)
layer = 'Lai_1km'
# the full dataset specification
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%(filename,layer)

# make a raster mask
# from the layer IRELAND in world.shp
mask = raster_mask(file_spec,\n                    target_vector_file = "files/data/world.shp",\n                    attribute_filter = "NAME = 'IRELAND'")

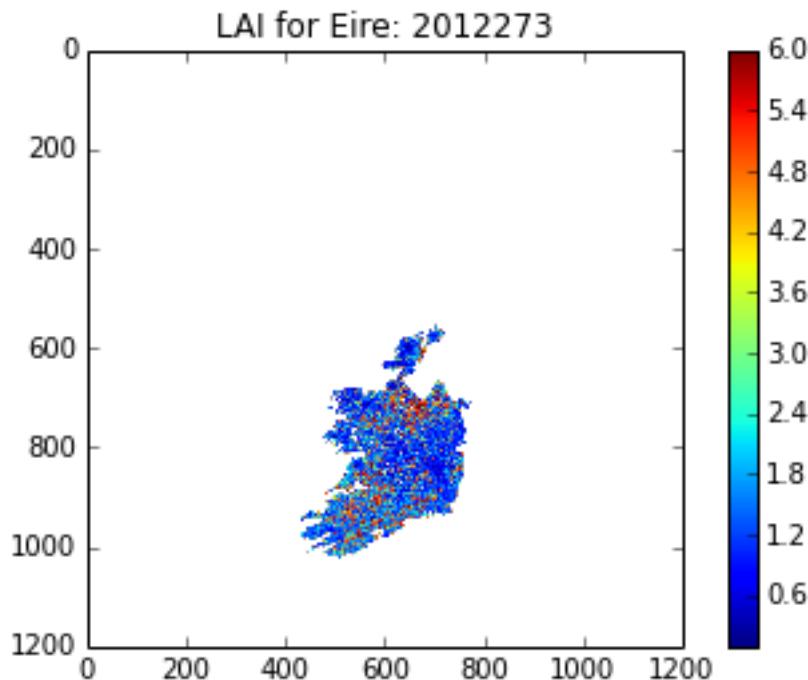
# get the LAI data
data = getLAI(filename)

# reset the data mask
# 'mask' is True for Ireland
# so take the opposite
data['Lai_1km'] = ma.array(data['Lai_1km'],mask=mask)
data['LaiStdDev_1km'] = ma.array(data['Lai_1km'],mask=mask)

plt.title('LAI for Eire: 2012273')
plt.imshow(data['Lai_1km'],vmax=6)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x47fc61b8>

```



Obviously, we will need to specify a set of filenames again.

These have a simple filename pattern, so `glob` would be appropriate (don't forget to `sort`):

```

from glob import glob

filenames = sort(glob('files/data/MCD15A2.A2012???.h17v03.005.*.hdf'))

```

```
print filenames

['files/data/MCD15A2.A2012001.h17v03.005.2012017211237.hdf',
 'files/data/MCD15A2.A2012009.h17v03.005.2012019044037.hdf',
 'files/data/MCD15A2.A2012017.h17v03.005.2012026072526.hdf',
 'files/data/MCD15A2.A2012025.h17v03.005.2012052124839.hdf',
 'files/data/MCD15A2.A2012033.h17v03.005.2012042060649.hdf',
 'files/data/MCD15A2.A2012041.h17v03.005.2012050092057.hdf',
 'files/data/MCD15A2.A2012049.h17v03.005.2012068144447.hdf',
 'files/data/MCD15A2.A2012057.h17v03.005.2012068140544.hdf',
 'files/data/MCD15A2.A2012065.h17v03.005.2012075021749.hdf',
 'files/data/MCD15A2.A2012073.h17v03.005.2012083010304.hdf',
 'files/data/MCD15A2.A2012081.h17v03.005.2012090131602.hdf',
 'files/data/MCD15A2.A2012089.h17v03.005.2012107201245.hdf',
 'files/data/MCD15A2.A2012097.h17v03.005.2012108125047.hdf',
 'files/data/MCD15A2.A2012105.h17v03.005.2012116125519.hdf',
 'files/data/MCD15A2.A2012113.h17v03.005.2012122072153.hdf',
 'files/data/MCD15A2.A2012121.h17v03.005.2012137221611.hdf',
 'files/data/MCD15A2.A2012129.h17v03.005.2012142001241.hdf',
 'files/data/MCD15A2.A2012137.h17v03.005.2012153021910.hdf',
 'files/data/MCD15A2.A2012145.h17v03.005.2012160130927.hdf',
 'files/data/MCD15A2.A2012153.h17v03.005.2012166161748.hdf',
 'files/data/MCD15A2.A2012161.h17v03.005.2012170080216.hdf',
 'files/data/MCD15A2.A2012169.h17v03.005.2012181134242.hdf',
 'files/data/MCD15A2.A2012177.h17v03.005.2012188150145.hdf',
 'files/data/MCD15A2.A2012185.h17v03.005.2012208181105.hdf',
 'files/data/MCD15A2.A2012193.h17v03.005.2012202144013.hdf',
 'files/data/MCD15A2.A2012201.h17v03.005.2012215131931.hdf',
 'files/data/MCD15A2.A2012209.h17v03.005.2012219144450.hdf',
 'files/data/MCD15A2.A2012217.h17v03.005.2012228215213.hdf',
 'files/data/MCD15A2.A2012225.h17v03.005.2012234105932.hdf',
 'files/data/MCD15A2.A2012233.h17v03.005.2012242093511.hdf',
 'files/data/MCD15A2.A2012241.h17v03.005.2012250182515.hdf',
 'files/data/MCD15A2.A2012249.h17v03.005.2012261231425.hdf',
 'files/data/MCD15A2.A2012257.h17v03.005.2012270114223.hdf',
 'files/data/MCD15A2.A2012265.h17v03.005.2012276134731.hdf',
 'files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf',
 'files/data/MCD15A2.A2012281.h17v03.005.2012297135831.hdf',
 'files/data/MCD15A2.A2012289.h17v03.005.2012299194634.hdf',
 'files/data/MCD15A2.A2012297.h17v03.005.2012306163257.hdf',
 'files/data/MCD15A2.A2012305.h17v03.005.2012314140451.hdf',
 'files/data/MCD15A2.A2012313.h17v03.005.2012322095802.hdf',
 'files/data/MCD15A2.A2012321.h17v03.005.2012335133638.hdf',
 'files/data/MCD15A2.A2012329.h17v03.005.2012340181739.hdf',
 'files/data/MCD15A2.A2012337.h17v03.005.2012346165133.hdf',
 'files/data/MCD15A2.A2012345.h17v03.005.2012356133200.hdf',
 'files/data/MCD15A2.A2012353.h17v03.005.2012363125132.hdf',
 'files/data/MCD15A2.A2012361.h17v03.005.2013007202756.hdf']
```

Now we just build a loop around that:

```
import sys
sys.path.insert(0,'files/python')
from raster_mask import raster_mask, getLAI
from glob import glob

# a layer (doesn't matter so much which: use for geometry info)
layer = 'Lai_1km'
# the full dataset specification

file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'

# list of filenames
```

```

filenames = sort(glob('files/data/MCD15A2.A2012???.h17v03.005.*.hdf'))

# build a mask: use filenames[0] for geometry
print 'building mask ...'
mask = raster_mask(file_template%(filenames[0],layer), \
                    target_vector_file = "files/data/world.shp", \
                    attribute_filter = "NAME = 'IRELAND'")

lai = []
print 'looping over files...'
for filename in filenames:
    # get the LAI data
    print filename
    data = getLAI(filename)

    # reset the data mask
    lai.append(ma.array(data['Lai_1km'],mask=mask))

# convert to masked array
lai = ma.array(lai)

building mask ...
looping over files...
files/data/MCD15A2.A2012001.h17v03.005.2012017211237.hdf
files/data/MCD15A2.A2012009.h17v03.005.2012019044037.hdf
files/data/MCD15A2.A2012017.h17v03.005.2012026072526.hdf
files/data/MCD15A2.A2012025.h17v03.005.2012052124839.hdf
files/data/MCD15A2.A2012033.h17v03.005.2012042060649.hdf
files/data/MCD15A2.A2012041.h17v03.005.2012050092057.hdf
files/data/MCD15A2.A2012049.h17v03.005.2012068144447.hdf
files/data/MCD15A2.A2012057.h17v03.005.2012068140544.hdf
files/data/MCD15A2.A2012065.h17v03.005.2012075021749.hdf
files/data/MCD15A2.A2012073.h17v03.005.2012083010304.hdf
files/data/MCD15A2.A2012081.h17v03.005.2012090131602.hdf
files/data/MCD15A2.A2012089.h17v03.005.2012107201245.hdf
files/data/MCD15A2.A2012097.h17v03.005.2012108125047.hdf
files/data/MCD15A2.A2012105.h17v03.005.2012116125519.hdf
files/data/MCD15A2.A2012113.h17v03.005.2012122072153.hdf
files/data/MCD15A2.A2012121.h17v03.005.2012137221611.hdf
files/data/MCD15A2.A2012129.h17v03.005.2012142001241.hdf
files/data/MCD15A2.A2012137.h17v03.005.2012153021910.hdf
files/data/MCD15A2.A2012145.h17v03.005.2012160130927.hdf
files/data/MCD15A2.A2012153.h17v03.005.2012166161748.hdf
files/data/MCD15A2.A2012161.h17v03.005.2012170080216.hdf
files/data/MCD15A2.A2012169.h17v03.005.2012181134242.hdf
files/data/MCD15A2.A2012177.h17v03.005.2012188150145.hdf
files/data/MCD15A2.A2012185.h17v03.005.2012208181105.hdf
files/data/MCD15A2.A2012193.h17v03.005.2012202144013.hdf
files/data/MCD15A2.A2012201.h17v03.005.2012215131931.hdf
files/data/MCD15A2.A2012209.h17v03.005.2012219144450.hdf
files/data/MCD15A2.A2012217.h17v03.005.2012228215213.hdf
files/data/MCD15A2.A2012225.h17v03.005.2012234105932.hdf
files/data/MCD15A2.A2012233.h17v03.005.2012242093511.hdf
files/data/MCD15A2.A2012241.h17v03.005.2012250182515.hdf
files/data/MCD15A2.A2012249.h17v03.005.2012261231425.hdf
files/data/MCD15A2.A2012257.h17v03.005.2012270114223.hdf
files/data/MCD15A2.A2012265.h17v03.005.2012276134731.hdf
files/data/MCD15A2.A2012273.h17v03.005.2012297134400.hdf
files/data/MCD15A2.A2012281.h17v03.005.2012297135831.hdf
files/data/MCD15A2.A2012289.h17v03.005.2012299194634.hdf
files/data/MCD15A2.A2012297.h17v03.005.2012306163257.hdf
files/data/MCD15A2.A2012305.h17v03.005.2012314140451.hdf
files/data/MCD15A2.A2012313.h17v03.005.2012322095802.hdf

```

```
files/data/MCD15A2.A2012321.h17v03.005.2012335133638.hdf
files/data/MCD15A2.A2012329.h17v03.005.2012340181739.hdf
files/data/MCD15A2.A2012337.h17v03.005.2012346165133.hdf
files/data/MCD15A2.A2012345.h17v03.005.2012356133200.hdf
files/data/MCD15A2.A2012353.h17v03.005.2012363125132.hdf
files/data/MCD15A2.A2012361.h17v03.005.2013007202756.hdf

# make sure we can load it, then plot it
import numpy.ma as ma
import numpy as np

# now plot and save images
# plot the data
import pylab as plt

cmap = plt.cm.Greens
#plt.ioff()

for i,f in enumerate(filenames):
    fig = plt.figure(figsize=(7,7))
    plt.imshow(lai[i],cmap=cmap,interpolation='none',vmin=0.,vmax=6.)
    # remember filenames of the form
    # files/data/MCD15A2.A2011185.h09v05.005.20111213154534.hdf'
    file_id = f.split('/')[-1].split('.')[5][1:]
    print file_id
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('files/images/lai_eire_%s.jpg'%file_id)
    plt.close(fig)

2012001
2012009
2012017
2012025
2012033
2012041
2012049
2012057
2012065
2012073
2012081
2012089
2012097
2012105
2012113
2012121
2012129
2012137
2012145
2012153
2012161
2012169
2012177
2012185
2012193
2012201
2012209
2012217
2012225
2012233
2012241
```

```

2012249
2012257
2012265
2012273
2012281
2012289
2012297
2012305
2012313
2012321
2012329
2012337
2012345
2012353
2012361

# now make a movie ...

import os

cmd = 'convert -delay 100 -loop 0 files/images/lai_eire_*.jpg files/images/lai_eire.gif'
os.system(cmd)

0

```

It should be obvious how to adapt this to generate a similar LAI Std Dev dataset.

An average is simply obtained:

```

try:
    lai_mean = lai.mean(axis=(1,2))
except:
    pass

```

Though in older versions of numpy, you have to do each axis separately:

```

np.array(lai.mean(axis=1).mean(axis=1))

array([ 0.          ,  0.          ,  0.          ,  1.13194208,  0.8385127 ,
       0.7581951 ,  1.62490352,  1.77427215,  1.68431605,  1.53881657,
       2.46867764,  2.22713124,  1.97079892,  2.19423149,  1.83186409,
       1.68775959,  2.5530641 ,  1.82426569,  3.350427  ,  1.79795768,
       2.29830478,  2.30001168,  1.86932879,  1.74646726,  2.21955 ,
       2.02689304,  2.61823983,  2.63460711,  2.18868341,  2.2350088 ,
       2.10244737,  2.64785816,  2.38064128,  1.93491947,  2.01097991,
       1.92061334,  1.6725845 ,  1.83265805,  1.70210771,  0.99212715,
       0.          ,  0.          ,  0.          ,  0.          ,  0.          ,  0.          ])

```

There is plenty to criticise with such an average though: it takes no account of the uncertainty in each data value (that is known), and it takes no account of data gaps.

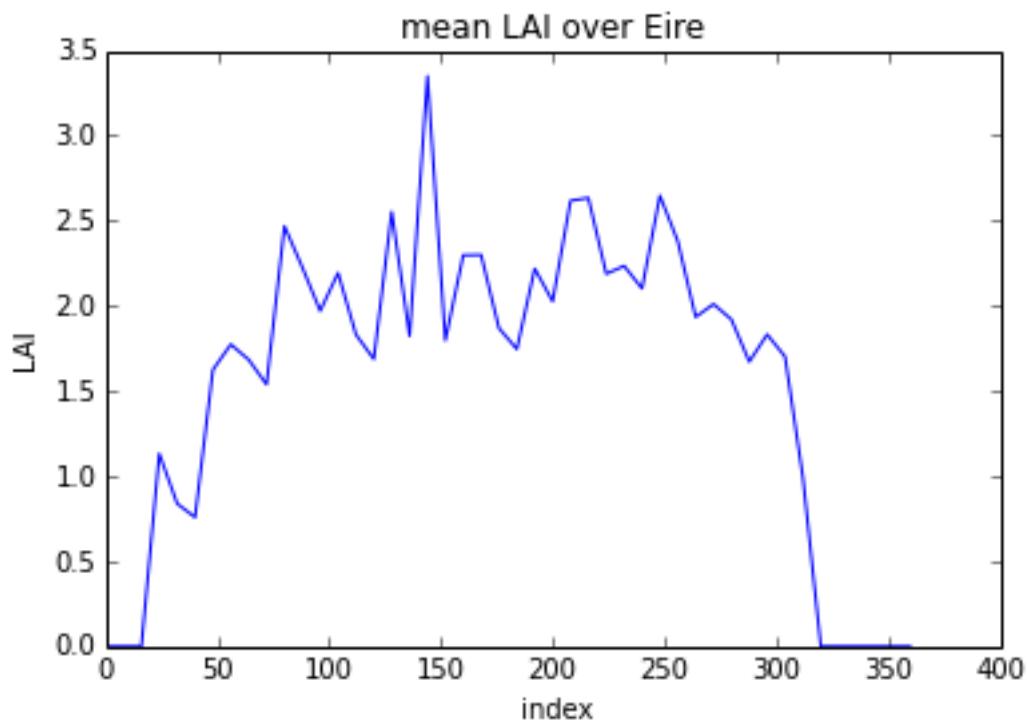
We will return to these issues next week.

```

plt.plot(np.arange(lai.shape[0])*8,np.array(lai.mean(axis=1).mean(axis=1)))
plt.title('mean LAI over Eire')
plt.xlabel(' index')
plt.ylabel('LAI')

<matplotlib.text.Text at 0x6fa33490>

```



CHAPTER
FORTYTWO

5. FUNCTION FITTING AND INTERPOLATION

In today's session, we will be using some of the LAI datasets we examined last week (masked by national boundaries) and doing some analysis on them.

- 5.1 Making 3D datasets and Movies First, we will examine how to improve our data reading function by extracting only the area we are interested in. This involves querying the 'country' mask to find its limits and passing this information through to the reader.
- 5.2 Interpolation Then we will look at methods to interpolate and smooth over gaps in datasets using various methods.
- 5.3 Function Fitting Finally, we will look at fitting models to datasets, in this case a model describing LAI phenology.

42.1 5.1 Making 3D datasets and Movies

First though, we will briefly go over once more the work we did on downloading data (ussssing `wget`), generating 3D masked datasets, and making movies.

This time, we will concentrate more on generating functions that we can re-use for other purposes.

42.1.1 5.1.1 Downloading data

We start by filtering the file `files/data/robot.txt` to get only lines (containing urls) for a particular tile and year.

We might easily do this in unix:

```
# filter LAI MODIS files for this year and tile (in bash)
tile='h17v03'
year='2005'
ofile=files/data/modis_lai_${tile}_${year}.txt

grep $tile < files/data/robot.txt | grep \/$year > $ofile

# have a look at the first few
head -4 < $ofile

http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.01/MCD15A2.A2005001.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.09/MCD15A2.A2005009.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.17/MCD15A2.A2005017.h17v03.00
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.25/MCD15A2.A2005025.h17v03.00
```

Now download the datasets:

```
tile='h17v03'
year='2005'
ofile=files/data/modis_lai_${tile}_${year}.txt
```

```
# go into the directory we want the data
pushd files/data
# get the urls from the file
# --cut-dirs=4 this time as there are 4 layers of directory we
# wish to ignore with this dataset
wget --quiet -nc -nH --cut-dirs=4 -i ../$ofile
# go back to where we were
popd

/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter5_Interpolation/files/data ...
/archive/rsu_raid_0/plewis/public_html/geogg122_local/geogg122/Chapter5_Interpolation
```

42.1.2 5.1.2 Read from an ASCII file

The ASCII file `files/data/modis_lai_{tile}_{year}.txt` contains lines of urls.

Each line (each url) is a string such as:

```
http://e4ftl01.cr.usgs.gov/MODIS_Composites/MOTA/MCD15A2.005/2005.01.25/MCD15A2.A2005025...
```

The *filename* here is `MCD15A2.A2005025.h17v03.005.2007353055037.hdf`, so we can split the url on the field / to get this:

Let's read the filenames from the text file that has the urls in it and load it into a list that we will call `filelist`:

```
import numpy as np

tile = 'h17v03'
year = '2005'

# specify the file with the urls in
ifile= 'files/data/modis_lai_%s_%s.txt'%(tile,year)

# one way to read the data from the file
fp = open(ifile)
lines = fp.readlines()
filelist = []
for url in lines:
    filename = url.split('/')[-1].strip()
    filelist.append(filename)
fp.close()

# show the first few
print filelist[:5]

['MCD15A2.A2005001.h17v03.005.2007350235547.hdf', 'MCD15A2.A2005009.h17v03.005.2007351235445.hdf']

# a neater way:
fp = open(ifile)
filelist = [url.split('/')[-1].strip() for url in fp.readlines()]
fp.close()

# show the first few
print filelist[:5]

['MCD15A2.A2005001.h17v03.005.2007350235547.hdf', 'MCD15A2.A2005009.h17v03.005.2007351235445.hdf']

# an even neater way using np.loadtxt
# But don't worry if you don't quite get this one yet!

# define a function get_filename(f)
# When a function is 'small' its easier to use a lambda definition!
```

```

get_filename = lambda f: f.split('/')[-1]
filelist = np.loadtxt(ifile,dtype='str',converters={0:get_filename})

print filelist[:5]

['MCD15A2.A2005001.h17v03.005.2007350235547.hdf'
 'MCD15A2.A2005009.h17v03.005.2007351235445.hdf'
 'MCD15A2.A2005017.h17v03.005.2007352033411.hdf'
 'MCD15A2.A2005025.h17v03.005.2007353055037.hdf'
 'MCD15A2.A2005033.h17v03.005.2007355050158.hdf']

```

42.1.3 5.1.3 Read Just The Data We Want

Last time, we generated a function to read MODIS LAI data.

We have now included such a function in the directory ‘files/python <files/python>‘ called ‘get_lai.py <files/python/get_lai.py>‘.

The only added sophistication is that when we call `ReadAsArray`, we give it the starting cols, rows, and number of cols and rows to read (e.g. `xsize=600, yoff=300, xoff=300, ysize=600`):

```

# Now we have a list of filenames
# load read_lai
import sys
sys.path.insert(0,'files/python')

from get_lai import get_lai

help(get_lai)

Help on function get_lai in module get_lai:

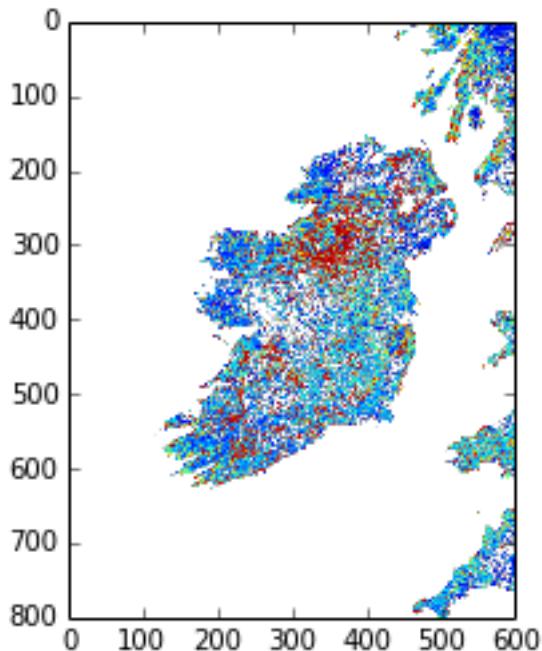
get_lai(filename, qc_layer='FparLai_QC', scale=[0.1, 0.1], mincol=0, minrow=0, ncol=None, nrow=None,
        xsize=600, yoff=300, xoff=300, ysize=600)

# e.g. for reading a single file:

lai_file0 = get_lai('files/data/%s'%filelist[20],ncol=600,mincol=300,minrow=400,nrow=800)
plt.imshow(lai_file0['Lai_1km'])

<matplotlib.image.AxesImage at 0x10342b50>

```



```
print type(lai_file0)
print lai_file0.keys()

<type 'dict'>
['Lai_1km', 'LaiStdDev_1km']
```

The function returns a dictionary with has keys ['Lai_1km', 'LaiStdDev_1km', 'FparLai_QC']:

```
print lai_file0['Lai_1km'].shape

(800, 600)
```

Each of these datasets is of shape (1200, 1200), but we have read only 600 (columns) and 800 (rows) in this case. Note that the numpy indexing is (rows,cols).

We know how to create a mask from a vector dataset from thelast session:

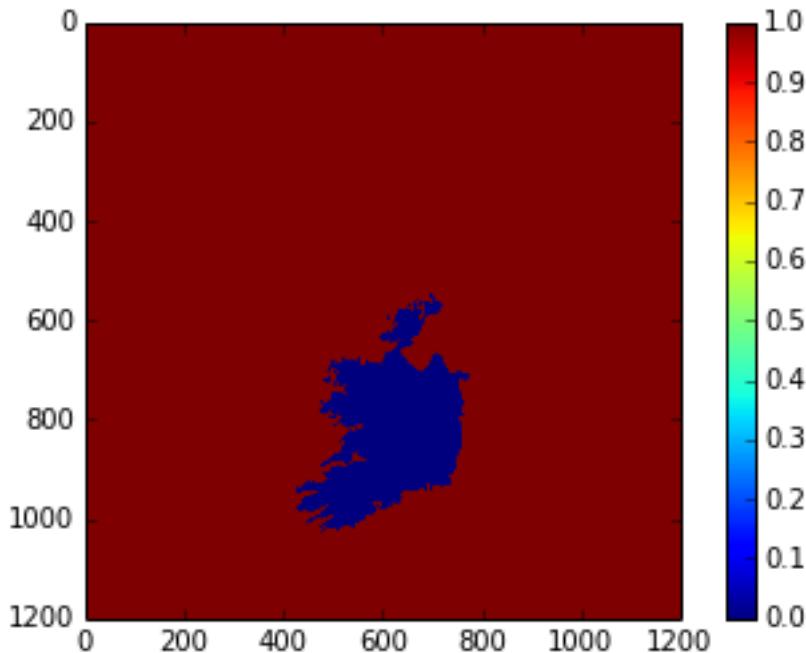
```
from raster_mask import raster_mask

# make a raster mask
# from the layer IRELAND in world.shp
filename = filelist[0]
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%('files/data/%s'%filename,'Lai_1km')

mask = raster_mask(file_spec,\n                  target_vector_file = "files/data/world.shp",\n                  attribute_filter = "NAME = 'IRELAND'")

plt.imshow(mask)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar instance at 0x10ae5bd8>
```



In this case, the data we want is only a small section of the whole spatial dataset.

It would be convenient to extract *only* the part we want.

We can use `numpy.where()` to help with this:

```
# The mask is False for the area we want
rowpix,colpix = np.where(mask == False)

print rowpix,colpix
[ 548  548  548 ..., 1024 1025 1025] [693 694 695 ..., 476 473 474]
```

`rowpix` and `colpix` are lists of pixel coordinates where the condition we specified is True (i.e. where `mask` is False).

If we wanted to find the bounds of this area, we simply need to know the minimum and maximum column and row in these lists:

```
mincol,maxcol = min(colpix),max(colpix)
minrow,maxrow = min(rowpix),max(rowpix)

# think about why the + 1 here!!!
# what if maxcol and mincol were the same?
ncol = maxcol - mincol + 1
nrow = maxrow - minrow + 1

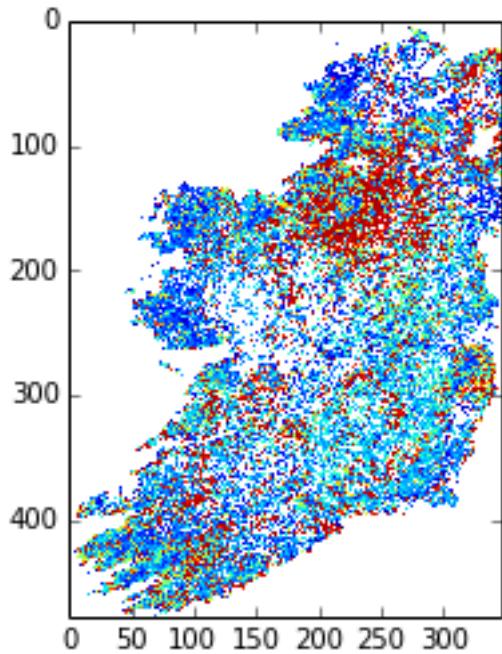
print minrow,mincol,nrow,ncol
548 422 478 348
```

We could use this information to extract *only* the area we want when we read the data:

```
lai_file0 = get_lai('files/data/%s'%filelist[20],\
                    ncol=ncol,nrow=nrow,mincol=mincol,minrow=minrow)

plt.imshow(lai_file0['Lai_1km'],interpolation='none')
```

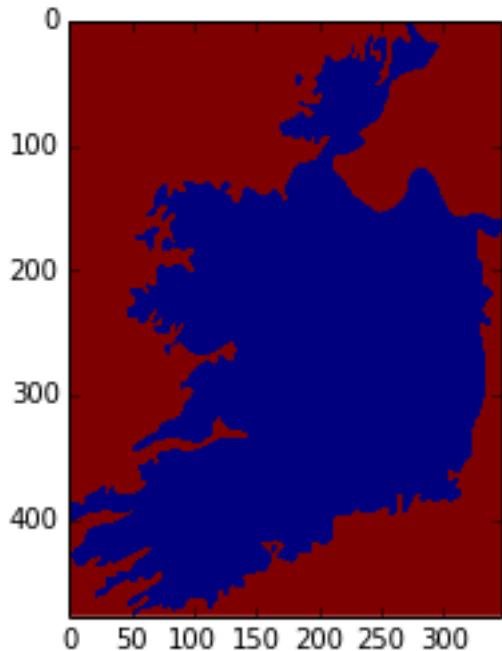
```
<matplotlib.image.AxesImage at 0x10ddc2d0>
```



Now, lets extract this portion of the mask:

```
small_mask = mask[minrow:minrow+nrow,mincol:mincol+ncol]  
  
plt.imshow(small_mask,interpolation='none')
```

```
<matplotlib.image.AxesImage at 0x2ae9e41a3750>
```



And combine the country mask with the small dataset:

As a recap, we can use the function `raster_mask` that we gave you last time to develop a raster mask (!) from an ESRI shapefile ([files/data/world.shp](#) here).

We can then combine this mask with the QC-derived mask in the LAI dataset.

The LAI mask (that will be `lai.mask` in the code below) is `False` for good data, as is the country mask.

To combine them, we want some operator X for which:

```
True X True == True
True X False == True
False X True == True
False X False == False
```

The operator to use then is an *or*, here, a bitwise or, `|`.

```
lai_file0 = get_lai('files/data/%s'%filelist[20], \
                    ncol=ncol, nrow=nrow, mincol=mincol, minrow=minrow)

layer = 'Lai_1km'
lai = lai_file0[layer]
small_mask = mask[minrow:minrow+nrow, mincol:mincol+ncol]

# combined mask
new_mask = small_mask | lai.mask

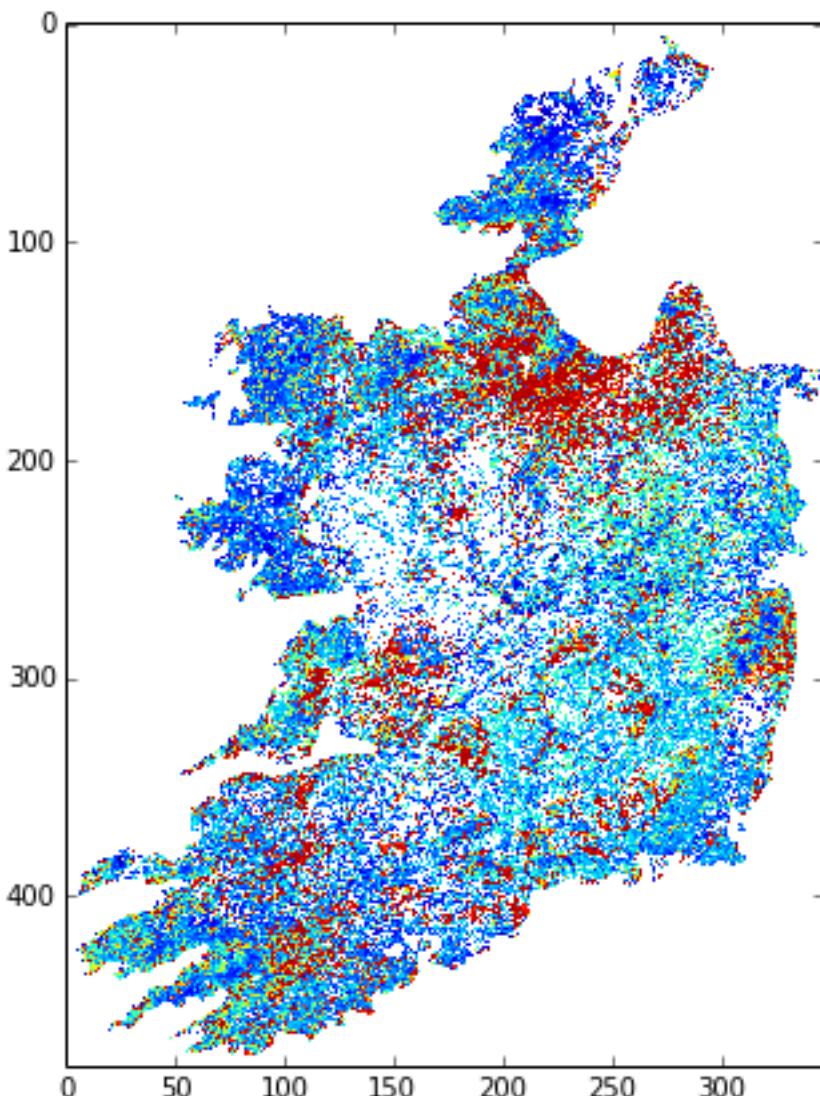
plt.figure(figsize=(7,7))
plt.imshow(new_mask, interpolation='none')

lai = ma.array(lai, mask=new_mask)

plt.figure(figsize=(7,7))
plt.imshow(lai, interpolation='none')

<matplotlib.image.AxesImage at 0x2ae9e4275f50>
```





We should be used to writing loops around such functions.

In this case, we read *all* of the files in `filelist` and put the data into the dictionary called `lai` here.

Because there are multiple layers in the datasets, we loop over layer and append to each list individually:

```
# load 'em all ...
# for United Kingdom here

import numpy.ma as ma
from raster_mask import raster_mask

country = 'UNITED KINGDOM'

# make a raster mask
# from the layer UNITED KINGDOM in world.shp
filename = filelist[0]
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
file_spec = file_template%('files/data/%s'%filename,'Lai_1km')

mask = raster_mask(file_spec,
                    target_vector_file = "files/data/world.shp",
                    attribute_filter = "NAME = '%s'"%country)
# extract just the area we want
```

```
# by getting the min/max rows/cols
# of the data mask
# The mask is False for the area we want
rowpix,colpix = np.where(mask == False)
mincol,maxcol = min(coli),max(coli)
minrow,maxrow = min(rowpix),max(rowpix)
ncol = maxcol - mincol + 1
nrow = maxrow - minrow + 1
# and make a small mask
small_mask = mask[minrow:maxrow,mincol:maxcol+1]

# data_fields with empty lists
data_fields = {'LaiStdDev_1km':[],'Lai_1km':[]}

# make a dictionary and put the filenames in it
# along with the mask and min/max info
lai = {'filenames':np.sort(filelist),\
        'minrow':minrow,'mincol':mincol,\n        'mask':small_mask}

# combine the dictionaries
lai.update(data_fields)

# loop over each filename
for f in np.sort(lai['filenames']):
    this_lai = get_lai('files/data/%s'%f,\n                      mincol=mincol,ncol=ncol,\n                      minrow=minrow,nrow=nrow)
    for layer in data_fields.keys():
        # apply the mask
        new_mask = this_lai[layer].mask | small_mask
        this_lai[layer] = ma.array(this_lai[layer],mask=new_mask)
        lai[layer].append(this_lai[layer])

# have a look at one of these

i = 20

import pylab as plt

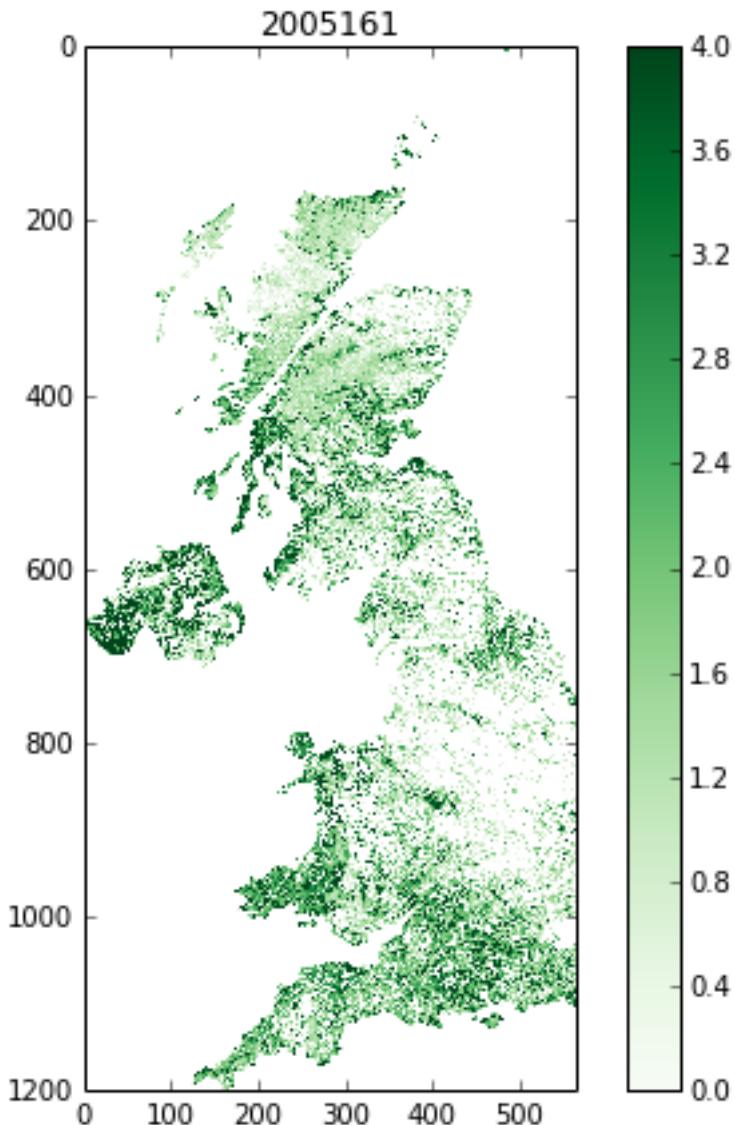
# just see what the shape is ...
print lai['Lai_1km'][i].shape

root = 'files/images/lai_uk'

cmap = plt.cm.Greens

f = lai['filenames'][i]
fig = plt.figure(figsize=(7,7))
# get some info from filename
file_id = f.split('/')[-1].split('.')[5][1:]
print file_id
plt.imshow(lai['Lai_1km'][i],cmap=cmap,interpolation='none',vmax=4.,vmin=0.0)
# plot a jpg
plt.title(file_id)
plt.colorbar()
plt.savefig('files/images/lai_uk_%s.jpg'%file_id)

(1200, 566)
2005161
```



```
# that's quite good, so put as a function:
import numpy.ma as ma
import numpy as np
import sys
sys.path.insert(0,'files/python')
from get_lai import get_lai
from raster_mask import raster_mask

def read_lai(filelist,datadir='files/data',country=None):
    '''
    Read MODIS LAI data from a set of files
    in the list filelist. Data assumed to be in
    directory datadir.

    Parameters:
    filelist : list of LAI files

    Options:
    datadir : data directory
    country : country name (in files/data/world.shp)

    Returns:

```

```
lai dictionary
'''
if country:
    # make a raster mask
    # from the layer UNITED KINGDOM in world.shp
    file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
    file_spec = file_template%'files/data/%s'%filelist[0], 'Lai_1km')

    mask = raster_mask(file_spec,\n        target_vector_file = "files/data/world.shp",\n        attribute_filter = "NAME = '%s'%"country)
    # extract just the area we want
    # by getting the min/max rows/cols
    # of the data mask
    # The mask is False for the area we want
    rowpix,colpix = np.where(mask == False)
    mincol,maxcol = min(coli),max(coli)
    minrow,maxrow = min(rowpi),max(rowpi)
    ncol = maxcol - mincol + 1
    nrow = maxrow - minrow + 1
    # and make a small mask
    small_mask = mask[minrow:minrow+nrow,mincol:mincol+ncol]
else:
    # no country
    mincol = 0
    maxcol = 0
    ncol = None
    nrow = None

# data_fields with empty lists
data_fields = {'LaiStdDev_1km':[], 'Lai_1km':[]}

# make a dictionary and put the filenames in it
# along with the mask and min/max info
lai = {'filenames':np.sort(filelist),\
        'minrow':minrow,'mincol':mincol,\n        'mask':small_mask}

# combine the dictionaries
lai.update(data_fields)

# loop over each filename
for f in np.sort(lai['filenames']):
    this_lai = get_lai('files/data/%s'%f,\n                      mincol=mincol,ncol=ncol,\n                      minrow=minrow,nrow=nrow)
    for layer in data_fields.keys():
        # apply the mask
        if country:
            new_mask = this_lai[layer].mask | small_mask
            this_lai[layer] = ma.array(this_lai[layer],mask=new_mask)
        lai[layer].append(this_lai[layer])
    for layer in data_fields.keys():
        lai[layer] = ma.array(lai[layer])

return lai

# test this ... the one in the file
# does a cutout of the data area as well
# which will keep the memory
# requirements down
from get_lai import read_lai
```

```

lai = read_lai(filelist,country='IRELAND',verbose=True)

# have a look at one of these

i = 20

# just see what the shape is ...
print lai['Lai_1km'][i].shape

root = 'files/images/lai_eire'

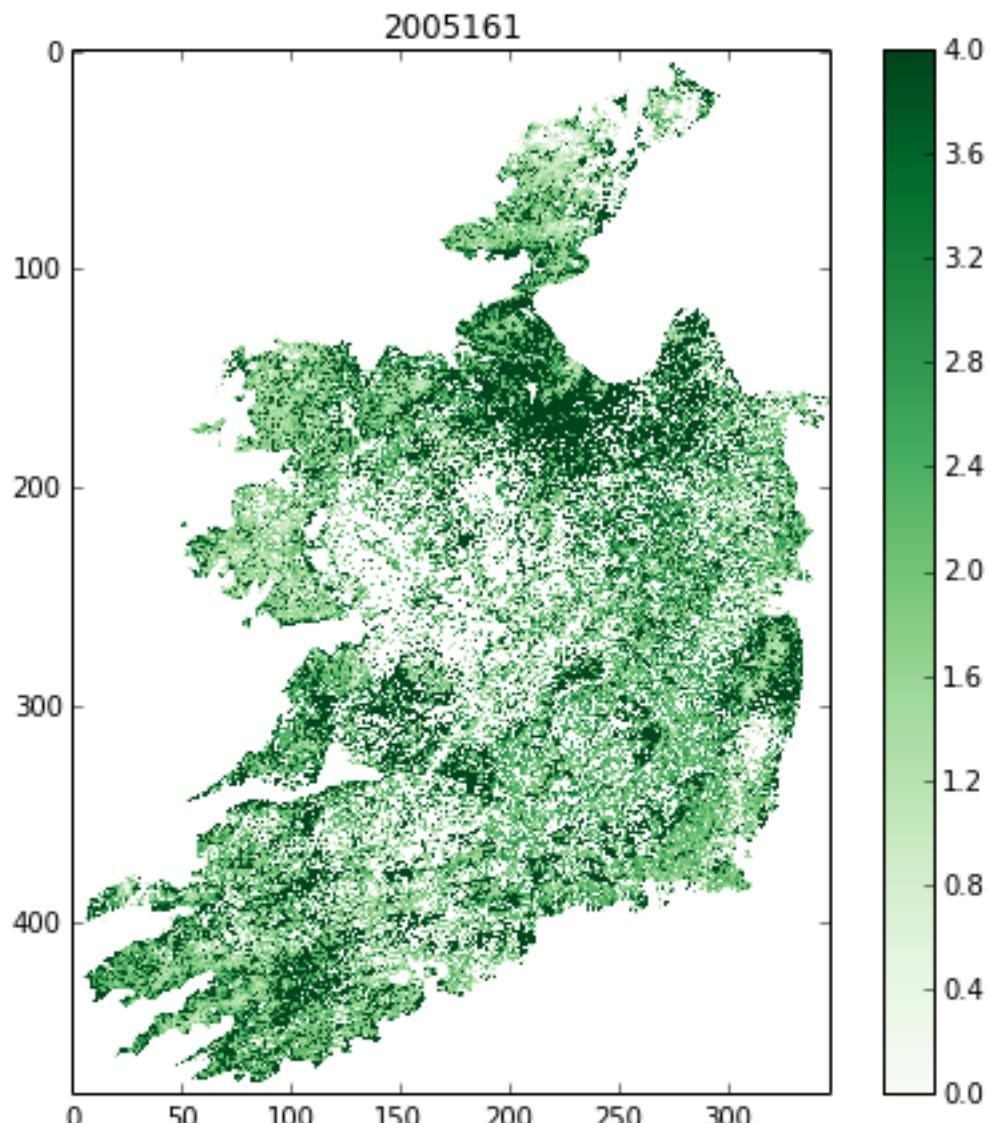
cmap = plt.cm.Greens

f = lai['filenames'][i]
fig = plt.figure(figsize=(7,7))
# get some info from filename
file_id = f.split('/')[-1].split('.')[ -5 ][1:]
print file_id
plt.imshow(lai['Lai_1km'][i],cmap=cmap,interpolation='none',vmax=4.,vmin=0.0)
# plot a jpg
plt.title(file_id)
plt.colorbar()
plt.savefig('%s_%s.jpg'%(root,file_id))

creating mask of IRELAND
...
MCD15A2.A2005001.h17v03.005.2007350235547.hdf
...
MCD15A2.A2005009.h17v03.005.2007351235445.hdf
...
MCD15A2.A2005017.h17v03.005.2007352033411.hdf
...
MCD15A2.A2005025.h17v03.005.2007353055037.hdf
...
MCD15A2.A2005033.h17v03.005.2007355050158.hdf
...
MCD15A2.A2005041.h17v03.005.2007357014602.hdf
...
MCD15A2.A2005049.h17v03.005.2007360165724.hdf
...
MCD15A2.A2005057.h17v03.005.2007361230641.hdf
...
MCD15A2.A2005065.h17v03.005.2007365024202.hdf
...
MCD15A2.A2005073.h17v03.005.2008001043631.hdf
...
MCD15A2.A2005081.h17v03.005.2008003173048.hdf
...
MCD15A2.A2005089.h17v03.005.2008005154542.hdf
...
MCD15A2.A2005097.h17v03.005.2008007175837.hdf
...
MCD15A2.A2005105.h17v03.005.2008018085544.hdf
...
MCD15A2.A2005113.h17v03.005.2008021020137.hdf
...
MCD15A2.A2005121.h17v03.005.2008021193749.hdf
...
MCD15A2.A2005129.h17v03.005.2008024061330.hdf
...
MCD15A2.A2005137.h17v03.005.2008032075236.hdf
...
MCD15A2.A2005145.h17v03.005.2008033192556.hdf
...
MCD15A2.A2005153.h17v03.005.2008035054421.hdf
...
MCD15A2.A2005161.h17v03.005.2008036173810.hdf
...
MCD15A2.A2005169.h17v03.005.2008039132812.hdf
...
MCD15A2.A2005177.h17v03.005.2008042090537.hdf
...
MCD15A2.A2005185.h17v03.005.2008044115459.hdf
...
MCD15A2.A2005193.h17v03.005.2008046140018.hdf
...
MCD15A2.A2005201.h17v03.005.2008050015227.hdf
...
MCD15A2.A2005209.h17v03.005.2008052203557.hdf
...
MCD15A2.A2005217.h17v03.005.2008055145215.hdf
...
MCD15A2.A2005225.h17v03.005.2008057010213.hdf
...
MCD15A2.A2005233.h17v03.005.2008060214119.hdf
...
MCD15A2.A2005241.h17v03.005.2008063115631.hdf
...
MCD15A2.A2005249.h17v03.005.1998144165707.hdf
...
MCD15A2.A2005257.h17v03.005.2008067051936.hdf
...
MCD15A2.A2005265.h17v03.005.2008069073121.hdf
...
MCD15A2.A2005273.h17v03.005.2008071050025.hdf
...
MCD15A2.A2005281.h17v03.005.2008072202421.hdf
...
MCD15A2.A2005289.h17v03.005.2008074194126.hdf
...
MCD15A2.A2005297.h17v03.005.2008077061121.hdf

```

```
... MCD15A2.A2005305.h17v03.005.2008080055607.hdf
... MCD15A2.A2005313.h17v03.005.2008083165435.hdf
... MCD15A2.A2005321.h17v03.005.2008084043211.hdf
... MCD15A2.A2005329.h17v03.005.2008086063619.hdf
... MCD15A2.A2005337.h17v03.005.2008087175845.hdf
... MCD15A2.A2005345.h17v03.005.2008088144615.hdf
... MCD15A2.A2005353.h17v03.005.2008091004441.hdf
... MCD15A2.A2005361.h17v03.005.2008091025114.hdf
... done
(478, 348)
2005161
```



```
# make a movie

import pylab as plt
import os

# just see what the shape is ...
print lai['Lai_1km'].shape

root = 'files/images/lai_country_eire'

cmap = plt.cm.Greens
```

```
for i,f in enumerate(lai['filenames']):
    fig = plt.figure(figsize=(7,7))
    # get some info from filename
    file_id = f.split('/')[-1].split('.')[5][1:]
    print file_id
    plt.imshow(lai['Lai_1km'][i],cmap=cmap,interpolation='none',vmax=4.,vmin=0.0)
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('%s_%s.jpg'%(root,file_id))
    plt.close(fig)

cmd = 'convert -delay 100 -loop 0 {0}_*.jpg {0}_movie.gif'.format(root)
os.system(cmd)

(46, 478, 348)
2005001
2005009
2005017
2005025
2005033
2005041
2005049
2005057
2005065
2005073
2005081
2005089
2005097
2005105
2005113
2005121
2005129
2005137
2005145
2005153
2005161
2005169
2005177
2005185
2005193
2005201
2005209
2005217
2005225
2005233
2005241
2005249
2005257
2005265
2005273
2005281
2005289
2005297
2005305
2005313
2005321
2005329
2005337
2005345
2005353
2005361
```

0

```
# The movie making works, so pack that into a function

import pylab as plt
import os

root = 'files/images/lai_eire'

def make_movie(lai,root,layer='Lai_1km',vmax=4.,vmin=0.,do_plot=False):
    """
    Make an animated gif from MODIS LAI data in
    dictionary 'lai'.

    Parameters:
    lai      : data dictionary
    root    : root file /directory name of frames and movie

    layer   : data layer to plot
    vmax    : max value for plotting
    vmin    : min value for plotting
    do_plot: set True if you want the individual plots
              to display

    Returns:
    movie name

    """
    cmap = plt.cm.Greens

    for i,f in enumerate(lai['filenames']):
        fig = plt.figure(figsize=(7,7))
        # get some info from filename
        file_id = f.split('/')[-1].split('.')[1:-5]
        print file_id
        plt.imshow(lai[layer][i],cmap=cmap,interpolation='none',\
                   vmax=vmax,vmin=vmin)
        # plot a jpg
        plt.title(file_id)
        plt.colorbar()
        plt.savefig('%s_%s.jpg'%(root,file_id))
        if not do_plot:
            plt.close(fig)

    cmd = 'convert -delay 100 -loop 0 {0}_*.jpg {0}_movie.gif'.format(root)
    os.system(cmd)
    return '{0}_movie.gif'.format(root)

# test it

lai_uk = read_lai(filelist,country='UNITED KINGDOM')
root = 'files/images/lai_UK'
movie = make_movie(lai_uk,root)
print movie

2005001
2005009
2005017
2005025
2005033
2005041
```

```
2005049  
2005057  
2005065  
2005073  
2005081  
2005089  
2005097  
2005105  
2005113  
2005121  
2005129  
2005137  
2005145  
2005153  
2005161  
2005169  
2005177  
2005185  
2005193  
2005201  
2005209  
2005217  
2005225  
2005233  
2005241  
2005249  
2005257  
2005265  
2005273  
2005281  
2005289  
2005297  
2005305  
2005313  
2005321  
2005329  
2005337  
2005345  
2005353  
2005361  
files/images/lai_UK_movie.gif
```

42.2 5.2 Interpolation

42.2.1 5.2.1 Univariate interpolation

So, we can load the data we want from multiple MODIS hdf files that we have downloaded from the NASA server into a 3D masked numpy array, with a country boundary mask (projected int the raster data coordinate system) from a vector dataset.

Let's start to explore the data then.

You should have an array of LAI for Ireland:

```
type(lai['Lai_1km'])  
  
numpy.ma.core.MaskedArray
```

Let's plot the LAI for some given pixels.

First, we might like to identify which pixels actually have any data.

A convenient function for this would be `np.where` that returns the indices of items that are `True`.

Since the data mask is `False` for good data, we take the complement `~` so that good data are ‘`True`’:

```
data = lai['Lai_1km']
np.where(~data.mask)

(array([ 3,  3,  3, ..., 39, 39, 39]),
 array([326, 328, 329, ..., 472, 472, 475]),
 array([ 82, 145,  83, ...,  86,  87,  51]))
```

An example good pixel this is (3,329,145). Let’s look at this for all time periods:

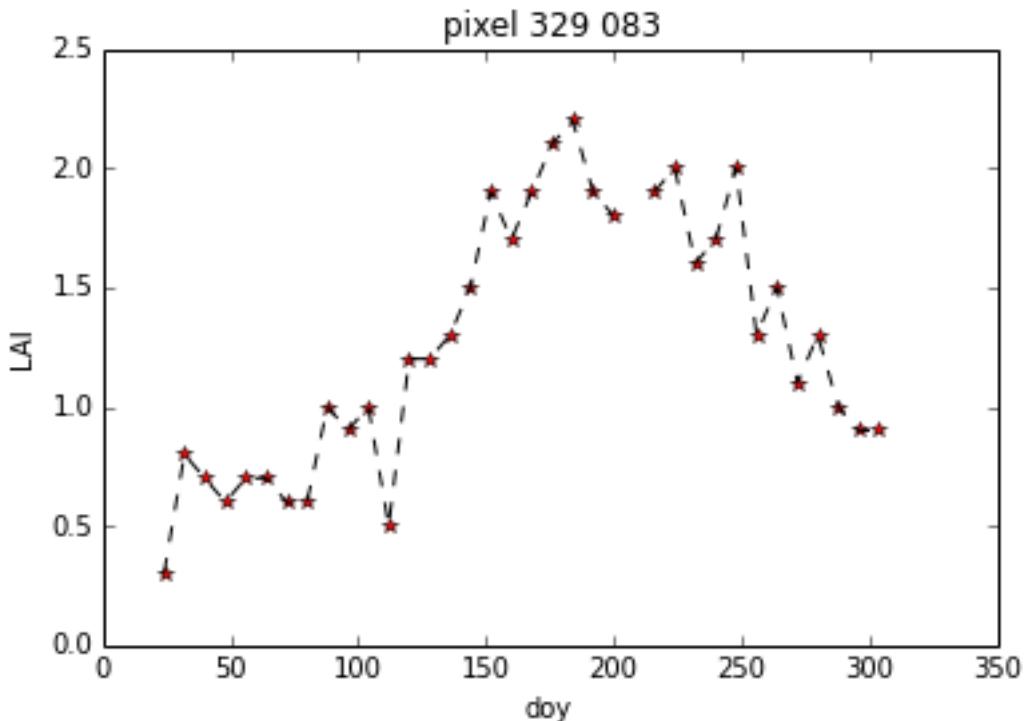
```
data = lai['Lai_1km']

r = 329
c = 83

pixel = data[:,r,c]

# plot red stars at the data points
plt.plot(np.arange(len(pixel))*8,pixel,'r*')
# plot a black (k) dashed line (--)
plt.plot(np.arange(len(pixel))*8,pixel,'k--')
plt.xlabel('doy')
plt.ylabel('LAI')
plt.title('pixel %03d %03d'%(r,c))

<matplotlib.text.Text at 0x11e77950>
```



The data follow the trend of what we might expect for LAI development, but they are clearly a little noisy.

We also have access to uncertainty information (standard deviation):

```
# copy the data in case we change it any
data = lai['Lai_1km'].copy()
```

```

sd    = lai['LaiStdDev_1km'].copy()

r = 329
c = 83

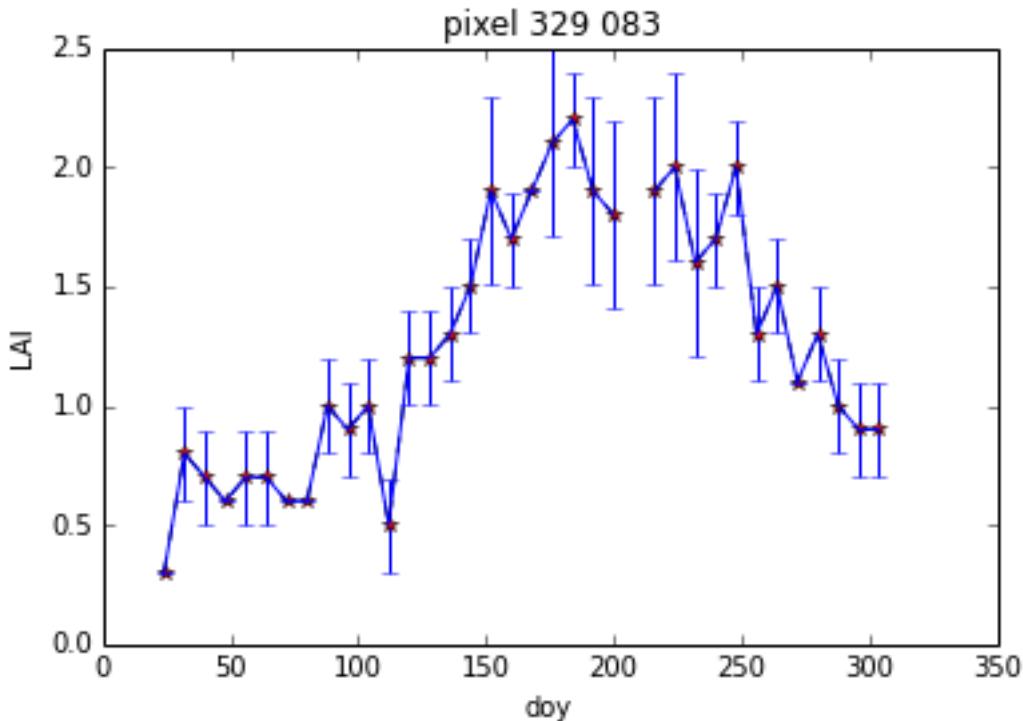
pixel    = data[:,r,c]
pixel_sd = sd[:,r,c]

x = np.arange(len(pixel))*8

# plot red stars at the data points
plt.plot(x,pixel,'r*')
# plot a black (k) dashed line (--)
plt.plot(x,pixel,'k--')
# plot error bars:
# 1.96 because that is the 95% confidence interval
plt.errorbar(x,pixel,yerr=pixel_sd*1.96)
plt.xlabel('doy')
plt.ylabel('LAI')
plt.title('pixel %03d %03d' %(r,c))

<matplotlib.text.Text at 0x2ae9e53409d0>

```



We would generally expect LAI to be quite smoothly varying over time. Visualising the data with 95% confidence intervals is quite useful as we can now ‘imagine’ some smooth line that would generally go within these bounds.

Some of the uncertainty estimates are really rather small though, which are probably not reliable.

Let’s inflate them:

```

data = lai['Lai_1km'].copy()
sd    = lai['LaiStdDev_1km'].copy()

r = 329
c = 83

pixel    = data[:,r,c]

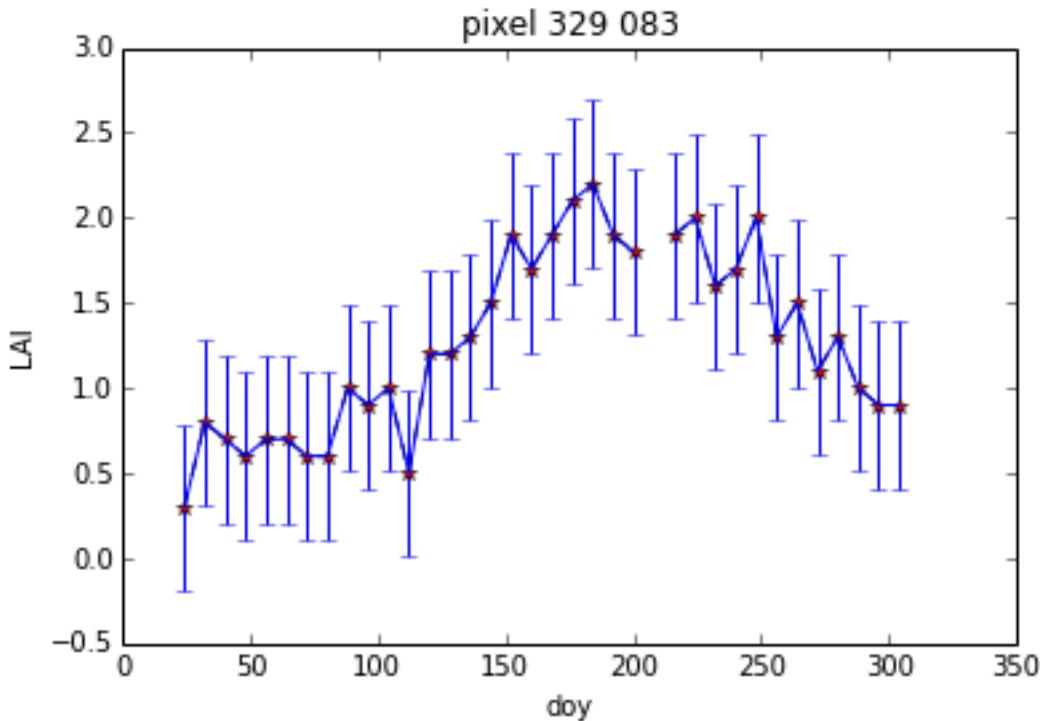
```

```
pixel_sd = sd[:,r,c]
# threshold
thresh = 0.25
pixel_sd[pixel_sd<thresh] = thresh

x = np.arange(len(pixel))*8

# plot red stars at the data points
plt.plot(x,pixel,'r*')
# plot a black (k) dashed line (--)
plt.plot(x,pixel,'k--')
# plot error bars:
# 1.96 because that is the 95% confidence interval
plt.errorbar(x,pixel,yerr=pixel_sd*1.96)
plt.xlabel('doy')
plt.ylabel('LAI')
plt.title('pixel %03d %03d'%(r,c))

<matplotlib.text.Text at 0x2ae9e5554910>
```



This is perhaps a bit more realistic ...

The data now have some missing values (data gaps) and, as we have noted, are a little noisy.

A Python module we can use for many scientific functions is ‘scipy’
<http://docs.scipy.org/doc/scipy>‘, in particular here, the ‘scipy’ interpolation functions
<http://docs.scipy.org/doc/scipy/reference/interpolate.html>‘.

We need to make a careful choice of the interpolation functions.

We might, in many circumstances simply want something that interpolates between data points, i.e. that goes through the data points that we have.

Many interpolators will not provide extrapolation, so in the example above we could not get an estimate of LAI prior to the first sample and after the last.

The best way to deal with that would be to have multiple years of data.

Instead here, we will repeat the dataset three times to mimic this:

```

from scipy import interpolate

pixel = data[:,r,c]

# original x,y
y_ = pixel
x_ = (np.arange(len(y_))*8.+1)[~pixel.mask]
y_ = y_[~pixel.mask]

# extend: using np.tile() to repeat data
y_extend = np.tile(y_, 3)
# extend: using vstack to stack 3 different arrays
x_extend = np.hstack((x_-46*8, x_, x_+46*8))

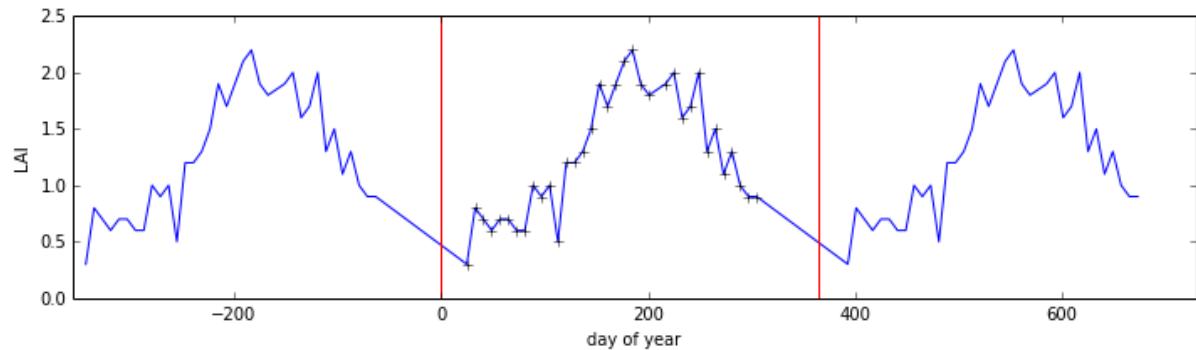
```

```

# plot the extended dataset
plt.figure(figsize=(12, 3))
plt.plot(x_extend,y_extend,'b')
plt.plot(x_,y_, 'k+')
plt.plot([0.,0.],[0.,2.5],'r')
plt.plot([365.,365.],[0.,2.5],'r')
plt.xlim(-356,2*365)
plt.xlabel('day of year')
plt.ylabel('LAI')

```

<matplotlib.text.Text at 0x2ae9e6108550>



```

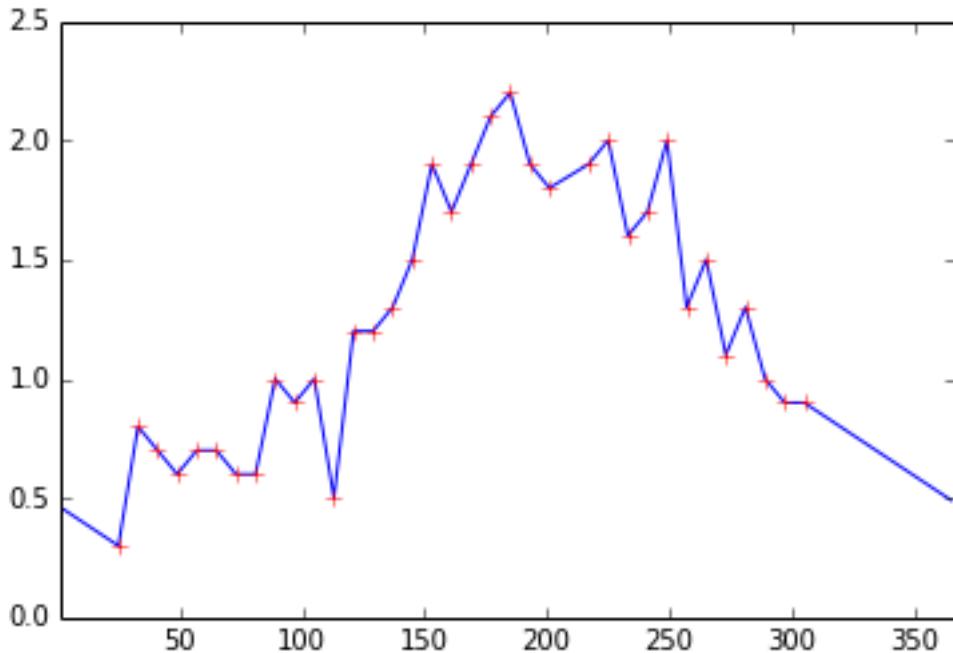
# define xnew at 1 day interval
xnew = np.arange(1.,366.)

# linear interpolation
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
ynew = f(xnew)

plt.plot(xnew,ynew)
plt.plot(x_,y_, 'r+')
plt.xlim(1,366)

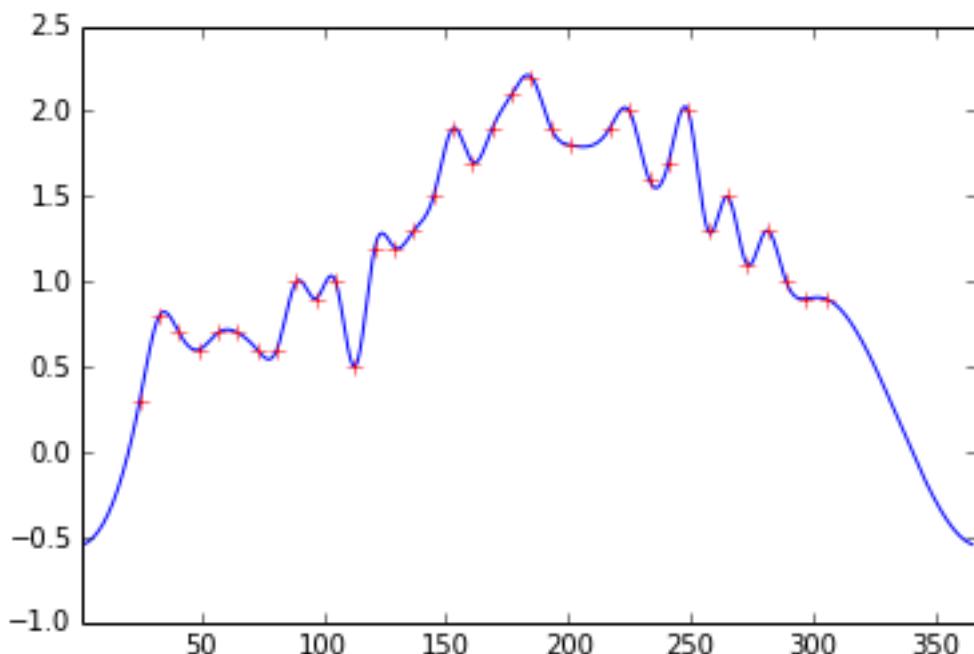
(1, 366)

```



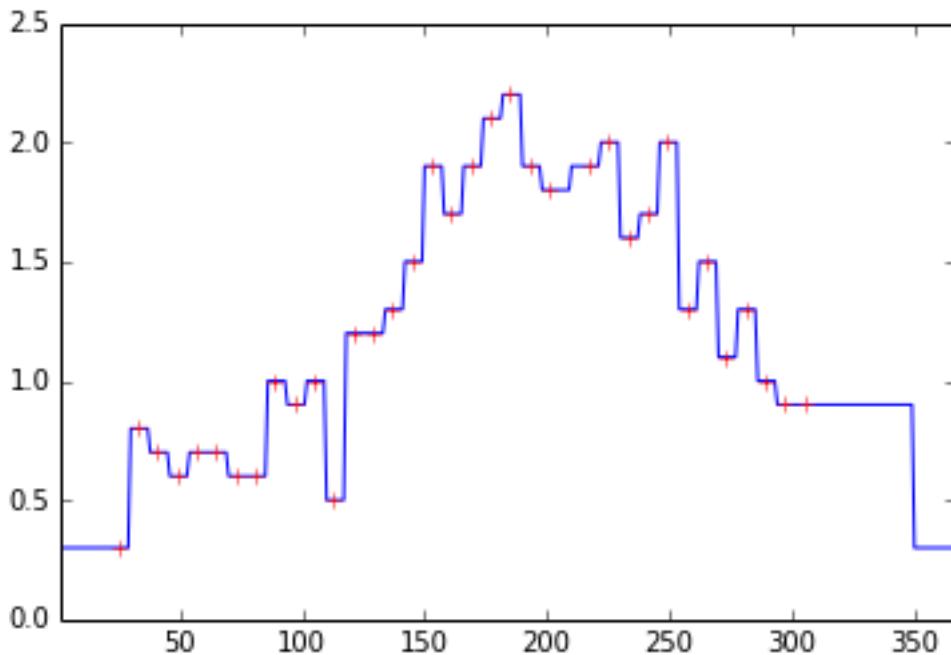
```
# cubic interpolation
f = interpolate.interp1d(x_extend,y_extend,kind='cubic')
ynew = f(xnew)
plt.plot(xnew,ynew)
plt.plot(x_,y_,'r+')
plt.xlim(1,366)
```

(1, 366)



```
# nearest neighbour interpolation
f = interpolate.interp1d(x_extend,y_extend,kind='nearest')
ynew = f(xnew)
plt.plot(xnew,ynew)
plt.plot(x_,y_,'r+')
plt.xlim(1,366)
```

(1, 366)



Depending on the problem you are trying to solve, different interpolation schemes will be appropriate. For categorical data (e.g. ‘snow’, coded as 1 and ‘no snow’ coded as 0), for instance, a nearest neighbour interpolation might be a good idea.

42.2.2 5.2.2 Smoothing

One issue with the schemes above is that they go exactly through the data points, but a more realistic description of the data might be one that incorporated the uncertainty information we have. Visually, this is quite easy to imagine, but how can we implement such ideas?

One way of thinking about this is to think about other sources of information that we might bring to bear on the problem. One such would be that we expect the function to be ‘quite smooth’. This allows us to consider applying smoothness as an additional constraint to the solution.

Many such problems can be phrased as convolution operations.

Convolution is a form of digital filtering that combines two sequences of numbers y and w to give a third, the result z that is a filtered version of y , where for each element j of y :

$$z_j = \sum_{i=-n}^{i=n} w_i y_{j+i}$$

where n is the half width of the filter w . For a smoothing filter, the elements of this will sum to 1 (so that the magnitude of y is not changed).

To illustrate this in Python:

```
# a simple box smoothing filter
# filter width 11
w = np.ones(11)
# normalise
w = w/w.sum()
# half width
n = len(w)/2

# Take the linear interpolation of the LAI above as the signal
```

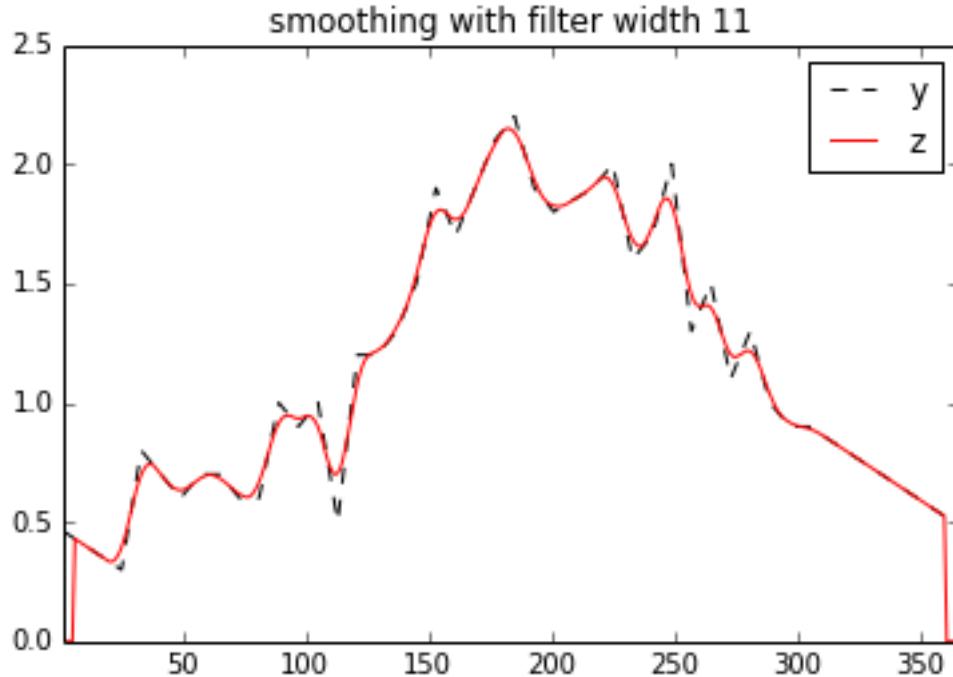
```
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

# where we will put the result
z = np.zeros_like(y)

# This is a straight implementation of the
# equation above
for j in xrange(n,len(y)-n):
    for i in xrange(-n,n+1):
        z[j] += w[n+i] * y[j+i]

plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
plt.legend(loc='best')
plt.title('smoothing with filter width %d'%len(w))

<matplotlib.text.Text at 0x2ae9e579e3d0>
```



As we suggested, the result of convolving y with the filter w (of width 31 here) is z , a smoothed version of y .

You might notice that the filter is only applied once we are n samples into the signal, so we get ‘edge effects’. There are various ways of dealing with edge effects, such as repeating the signal (as we did above, for much the same reason), reflecting the signal, or assuming the signal to be some constant value (e.g. 0) outside of its defined domain.

If we make the filter wider (width 31 now):

```
# a simple box smoothing filter
# filter width 31
w = np.ones(31)
# normalise
w = w/w.sum()
# half width
n = len(w)/2
```

```

# Take the linear interpolation of the LAI above as the signal
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

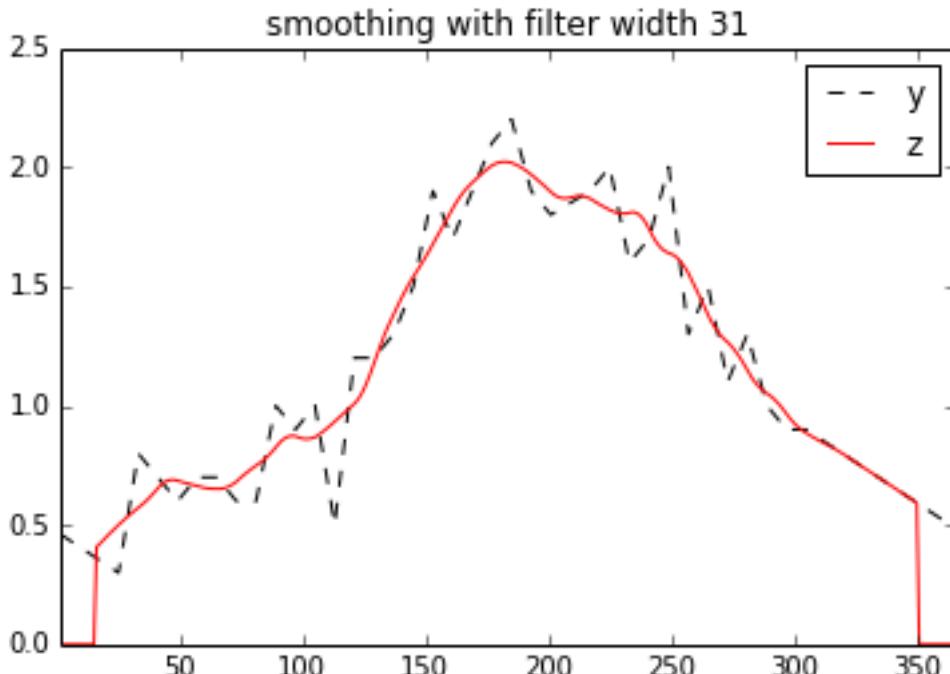
# where we will put the result
z = np.zeros_like(y)

# This is a straight implementation of the
# equation above
for j in xrange(n,len(y)-n):
    for i in xrange(-n,n+1):
        z[j] += w[n+i] * y[j+i]

plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
plt.legend(loc='best')
plt.title('smoothing with filter width %d'%len(w))

<matplotlib.text.Text at 0x2ae9e61f8cd0>

```



Then the signal is ‘more’ smoothed.

There are *many* filters implemented in ‘scipy.signal <<http://docs.scipy.org/doc/scipy/reference/signal.html>>’ __ that you should look over.

A very commonly used smoothing filter is the Savitsky-Golay filter for which you define the window size and filter order.

As with most filters, the filter width controls the degree of smoothing (see examples above). The filter order (related to polynomial order) in essence controls the shape of the filter and defines the ‘peakiness’ of the response.

```

import sys
sys.path.insert(0,'files/python')
# see http://wiki.scipy.org/Cookbook/SavitzkyGolay
from savitzky_golay import *

```

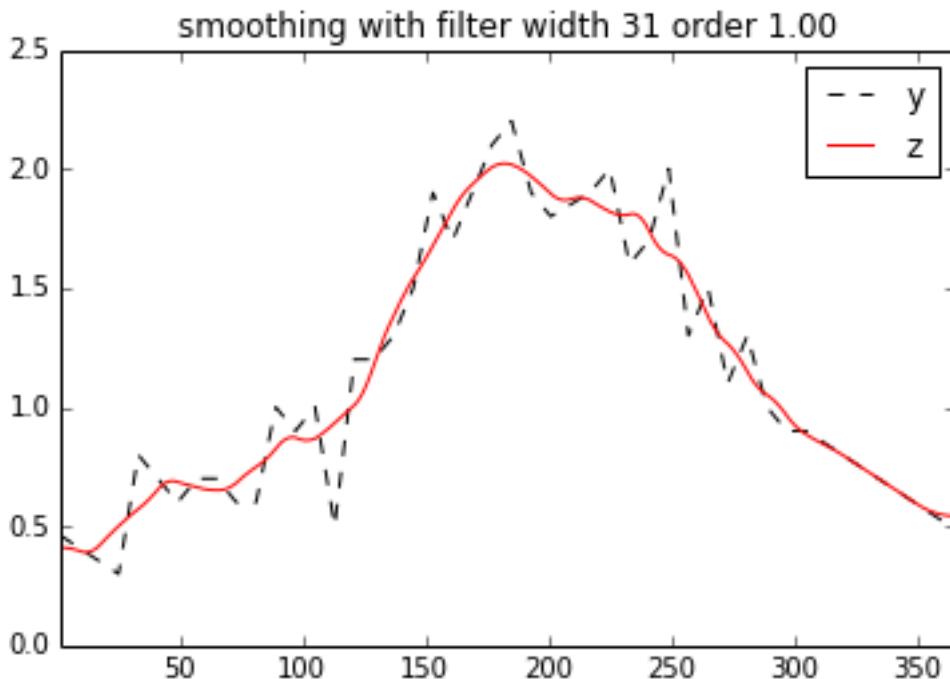
```
window_size = 31
order = 1

# Take the linear interpolation of the LAI above as the signal
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

z = savitzky_golay(y,window_size,order)

plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
plt.legend(loc='best')
plt.title('smoothing with filter width %d order %.2f'%(window_size,order))

<matplotlib.text.Text at 0x2ae9e65e6890>
```



```
import sys
sys.path.insert(0,'files/python')
# see http://wiki.scipy.org/Cookbook/SavitzkyGolay
from savitzky_golay import *

window_size = 61
order = 2

# Take the linear interpolation of the LAI above as the signal
# linear interpolation
x = xnew
f = interpolate.interp1d(x_extend,y_extend,kind='linear')
y = f(x)

z = savitzky_golay(y,window_size,order)

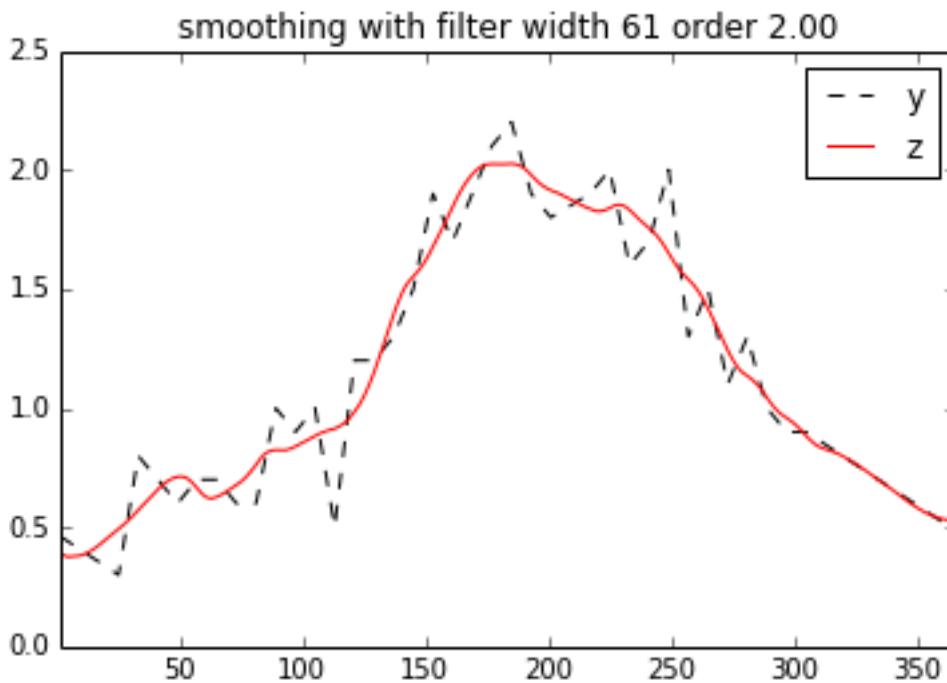
plt.plot(x,y,'k--',label='y')
plt.plot(x,z,'r',label='z')
plt.xlim(x[0],x[-1])
```

```

plt.legend(loc='best')
plt.title('smoothing with filter width %d order %.2f'%(window_size,order))

<matplotlib.text.Text at 0x2ae9e663bb50>

```



If the samples y have uncertainty (standard deviation σ_j for sample j) associated with them, we can incorporate this into smoothing, although many of the methods in `scipy` and `numpy` do not directly allow for this.

Instead, we call an optimal interpolation scheme (a regulariser) here that achieves this. This also has the advantage of giving an estimate of uncertainty for the smoothed samples.

In this case, the parameters are: `order` (as above, but only integer in this implementation) and `wsd` which is an estimate of the variation (standard deviation) in the signal that control smoothness.

```

tile = 'h17v03'
year = '2005'

# specify the file with the urls in
ifile= 'files/data/modis_lai_%s_%s.txt'%(tile,year)

fp = open(ifile)
filelist = [url.split('/')[-1].strip() for url in fp.readlines()]
fp.close()
import sys
sys.path.insert(0,'files/python')

from get_lai import *

try:
    data = lai['Lai_1km']
    sd = lai['LaiStdDev_1km']
except:
    lai = read_lai(filelist,country='IRELAND')
    data = lai['Lai_1km']
    sd = lai['LaiStdDev_1km']

thresh = 0.25
sd[sd

```

```
r = 472
c = 84
from smoothn import *

# this is about the right amount of smoothing here
gamma = 5.

pixel = data[:,r,c]
pixel_sd = sd[:,r,c]

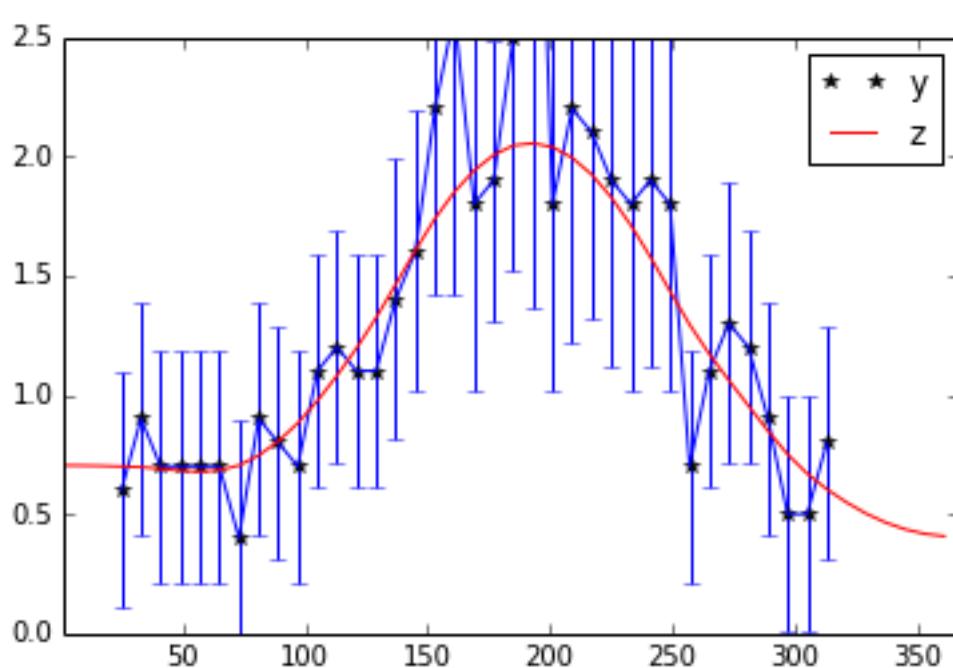
x = np.arange(46)*8+1

order = 2
z = smoothn(pixel,s=gamma, sd=pixel_sd, smoothOrder=2.0)[0]

# plot
plt.plot(x,pixel,'k*',label='y')
plt.errorbar(x,pixel,pixel_sd*1.96)
plt.plot(x,z,'r',label='z')
# lower and upper bounds of 95% CI

plt.xlim(1,366)
plt.ylim(0.,2.5)
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x2b75e4be2790>
```



```
# test it on a new pixel

r = 472
c = 86

gamma = 5

pixel = data[:,r,c]
pixel_sd = sd[:,r,c]

x = np.arange(46)*8+1
```

```

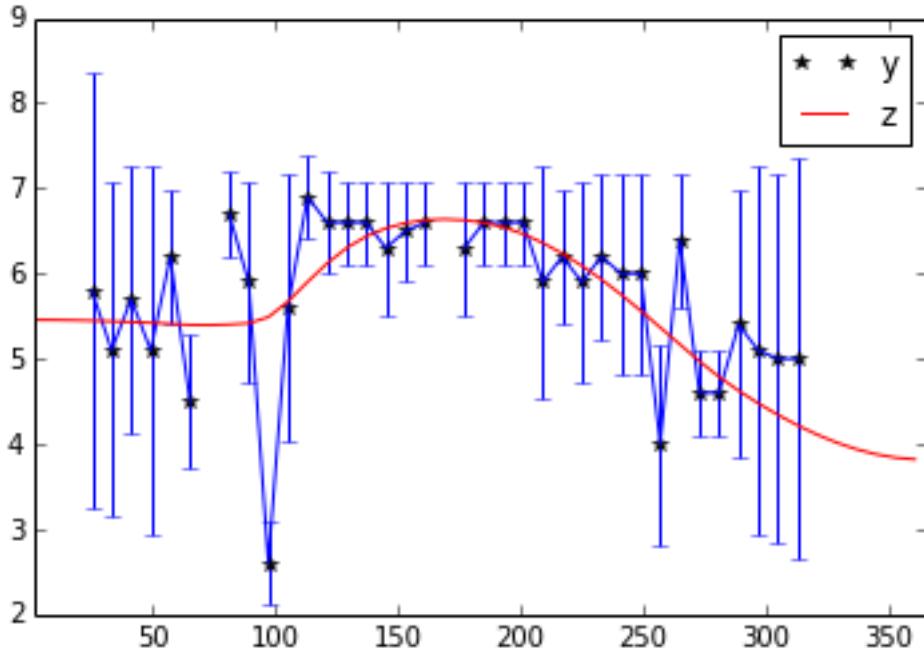
order = 2
z = smoothn(pixel,s=gamma, sd=pixel_sd,smoothOrder=2.0) [0]

# plot
plt.plot(x,pixel,'k*',label='y')
plt.errorbar(x,pixel,pixel_sd*1.96)
plt.plot(x,z,'r',label='z')

plt.xlim(1,366)
plt.legend(loc='best')
z.ndim

```

1



```

# and test it on a new pixel

r = 472
c = 84

#r = 9
#c = 277
gamma = 5.

pixel = data[:,r,c]
pixel_sd = sd[:,r,c]

x = np.arange(46)*8+1

order = 2
# solve for gamma - degree of smoothness
zz = smoothn(pixel, sd=pixel_sd, smoothOrder=2.0)
z = zz[0]
print zz[1],zz[2]

gamma = zz[1]

# plot
plt.plot(x,pixel,'k*',label='y')
plt.errorbar(x,pixel,pixel_sd*1.96)

```

```

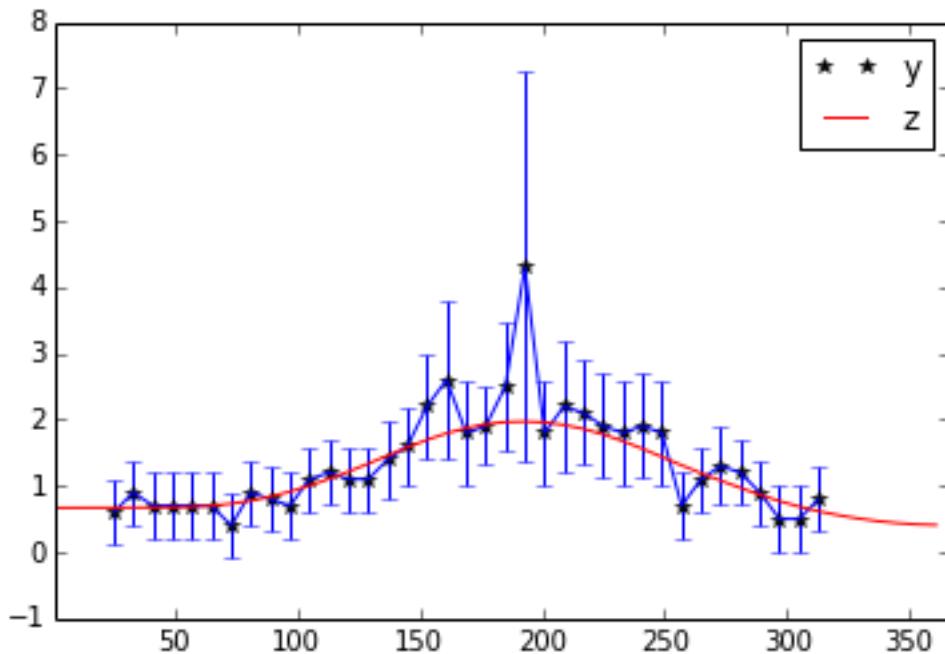
plt.plot(x,z,'r',label='z')

plt.xlim(1,366)
plt.legend(loc='best')

7.56265788653 True

<matplotlib.legend.Legend at 0x2b75e6878890>

```



To apply this approach to our 3D dataset, we could simply loop over all pixels.

Note that *any* per-pixel processing will be slow ... but this is quite a fast smoothing method, so is feasible here.

```

# we have put in an axis control to smoothn
# here so it will only smooth over day
# This will take a few minutes to process
# we switch on verbose mode to get some feedback
# on progress

# make a mask of pixels where there is at least 1 sample
# over the time period
mask = (data.mask.sum(axis=0) == 0)
mask = np.array([mask]*data.shape[0])

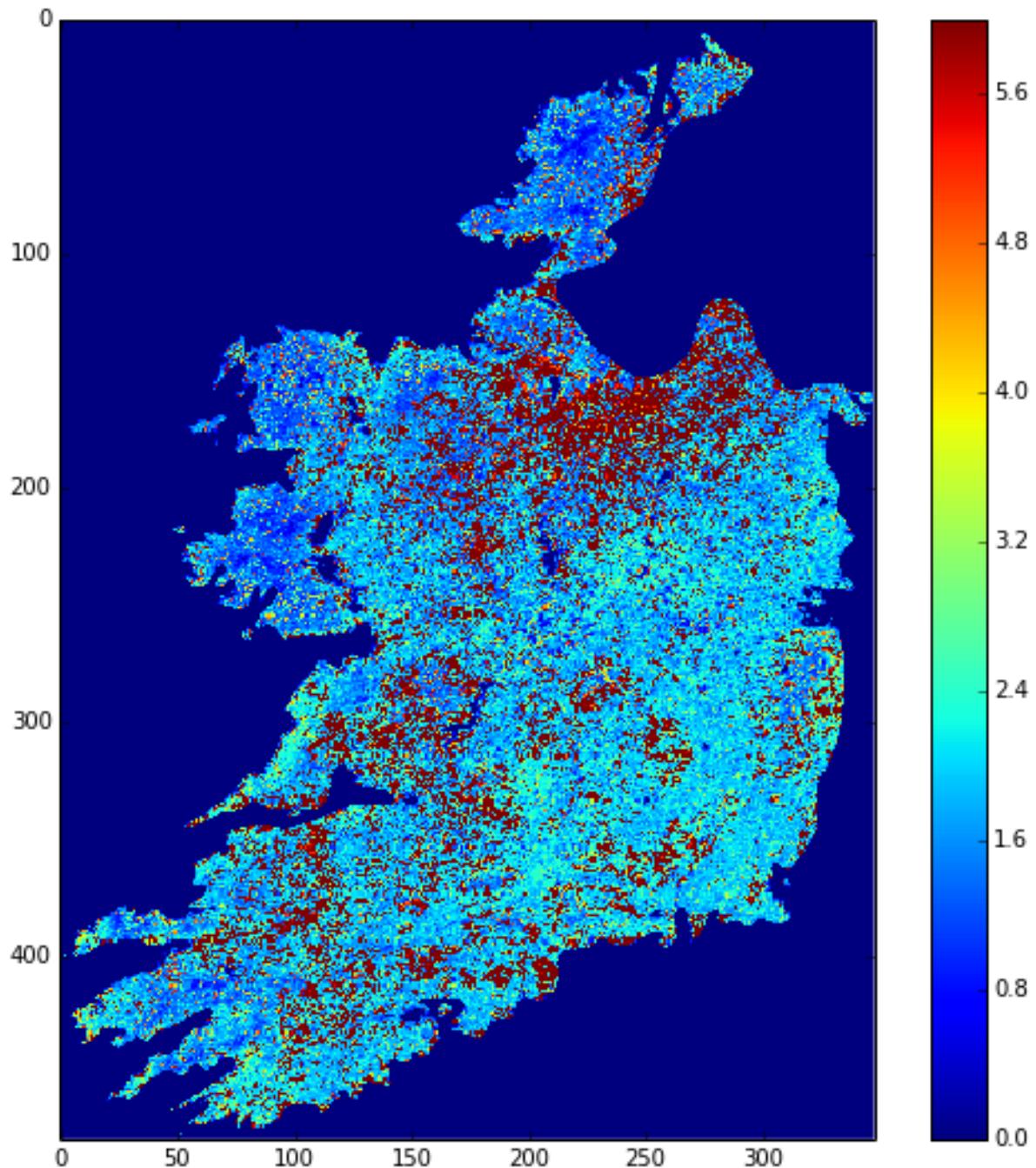
z = smoothn(data,s=5.0, sd=sd, smoothOrder=2.0, axis=0, TolZ=0.05, verbose=True)[0]
z = ma.array(z,mask=mask)

tol 1.0 nit 0
tol 1.03767913976 nit 1
tol 0.695375818129 nit 2
tol 0.55340286659 nit 3
tol 0.379048609608 nit 4
tol 0.297133997656 nit 5
tol 0.211254020382 nit 6
tol 0.161703395437 nit 7
tol 0.118022633002 nit 8
tol 0.089141179031 nit 9
tol 0.0662378920796 nit 10

```

```
plt.figure(figsize=(9,9))
plt.imshow(z[20],interpolation='none',vmax=6)
plt.colorbar()

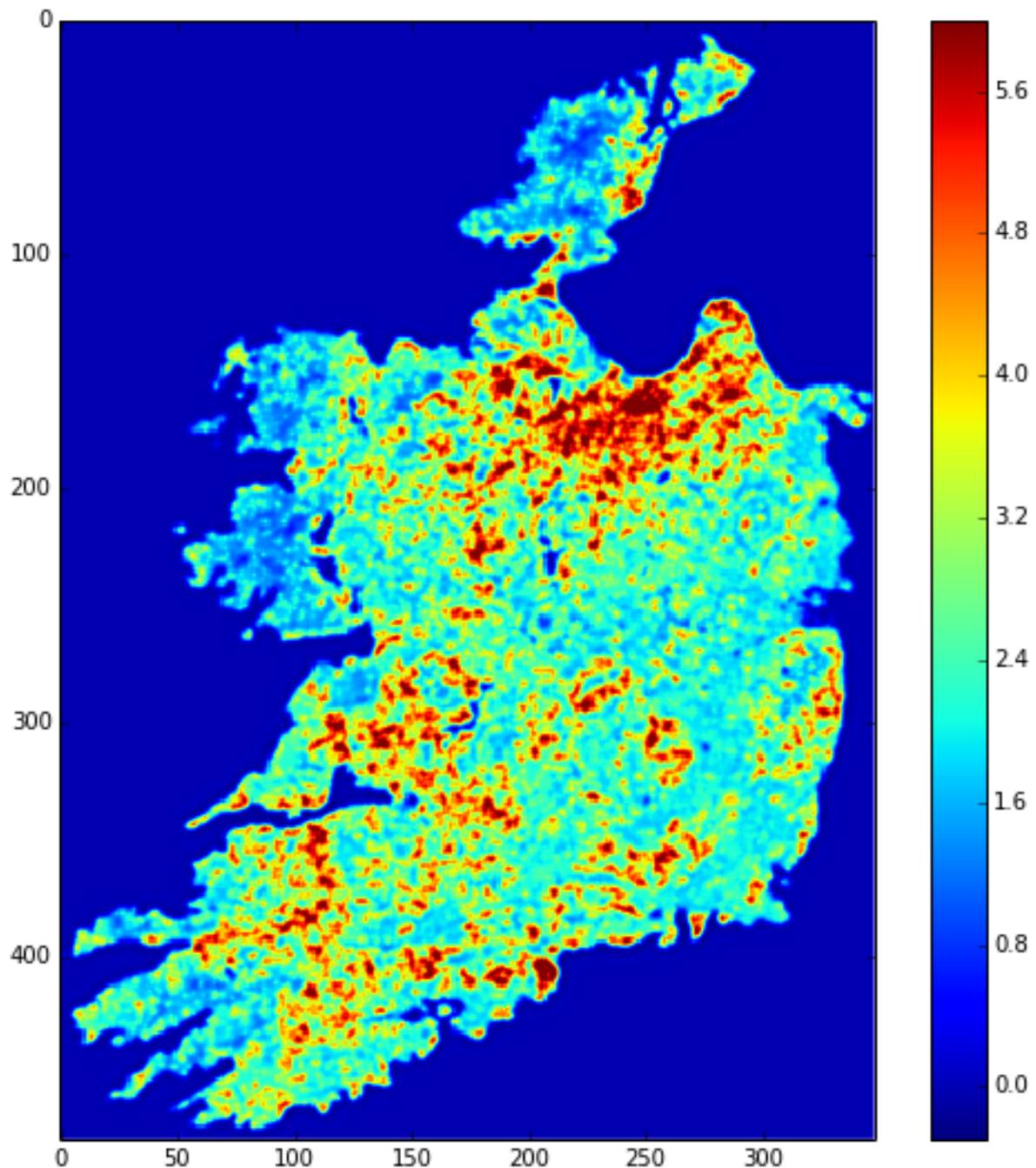
<matplotlib.colorbar.Colorbar instance at 0x2b75d2d9fd40>
```



```
# similarly, take frame 20
# and smooth that

ZZ = smoothn(z[20],smoothOrder=2.)
# self-calibrated smoothness term
s = ZZ[1]
print 's =',s
Z = ZZ[0]
plt.figure(figsize=(9,9))
plt.imshow(Z,interpolation='none',vmax=6)
```

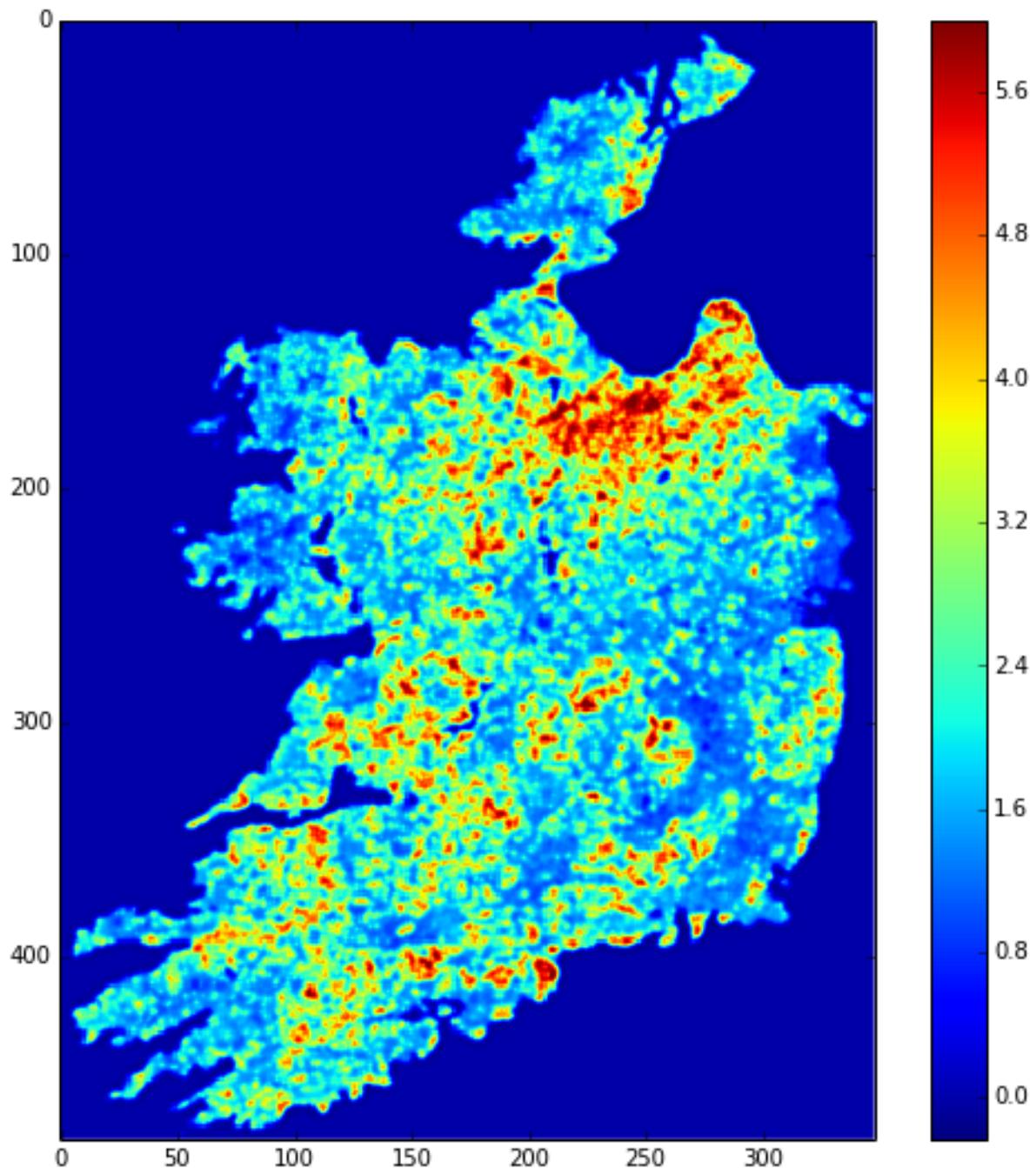
```
plt.colorbar()  
  
s = 0.731142059593  
  
<matplotlib.colorbar.Colorbar instance at 0x1857ee18>
```



```
# similarly, take frame 20  
# and smooth that  
  
ZZ = smoothn(z,s=s,smoothOrder=2.,axis=(1,2),verbose=True)  
  
Z = ZZ[0]  
plt.figure(figsize=(9,9))  
plt.imshow(Z[30],interpolation='none',vmax=6)  
plt.colorbar()
```

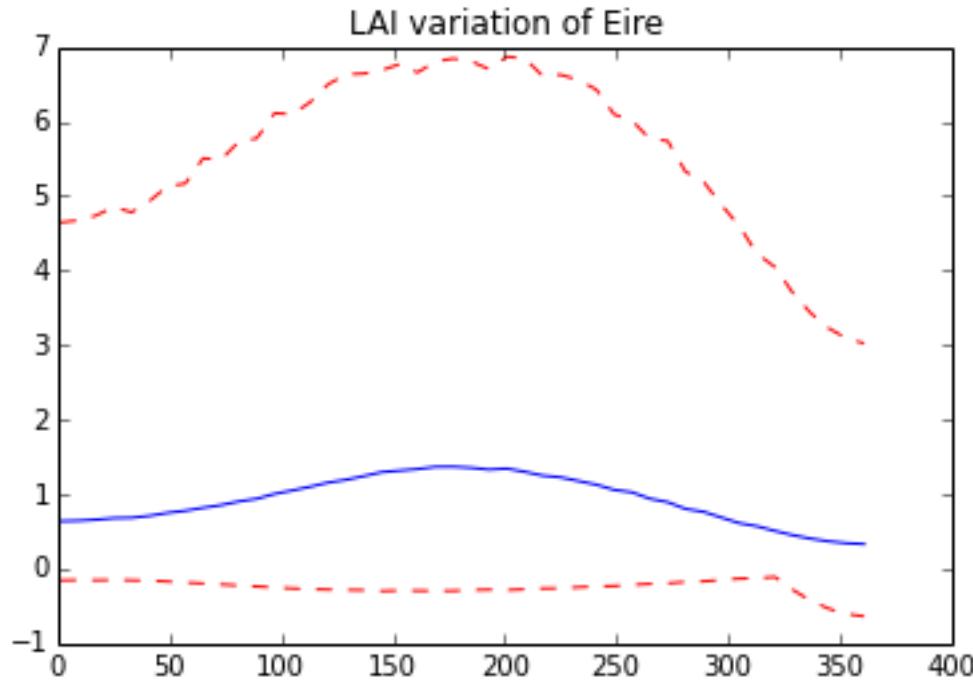
```
tol 1.0 nit 0
```

```
<matplotlib.colorbar.Colorbar instance at 0x23289dd0>
```



```
x = np.arange(46)*8+1.  
try:  
    plt.plot(x,np.mean(Z,axis=(1,2)))  
    plt.plot(x,np.min(Z,axis=(1,2)), 'r--')  
    plt.plot(x,np.max(Z,axis=(1,2)), 'r--')  
except:  
    plt.plot(x,np.mean(Z,axis=2).mean(axis=1))  
    plt.plot(x,np.min(Z,axis=2).min(axis=1), 'r--')  
    plt.plot(x,np.max(Z,axis=2).max(axis=1), 'r--')  
  
plt.title('LAI variation of Eire')
```

```
<matplotlib.text.Text at 0x232b2ed0>
```



```
# or doing this pixel by pixel ...
# which is slower than using axis

order = 2

# pixels that have some data
mask = (~data.mask).sum(axis=0)

odata = np.zeros((46,) + mask.shape)

rows,cols = np.where(mask>0)

len_x = len(rows)
order = 2
gamma = 5.

for i in xrange(len_x):
    r,c = rows[i],cols[i]
    # progress bar
    if i%(len_x/20) == 0:
        print '... %4.2f percent'%(i*100./float(len_x))
    pixel    = data[:,r,c]
    pixel_sd = sd[:,r,c]

    zz = smoothn(pixel,s=gamma, sd=pixel_sd,smoothOrder=order,TolZ=0.05)
    odata[:,rows[i],cols[i]] = zz[0]

...
0.00 percent
...
5.00 percent
...
10.00 percent
...
15.00 percent
...
20.00 percent
...
25.00 percent
...
30.00 percent
...
35.00 percent
...
40.00 percent
```

```
... 45.00 percent
... 50.00 percent
... 55.00 percent
... 60.00 percent
... 65.00 percent
... 70.00 percent
... 75.00 percent
... 80.00 percent
... 85.00 percent
... 90.00 percent
... 95.00 percent

import pylab as plt
import os

root = 'files/images/lai_eire_colourZ'

for i,f in enumerate(lai['filenames']):
    fig = plt.figure(figsize=(7,7))
    # get some info from filename
    file_id = f.split('/')[-1].split('.')[1:-5][1:]
    print file_id
    plt.imshow(Z[i],interpolation='none',vmax=6.,vmin=0.0)
    # plot a jpg
    plt.title(file_id)
    plt.colorbar()
    plt.savefig('%s_%s.jpg'%(root,file_id))
    plt.close(fig)

2005001
2005009
2005017
2005025
2005033
2005041
2005049
2005057
2005065
2005073
2005081
2005089
2005097
2005105
2005113
2005121
2005129
2005137
2005145
2005153
2005161
2005169
2005177
2005185
2005193
2005201
2005209
2005217
2005225
2005233
2005241
2005249
2005257
2005265
```

```
2005273
2005281
2005289
2005297
2005305
2005313
2005321
2005329
2005337
2005345
2005353
2005361

cmd = 'convert -delay 100 -loop 0 {0}_*.jpg {0}_movie2.gif'.format(root)
os.system(cmd)

0
```

42.2.3 5.3 Function fitting

Sometimes, instead of applying some arbitrary smoothing function to data, we want to extract particular information from the time series.

One way to approach this is to fit some function to the time series at each location.

Let us suppose that we wish to characterise the phenology of vegetation in Ireland.

One way we could do this would be to look in the lai data for the most rapid changes.

Another would be to explicitly fit some mathematical function to the LAI data that would expect to describe typical LAI trajectories.

One example of such a function is the double logistic. A logistic function is:

$$\hat{y} = p_0 - p_1 \left(\frac{1}{1 + e^{p_2(t-p_3)}} + \frac{1}{1 + e^{p_4(t-p_5)}} - 1 \right)$$

We can give a function for a double logistic:

```
def dbl_logistic_model ( p, t ):
    """A double logistic model, as in Sobrino and Juliean, or Zhang et al"""
    return p[0] - p[1]* ( 1./(1+np.exp(p[2]*(t-p[3]))) + \
                         1./(1+np.exp(-p[4]*(t-p[5]))) ) - 1 )

tile = 'h17v03'
year = '2005'

# specify the file with the urls in
ifile= 'files/data/modis_lai_%s_%s.txt'%(tile,year)

fp = open(ifile)
filelist = [url.split('/')[-1].strip() for url in fp.readlines()]
fp.close()
import sys
sys.path.insert(0,'files/python')

from get_lai import *

try:
```

```

data = lai['Lai_1km']
sd = lai['LaiStdDev_1km']
except:
    lai = read_lai(filelist, country='IRELAND')
    data = lai['Lai_1km']
    sd = lai['LaiStdDev_1km']

thresh = 0.25
sd[sd# test pixel
r = 472
c = 84

y = data[:,r,c]
mask = ~y.mask
y = np.array(y[mask])
x = (np.arange(46)*8+1.) [mask]
unc = np.array(sd[:,r,c] [mask])

```

And see what this looks like:

```

# define x (time)
x_full = np.arange(1,366)

# some default values for the parameters
p = np.zeros(6)

# some stats on y
ysd = np.std(y)
ymean = np.mean(y)

# some rough guesses at the parameters

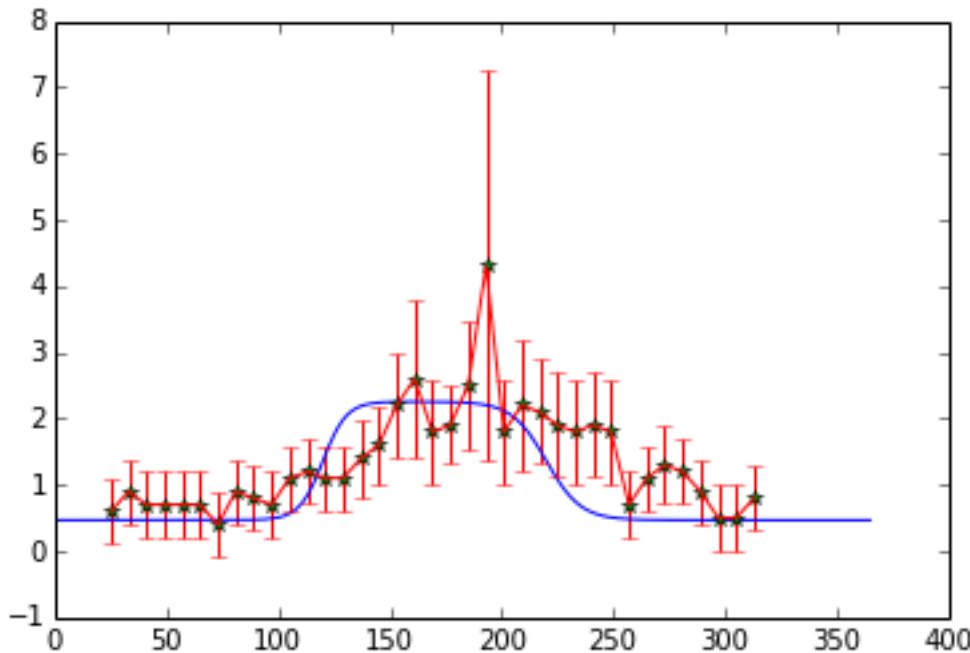
p[0] = ymean - 1.151*ysd;      # minimum (1.151 is 75% CI)
p[1] = 2*1.151*ysd           # range
p[2] = 0.19                   # related to up slope
p[3] = 120                     # midpoint of up slope
p[4] = 0.13                   # related to down slope
p[5] = 220                     # midpoint of down slope

y_hat = dbl_logistic_model(p,x_full)

plt.clf()
plt.plot(x_full,y_hat)
plt.plot(x,y,'*')
plt.errorbar(x,y,unc*1.96)

<Container object of 3 artists>

```



We could manually ‘tweak’ the parameters until we got a better ‘fit’ to the observations.

First though, let’s define a measure of ‘fit’:

$$Z_i = \frac{\hat{y}_i - y_i}{\sigma_i}$$

$$Z^2 = \sum_i Z_i^2 = \sum_i \left(\frac{\hat{y}_i - y_i}{\sigma_i} \right)^2$$

and implement this as a mismatch function where we have data points:

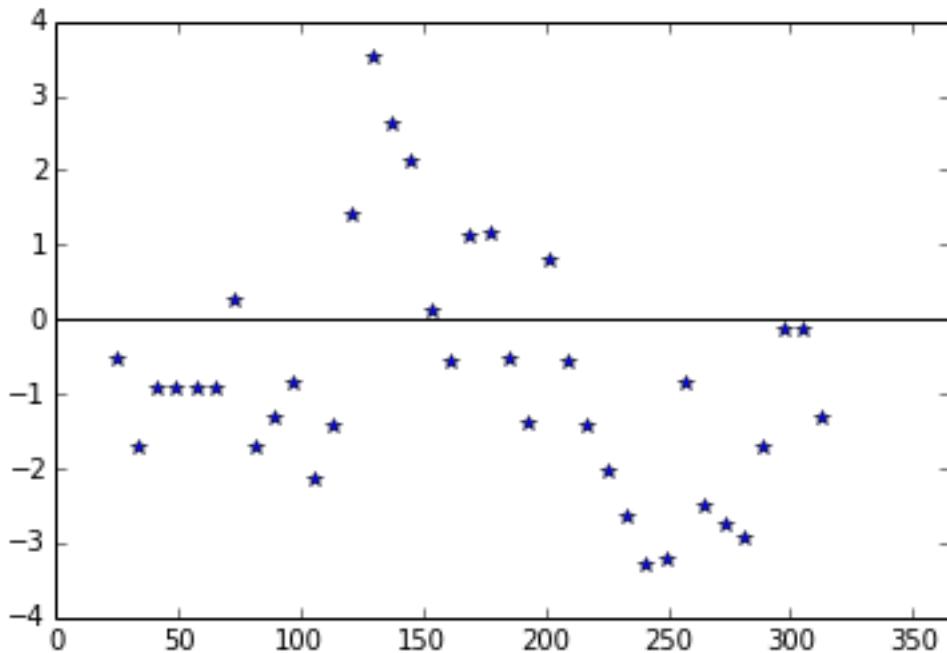
```
def mismatch_function(p, x, y, unc):
    y_hat = dbl_logistic_model(p, x)
    diff = (y_hat - y) / unc
    return diff
```

```
Z = mismatch_function(p,x,y,unc)
```

```
plt.plot([1,365.],[0,0.], 'k-')
plt.xlim(0,365)
plt.plot(x,Z, '.*')
```

```
print 'Z^2 =', (Z**2).sum()
```

```
Z^2 = 113.325251358
```



Now lets change p a bit:

```

p[0] = ymean - 1.151*ysd;      # minimum (1.151 is 75% CI)
p[1] = 2*1.151*ysd            # range
p[2] = 0.19                    # related to up slope
p[3] = 140                     # midpoint of up slope
p[4] = 0.13                    # related to down slope
p[5] = 220                     # midpoint of down slope

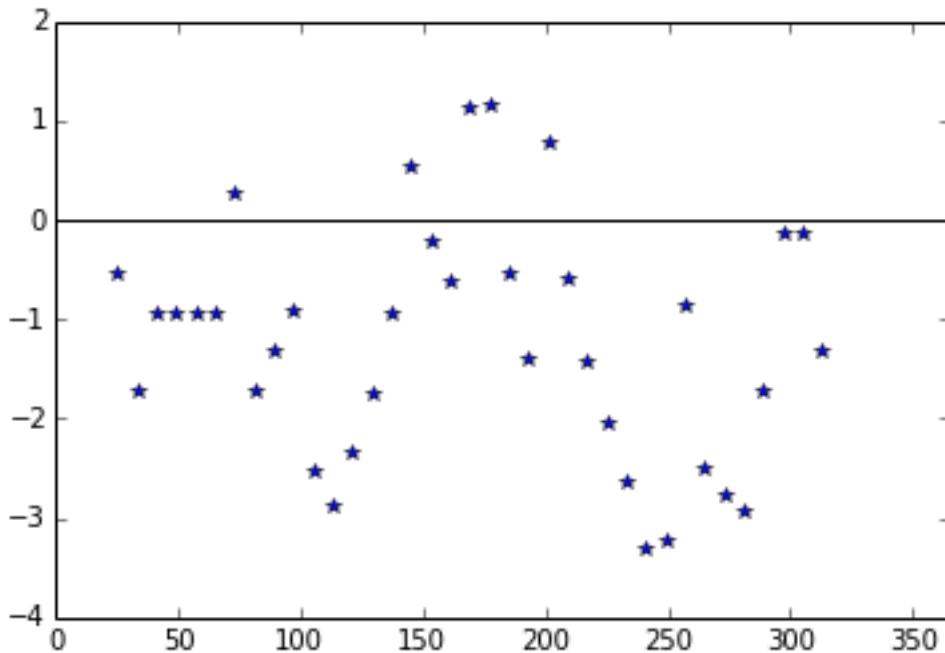
Z = mismatch_function(p,x,y,unc)

plt.plot([1,365.],[0,0.], 'k-')
plt.xlim(0,365)
plt.plot(x,Z, '*')

print 'Z^2 =', (Z**2).sum()

Z^2 = 105.478274642

```



We have made the mismatch go down a little ...

Clearly it would be tedious (and impractical) to do a lot of such tweaking, so we can use methods that seek the minimum of some function.

One such method is implemented in `scipy.optimize.leastsq`:

```
from scipy import optimize

# initial estimate is in p
print 'initial parameters:',p[0],p[1],p[2],p[3],p[4],p[5]

# set some bounds for the parameters
bound = np.array([(0.,10.), (0.,10.), (0.01,1.), (50.,300.), (0.01,1.), (50.,300.)])

# test pixel
r = 472
c = 84

y = data[:,r,c]
mask = ~y.mask
y = np.array(y[mask])
x = (np.arange(46)*8+1.) [mask]
unc = np.array(sd[:,r,c] [mask])

# define function to give Z^2

def sse(p,x,y,unc):
    '''Sum of squared error'''
    # penalise p[3] > p[5]
    err = np.max([0.,(p[3] - p[5])])*1e4
    return (mismatch_function(p,x,y,unc)**2).sum() +err

# we pass the function:
#
# sse           : the name of the function we wrote to give
#                   sum of squares of Z_i
# p             : an initial estimate of the parameters
```

```

# args=(x,y,unc)      : the other information (other than p) that
#                      mismatch_function needs
# approx_grad          : if we dont have a function for the gradient
#                      we have to get the solver to approximate it
#                      which takes time ... see if you can work out
#                      d_sse / dp and use that to speed this up!

psolve = optimize.fmin_l_bfgs_b(sse,p,approx_grad=True,iprint=-1,\n                                args=(x,y,unc),bounds=bound)

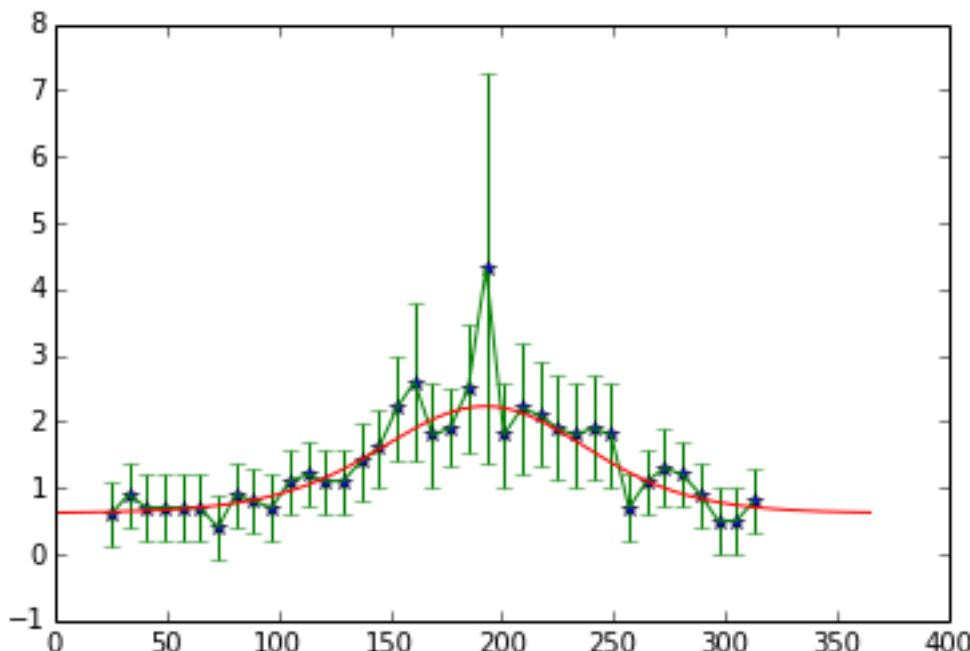
print psolve[1]
pp = psolve[0]
plt.plot(x,y,'*')
plt.errorbar(x,y,unc*1.96)
y_hat = dbl_logistic_model(pp,x_full)
plt.plot(x_full,y_hat)

print 'solved parameters: ',pp[0],pp[1],pp[2],pp[3],pp[4],pp[5]

# if we define the phenology as the parameter p[3]
# and the 'length' of the growing season:
print 'phenology',pp[3],pp[5]-pp[3]

initial parameters: 1.04703335612 1.62445180627 0.19 140.0 0.13 220.0
23.2882812239
solved parameters:  0.614031423522 2.92120123969 0.0357337165804 159.497584338 0.0381324783625 22
phenology 159.497584338 67.5051369225

```



```

# and run over each pixel ... this will take some time

# pixels that have some data
mask = (~data.mask).sum(axis=0)

pdata = np.zeros((7,) + mask.shape)

rows,cols = np.where(mask>0)
len_x = len(rows)

# lets just do some random ones to start with

```

```
#rows = rows[::10]
#cols = cols[::10]

len_x = len(rows)

for i in xrange(len_x):
    r,c = rows[i],cols[i]
    # progress bar
    if i%(len_x/40) == 0:
        print '... %4.2f percent' %(i*100./float(len_x))

    y = data[:,r,c]
    mask = ~y.mask
    y = np.array(y[mask])
    x = (np.arange(46)*8+1.) [mask]
    unc = np.array(sd[:,r,c] [mask])

    # need to get an initial estimate of the parameters

    # some stats on y
    ysd = np.std(y)
    ymean = np.mean(y)

    p[0] = ymean - 1.151*ysd;      # minimum (1.151 is 75% CI)
    p[1] = 2*1.151*ysd           # range
    p[2] = 0.19                   # related to up slope
    p[3] = 140                     # midpoint of up slope
    p[4] = 0.13                   # related to down slope
    p[5] = 220                     # midpoint of down slope

    # set factr to quite large number (relative error in solution)
    # as it'll take too long otherwise
    psolve = optimize.fmin_l_bfgs_b(sse,p,approx_grad=True,iprint=-1,
                                    args=(x,y,unc),bounds=bound,factr=1e12)

    pdata[:-1,rows[i],cols[i]] = psolve[0]
    pdata[-1,rows[i],cols[i]] = psolve[1] # sse

    ... 0.00 percent
    ... 2.50 percent
    ... 5.00 percent
    ... 7.50 percent
    ... 10.00 percent
    ... 12.50 percent
    ... 15.00 percent
    ... 17.50 percent
    ... 19.99 percent
    ... 22.49 percent
    ... 24.99 percent
    ... 27.49 percent
    ... 29.99 percent
    ... 32.49 percent
    ... 34.99 percent
    ... 37.49 percent
    ... 39.99 percent
    ... 42.49 percent
    ... 44.99 percent
    ... 47.49 percent
    ... 49.99 percent
    ... 52.49 percent
    ... 54.99 percent
```

```
... 57.49 percent
... 59.98 percent
... 62.48 percent
... 64.98 percent
... 67.48 percent
... 69.98 percent
... 72.48 percent
... 74.98 percent
... 77.48 percent
... 79.98 percent
... 82.48 percent
... 84.98 percent
... 87.48 percent
... 89.98 percent
... 92.48 percent
... 94.98 percent
... 97.47 percent
... 99.97 percent

plt.figure(figsize=(10,10))
plt.imshow(pdata[3],interpolation='none',vmin=137,vmax=141)
plt.title('green up day')
plt.colorbar()

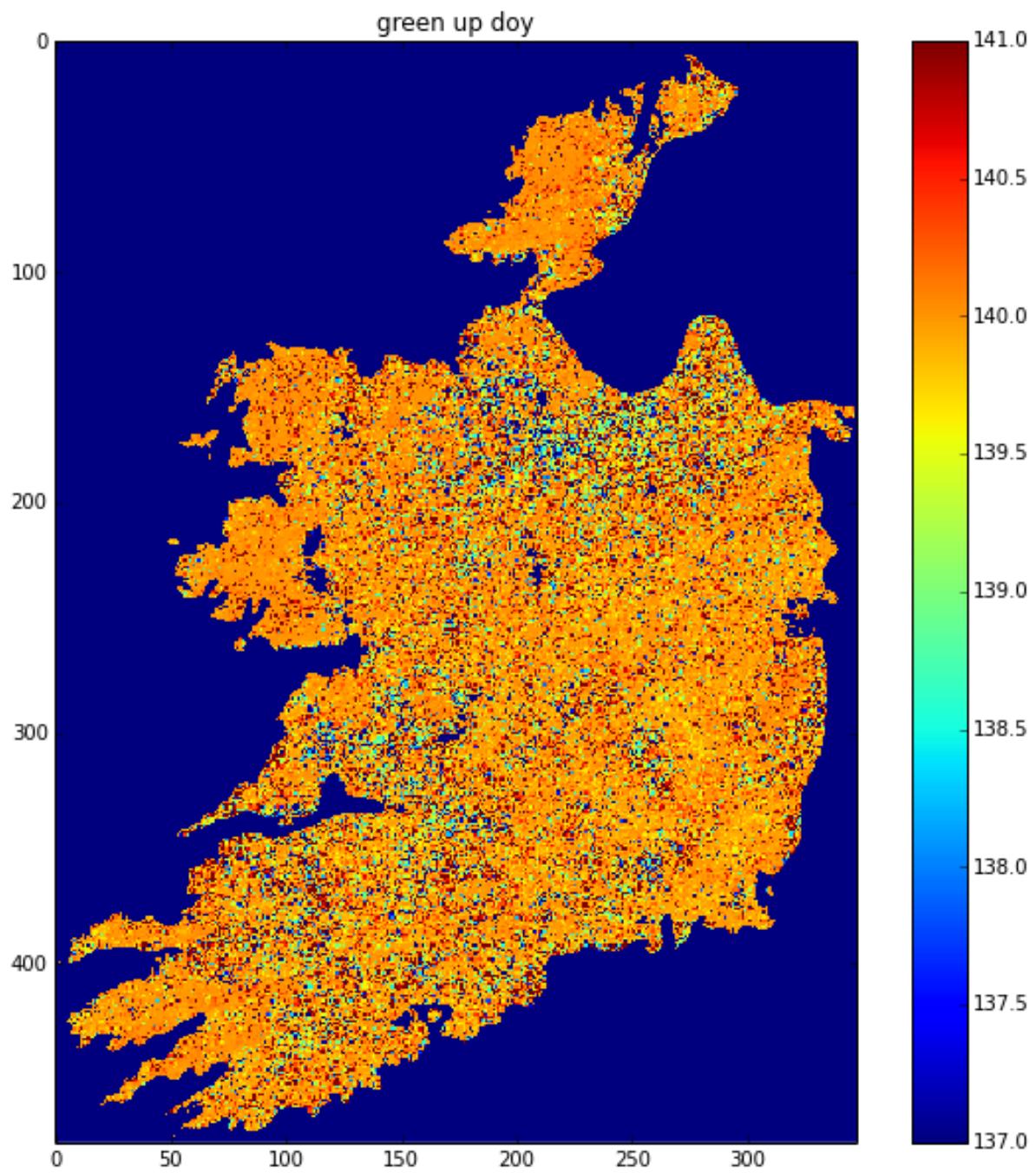
plt.figure(figsize=(10,10))
plt.imshow(pdata[5]-pdata[3],interpolation='none',vmin=74,vmax=84)
plt.title('season length')
plt.colorbar()

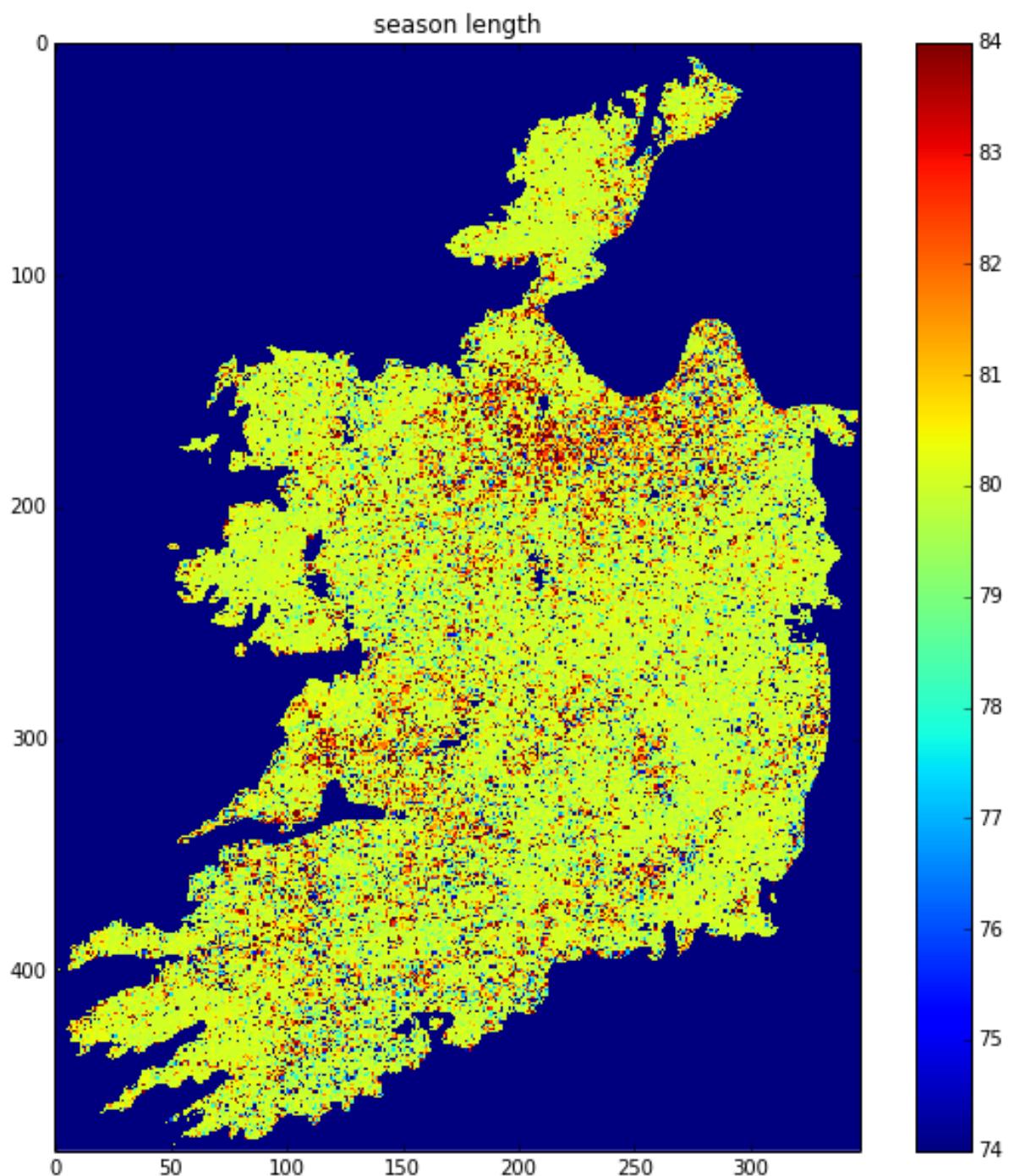
plt.figure(figsize=(10,10))
plt.imshow(pdata[0],interpolation='none',vmin=0.,vmax=6.)
plt.title('min LAI')
plt.colorbar()

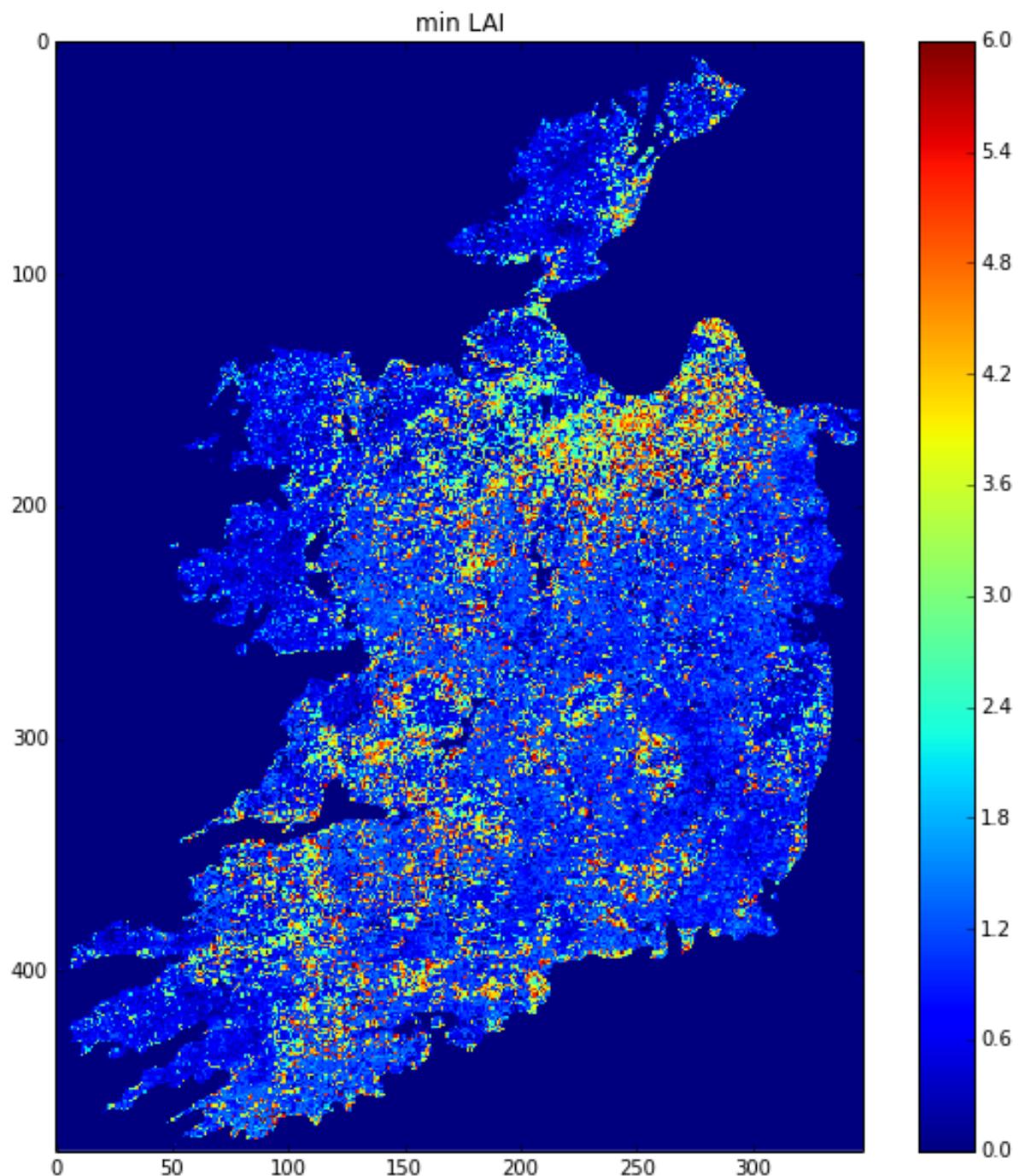
plt.figure(figsize=(10,10))
plt.imshow(pdata[1]+pdata[0],interpolation='none',vmin=0.,vmax=6.)
plt.title('max LAI')
plt.colorbar()

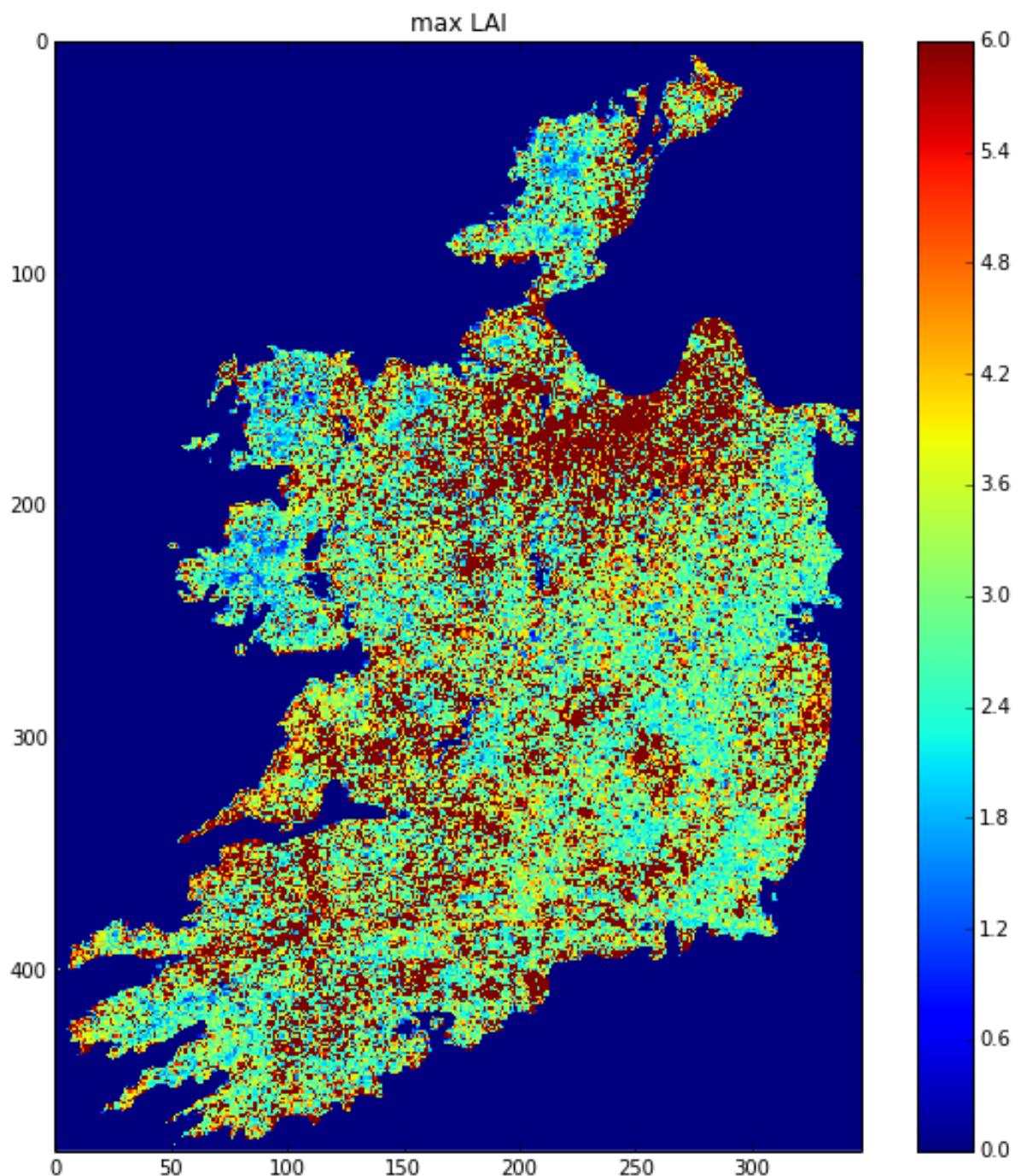
plt.figure(figsize=(10,10))
plt.imshow(np.sqrt(pdata[-1]),interpolation='none',vmax=np.sqrt(500))
plt.title('RSSE')
plt.colorbar()

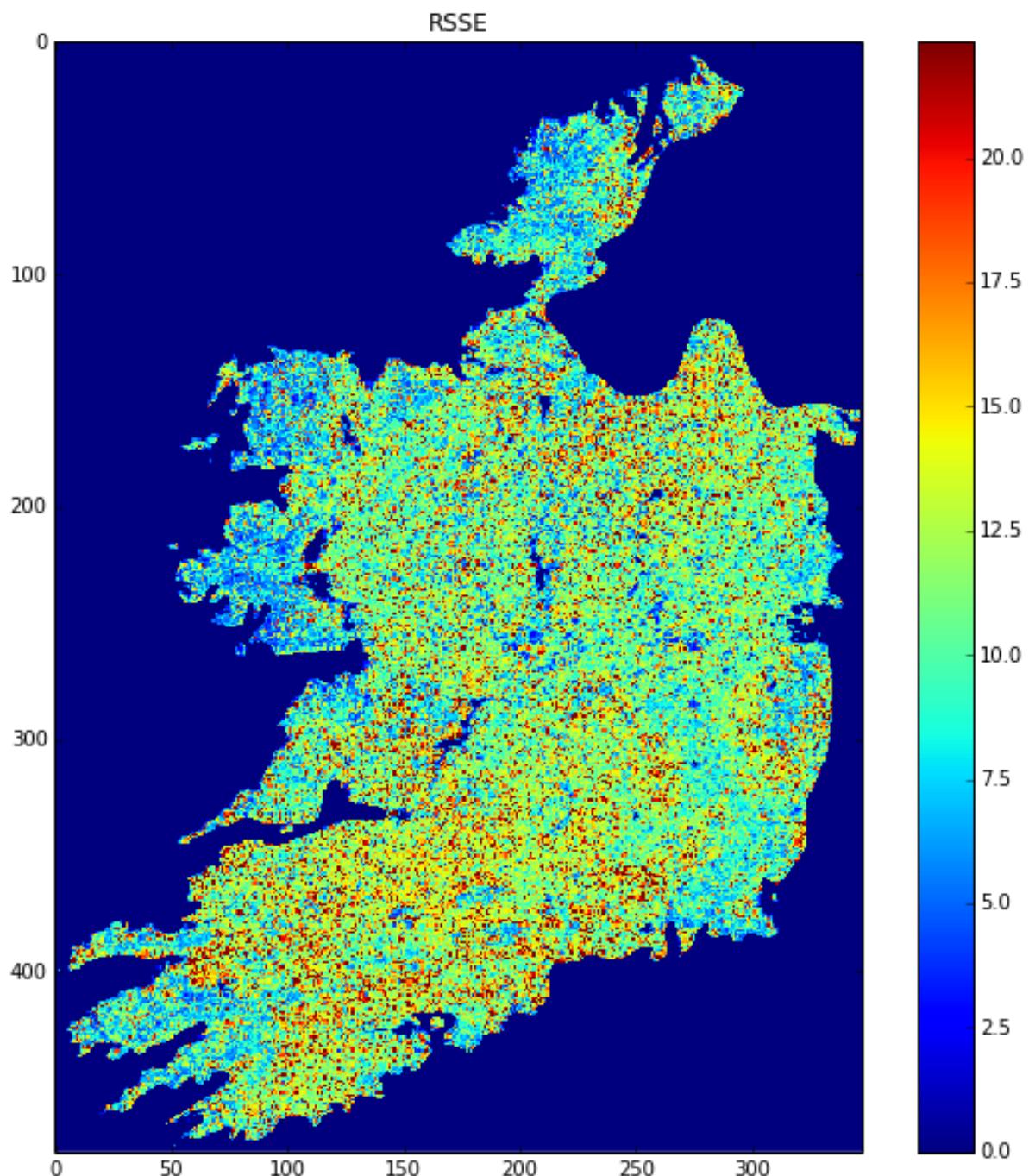
<matplotlib.colorbar.Colorbar instance at 0x2b75e5deefc8>
```











```
# check a few pixels
c = 200

for r in xrange(200,400,25):
    y = data[:,r,c]
    mask = ~y.mask
    y = np.array(y[mask])
    x = (np.arange(46)*8+1.)[mask]
    unc = np.array(sd[:,r,c][mask])

    x_full = np.arange(1,366)

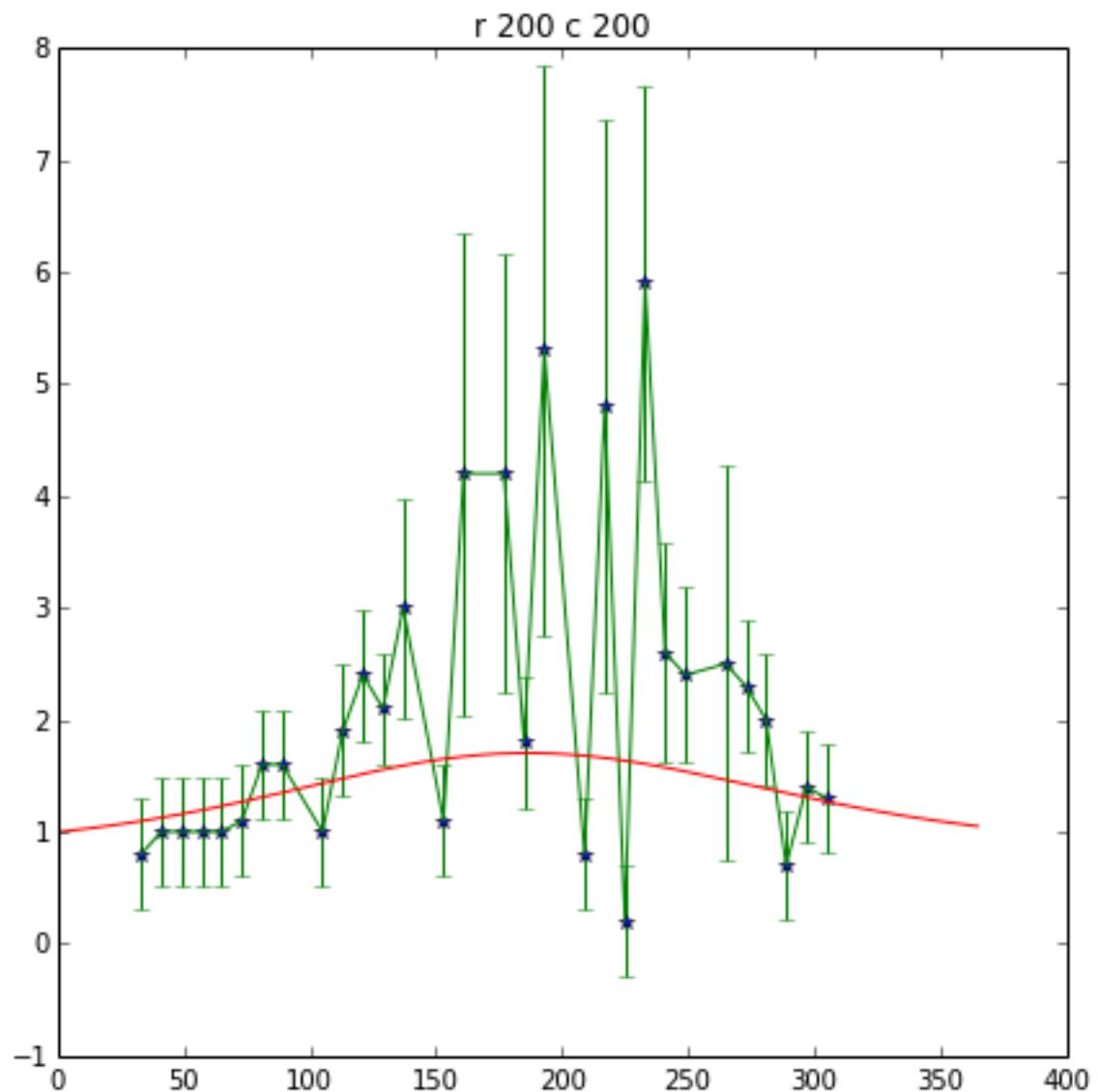
    # some default values for the parameters
    pp = pdata[:-1,r,c]
```

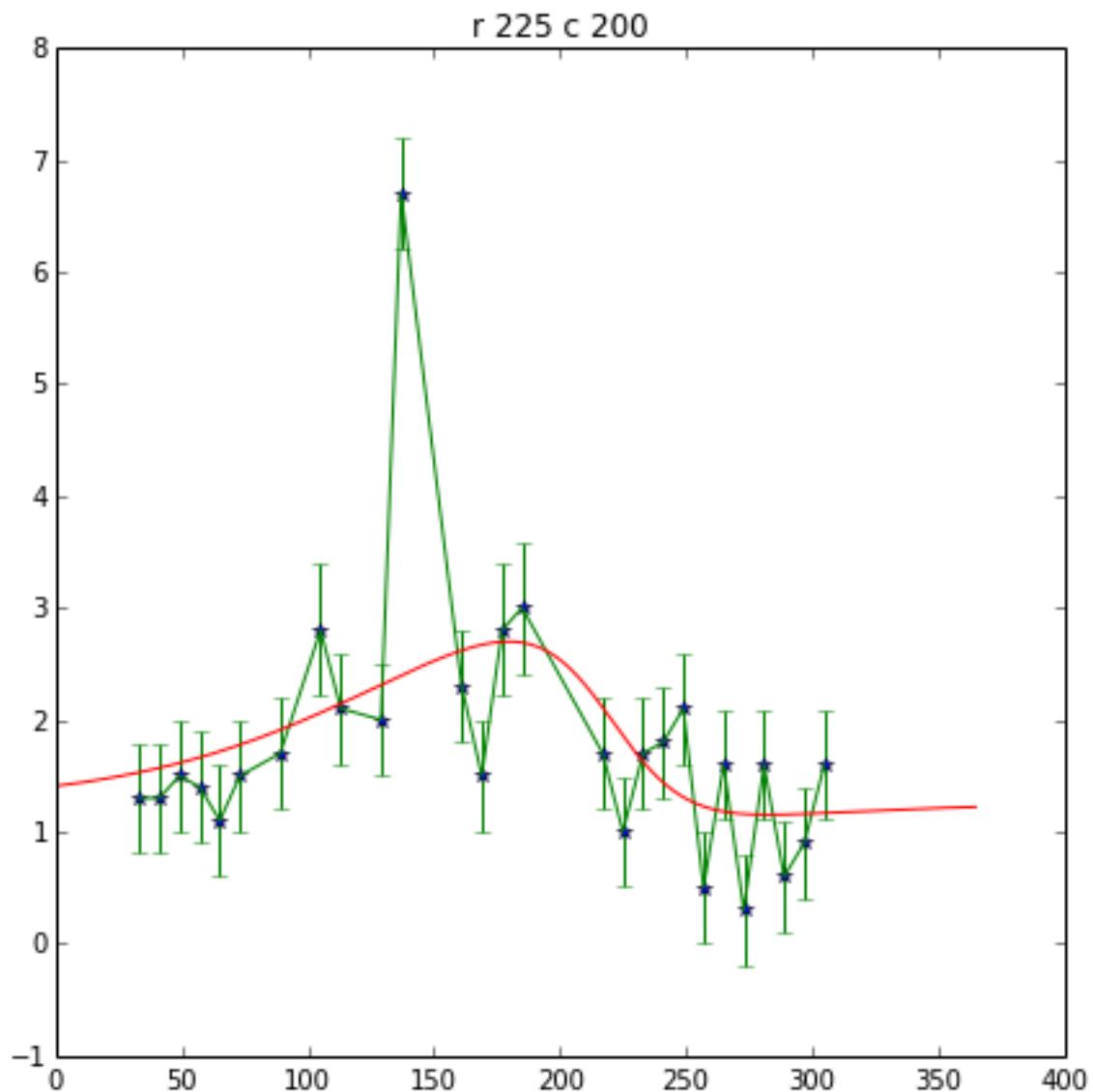
```
plt.figure(figsize=(7,7))
plt.title('r %d c %d'%(r,c))
plt.plot(x,y,'*')
plt.errorbar(x,y,unc*1.96)
y_hat = dbl_logistic_model(pp,x_full)
plt.plot(x_full,y_hat)

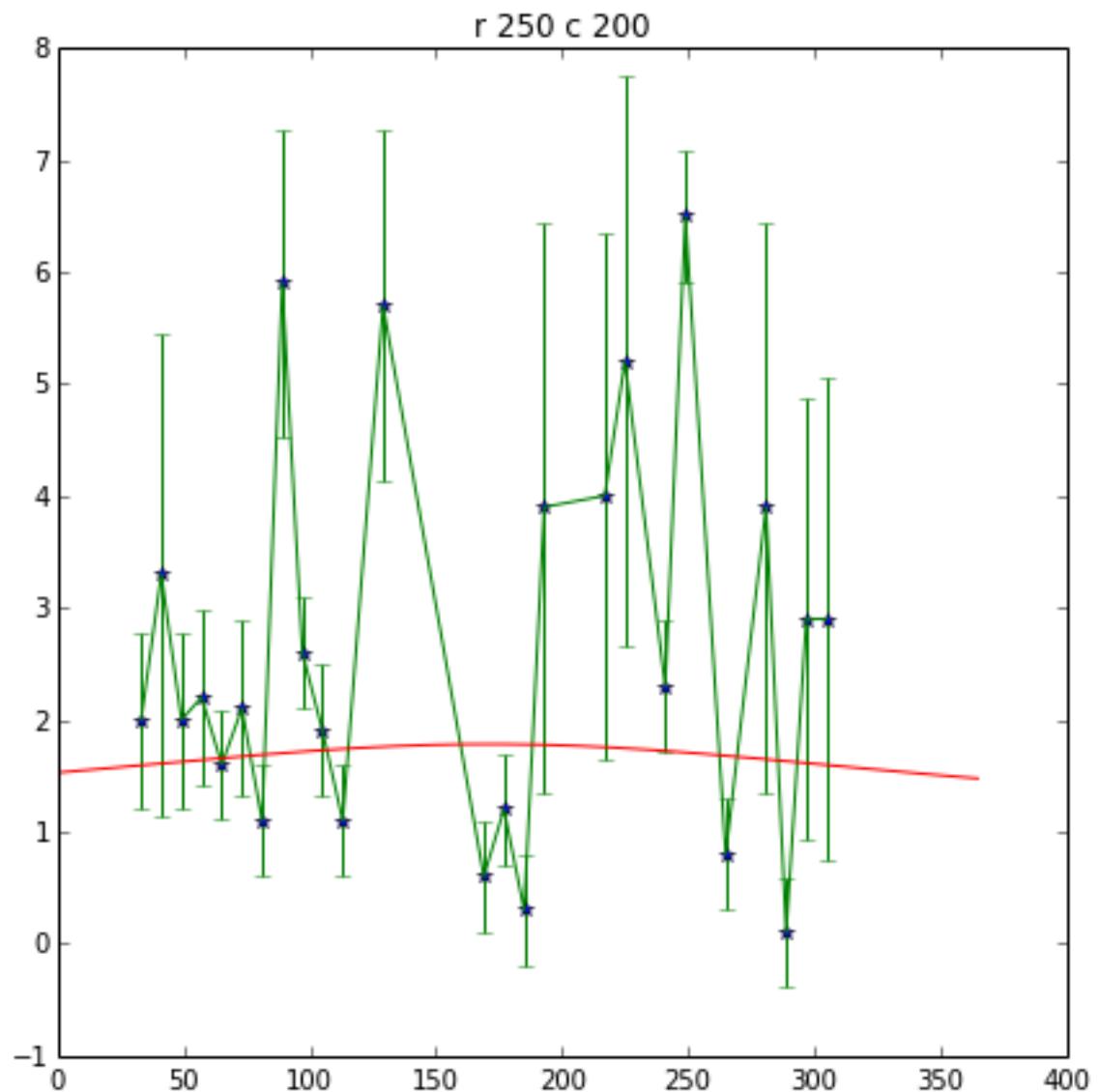
print 'solved parameters: ',pp[0],pp[1],pp[2],pp[3],pp[4],pp[5]

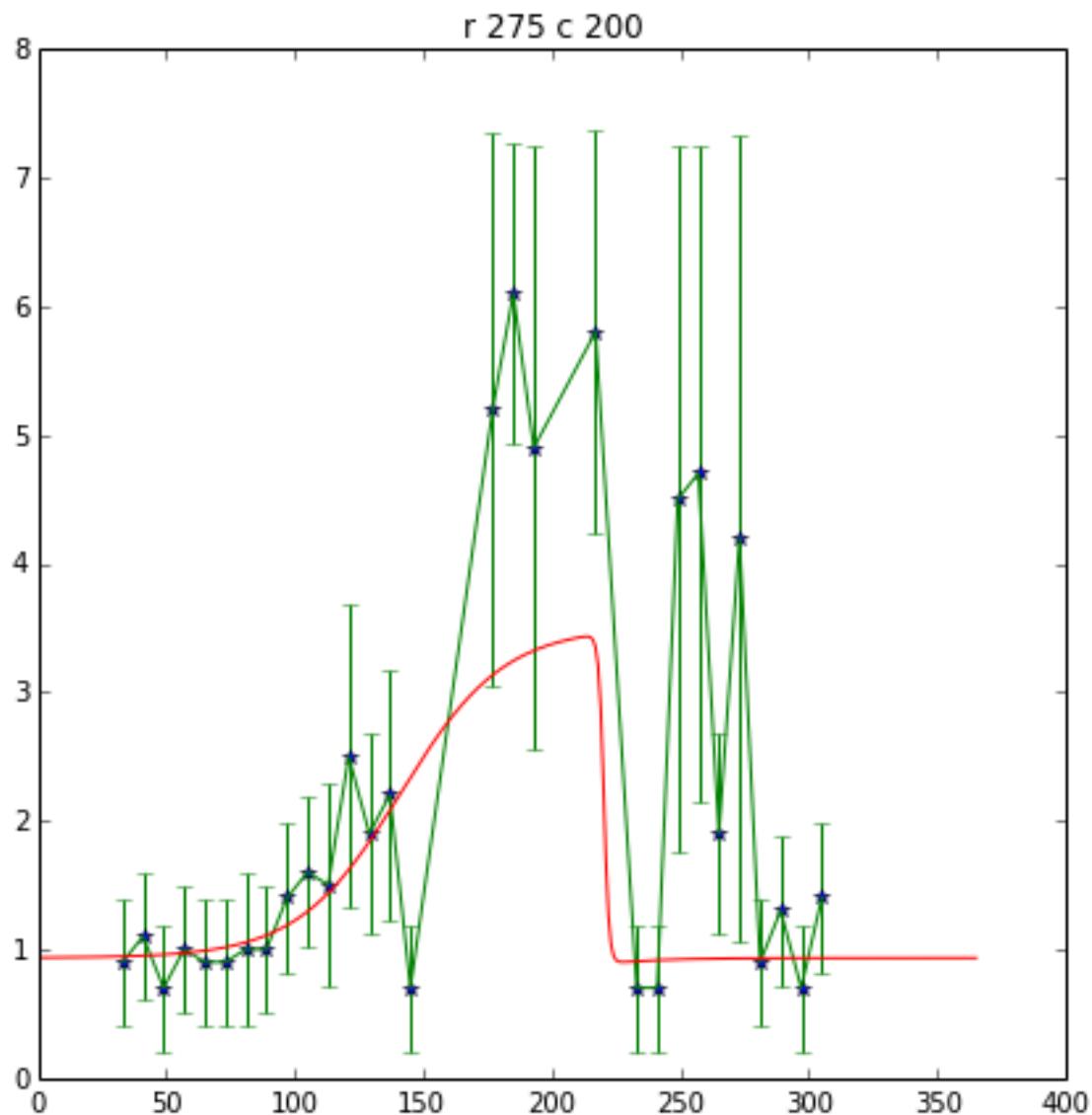
# if we define the phenology as the parameter p[3]
# and the 'length' of the growing season:
print 'phenology',pp[3],pp[5]-pp[3]

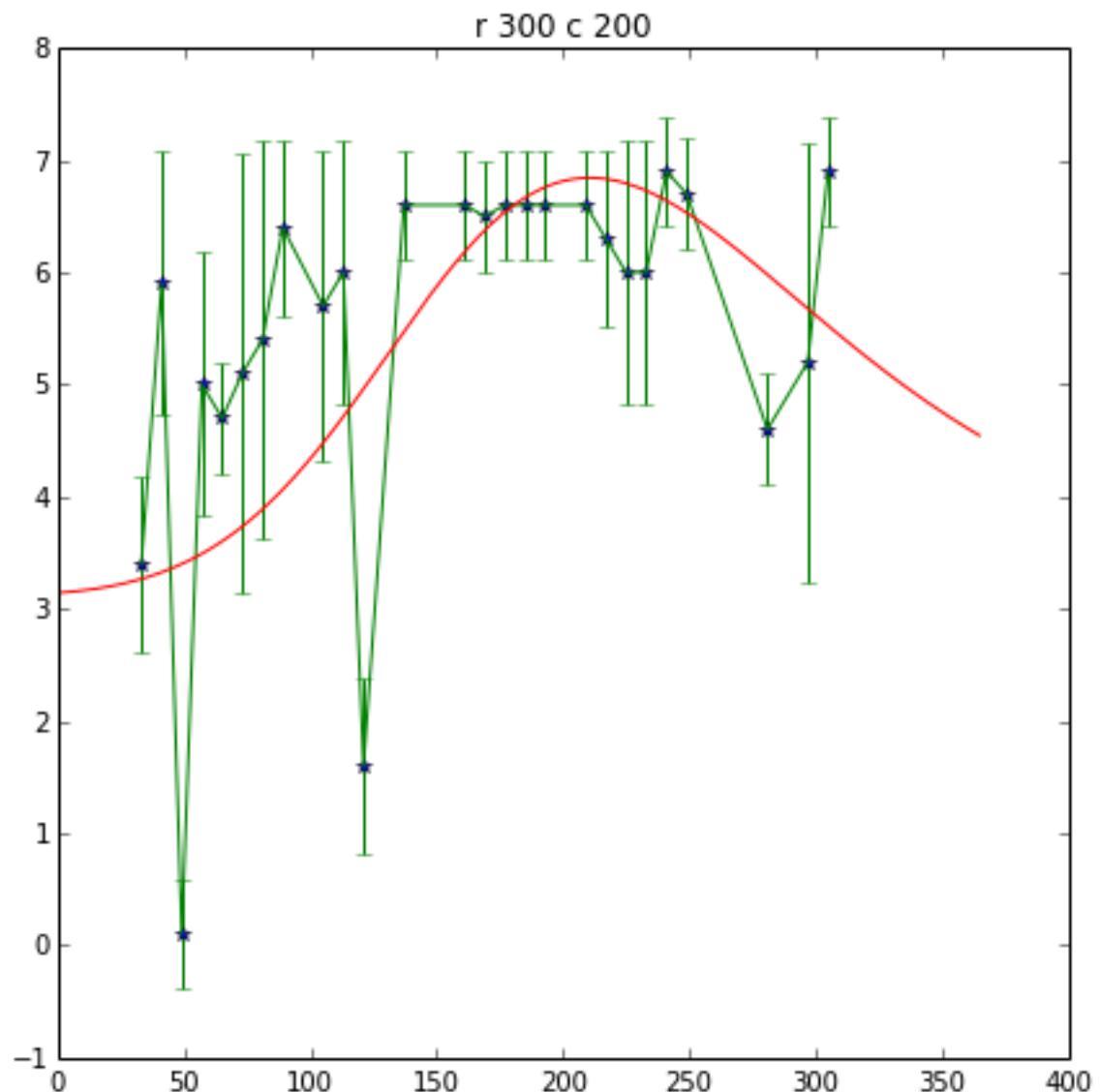
solved parameters:  0.837899998448 2.82427082241 0.0164922480786 139.933827392 0.015218312802 219
phenology 139.933827392 79.6862154345
solved parameters:  1.25111026954 2.3270241822 0.0190331997407 137.746147061 0.0650489530419 220.
phenology 137.746147061 82.3029586718
solved parameters:  1.24735745202 2.64641924245 0.01 139.923207754 0.0103960017819 220.21982824
phenology 139.923207754 80.2966204854
solved parameters:  0.932759824658 2.57001493023 0.0499114191497 141.274250409 1.0 220.241108243
phenology 141.274250409 78.9668578339
solved parameters:  3.3118926764 10.0 0.0227526645775 141.59060759 0.0129963184052 218.391152509
phenology 141.59060759 76.8005449187
solved parameters:  1.00690819411 2.5680266473 0.0244446244614 139.049623755 0.0196297496409 220.
phenology 139.049623755 81.1447880828
solved parameters:  1.41787972842 0.735065812424 0.183737093991 139.730989265 0.0642603310649 219
phenology 139.730989265 80.2444864894
solved parameters:  1.03836639365 1.58477670269 0.0296244335923 139.895942883 0.0436631925744 220
phenology 139.895942883 80.1405620773
```

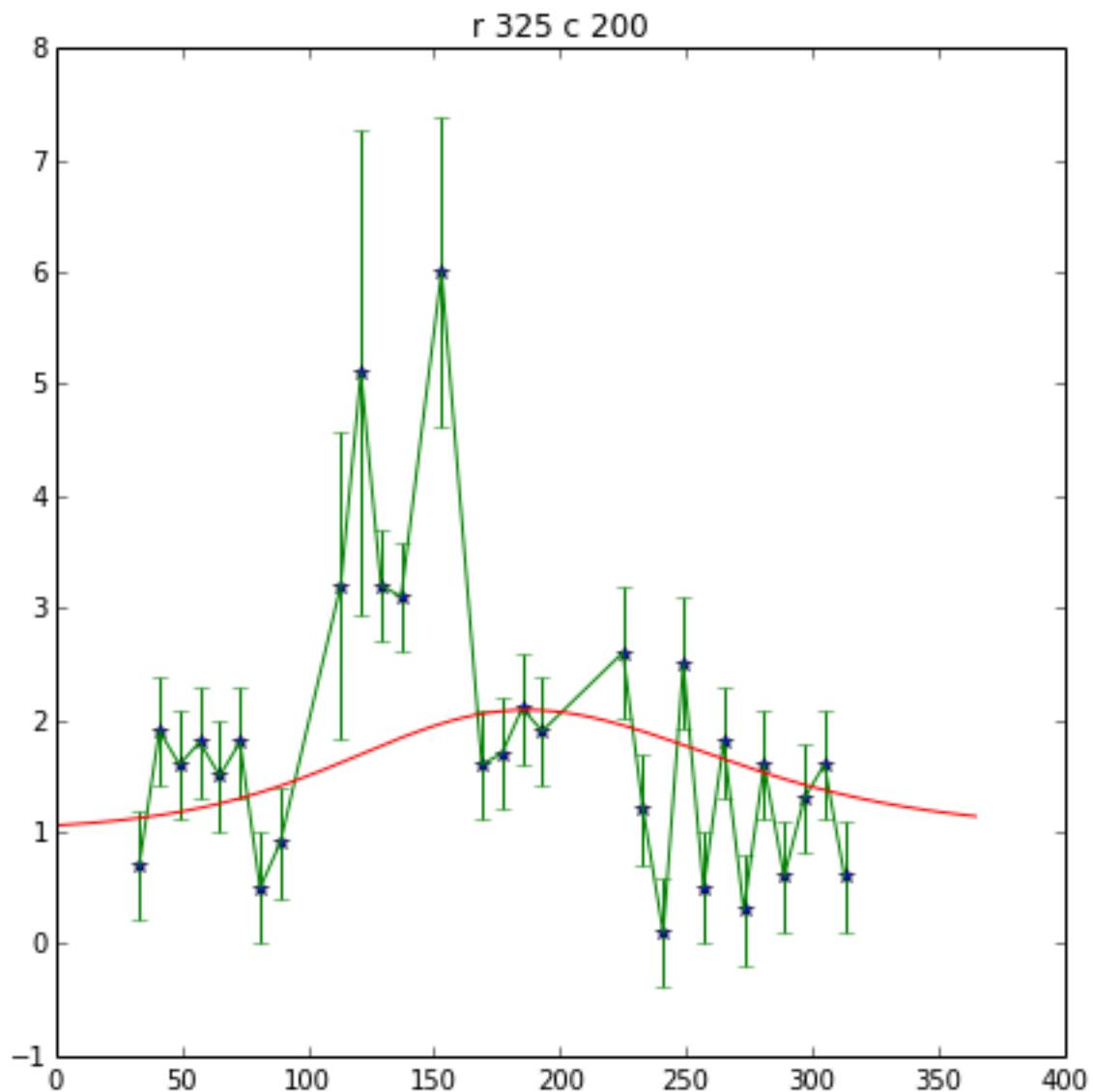


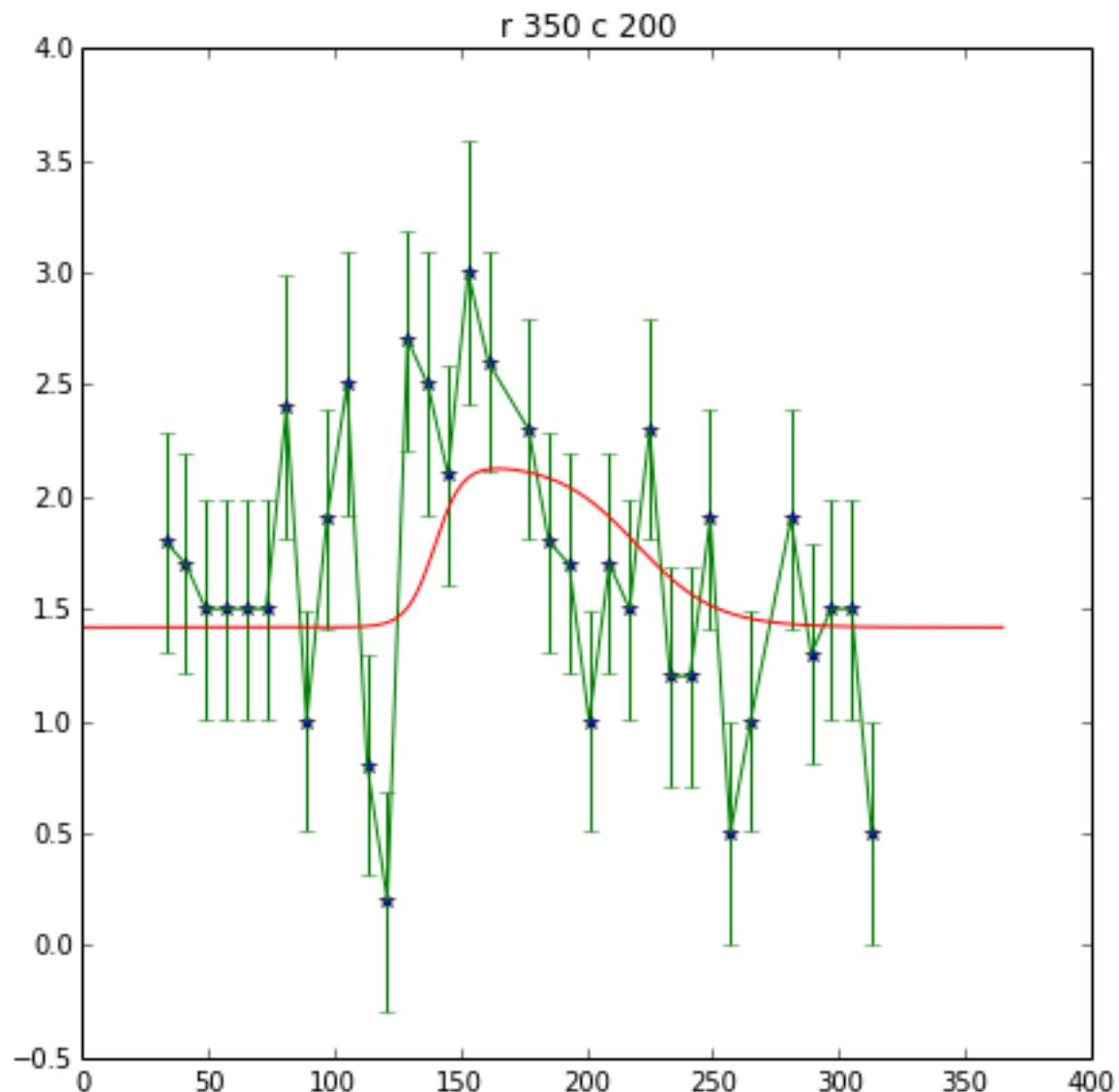


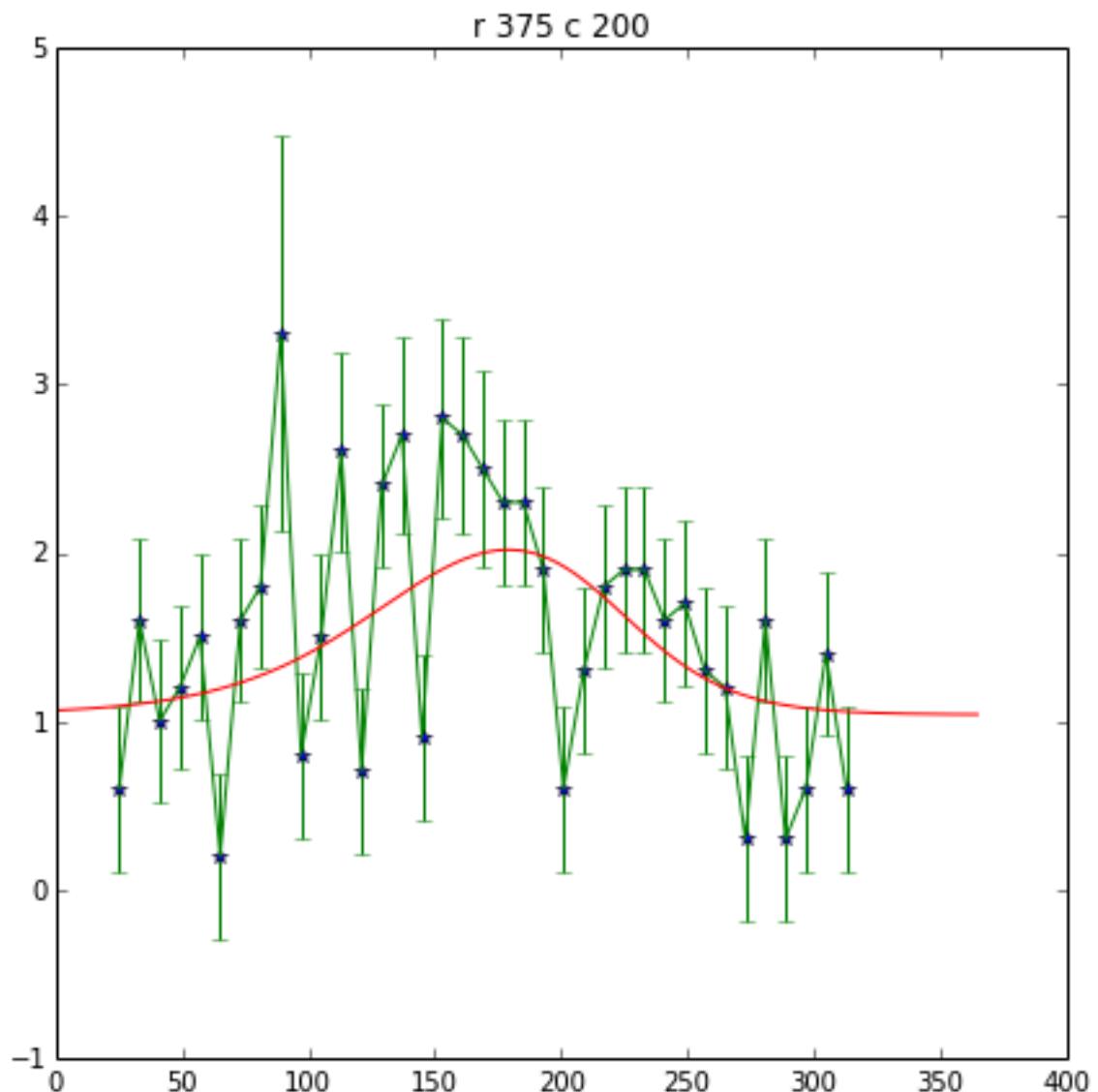












**CHAPTER
FORTYTHREE**

ENVI

The purpose of this week's practical (Reading Week) is for you to become familiar with using the software package ENVI.

You should do one by following the [Classification](#) practical.

There is no direct supervision for this session.

You are expected to get on with this yourselves.

6A. ASSESSED PRACTICAL

44.1 6.1 Introduction

44.1.1 6.1.1 Site, Data and Task overview

These notes describe the practical you must submit for assessment in this course.

The practical comes in two parts: (1) data preparation; (2) modelling.

It is important that you complete both parts of this exercise, as you will need to make use of the code and results in the work you submit for assessment for this course.

- **Data Preparation**

The first task you must complete is to produce a dataset of the proportion of HUC catchment 13010001 (Rio Grande headwaters in Colorado, USA) that is covered by snow for **two consecutive years**, along with associated datasets on temperature (in C) and river discharge at the Del Norte monitoring station. You **may not** use data from the year 2005, as this is given to you in the illustrations above.

The dataset you produce must have a value for the mean snow cover, temperature and discharge in the catchment for every day over each year.

Your write up **must** include fully labelled graph(s) of snow cover, temperature and discharge for the catchment for each year (with units as appropriate), along with some summary statistics (e.g. mean or median, minimum, maximum, and the timing of these).

You **must** provide evidence of how you got these data (i.e. the code and commands you ran to produce the data).

- **Modelling**

You will have prepared two years of data above. Use one of these years to calibrate the (snowmelt) hydrological model (described below) and one year to test it.

The model parameter estimate *must* be objective (i.e. you can't just arbitrarily choose a set) and ideally optimal, and you *must* state the equation of the cost function you will try to minimise and explain the approach used.

You **must** state the values of the model parameters that you have estimated and show evidence for how you went about calculating them. Ideally, you should also state the uncertainty in these parameter estimates (not critical to pass this section though).

You **must** quantify the goodness of fit between your measured flow data and that produced by your model, both for the calibration exercise and the validation.

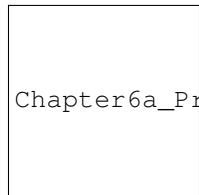
You **must** work individually on this task. If you do not, it will be treated as plagiarism. By reading these instructions for this exercise, we assume that you are aware of the UCL rules on plagiarism. You can find more information on this matter in your student handbook. If in doubt about what might constitute plagiarism, ask one of the course convenors.

What you are going to do is to build, calibrate and test a (snowmelt) hydrological model, driven by observations in the Rio Grande Headwaters in Colorado, USA.

You will need to process two years of data (N.B. *not* 2005 as that is given in the illustrations).

The purpose of the model is to describe the streamflow at the Del Norte measurement station, just on the edge of the catchment.

The average climate for Del Norte is:



Chapter6a_Practical/files/images/usco0103climatedelnorte

Further general information is available from various [websites](#), including [NOAA](#).

You can visualise the site [here](#).

First then, we should look at the streamflow data. These data are in the file delnorte.dat for the years 2000 to 2010 inclusive. You can get further data from <http://waterdata.usgs.gov> if you wish.

```
# load a pre-cooked version of the data for 2005 (NB -- Dont use this year!!!
# except perhaps for testing)

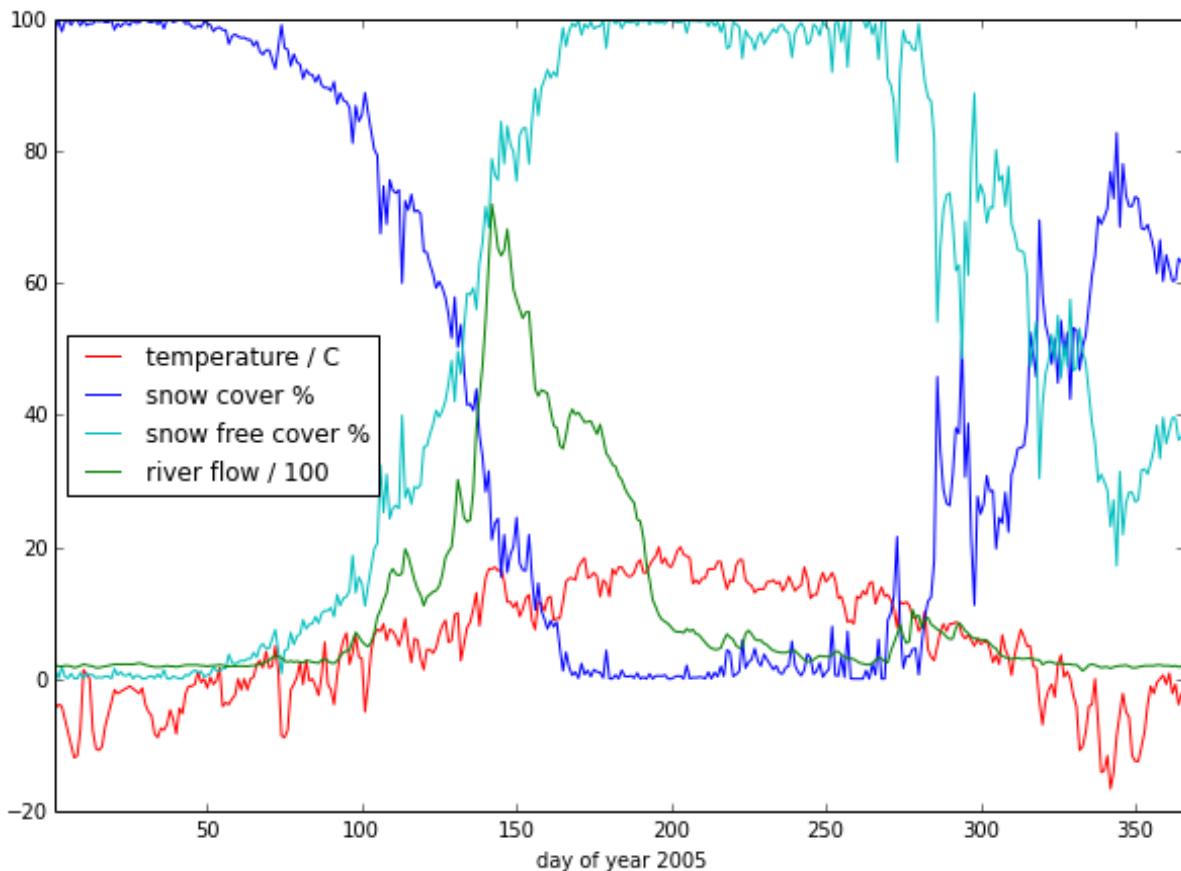
# load the data from a pickle file
import pickle
pkl_file = open('files/data/data.pkl', 'rb')
data = pickle.load(pkl_file)
pkl_file.close()

# set up plot
plt.figure(figsize=(10,7))
plt.xlim(data['doy'][0],data['doy'][-1]+1)
plt.xlabel('day of year 2005')

# plot data
plt.plot(data['doy'],data['temp'],'r',label='temperature / C')
plt.plot(data['doy'],data['snowprop']*100,'b',label='snow cover %')
plt.plot(data['doy'],100-data['snowprop']*100,'c',label='snow free cover %')
plt.plot(data['doy'],data['flow']/100.,'g',label='river flow / 100')

plt.legend(loc='best')

<matplotlib.legend.Legend at 0x1a2cae10>
```



we have plotted the streamflow (scaled) in green, the snow cover in blue, and the non snow cover in cyan and the temperature in red. It should be apparent that the hydrology is snow melt dominated, and to describe this (i.e. to build the simplest possible model) we can probably just apply some time lag function to the snow cover.

44.1.2 6.1.2 The Model

We will build a mass balance model, in terms of ‘snow water equivalent’:

The basis of a model is going to be something of the form:

$$SWE = k * \text{snowProportion}$$

where SWE is the ‘snow water equivalent’, the amount of snow in the entire snow pack in the catchment. snowProportion here then, is the proportion of snow cover in the catchment. We lump together density and volume terms into the coefficient k .

SWE then is the ‘mass’ (of water) that is available for melting on a particular day. We can obtain snowProportion from satellite data, so we only need the area / density term k , which we can suppose to be constant over time.

The simplest model of snowmelt is one where we assume that a proportion of this SWE is released (melted) as a function of temperature. In its simplest form, this is simply a temperature threshold:

```
meltDays = np.where(temperature > tempThresh) [0]
```

On these melt days then, we add $k * \text{snowProportion}$ of water into the system. For the present, we will ignore direct precipitation. So:

```
for d in meltDays: water = K * snowProportion[d]
```

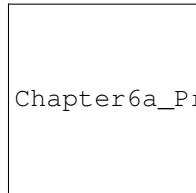
Now we have a mechanism to release snow melt into the catchment, but there will always be some delay in the water reaching the monitoring station from far away regions, compared to nearby areas. The function that describes this delay can be called a network response function. It is often modelled as a Laplace function (an

exponential). The idea is that if we have a ‘flash’ input to the catchment, this network response function will give us what we would measure as a hydrograph at the monitoring station (or elsewhere).

We can parameterise this with a decay factor, p, so that if the amount of water on day d is 1, the amount on day d+1 is p, on d+2, p^2 etc:

```
n = np.arange(len(snowProportion)) - d m = p ** n m[n<0] = 0
```

so here, m is the decay function:



Chapter6a_Practical/files/images/laplace.png

for day 150. This model will transfer a large amount of water of the peak day, then less and less as time goes by. So, a simple model then is of the form:

```
def model_accum(data,tempThresh,k,p):
    meltDays = np.where(data['temp'] > tempThresh) [0]
    accum = data['snowprop']*0.
    for d in meltDays:
        water = k * data['snowprop'][d]
        n = np.arange(len(data['snowprop'])) - d
        m = p ** n
        m[n<0] = 0
        accum += m * water
    return accum

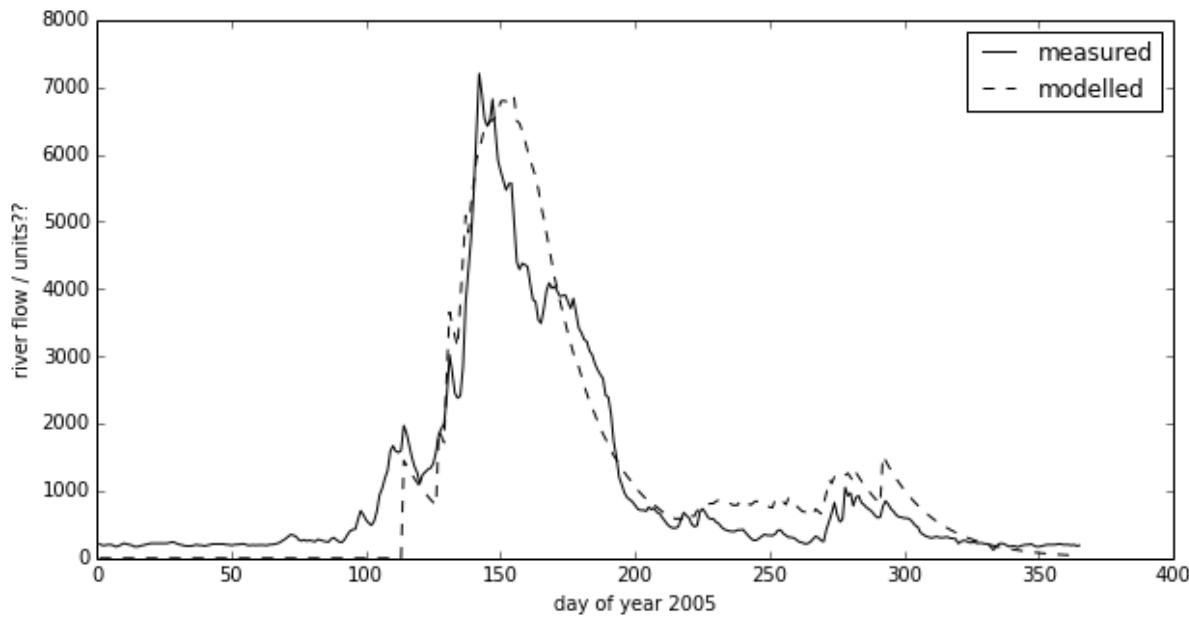
tempThresh = 8.5
k = 2000.0
p = 0.95

# test it
accum = model_accum(data,tempThresh,k,p)
```

This is a very simple model. It has three parameters (tempThresh, k, p) and is driven only by temperature and snow cover data. And yet, we see that even with a rough guess at what the parameters ought to be, we can get a reasonable match with the observed flow data:

```
plt.figure(figsize=(10,5))
plt.plot(data['doy'],data['flow'],'k',label='measured')
plt.plot(data['doy'],accum,'k--',label='modelled')
plt.ylabel('river flow / units??')
plt.xlabel('day of year 2005')
plt.legend(loc='best')

<matplotlib.legend.Legend at 0x1c0b4ed0>
```



44.2 6.2 Data Preparation

44.2.1 6.2.1 Statement of the task

The first task you must complete is to produce a dataset of the proportion of HUC catchment 13010001 (Rio Grande headwaters) that is covered by snow for **two years**, along with associated datasets on temperature and river discharge at the Del Norte monitoring station.

The dataset you produce must have a value for the mean snow cover, temperature and discharge in the catchment for every day over each year.

Your write up **must** include a fully labelled graph of snow cover, temperature and discharge for the catchment for each year, along with some summary statistics (e.g. mean or median, minimum, maximum, and the timing of these).



You should aim to complete this task soon after Reading Week.

44.2.2 6.2.2 Some Advice

You would probably want to use a **daily** snow product for this task, such as that available from MODIS, so make sure you know what that is and explore the characteristics of the dataset.

You will notice from the figure above (the figure should give you some clue as to a suitable data product) that there will be areas of each image for which you have no information (described in the dataset QC). You will need to decide what to do about ‘missing data’. For instance, you might consider interpolating over missing values.

The simplest thing might be to produce a mean snow cover over what samples are available (ignoring the missing values). But whilst that may be sufficient to pass this section, it is far from ideal.

Whilst you only need to produce an average daily value for the catchment, a better approach would be to try to estimate snow cover for each pixel in the catchment (e.g. so you could do spatially explicit modelling with such data). I stress that this is not strictly necessary, but would be an interesting thing to do if you feel able.

However you decide to process the data, you must give a rationale for why you have taken the approach you have done.

You will notice that if you use MODIS data, you have access to both data from Terra (MOD10A) and Aqua (MYD10A), which potentially gives you two samples per day. Think about how to take that into account. Again, the simplest thing to do might be to just use one of these. That is likely to be sufficient, but it would be much better to include both datasets.

You should be able to hunt around to find the temperature and discharge data you want, but we take you through finding them in the advice below.

44.2.3 6.2.3 Data Advice

6.2.3.1 MODIS snow cover data

For MODIS data, you will need to work out which data product you want and how to download it. To help you with this, we have included urls of the MODIS Terra snow data product MOD10A1 and Aqua product MYD10A1 in the files files/data/robot_snow.????.txt:

```
!ls -l files/data/robot_snow.????.txt

-rw-rw-r-- 1 plewis plewis 9034752 Nov 6 09:12 files/data/robot_snow.2000.txt
-rw-rw-r-- 1 plewis plewis 10820568 Nov 6 09:12 files/data/robot_snow.2001.txt
-rw-rw-r-- 1 plewis plewis 16321938 Nov 6 09:12 files/data/robot_snow.2002.txt
-rw-rw-r-- 1 plewis plewis 22423068 Nov 6 09:12 files/data/robot_snow.2003.txt
-rw-rw-r-- 1 plewis plewis 7633374 Nov 6 09:12 files/data/robot_snow.2004.txt
-rw-rw-r-- 1 plewis plewis 18872448 Nov 6 09:12 files/data/robot_snow.2005.txt
-rw-rw-r-- 1 plewis plewis 11433078 Nov 6 09:12 files/data/robot_snow.2006.txt
-rw-rw-r-- 1 plewis plewis 22663686 Nov 6 09:12 files/data/robot_snow.2007.txt
-rw-rw-r-- 1 plewis plewis 22668990 Nov 6 09:12 files/data/robot_snow.2008.txt
-rw-rw-r-- 1 plewis plewis 22705317 Nov 6 09:12 files/data/robot_snow.2009.txt
-rw-rw-r-- 1 plewis plewis 22712370 Nov 6 09:13 files/data/robot_snow.2010.txt
-rw-rw-r-- 1 plewis plewis 17129166 Nov 6 09:13 files/data/robot_snow.2011.txt
-rw-rw-r-- 1 plewis plewis 22756824 Nov 6 09:13 files/data/robot_snow.2012.txt
-rw-rw-r-- 1 plewis plewis 18104898 Nov 6 09:13 files/data/robot_snow.2013.txt

!head -10 < files/data/robot_snow.2007.txt

ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h00v08.005.2008309053908
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h00v09.005.2008309053510
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h00v10.005.2008309053824
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v08.005.2008309053759
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v09.005.2008309053913
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v10.005.2008309053817
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h01v11.005.2008309053918
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h02v06.005.2008309053503
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h02v08.005.2008309053822
ftp://n4ft101u.ecs.nasa.gov/MOSA/MYD10A1.005/2007.01.01/MYD10A1.A2007001.h02v09.005.2008309054221
```

We can use the usual tools to explore the MODIS hdf files:

```
import gdal
target_vector_file = file
modis_file = 'files/data/MYD10A1.A2003026.h09v05.005.2008047035848.hdf'
g = gdal.Open(modis_file)
data_layer = 'MOD_Grid_Snow_500m:Fractional_Snow_Cover'

subdatasets = g.GetSubDatasets()
```

```

for fname, name in subdatasets:
    print name
    print "\t", fname

fname = 'HDF4_EOS:EOS_GRID:"%s":%s'%(modis_file,data_layer)
raster = gdal.Open(fname)

[2400x2400] Snow_Cover_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Spatial_QA MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m
[2400x2400] Snow_Albedo_Daily_Tile MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m
[2400x2400] Fractional_Snow_Cover MOD_Grid_Snow_500m (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"files/data/MYD10A1.A2003026.h09v05.2008047035848.hdf":MOD_Grid_Snow_500m

```

6.2.3.2 Boundary Data

Boundary data, such as catchments, might typically come as ESRI shapefiles or may be in other vector formats. There tends to be variable quality among different databases, but a reliable source for catchment data the USA is the [USGS](#). One set of catchments in the tile we have is the Rio Grande headwaters, which we can [see](#) has a HUC 8-digit code of 13010001. The full dataset is easily found from the [USGS](#) or locally. Literature and associated data concerning this area can be found [here](#). Associated GIS data are [here](#), including the watershed boundary data.

Data more specific to our particular catchment of interest can be found on the [Rio Grande Data Project pages](#).

You should download the file [Hydrologic_Units.zip](#) or get this locally. Obviously, you will need to unzip this file to get at the shapefile '`Hydrologic_Units/HUC_Polygons.shp`' within it.

You can explore the shape file with the following:

```
ogrinfo files/data/Hydrologic_Units/HUC_Polygons.shp HUC_Polygons | head -89 | tail -16

OGRFeature(HUC_Polygons):2
  HUC (Integer) = 13010001
  REG_NAME (String) = Rio Grande Region
  SUB_NAME (String) = Rio Grande Headwaters
  ACC_NAME (String) = Rio Grande Headwaters
  CAT_NAME (String) = Rio Grande Headwaters. Colorado.
  HUC2 (Integer) = 13
  HUC4 (Integer) = 1301
  HUC6 (Integer) = 130100
  REG (Integer) = 13
  SUB (Integer) = 1301
  ACC (Integer) = 130100
  CAT (Integer) = 13010001
  CAT_NUM (String) = 13010001
  Shape_Leng (Real) = 313605.66409400001
  Shape_Area (Real) = 3458016895.23000001907
```

This tells us that we want **HUC feature 2** (catchment 13010001).

We can produce a mask with `raster_mask`, but in this case, we need to use a function `raster_mask2`:

```

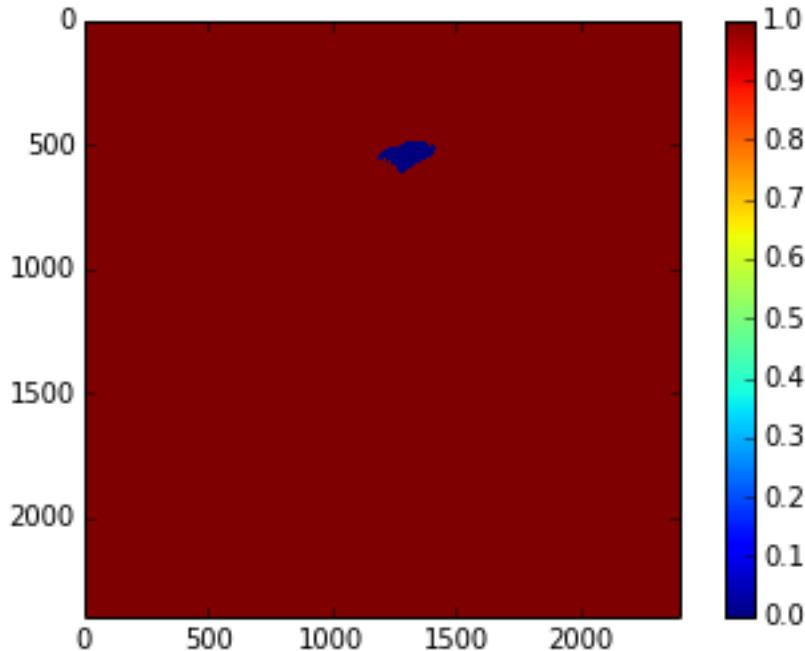
import sys
sys.path.insert(0,'files/python')

from raster_mask import *

m = raster_mask2(fname,\n                 target_vector_file="files/data/Hydrologic_Units/HUC_Polygons.shp",\\
                  attribute_filter=2)
```

```
plt.imshow(m)
plt.colorbar()

<matplotlib.colorbar.Colorbar instance at 0x2b6b7fcd6cb0>
```



The catchment is only a very small portion of the dataset, so you should make sure that you perform masking when you read the dataset in and only extract the area of data that you want.

6.2.3.3 Discharge Data

The river discharge data are in the file `files/data/delnorte.dat <files/data/delnorte.dat>`.

If you examine the file:

```
!head -35 < files/data/delnorte.dat

# ----- WARNING -----
# The data you have obtained from this automated U.S. Geological Survey database
# have not received Director's approval and as such are provisional and subject to
# revision. The data are released on the condition that neither the USGS nor the
# United States Government may be held liable for any damages resulting from its use.
# Additional info: http://waterdata.usgs.gov/nwis/help/?provisional
#
# File-format description: http://waterdata.usgs.gov/?tab_delimited_format_info
# Automated-retrieval info: http://waterdata.usgs.gov/nwis/?automated_retrieval_info
#
# Contact: gs-w_support_nwisweb@usgs.gov
# retrieved: 2011-09-30 09:35:31 EDT (caww02)
#
# Data for the following 1 site(s) are contained in this file
#   USGS 08220000 RIO GRANDE NEAR DEL NORTE, CO
#
#
# Data provided for site 08220000
#   DD parameter statistic Description
#   01    00060      00003    Discharge, cubic feet per second (Mean)
#
# Data-value qualification codes included in this output:
```

```

#      A Approved for publication -- Processing and review completed.
#      e Value has been estimated.
# agency_cd site_no datetime          01_00060_00003  01_00060_00003_cd
# 5s      15s    20d    14n   10s
USGS    08220000    2000-01-01     190      A:e
USGS    08220000    2000-01-02     170      A:e
USGS    08220000    2000-01-03     160      A:e
USGS    08220000    2000-01-04     160      A:e
USGS    08220000    2000-01-05     170      A:e
USGS    08220000    2000-01-06     180      A:e
USGS    08220000    2000-01-07     170      A:e
USGS    08220000    2000-01-08     190      A:e
USGS    08220000    2000-01-09     190      A:e

```

you will see comment lines that start with #, followed by data lines.

The easiest way to read these data would be to use:

```

file = 'files/data/delnorte.dat'
data = np.loadtxt(file, usecols=(2, 3), unpack=True, dtype=str)

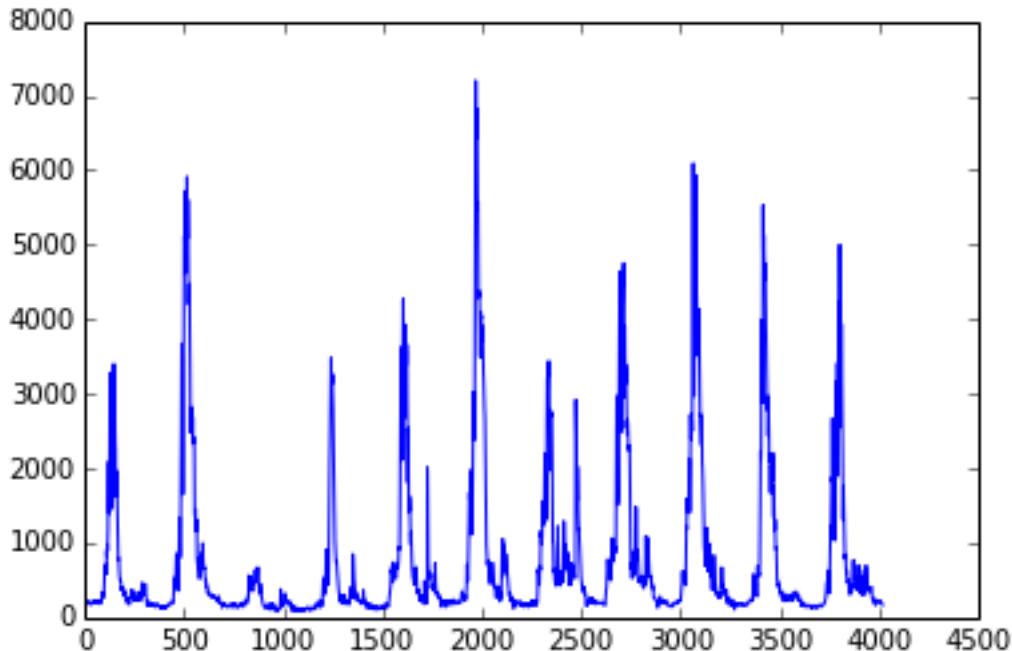
# so you have the dates in
data[0]

array(['2000-01-01', '2000-01-02', '2000-01-03', ..., '2010-12-29',
       '2010-12-30', '2010-12-31'],
      dtype='|S10')

# and the stream flow in data[1]
plt.plot(data[1].astype(float))

[<matplotlib.lines.Line2D at 0x2b6b7c573950>]

```



You will need to convert the date field (i.e. the data in `data[0]`) into the day of year.

This is readily accomplished using `datetime`:

```

import datetime
# transform the first one
ds = np.array(data[0][0].split('-')).astype(int)

```

```
print ds
year,doy = datetime.datetime(ds[0],ds[1],ds[2]).strftime('%Y %j').split()
print year,doy

[2000    1      1]
2000 001
```

6.2.3.4 Temperature data

We can directly access temperature data from [here](#).

The format of 'delNorteT.dat <files/data/delNorteT.dat>' is given here.

The first three fields are date fields (YEAR, MONTH and DAY), followed by TMAX, TMIN, PRCP, SNOW, SNDP.

You should read in the temperature data for the days and years that you want.

For temperature, you might take a **mean of TMAX and TMIN**.

Note that these are in Fahrenheit. You should convert them to Celcius.

Note also that there are missing data (values 9998 and 9999). You will need to filter these and interpolate the data in some way. A median might be a good approach, but any interpolation will suffice.

With that processing then, you should have a dataset, Temperature that will look something like (in cyan, for the year 2005):



44.3 6.3 Coursework

You need to submit your coursework in the usual manner by the usual submission date.

You **must** work individually on this task. If you do not, it will be treated as plagiarism. By reading these instructions for this exercise, we assume that you are aware of the UCL rules on plagiarism. You can find more information on this matter in your student handbook. If in doubt about what might constitute plagiarism, ask one of the course convenors.

44.3.1 6.3.1 Summary of coursework requirements

- **Data Preparation**

The first task you must complete is to produce a dataset of the proportion of HUC catchment 13010001 (Rio Grande headwaters in Colorado, USA) that is covered by snow for **two consecutive years**, along with associated datasets on temperature (in C) and river discharge at the Del Norte monitoring station.

You **may not** use data from the year 2005, as this is given to you in the illustrations above.

The dataset you produce must have a value for the mean snow cover, temperature and discharge in the catchment for every day over each year.

Your write up **must** include fully labelled graph(s) of snow cover, temperature and discharge for the catchment for each year (with units as appropriate), along with some summary statistics (e.g. mean or median, minimum, maximum, and the timing of these).

You **must** provide evidence of how you got these data (i.e. the code and commands you ran to produce the data).

- **Modelling**

You will have prepared two years of data above. Use one of these years to calibrate the (snowmelt) hydrological model (described below) and one year to test it.

The model parameter estimate *must* be objective (i.e. you can't just arbitrarily choose a set) and ideally optimal, and you *must* state the equation of the cost function you will try to minimise and explain the approach used.

You **must** state the values of the model parameters that you have estimated and show evidence for how you went about calculating them. Ideally, you should also state the uncertainty in these parameter estimates (not critical to pass this section though).

You **must** quantify the goodness of fit between your measured flow data and that produced by your model, both for the calibration exercise and the validation.

44.3.2 6.3.2 Summary of Advice

The first task involves pulling datasets from different sources. No individual part of that should be too difficult, but you must put this together from the material we have done so far. It is more a question of organisation then.

Perhaps think first about where you want to end up with on this (the ‘output’). This might for example be a dictionary with keys `temp`, `doy`, `snow` and `flow`, where each of these would be an array with 365 values (or 366 in a leap year).

Then consider the datasets you have: these are: (i) a stack of MODIS data with daily observations; (ii) temperature data in a file; (iii) flow data in a file.

It might be a little fiddly getting the data you want from the flow and temperature data files, but its not very complicated. You will need to consider flagging invalid observations and perhaps interpolating between these.

Processing the MODIS data might take a little more thought, but it is much the same process. Again, we read the datasets in, trying to make this efficient on data size by only using the area of the vector data mask as in a previous exercise. The data reading will be very similar to reading the MODIS LAI product, but you need to work out and implement what changes are necessary. As advised above you should use the `raster_mask2()` function for creating the spatial data masks. Again, you will need to interpolate or perhaps smooth between observations, and then process the snow cover proportions to get an average over the catchment.

The second task revolves around using the model that we have developed above in the function `model_accum()`. You have been through previous examples in Python where you attempt to estimate some model parameters given an initial estimate of the parameters and some cost function to be minimised. Solving the model calibration part of problem should follow those same lines then. Testing (validation) should be easy enough. Don’t forget to include the estimated parameters (and other relevant information, e.g. your initial estimate, uncertainties if available) in your write up.

There is quite a lot of data presentation here, and you need to provide *evidence* that you have done the task. Make sure you use images (e.g. of snow cover varying), graphs (e.g. modelled and predicted flow, etc.), and tables (e.g. model parameter estimates) throughout, as appropriate.

If, for some reason, you are unable to complete the first part of the practical, you should submit what you can for that first part, and continue with calibrating the model using the 2005 dataset that we used above. This would be far from ideal as you would not have completed the required elements for either part in that case, but it would generally be better than not submitting anything.

44.3.3 6.3.3 Further advice

There is plenty of scope here for going beyond the basic requirements, if you get time and are interested (and/or want a higher mark!).

You will be given credit for all additional work included in the write up, **once you have achieved the basic requirements**. So, there is no point (i.e. you will not get credit for) going off on all sorts of interesting lines of exploration here *unless* you have first completed the core task.

44.3.4 6.3.4 Structure of the Report

The required elements of the report are:

Introduction (5%)
Data Preparation (45%)
Modelling (45%)
Discussion/Conclusions (5%)

The figures in brackets indicate the percentage of marks that we will award for each section of the report.

Introduction (5%)

This should be of around 2-3 pages.

It should introduce the purpose of the study, being at a base level, ‘to build and calibrate a snow/hydrological model in Python’.

It should provide some background to building models of this sort (their purpose/role) and include some review of the types of models that might be built, with reference to the literature (journals).

A pass mark for this section will describe and explain the purpose of the study and examine some of the context to such modelling, with appropriate literature being cited. Higher or lower marks will depend on the depth that this goes into and the clarity of expression.

Data Preparation (45%)

This should contain around 3-4 pages of text, other than codes, figures and tables.

For a pass mark in this section, you must :

- introduce the study site, giving general site characteristics, with appropriate figures.
- provide an overview of the data used in the study (snow cover, temperature, flow data) and produce visualisations of the data you are using (images, graphs, tables as appropriate) alongside appropriate summary statistics.
- fully demonstrate how you got these data to this point of processing – i.e. submit appropriate Python codes and/or unix commands that when run in the sequence you describe would produce the data you have described.

The weighting here on the study site description is 5% and on the rest, 40%.

You can obtain higher marks here by going beyond the basics in your approaches to the data or modelling. You still need to demonstrate that you have done the core ‘pass’ material.

Modelling

This should contain around 3-4 pages of text, other than codes, figures and tables.

For a pass mark in this section, you must :

- provide an overview of the model that is constructed here, explaining the role of each parameter.
- explain the way in which model calibration and validation is to be undertaken.
- provide an overview of results of the calibration and validation along with relevant visualisations (images, graphs, tables as appropriate).

- fully demonstrate how you got these results – i.e. submit appropriate Python codes and/or unix commands that when run in the sequence you describe would produce the results you have described.

The weighting here is 5% for the model description and on the rest, 40%.

For a mildly improved mark, you should examine and discuss the model assumptions in the context of any modelling literature you looked at in the Introduction. For a significantly improved mark, you could try improving the model.

Discussion and Conclusions

This section should be around 2 pages. It should provide a discussion and analysis of your results and you should draw appropriate conclusions from these.

You can also use this section to critique the model/data/methods, and suggest ways that you would improve things. If you do this, you must give some indication of how that would be achieved. You will get no credit for simply saying ‘next time I would make the code more efficient’, for example.

Very good/excellent marks would normally require you to cite appropriate literature.

A sufficient effort for a pass would make a reasonable effort at discussing these results in the context of some literature and draw a few (non trivial) conclusions from the study.

44.3.5 6.3.5 Computer Code

General requirements

You will obviously need to submit computer codes as part of this assessment. Some flexibility in the style of these codes is to be expected. For example, some might write a class that encompasses the functionality for all tasks. Some people might have multiple versions of codes with different functionality. All of these, and other reasonable variations are allowed.

All codes needed to demonstrate that you have performed the core tasks are required to be included in the submission. You should include all codes that you make use of in the main body of the text in the main body. Any other codes that you want to refer to (e.g. something you tried out as an enhancement and didn’t quite get there) you can include in appendices.

All codes should be well-commented. Part of the marks you get for code will depend on the adequacy of the commenting.

Degree of original work required and plagiarism

If you use a piece of code verbatim that you have taken from the course pages or any other source, **you must acknowledge this** in comments in your text. **Not to do so is plagiarism.** Where you have taken some part (e.g. a few lines) of someone else’s code, **you should also indicate this.** If some of your code is heavily based on code from elsewhere, **you must also indicate that.**

Some examples. You may recognise this snippet of code from above.

The first example is guilty of strong plagiarism, it does not seek to acknowledge the source of this code, even though it is just a direct copy, pasted into a method called `model()`:

```
def model(tempThresh=9.0, K=2000.0, p=0.96):
    '''need to comment this further ...
    '''

    import numpy as np
    meltDays = np.where(temperature > tempThresh)[0]
    accum = snowProportion*0.
    for d in meltDays:
        water = K * snowProportion[d]
```

```

n = np.arange(len(snowProportion)) - d
m = p ** n
m[np.where(n<0)] = 0
accum += m * water
return accum

```

This is **not** acceptable.

This should probably be something along the lines of:

```

def model(tempThresh=9.0,K=2000.0,p=0.96):
    '''need to comment this further ...

    This code is taken directly from
    "Modelling delay in a hydrological network"
    by P. Lewis http://www2.geog.ucl.ac.uk/~plewis/geogg122/DelNorte.html
    and wrapped into a method.
    '''

    # my code: make sure numpy is imported
    import numpy as np

    # code below verbatim from Lewis
    meltDays = np.where(temperature > tempThresh)[0]
    accum = snowProportion*0.
    for d in meltDays:
        water = K * snowProportion[d]
        n = np.arange(len(snowProportion)) - d
        m = p ** n
        m[np.where(n<0)] = 0
        accum += m * water
    # my code: return accumulator
    return accum

```

Now, we acknowledge that this is in essence a direct copy of someone else's code, and clearly state this. We do also show that we have added some new lines to the code, and that we have wrapped this into a method.

In the next example, we have seen that the way m is generated is in fact rather inefficient, and have re-structured the code. It is partially developed from the original code, and acknowledges this:

```

def model(tempThresh=9.0,K=2000.0,p=0.96):
    '''need to comment this further ...

    This code after the model developed in
    "Modelling delay in a hydrological network"
    by P. Lewis
    http://www2.geog.ucl.ac.uk/~plewis/geogg122/DelNorte.html

    My modifications have been to make the filtering more efficient.
    '''

    # my code: make sure numpy is imported
    import numpy as np

    # code below verbatim from Lewis unless otherwise indicated
    meltDays = np.where(temperature > tempThresh)[0]
    accum = snowProportion*0.

    # my code: pull the filter block out of the loop
    n = np.arange(len(snowProportion))
    m = p ** n

    for d in meltDays:
        water = K * snowProportion[d]

    # my code: shift the filter on by one day

```

```
# ...do something clever to shift it on by one day

    accum += m * water
# my code: return accumulator
return accum
```

This example makes it clear that significant modifications have been made to the code structure (and probably to its efficiency) although the basic model and looping comes from an existing piece of code. It clearly highlights what the actual modifications have been. Note that this is not a working example!!

Although you are supposed to do this piece of work on your own, there might be some circumstances under which someone has significantly helped you to develop the code (e.g. written the main part of it for you & you've just copied that with some minor modifications). You **must** acknowledge in your code comments if this has happened. On the whole though, this should not occur, as you **must** complete this work on your own.

If you take a piece of code from somewhere else and all you do is change the variable names and/or other cosmetic changes, you **must** acknowledge the source of the original code (with a URL if available).

Plagiarism in coding is a tricky issue. One reason for that is that often the best way to learn something like this is to find an example that someone else has written and adapt that to your purposes. Equally, if someone has written some tool/library to do what you want to do, it would generally not be worthwhile for you to write your own but to concentrate on using that to achieve something new. Even in general code writing (i.e. when not submitting it as part of your assessment) you and anyone else who ever has to read your code would find it of value to make reference to where you found the material to base what you did on. The key issue to bear in mind in this work, as it is submitted ‘as your own work’ is that, to avoid being accused of plagiarism and to allow a fair assessment of what you have done, you must clearly acknowledge which parts of it are your own, and the degree to which you could claim them to be your own.

For example, based on ... is absolutely fine, and you would certainly be given credit for what you have done. In many circumstances ‘taken verbatim from ...’ would also be fine (provided it is acknowledged) but then you would be given credit for what you had done with the code that you had taken from elsewhere (e.g. you find some elegant way of doing the graphs that someone has written and you make use of it for presenting your results).

The difference between what you submit here and the code you might write if this were not a piece submitted for assessment is that you the vast majority of the credit you will gain for the code will be based on the degree to which you demonstrate that you can write code to achieve the required tasks. There would obviously be some credit for taking codes from the coursenotes and bolting them together into something that achieves the overall aim: provided that worked, and you had commented it adequately and acknowledge what the extent of your efforts had been, you should be able to achieve a pass in that component of the work. If there was no original input other than bolting pieces of existing code together though, you be unlikely to achieve more than a pass. If you get less than a pass in another component of the coursework, that then puts you in danger of an overall fail.

Provided you achieve the core tasks, the more original work that you do/show (that is of good quality), the higher the mark you will get. Once you have achieved the core tasks, even if you try something and don't quite achieve it, it is probably worth including, as you may get marks for what you have done (or that fact that it was a good or interesting thing to try to do).

Documentation

Note: All methods/functions and classes must be documented for the code to be adequate. Generally, this will contain:

- some text on the purpose of the method (/function/class)
- some text describing the inputs and outputs, including reference to any relevant details such as datatype, shape etc where such things are of relevance to understanding the code.
- some text on keywords, e.g.:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.
```

Keyword arguments:

```
real -- the real part (default 0.0)
imag -- the imaginary part (default 0.0)
```

```
Example taken verbatim from:
http://www.python.org/dev/peps/pep-0257/
"""
if imag == 0.0 and real == 0.0: return complex_zero
```

You should look at the document on good docstring conventions when considering how to document methods, classes etc.

To demonstrate your documentation, you **must** include the help text generated by your code after you include the code. e.g.:

```
def print_something(this, stderr=False):
    '''This does something.

    Keyword arguments:
    stderr -- set to True to print to stderr (default False)
    '''

    if stderr:
        # import sys.stderr
        from sys import stderr

        # print to stderr channel, converting this to str
        print >> stderr, str(this)

        # job done, return
        return

    # print to stdout, converting this to str
    print str(this)

    return
```

Then the help text would be:

```
help(print_something)

Help on function print_something in module __main__:

print_something(this, stderr=False)
    This does something.

    Keyword arguments:
    stderr -- set to True to print to stderr (default False)
```

The above example represents a ‘good’ level of commenting as the code broadly adheres to the style suggestions and most of the major features are covered. It is not quite ‘very good/excellent’ as the description of the purpose of the method (rather important) is trivial and it fails to describe the input this in any way. An excellent piece would do all of these things, and might well tell us about any dependencies (e.g. requires sys if stderr set to True).

An inadequate example would be:

```
def print_something(this, stderr=False):
    '''This prints something'''
    if stderr:
        from sys import stderr
        print >> stderr, str(this)
        return
    print str(this)
```

It is inadequate because it still only has a trivial description of the purpose of the method, it tells us nothing about inputs/outputs and there is no commenting inside the method.

Word limit

There is no word limit per se on the computer codes, though as with all writing, you should try to be succinct rather than overly verbose.

Code style

A good to excellent piece of code would take into account issues raised in the [style guide](#). The ‘degree of excellence’ would depend on how well you take those points on board.

7.0 FIRE/ENSO TELECONNECTIONS

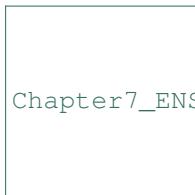
45.1 7.1 Introduction

There is much public and scientific interest in monitoring and predicting the activity of wildfires and such topics are often in the media.

Part of this interest stems from the role fire plays in issues such as land cover change, deforestation and forest degradation and Carbon emissions from the land surface to the atmosphere, but also of concern are human health impacts. The impacts of fire should not however be considered as wholly negative, as it plays a significant role in natural ecosystem processes.

For many regions of the Earth, there are large inter-annual variations in the timing, frequency and severity of wildfires. Whilst anthropogenic activity accounts for a large and probably increasing proportion of fires started, this is not in itself a new phenomenon.

Fires spread where: (i) there is an ignition source (lightning or man, mostly); (ii) sufficient combustible fuel to maintain the fire. The latter is strongly dependent on fuel loads and moisture content, as well as meteorological conditions. Generally then, when conditions are drier (and there is sufficient fuel and appropriate weather conditions), we would expect fire spread to increase. If the number of ignitions remained approximately constant, this would mean more fires. Many models of fire activity predict increases in fire frequency in the coming decades, although there may well be different behaviours in different parts of the world.



Chapter7_ENSO/files/images/492949main_Figure-2-Wildfires_s3.jpg

Satellite data has been able to provide us with increasingly useful tools for monitoring wildfire activity, particularly since 2000 with the MODIS instruments on the NASA Terra and Aqua (2002) satellites. A suite of ‘fire’ products have been generated from these data that have been used in a large number of publications and practical/management projects.

There is growing evidence of ‘teleconnection’ links between fire occurrence and large scale climate patterns, such as ENSO.

The proposed mechanisms are essentially that such climatic patterns are linked to local water status and temperature and thus affect the ability of fires to spread. For some regions of the Earth, empirical models built from such considerations have quite reasonable predictive skill, meaning that fire season severity might be predicted some months ahead of time.

45.2 7.2 A Practical Exercise

45.2.1 7.2.1 In This Session

In this session, you will be working in groups (of 3 or 4) to build a computer code in python to explore links between fire activity and Sea Surface Temperature anomalies.

This is a team exercise, but does not form part of your formal assessment for this course. You should be able to complete the exercise in the 3 hour session, if you work effectively as a team. Staff will be on hand to provide pointers.

You should be able to complete the exercise using coding skills and python modules that you have previously experience of, though we will also provide some pointers to get you started.

45.2.2 7.2.2 Statement of the problem

Using monthly fire count data from MODIS Terra, develop and test a predictive model for the number of fires per unit area per year driven by Sea Surface Temperature anomaly data.

45.2.3 7.2.3 Datasets

We suggest that the datasets you use of this analysis, following Chen et al. (2011), are:

- MODIS Terra fire counts (2001-2011) (MOD14CMH). The particular dataset you will want from the file is ‘SUBDATASET_2 [360x720] CloudCorrFirePix (16-bit integer)’.
- Climate index data from NOAA

If you ever wish to take this study further, you can find various other useful datasets such as these.

Fire Data

The MOD14CMH CMG data are available from the [UMD ftp server](#) but the data you will need are also directly available from /data/geospatial_10/ucfajlg/MOD14CMH/. Note that, if you are on the UCL system, you do not need to copy the data, just use them from where they are.

If for any reason, you *did* want to copy or update them, use the following unix command:

```
wget 'ftp://fire:burnt@fuoco.geog.umd.edu/modis/C5/cmg/monthly/hdf/*'
```

The data are in files/data and are in HDF format, so you should know how to read them into a numpy array in python.

```
!ls -l files/data/*hdf | head -10
```

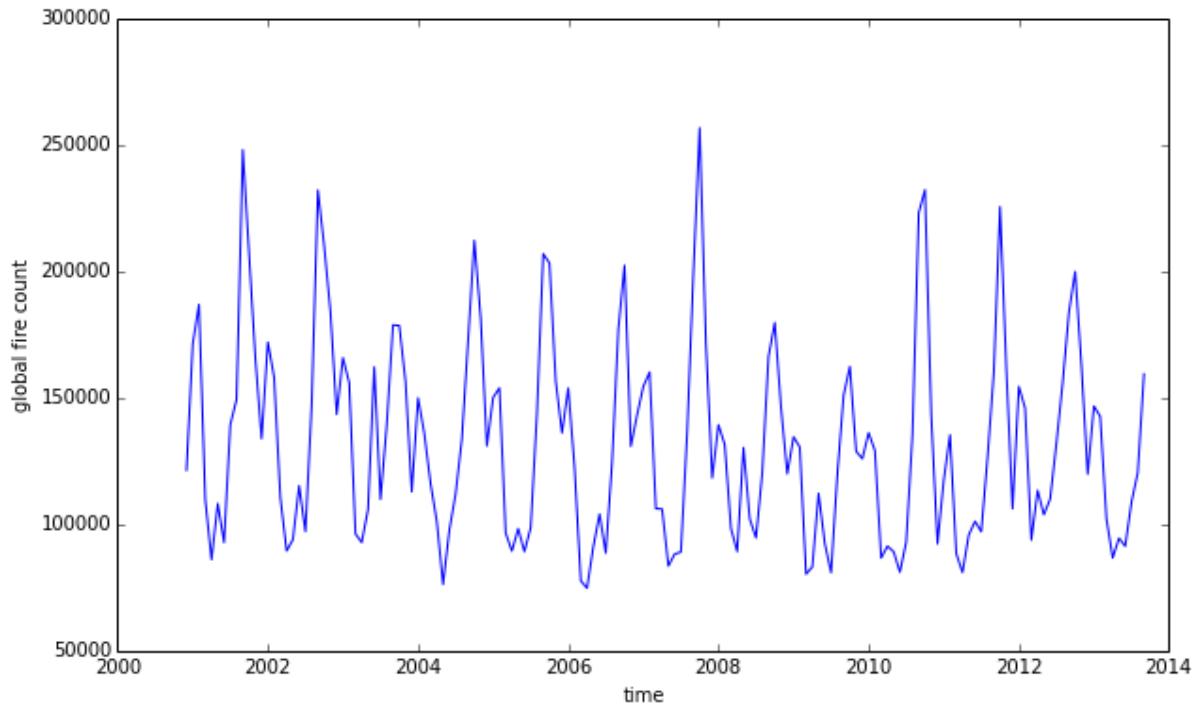
```
-rw-rw-r-- 1 plewis plewis 961993 Nov  1 2007 files/data/MOD14CMH.200011.005.01.hdf
-rw-rw-r-- 1 plewis plewis 923238 Nov  1 2007 files/data/MOD14CMH.200012.005.01.hdf
-rw-rw-r-- 1 plewis plewis 953227 Mar 22 2008 files/data/MOD14CMH.200101.005.01.hdf
-rw-rw-r-- 1 plewis plewis 951917 Apr 23 2008 files/data/MOD14CMH.200102.005.01.hdf
-rw-rw-r-- 1 plewis plewis 959282 Apr 23 2008 files/data/MOD14CMH.200103.005.01.hdf
-rw-rw-r-- 1 plewis plewis 943794 Apr 23 2008 files/data/MOD14CMH.200104.005.01.hdf
-rw-rw-r-- 1 plewis plewis 988588 Apr 23 2008 files/data/MOD14CMH.200105.005.01.hdf
-rw-rw-r-- 1 plewis plewis 898137 Mar 22 2008 files/data/MOD14CMH.200106.005.01.hdf
-rw-rw-r-- 1 plewis plewis 962767 Mar 22 2008 files/data/MOD14CMH.200107.005.01.hdf
-rw-rw-r-- 1 plewis plewis 982616 Nov  5 2007 files/data/MOD14CMH.200108.005.01.hdf
ls: write error: Broken pipe
```

If you are **really** stuck on reading the data, or just want to move on to the next parts, you can use ‘files/python/reader.py <files/python/reader.py>’ which will create a masked array in data, and an array of years (year) and months (month):

```
run files/python/reader

plt.figure(figsize=(10, 6))
x = year + month/12.
y = np.sum(data, axis=(1, 2))
plt.plot(x, y)
plt.ylabel('global fire count')
plt.xlabel('time')

<matplotlib.text.Text at 0x2b4488089490>
```



This dataset is at 0.5 degree resolution and we want to perform tha analysis as 5 degrees.

We need to shrink the dataset by a factor of 10 then.

There are different ways to achive this, but one way would be to reorganise the data:

```
rdata = [data[:, i::10, j::10] for i in xrange(10) for j in xrange(10)]
rdata = ma.array(rdata)

print rdata.shape
```

(100, 154, 36, 72)

So, we have made the dataset which as (154, 360, 720) into a shape (100, 154, 36, 72).

We can now get the total fire counts easily at 5 degrees by summing over those 100 cells (axis=0):

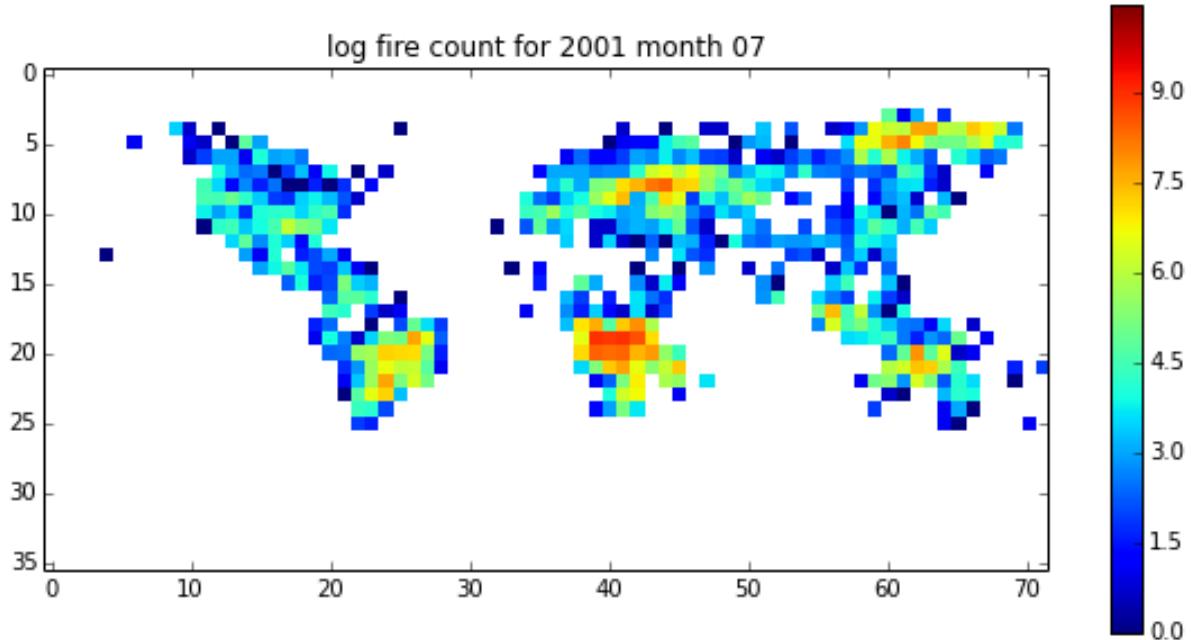
```
fdata = rdata.sum(axis=0)
print fdata.shape

lf = np.log(fdata)
vmax = np.max(lf[lf>0])

plt.figure(figsize=(10, 5))
plt.imshow(lf[8], interpolation='nearest', vmax=vmax)
plt.colorbar()
plt.title('log fire count for %d month %02d' % (year[8], month[8]))
```

(154, 36, 72)

<matplotlib.text.Text at 0x1e19add0>



```
# or even make a movie
lf = np.log(fdata)
vmax = np.max(lf[lf>0])

root = 'files/images/'
for i in xrange(lf.shape[0]):
    fig = plt.figure(figsize=(10,5))
    plt.imshow(np.log(fdata[i]), interpolation='nearest', vmax=vmax)
    plt.colorbar()
    file_id = '%d month %02d'%(year[i],month[i])
    plt.title('log fire count for %s'%file_id)
    plt.savefig('%s_%s.jpg'%(root,file_id.replace(' ', '_')))
    plt.close(fig)

-c:2: RuntimeWarning: divide by zero encountered in log
-c:2: RuntimeWarning: invalid value encountered in log
-c:8: RuntimeWarning: divide by zero encountered in log
-c:8: RuntimeWarning: invalid value encountered in log

cmd = 'convert -delay 100 -loop 0 {0}_*month*.jpg {0}fire_movie3.gif'.format(root)
os.system(cmd)

0
```

The information we want is the peak fire count and to know which month this occurred in.

To do this, we might reorder the data first:

```
nlatlon = fdata.shape[1:]
min_year = year[0]
max_year = year[-1]
# number of years
nyears = max_year - min_year + 1

# set up a big array
```

```

f2data = np.zeros((12,nyears)+nlatlon)
f2datam = np.ones((12,nyears)+nlatlon).astype(bool)

for i,(y,m) in enumerate(zip(year-year[0],month-1)):
    f2data[m,y] = fdata[i]
    f2datam[m,y] = (fdata[i] <= 0)
# mask it
f2data = ma.array(f2data,mask=f2datam)
print f2data.shape

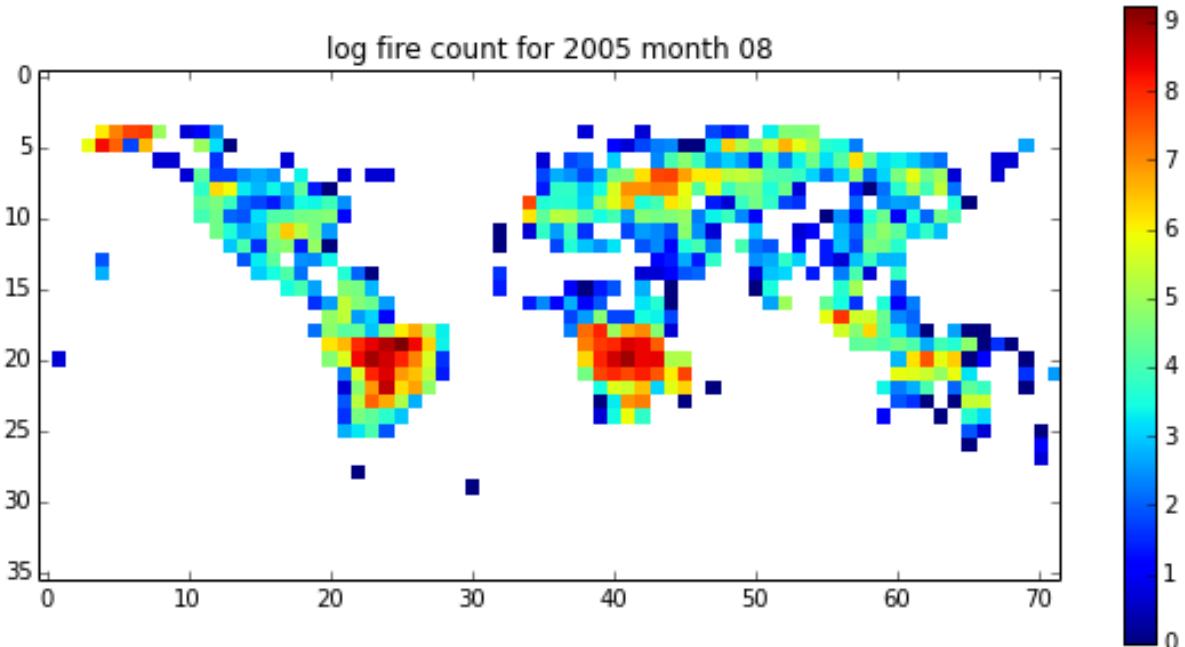
# test it
m = 8
y = 2005
plt.figure(figsize=(10,5))
plt.imshow(np.log(f2data[m-1,y-year[0]]),interpolation='nearest')
plt.colorbar()
plt.title('log fire count for %d month %02d'%(y,m))

(12, 14, 36, 72)

-c:22: RuntimeWarning: divide by zero encountered in log

<matplotlib.text.Text at 0x2b4489c91ad0>

```



```

# which month has the highest fire count
# NB 0-based here but we use a masked array

# total fire count summed over month (axis 0)
fmask = f2data.sum(axis=0) == 0

# which month (axis 0) has the max value?
fire_month = np.argmax(f2data,axis=0)

# masked array of this
fire_month = ma.array(fire_month,mask=fmask)

y = 2005
plt.figure(figsize=(10,5))
plt.imshow(fire_month[y-year[0]],interpolation='nearest')

```

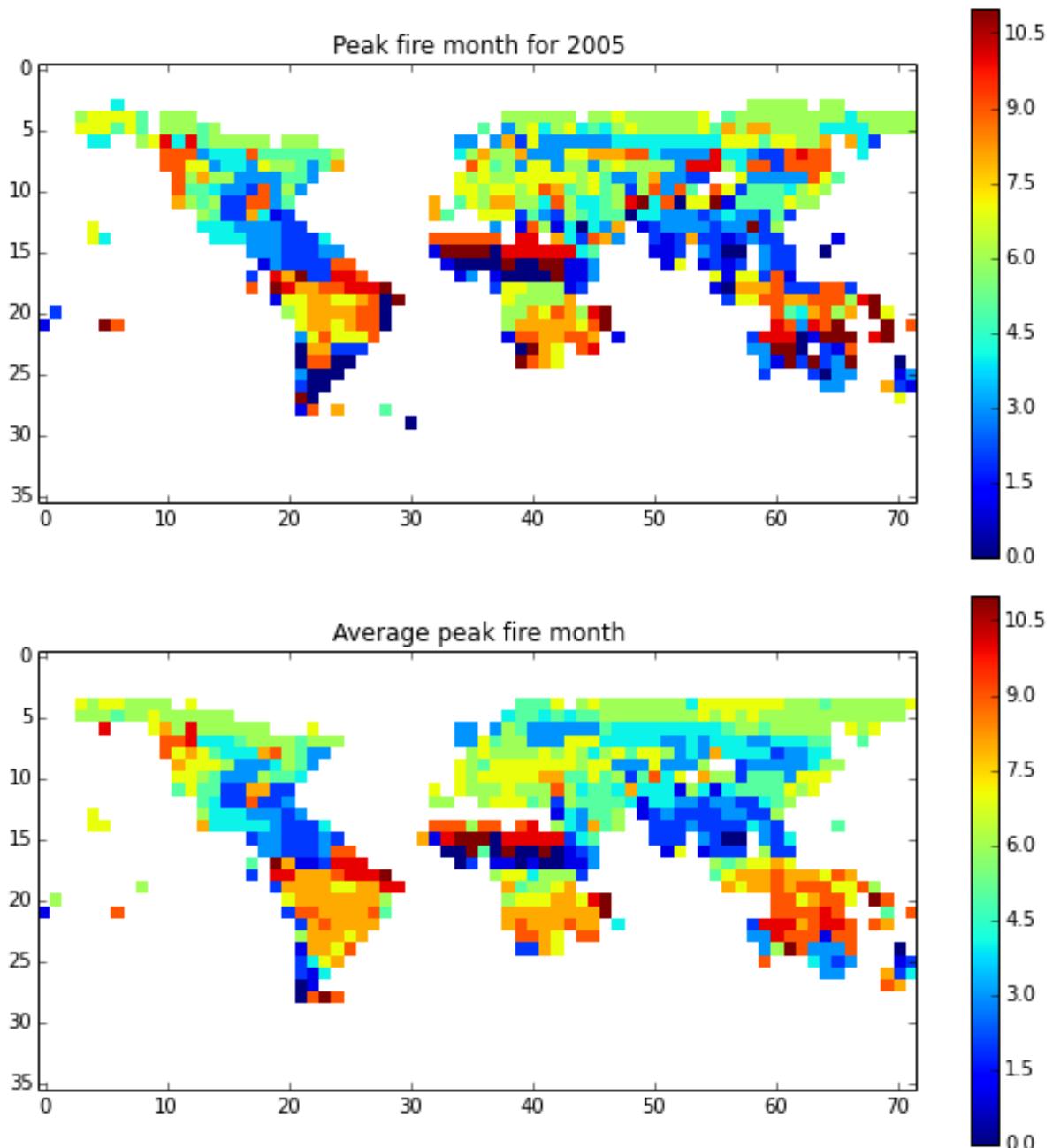
```

plt.colorbar()
plt.title('Peak fire month for %d'%(y))

# suppose this is the same for all years:
av_fire_month = np.median(fire_month, axis=0).astype(int)
plt.figure(figsize=(10,5))
plt.imshow(av_fire_month, interpolation='nearest')
plt.colorbar()
plt.title('Average peak fire month')

<matplotlib.text.Text at 0x134d27d0>

```



```

# and now get the fire count for that month
# lets try this by hand first

```

```
peak_count = np.zeros_like(f2data[0])
```

```

y = 2001
m = 0

fmask = (av_fire_month == m)
peak_count[y-year[0]][fmask] = f2data[m,y-year[0]][fmask]

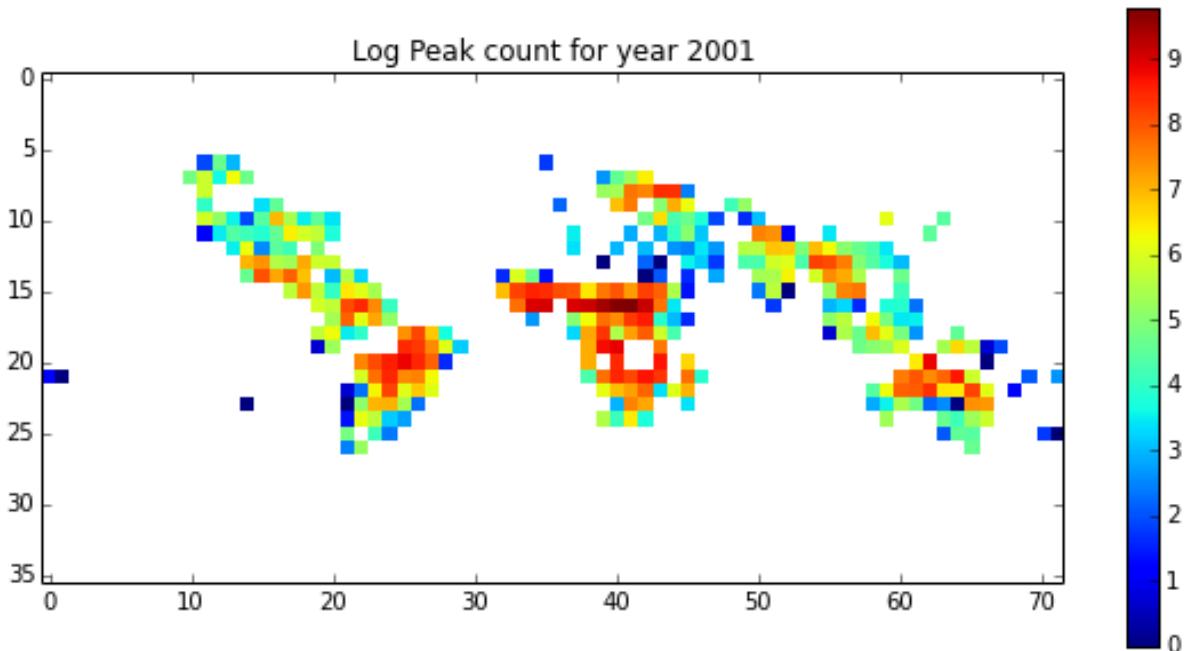
# and now extend it
peak_count = np.zeros_like(f2data[0])

for m in xrange(f2data.shape[0]):
    fmask = (av_fire_month == m)
    for y in xrange(f2data.shape[1]):
        peak_count[y][fmask] = f2data[m,y][fmask]

# test it
y = 1
plt.figure(figsize=(10,5))
plt.imshow(np.log(peak_count[y]),interpolation='nearest')
plt.colorbar()
plt.title('Log Peak count for year %d'%(min_year+y))

<matplotlib.text.Text at 0x2b44901b2090>

```



In summary, we have developed the following datasets:

```

print 'peak_count',peak_count.shape
print 'av_fire_month',av_fire_month.shape
print 'min_year',min_year

peak_count (14, 36, 72)
av_fire_month (36, 72)
min_year 2000

```

Climate Data

The climate data you will want will be some form of Sea Surface Temperature (SST) anomaly measure. There is a long list of such measures on <http://www.esrl.noaa.gov/psd/data/climateindices/list>.

Examples would be [AMO](#) or [ONI](#). Note that some of these measures are smoothed and others not.

Suppose we had selected AMO and we want to read directly from the url:

```
import urllib2

url = 'http://www.esrl.noaa.gov/psd/data/correlation/amon.us.data'

req = urllib2.Request ( url )
raw_data = urllib2.urlopen(req).readlines()

# we notice from inspection that
# we want data from rows 1 to -4
raw_data[:2]

[' 1948      2013n',
 ` 1948   -0.006   -0.018    0.037   -0.061    0.005    0.064   -0.030   -0.013   -0.043
raw_data[-10:-4]

[' 2008    0.051    0.150    0.185    0.071    0.193    0.287    0.237    0.201    0.228
` 2009   -0.032   -0.137   -0.139   -0.103   -0.039    0.152    0.259    0.182    0.086
` 2010    0.068    0.201    0.313    0.457    0.486    0.476    0.482    0.559    0.481
` 2011    0.173    0.134    0.082    0.119    0.172    0.206    0.126    0.180    0.183
` 2012   -0.041    0.028    0.048    0.109    0.191    0.332    0.412    0.468    0.482
` 2013    0.155    0.144    0.186    0.168    0.132    0.078    0.218    0.226    0.290

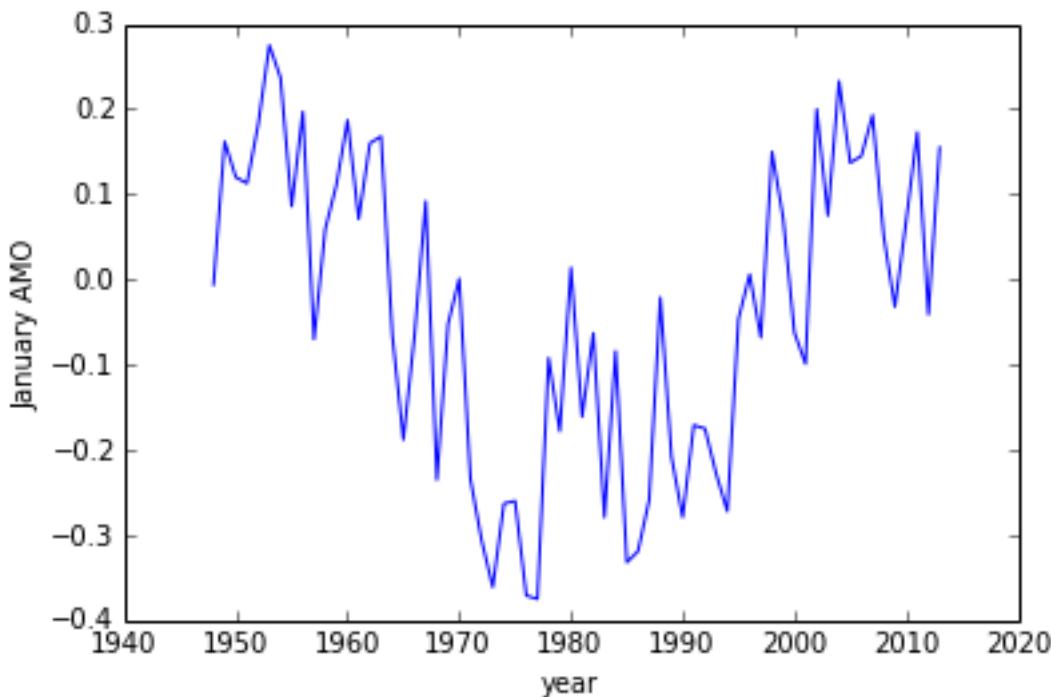
cdata = np.array([r.split() for r in raw_data[1:-4]]).astype(float)

cmask = (cdata < -50 )
cdata = ma.array(cdata,mask=cmask).T
cyears = cdata[0]
cdata = cdata[1:]

# now we have the climate data as a masked array
# column 0 is years, column 1 is Jan etc.

plt.plot(cyears,cdata[0])
plt.xlabel('year')
plt.ylabel('January AMO')
print cdata.shape

(12, 66)
```



45.2.4 7.2.4 Code to perform correlation analysis

The idea here is, for a particular (or set of) SST anomaly measures, work out which ‘lag’ month gives the highest correlation coefficient with fire count.

By ‘lag’ month, we mean that e.g. if the peak fire month for a particular pixel was September, which month prior to that has a set of SST anomalies over the sample years that is most strongly correlated with fire count.

So, if we were using a single SST anomaly measure (e.g. AMO or ONI) and sample years 2001 to 2009 to build our model, then we would do a linear regression of fire count for a particular pixel over these years against e.g. AMO data for September (lag 0) then August (lag 1) then July (lag 2) etc. and see which produced the highest R^2 .

Before we get into that, let’s look again at the data structure we have:

```
# climate data
print 'cdata', cdata.shape
print 'cyears', cyears.shape

# From the fire data

print 'peak_count', peak_count.shape
print 'av_fire_month', av_fire_month.shape
print 'min_year', min_year

cdata (12, 66)
cyears (66,)
peak_count (14, 36, 72)
av_fire_month (36, 72)
min_year 2000
```

So, if we want to select data for particular years:

```
# which years (inclusive)
years = [2001, 2010]

ypeak_count = peak_count[years[0]-min_year:years[1] - min_year + 1]
ycdata = cdata[:, years[0] - cyears[0]:years[1] - cyears[0] + 1]
```

```
# check the shape
print ycdata.shape,ypeak_count.shape,av_fire_month.shape

(12, 10) (10, 36, 72) (36, 72)

We need to consider a little carefully the implementation of lag ...

# we will need to access ycdata[month - n][year]
# which is a bit fiddly as e.g. -3 will be interpreted as
# October for that same year, rather than the previous year
y = 2001 - min_year
m = 2
lag = 5
print m - lag,y

-3 1

# so one way to fix this is to decrease y by one
# if m - lag is -ve
Y = y - (m - lag < 0)
print m-lag,Y

-3 0

from scipy.stats import linregress

# examine an example row col
# for a given month over all years

c = 24
r = 19
m = av_fire_month[r,c]
# pull the data
yyears = np.arange(years[1]-years[0]+1)

R2 = np.array([linregress(\ 
    ycdata[m-n,yyears - (m - n < 0)],\
    ypeak_count[yyears - (m - n < 0),r,c]\ 
) [2] for n in xrange(12)]) 

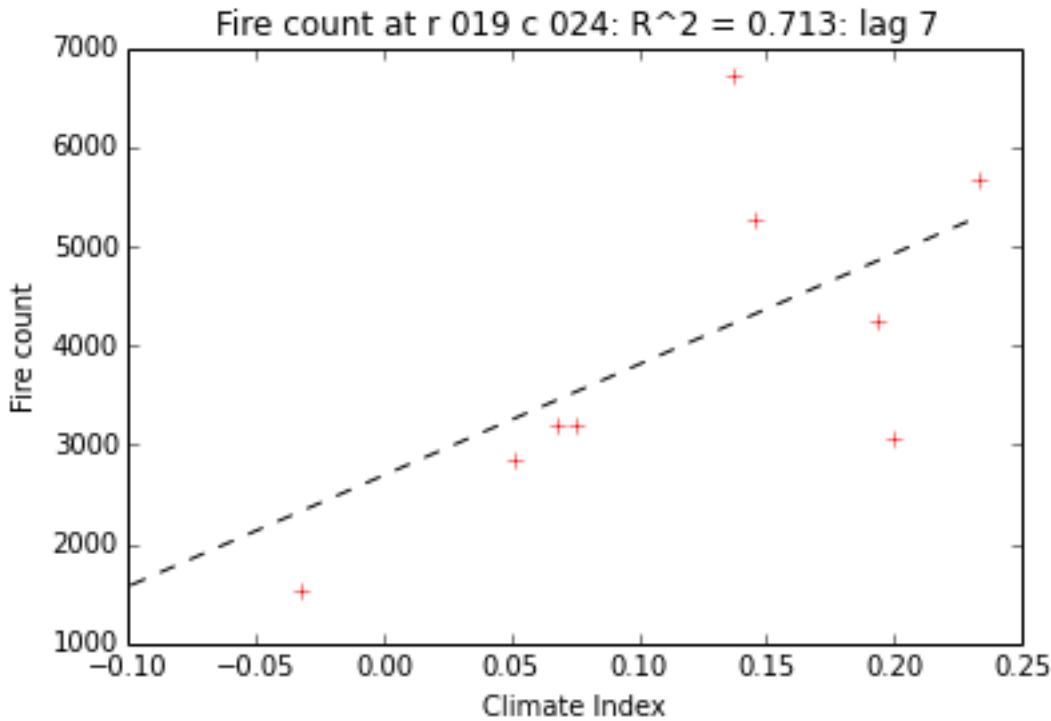
n = np.argmax(R2)

x = ycdata[m-n,yyears - (m - n < 0)]
y = ypeak_count[yyears - (m - n < 0),r,c]
slope,intercept,R,p,err = linregress(x,y)

print slope,intercept,p,err
plt.plot(ycdata[m-n],y,'r+')
plt.xlabel('Climate Index')
plt.ylabel('Fire count')
plt.plot([x.min(),x.max()],\
         [intercept+slope*x.min(),intercept+slope*x.max()],'k--')
plt.title('Fire count at r %03d c %03d: R^2 = %.3f: lag %d'%(r,c,R2[n],n))

11175.1548059 2691.89246835 0.0207363379213 3889.85050593

<matplotlib.text.Text at 0x166679d0>
```



```
# looper

data_mask = ypeak_count.sum(axis=0)>100

rs,cs = np.where(data_mask)

results = {'intercept':0,'slope':0,'p':0,'R':0,'stderr':0,'lag':0}
for k in results.keys():
    results[k] = np.zeros_like(av_fire_month).astype(float)
    results[k] = ma.array(results[k],mask=~data_mask)

for r,c in zip(rs,cs):
    m = av_fire_month[r,c]
    # pull the data
    yyears = np.arange(years[1]-years[0]+1)
    R2 = np.array([
        linregress(
            ycdata[m-n,yyears - (m - n < 0)], \
            ypeak_count[yyears - (m - n < 0),r,c]\n        )[2] for n in xrange(12)])
    n = np.argmax(R2)
    results['lag'][r,c] = n
    x = ycdata[m-n,yyears - (m - n < 0)]
    y = ypeak_count[yyears - (m - n < 0),r,c]
    results['slope'][r,c],results['intercept'][r,c],\
    results['R'][r,c],results['p'][r,c],\
    results['stderr'][r,c] = linregress(x,y)

plt.figure(figsize=(10,4))
plt.imshow(results['R'],interpolation='nearest')
plt.colorbar()
plt.title('R')

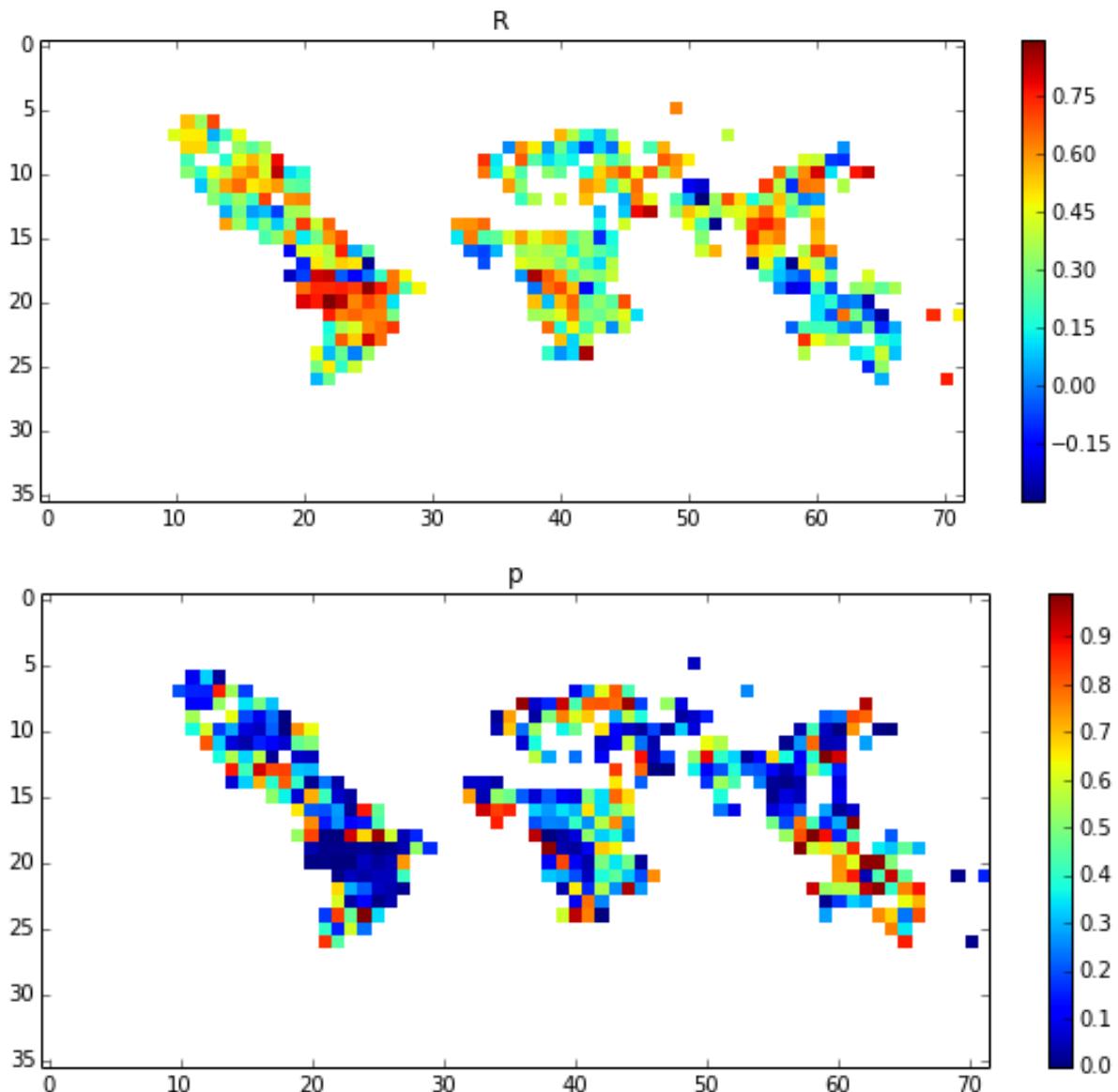
plt.figure(figsize=(10,4))
plt.imshow(results['p'],interpolation='nearest')
plt.colorbar()
```

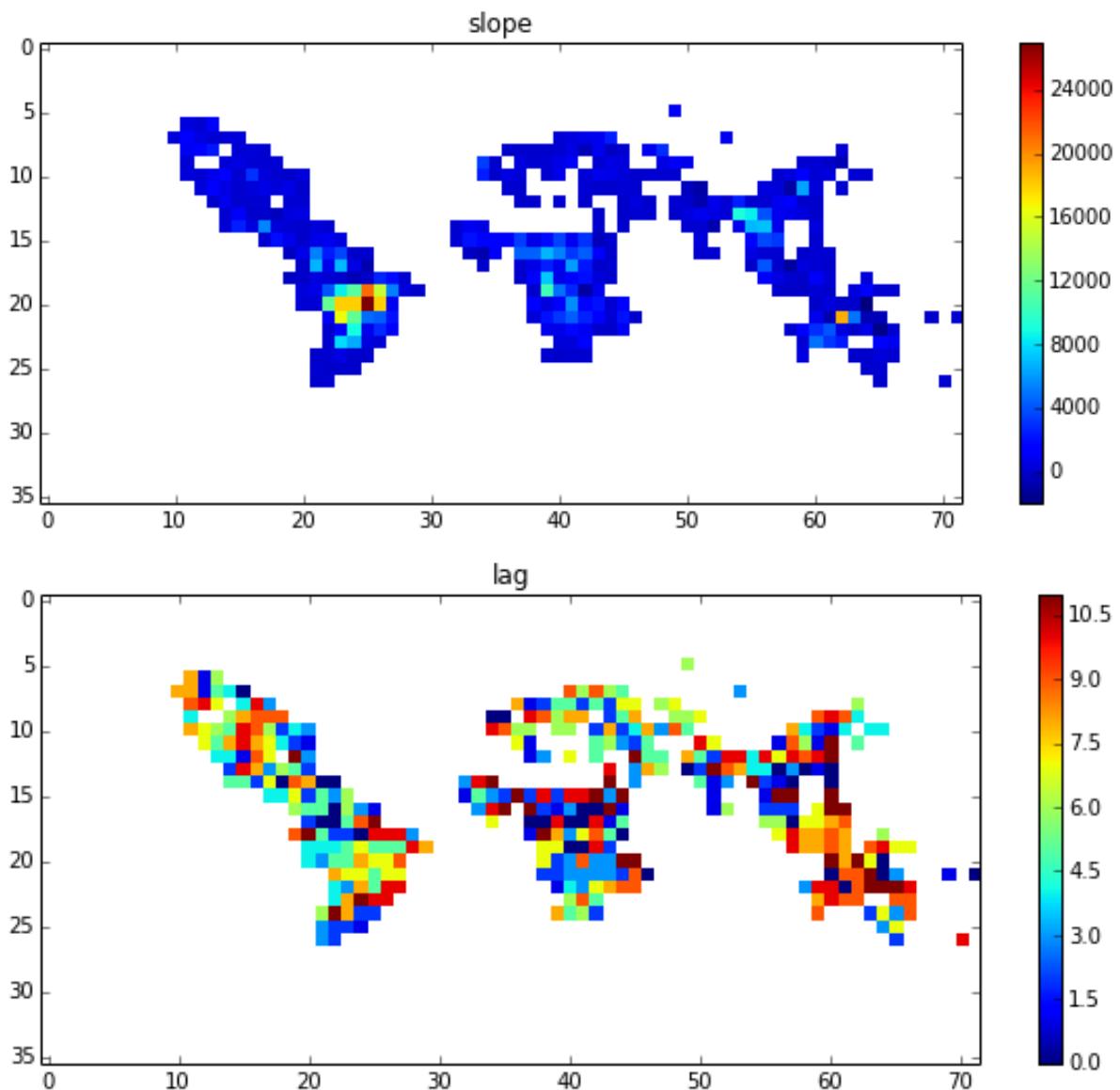
```
plt.title('p')

plt.figure(figsize=(10, 4))
plt.imshow(results['slope'], interpolation='nearest')
plt.colorbar()
plt.title('slope')

plt.figure(figsize=(10, 4))
plt.imshow(results['lag'], interpolation='nearest')
plt.colorbar()
plt.title('lag')

<matplotlib.text.Text at 0x6d69890>
```





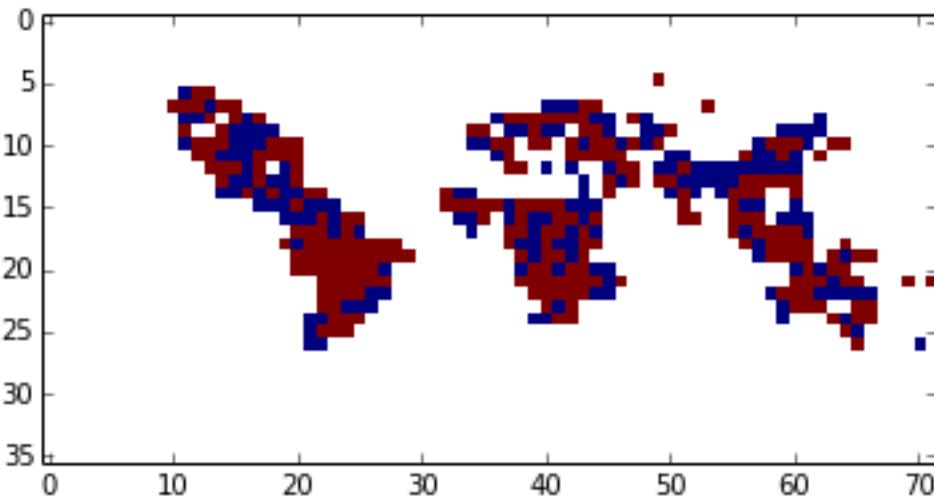
which we can now predict:

```
# prediction year
pyear = 2012

# which month?
M = av_fire_month - results['lag']
Y = np.zeros_like(M) + pyear
Y[M<0] -= 1

# lets look at that ...
plt.imshow(Y, interpolation='nearest')

<matplotlib.image.AxesImage at 0x7d6ab90>
```

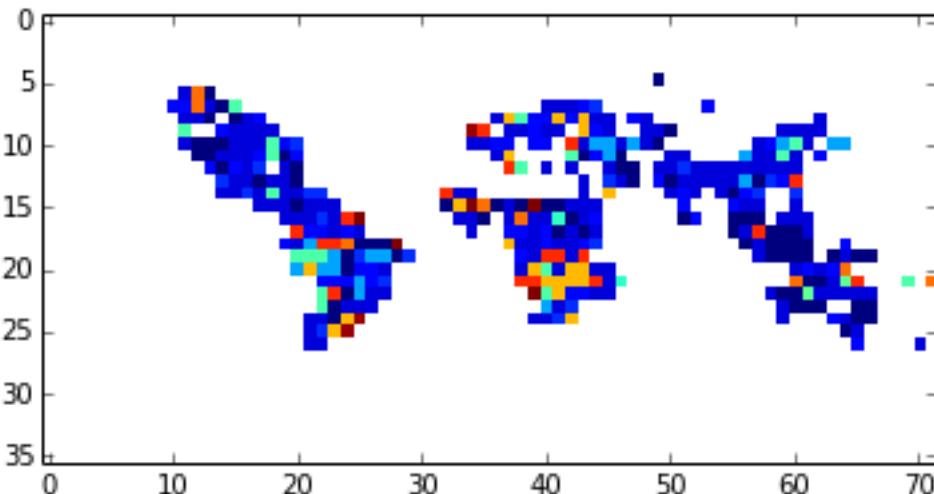


```
# climate data
scdata = np.zeros_like(Y).astype(float)

for y in [pyear,pyear-1]:
    for m in xrange(12):
        scdata[(Y == y) & (M == m)] = cdata[m,y-cyears[0]]

plt.imshow(scdata,interpolation='nearest')

<matplotlib.image.AxesImage at 0x7d5a590>
```



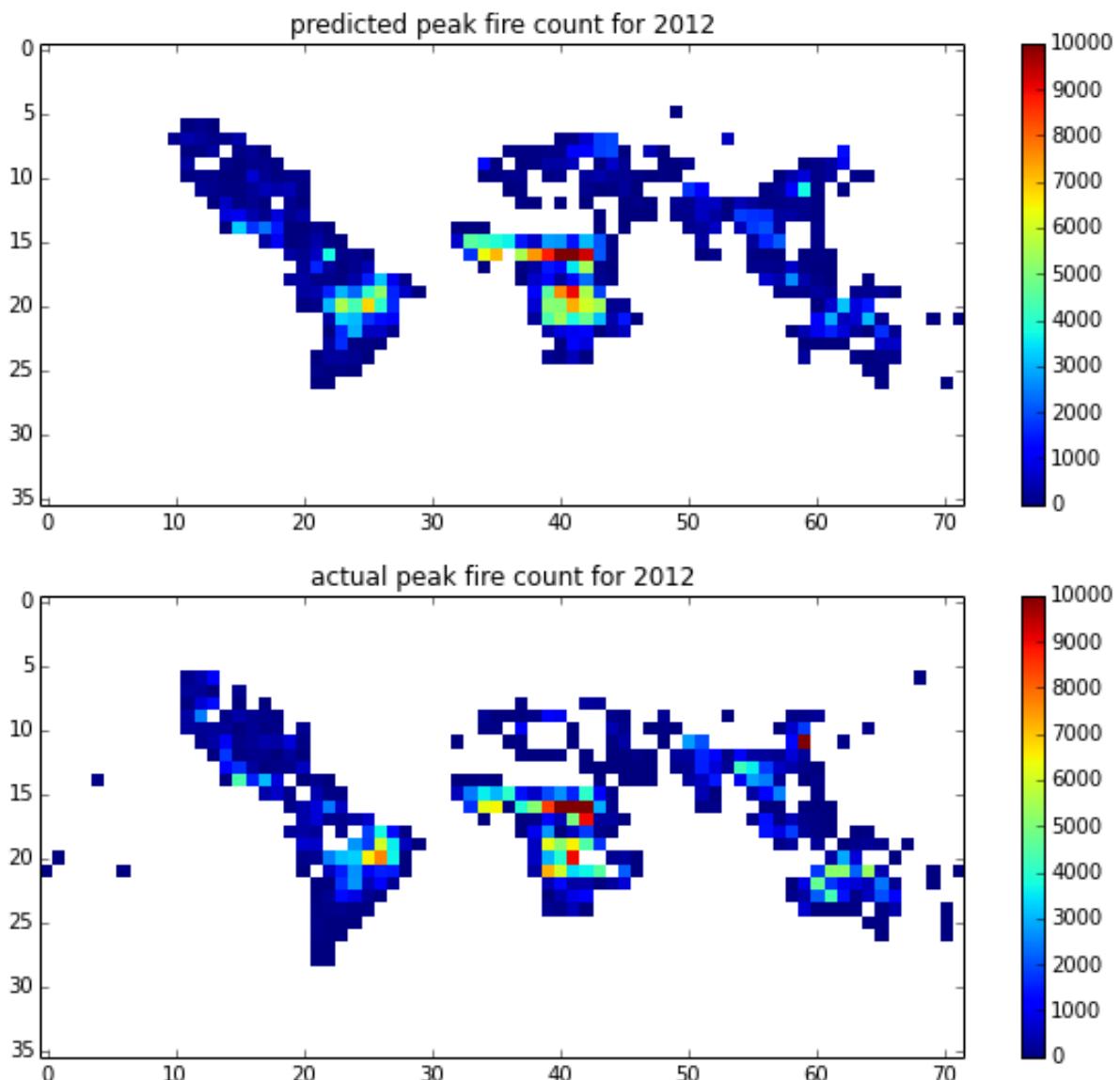
```
# now predict

fc_predict = results['intercept'] + results['slope'] * scdata

plt.figure(figsize=(10,4))
plt.imshow(fc_predict,interpolation='nearest',vmin=0,vmax=10000)
plt.colorbar()
plt.title('predicted peak fire count for %d'%pyear)

plt.figure(figsize=(10,4))
plt.imshow(peak_count[pyear-min_year],\
           interpolation='nearest',vmin=0,vmax=10000)
plt.colorbar()
plt.title('actual peak fire count for %d'%pyear)
```

<matplotlib.text.Text at 0x83ec750>



```

x = peak_count[pyear-min_year].flatten()
y = fc_predict.flatten()

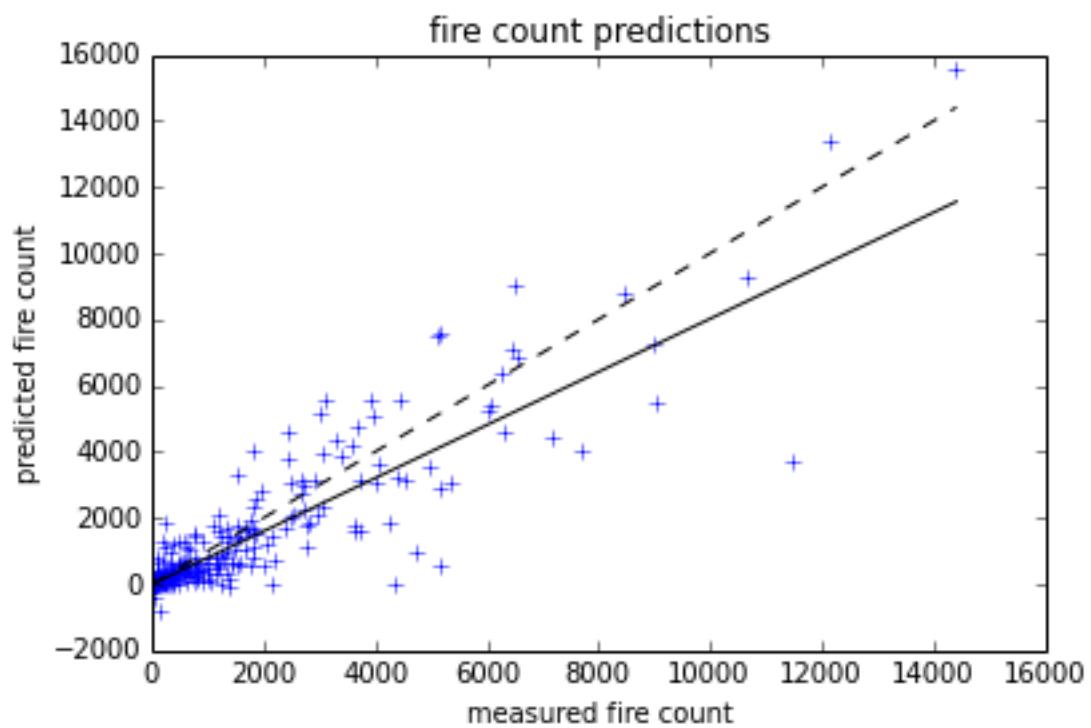
slope,intercept,R,p,err = linregress(x,y)

plt.plot(x,y,'+')
plt.xlabel('measured fire count')
plt.ylabel('predicted fire count')
cc = np.array([0.,x.max()])

plt.plot(cc,cc,'k--')
plt.plot(cc,slope*cc+intercept,'k-')
plt.title('fire count predictions')
print slope,intercept,R,p,err

<class 'numpy.ma.core.MaskedArray'>
0.802827703736 -6.97449131978 0.880785138861 0.0 0.00848080897451

```



CHAPTER
FORTYSIX

RECOLLISION PROBABILITY THEORY

A useful source of information quantifying vegetation amount is the Leaf Area Index (LAI). To interpret LAI from satellite data, we build models of radiative transfer.

One of the simplest such models that we can use at optical wavelengths uses what is known as ‘p theory’ or ‘recollision probability theory’ (Lewis and Disney, 2007; Huang et al., 2007).

The task today is to use p-theory to estimate LAI over some agricultural fields.

References

- P. Lewis and M. Disney (2007) Spectral invariants and scattering across multiple scales from within-leaf to canopy, *Remote Sensing of Environment* 109, 196-206.
- D. Huang, Y. Knyazikhin, R.E. Dickinson, M. Rautiainen, P. Stenberg, M. Disney, P. Lewis, A. Cescatti, Y. Tian, W. Verhoef, and R.B. Myneni (2007), Canopy spectral invariants for remote sensing and model applications, *Remote Sensing of Environment*, 106, 106-122
- Knyazikhin Y, Schull MA, Stenberg P, Mottus M, Rautiainen M, Yang Y, Marshak A, Latorre Carmona P, Kaufmann RK, Lewis P, Disney MI, Vanderbilt V, Davis AB, Baret F, Jacquemoud S, Lyapustin A, Myneni RB. (2013) Hyperspectral remote sensing of foliar nitrogen content. Proc Natl Acad Sci USA, 10.1073/pnas.1210196109.

CHAPTER
FORTYSEVEN

DATA

The dataset you have available is an airborne hyperspectral image (HYMAP) dataset for which you have a 512x512 pixel subscene taken in 125 wavebands with a spatial resolution of 4m. The data were obtained on 17 June 2000 over Barton Bendish Farms, Norfolk during the BNSC/NERC SHAC campaign.

The data are available as a compressed *flat binary* file in the directory ‘files/data <files/data>‘__.

First, uncompress the dataset:

```
# or try zcat if gzcat isn't there
!gzcat files/data/bbHYMAP.dat.gz > files/data/bbHYMAP.dat

!ls -l files/data/bbHYMAP.dat

-rw-r--r-- 1 plewis  staff  131072000 26 Nov 17:31 files/data/bbHYMAP.dat
```

The file is 131072000 bytes in 32 bit floating point format:

```
print 'nbands =', 131072000 / (512*512*4)

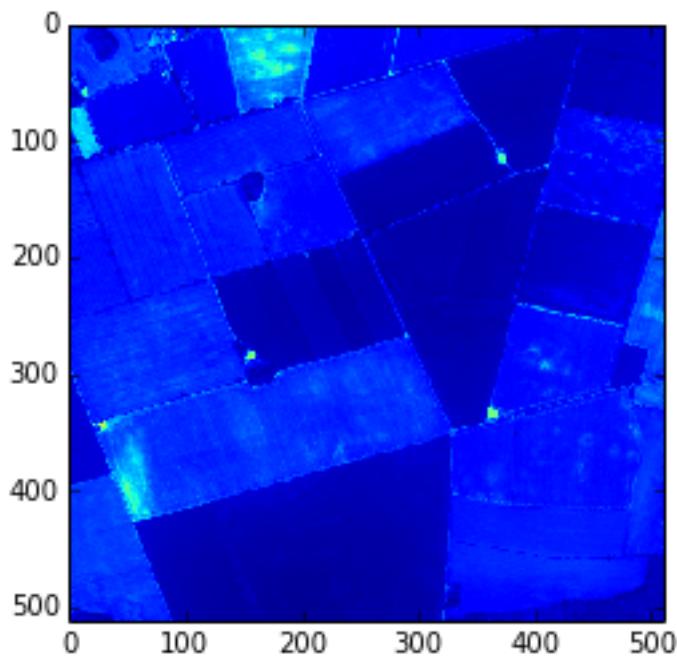
nbands = 125

# we will use memmap to read the data in
from numpy import memmap

hymap = memmap('files/data/bbHYMAP.dat', dtype=np.float32, mode='r', shape=(125, 512, 512))

plt.imshow(hymap[10], interpolation='nearest')

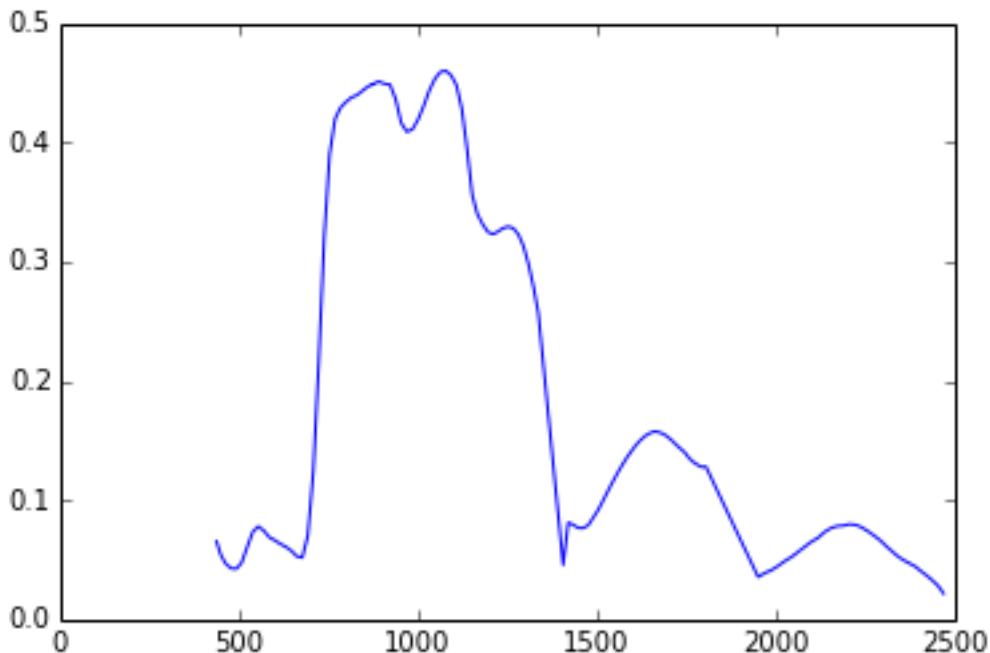
<matplotlib.image.AxesImage at 0x10598edd0>
```



The wavelengths associated with each band are stored in 'files/data/wavebands.dat'.

```
mean = hymap.mean(axis=(1,2))
wavelength = np.loadtxt('files/data/wavebands.dat')
plt.plot(wavelength,mean)
```

```
[<matplotlib.lines.Line2D at 0x10f089790>]
```



CHAPTER
FORTYEIGHT

THEORY

The simplest form of model in p-theory assumes that the photon recollision probability is constant with wavelength and scattering order. Under this assumption, the total scattering from the canopy, W is:

$$W = i_0 \frac{(1-p)\omega}{1-p\omega}$$

where i_0 is the canopy interception probability, ω is the leaf-level scattering (the leaf single scattering albedo) and p is the recollision probability: the probability that a photon, having intercepted a canopy element, will recollide with another element rather than escape the canopy.

We can develop from this a model of the canopy *reflectance* ρ (i.e. that portion scattered upwards, perhaps in a particular direction):

$$\rho = \frac{a\omega}{1-p\omega}$$

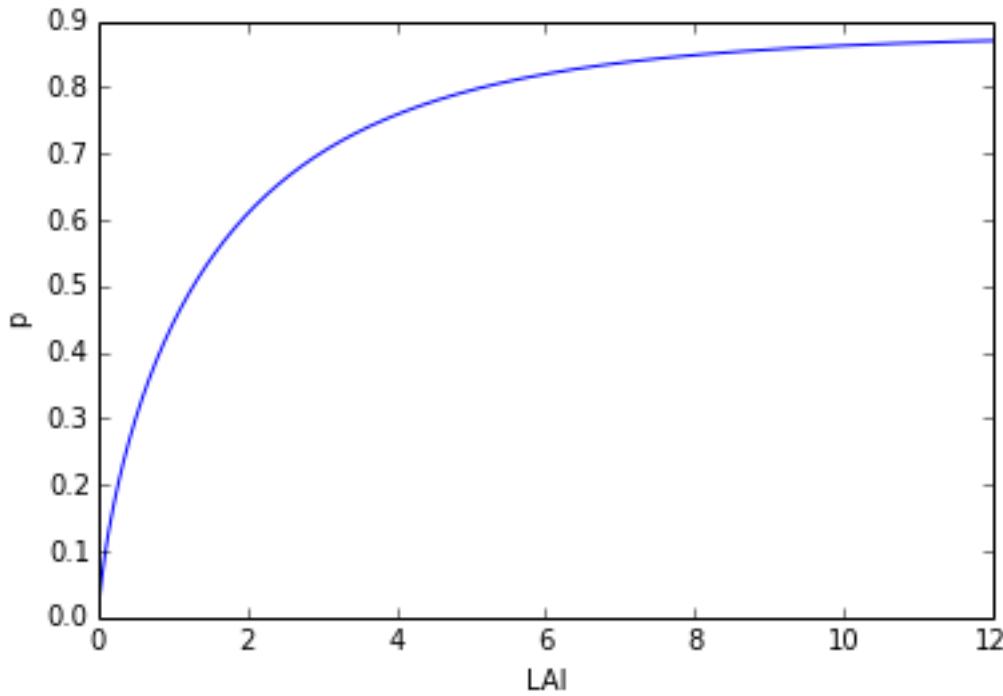
For a closed canopy, or one with only little soil influence, this will describe the spectral reflectance for some given leaf single scattering albedo spectrum and given p .

If we know ω then, and have a measurement of ρ , we can estimate p . From p , we can estimate LAI according to Lewis and Disney (2007) by:

$$p = 0.88 (1 - \exp(-0.7LAI^{0.75}))$$

```
LAI = np.arange(0,12,0.01)
p = 0.88*(1 - np.exp(-0.7 * LAI**0.75))
plt.plot(LAI,p)
plt.xlabel('LAI')
plt.ylabel('p')

<matplotlib.text.Text at 0x10f12f0d0>
```



There are several ways we could estimate p . An interesting feature exploited by Knyazikhin et al. (2013) follows from:

$$\frac{\rho}{\omega} = \frac{a}{1 - p\omega}$$

so

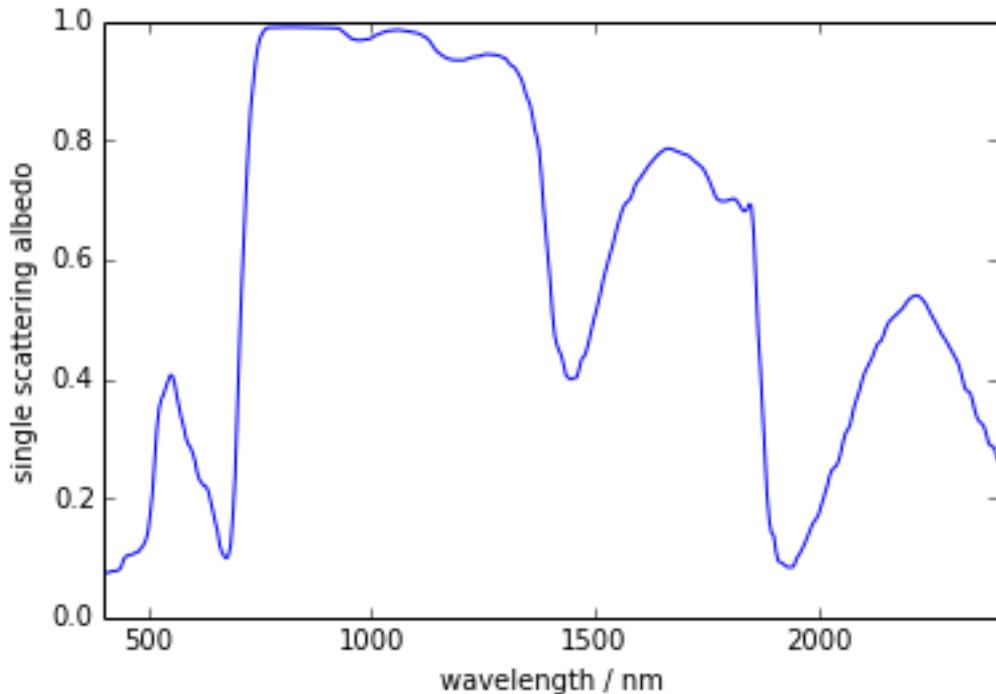
$$\frac{\rho}{\omega} = a + p\rho$$

So that if we plot $\frac{\rho}{\omega}$ as a function of ρ , this theory predicts we should see a straight line.

An example leaf single scattering albedo is given in ‘files/data/ssalbedo.dat’:

```
ssalbedo = np.loadtxt('files/data/ssalbedo.dat').T
plt.plot(ssalbedo[0], ssalbedo[1])
plt.xlabel('wavelength / nm')
plt.ylabel('single scattering albedo')
plt.xlim(ssalbedo[0][0], ssalbedo[0][-1])  

(400.0, 2400.0)
```



which is sampled every 1 nm from 400 to 2400 nm.

We will clearly need to resample this to the same wavebands as the hyperspectral data:

```
from scipy.interpolate import interp1d
from scipy.stats import linregress

f = interp1d(ssalbedo[0], ssalbedo[1])
omega = f(wavelength[wavelength<=2400])

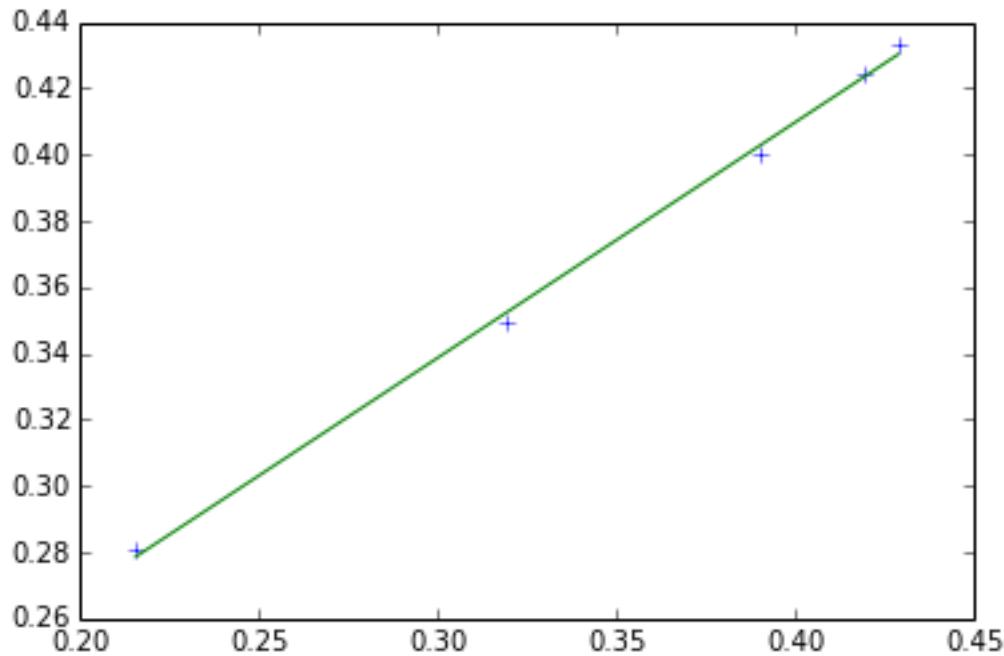
# wavelength
W = (wavelength >= 710) & (wavelength <= 790)

wave = wavelength[W]
# single scattering albedo
omega = f(wavelength[W])
# reflectance
rho = hymap[W]

# lets see if its a straight line!
mean = rho.mean(axis=(1,2))
plt.plot(mean, mean/omega, '+')

# linear fit
slope, intercept, r_value, p_value, std_err = linregress(mean, mean/omega)
x = np.array([mean[0], mean[-1]])
plt.plot(x, x*slope+intercept)
print 'slope', slope, 'intercept', intercept
```

```
slope 0.710882123721 intercept 0.125383329915
```



So, here, p is 0.71088

```
#p = 0.88*(1 - np.exp(-0.7 * LAI**0.75))
LAI = (np.log(1 - slope/0.88)/-0.7)**(4./3.)
print LAI

3.13529156174
```

Following Knyazikhin et al. (2013) we can calculate the DASF (Directional Area Scattering Function) from:

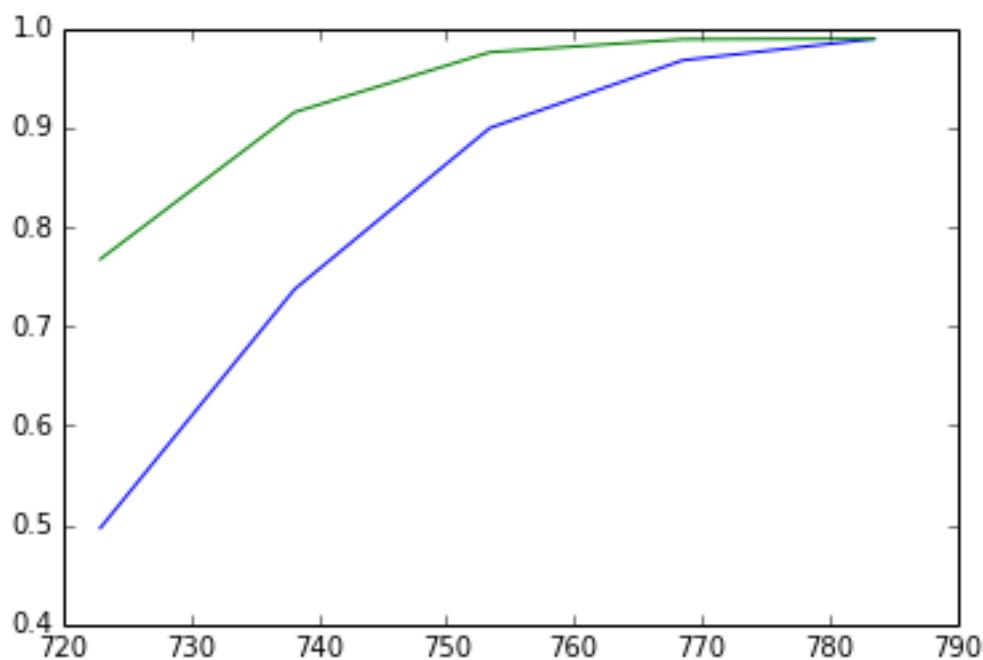
```
DASF = intercept/(1 - slope)
print DASF
```

```
0.43367546666
```

and from that, W :

```
W = mean/DASF
plt.plot(wave,W)
plt.plot(wave,omega)

[<matplotlib.lines.Line2D at 0x11b5d1410>]
```



which in turn can give access to leaf biochemistry.

CHAPTER
FORTYNINE

THE TASK

The output of this exercise should be a spatial dataset of LAI.

The processing for this task is quite straightforward, and you *should* be able to develop a neat algorithm given the information above.

You should work in teams for this exercise as in previous weeks, and assign tasks for people to complete after you have discussed an overall algorithm and defined any interfaces you will need.

If you finish the task quickly, you could explore the impact of the the leaf single scattering albedo assumed, and/or examine the impact of widening the wavelength range used here.

READ AND USE SOME DIFFERENT FILE FORMATS

In this session, we will learn to use some geospatial tools using GDAL in Python. A good set of working notes on how to use GDAL has been developed that you will find useful for background reading.

HDF(Hierarchical Data Format) and [HDF-EOS](#) are common formats for EO data so you need to have some idea how to use and manipulate them.

A hierarchical data format is essentially a format that ‘packs’ together various aspects of a dataset (metadata, raster data etc.) into a binary file. There are many tools for manipulating and reading HDF in python, but we will use one of the more generic tools, [gdal](#) here.

When using HDF files, we need to have some idea of the structure of the contents, although you can clearly explore that yourself in an interactive session. MODIS products have extensive information available to help you interpret the datasets, for example the MODIS LAI/fAPAR product [MOD15A2](#). We will use this as an example to explore a dataset.

You will need access to the file ‘MCD15A2.A2011185.h09v05.005.2011213154534.hdf <http://www2.geog.ucl.ac.uk/~plewis/geogg122/_images/MCD15A2.A2011185.h09v05.005.2011213154534.hdf>‘, which you might access from the [MODIS Land Products](#) site

Before going into the Python coding for GDAL, it is worthwhile looking over some of the tools that are provided with GDAL and that can be run from the shell. In particular, we can use the [gdalinfo](#) program, that takes a filename and will output a copious description of the data, including metadata, but also geographic projection, size, number of bands, etc.

```
!gdalinfo MCD15A2.A2011185.h09v05.005.2011213154534.hdf
```

```
Driver: HDF4/Hierarchical Data Format Release 4
Files: MCD15A2.A2011185.h09v05.005.2011213154534.hdf
Size is 512, 512
Coordinate System is ``
Metadata:
  HDFEOSVersion=HDFEOS_V2.9
  LOCALGRANULEID=MCD15A2.A2011185.h09v05.005.2011213154534.hdf
  PRODUCTIONDATETIME=2011-08-01T15:45:34.000Z
  DAYNIGHTFLAG=Day
  REPROCESSINGACTUAL=reprocessed
  LOCALVERSIONID=5.0.4
  REPROCESSINGPLANNED=further update is anticipated
  SCIENCEQUALITYFLAG=Not Investigated
  AUTOMATICQUALITYFLAGEXPLANATION=No automatic quality assessment is performed in the PGE
  AUTOMATICQUALITYFLAG=Passed
  SCIENCEQUALITYFLAGEXPLANATION=See http://landweb.nascom.nasa.gov/cgi-bin/QA_WWW/qaFlagPage.cgi?
  QAPERCENTMISSINGDATA=3
  QAPERCENTOUTOFBOUNDSDATA=3
  QAPERCENTCLOUDCOVER=23
  QAPERCENTINTERPOLATEDDATA=0
  PARAMETERNAME=MCD15A2
  VERSIONID=5
  SHORTNAME=MCD15A2
  INPUTPOINTER=MYD15A1.A2011192.h09v05.005.2011194222549.hdf, MYD15A1.A2011191.h09v05.005.2011194
```

```
GRINGPOINTLONGITUDE=-103.835851753394, -117.486656023174, -104.256722414513, -92.131858571552
GRINGPOINTLATITUDE=29.8360532722546, 39.9999999964079, 40.0742066197196, 29.9009502428382
GRINGPOINTSEQUENCENO=1, 2, 3, 4
EXCLUSIONGRINGFLAG=N
RANGEENDINGDATE=2011-07-11
RANGEENDINGTIME=23:59:59
RANGEBEGINNINGDATE=2011-07-04
RANGEBEGINNINGTIME=00:00:00
PGEVERSION=5.0.9
ASSOCIATEDSENSORSHORTNAME=MODIS
ASSOCIATEDPLATFORMSHORTNAME=Terra
ASSOCIATEDINSTRUMENTSHORTNAME=MODIS
ASSOCIATEDSENSORSHORTNAME=MODIS
ASSOCIATEDPLATFORMSHORTNAME=Aqua
ASSOCIATEDINSTRUMENTSHORTNAME=MODIS
QAPERCENTGOODQUALITY=100
QAPERCENTOTHERQUALITY=100
HORIZONTALTILENUMBER=09
VERTICALTILENUMBER=05
TileID=51009005
NDAYS_COMPOSITED=16
QAPERCENTGOODFPAR=100
QAPERCENTGOODLAI=100
QAPERCENTMAINMETHOD=100
QAPERCENTTEMPIRICALMODEL=0
NORTHBOUNDINGCOORDINATE=39.999999964079
SOUTHBOUNDINGCOORDINATE=29.999999973059
EASTBOUNDINGCOORDINATE=-92.3664205550513
WESTBOUNDINGCOORDINATE=-117.486656023174
ALGORITHMPACKAGEACCEPTANCEDATE=10-01-2004
ALGORITHMPACKAGEMATURITYCODE=Normal
ALGORITHMPACKAGENAME=MCDPR_15A2
ALGORITHMPACKAGEVERSION=5
GEOANYABNORMAL=False
GEOESTMAXRMSERROR=50.0
LONGNAME=MODIS/Terra+Aqua Leaf Area Index/FPAR 8-Day L4 Global 1km SIN Grid
PROCESSINGCENTER=MODAPS
SYSTEMFILENAME=MYD15A1.A2011192.h09v05.005.2011194222549.hdf, MYD15A1.A2011191.h09v05.005.20111
NUMBEROFGRANULES=1
GRANULEDAYNIGHTFLAG=Day
GRANULEBEGINNINGDATETIME=2011-07-29T21:15:23.000Z
GRANULEENDINGDATETIME=2011-07-29T21:15:23.000Z
CHARACTERISTICBINANGULARSIZE=30.0
CHARACTERISTICBINSIZE=926.625433055556
DATAOLUMNS=1200
DATAROWS=1200
GLOBALGRIDCOLUMNS=43200
GLOBALGRIDROWS=21600
NADIRDATARESOLUTION=1km
MAXIMUMOBSERVATIONS=1
SPSOPARAMETERS=5367, 2680
PROCESSINGENVIRONMENT=Linux minion5559 2.6.18-238.12.1.el5PAE #1 SMP Tue May 31 14:02:45 EDT 20
DESCRREVISION=5.0
ENGINEERING_DATA=
# MOD_PR15A2 (Vers 5.0.4 Rele 10.18.2006 23:59)
# MUM API Vers 2.5.8 Rev 104 Rel. 11.15.2000 10:49 (pgs)
# (c) 2000 J.M. Glassy, NTSG, LLSD
# portions of MUM API by Petr Votava, NTSG Lab, U.Montana
# HOST ECS_PGS_VirtualHost PROCESS 19540618
# PLATFORM Sys genericunix Vers unknown Release unknown Node (no nodename available)
# INIT-TIME Mon Aug 1 11:45:14 2011

YEARDAY 185 COMPOSITE_PERIOD 24 FIRSTDAY_IN_PERIOD 185
```

```

SDS[PGE34_ISG_MBRLUT] %ID 268697600 Rank 2 (664 120)
SDS[PGE34_OUTFIELD_PROP] %ID 268697601 Rank 2 (78 179)
SDS[PGE34_BROWSEFIELD_PROP] %ID 268697602 Rank 2 (77 164)
READANC SDS[PGE34_ISG_MBRLUT] RANK 2 (664 120)
READANC SDS[PGE34_OUTFIELD_PROP] RANK 2 (78 179)
READANC SDS[PGE34_BROWSEFIELD_PROP] RANK 2 (77 164)
READANC SDS[PGE34_ECSMETA_DICT] RANK 2 (73 110)
READANC SDS[PGE34_RELEASE_NOTES] RANK 2 (84 98)
FLDPROP SDSnam(PGE34_OUTFIELD_PROP)MoleName(PGE34_OUTFIELD_PROP)Status 3 Nelem 13962
BROWSE cvlBrwMol 0 Nelem 57600 Type 21 Status 3
BROWSE cvlBrwMol 1 Nelem 57600 Type 21 Status 3
BROWSE cvlBrwMol 2 Nelem 57600 Type 21 Status 3
BROWSE cvlBrwMol 3 Nelem 57600 Type 21 Status 3
BROWSE: NEW GRID ID 4194320
BROWSEFIELD 0 Sum 1725178 Average 29.951
BROWSEFIELD 1 Sum 581056 Average 10.0878
BROWSEFIELD 2 Sum 358279 Average 6.22012
BROWSEFIELD 3 Sum 8016362 Average 139.173
BROWSE-DONE: N-Pixels 57600 Invalid offsets: 0 OutOfRange 0

```

COMPOSITING FPAR FREQUENCIES

FPAR	0	34
FPAR	1	71
FPAR	2	21
FPAR	3	50
FPAR	4	221
FPAR	5	592
FPAR	6	4712
FPAR	7	1445
FPAR	8	522
FPAR	9	1210
FPAR	10	12969
FPAR	11	17195
FPAR	12	83945
FPAR	13	197436
FPAR	14	79525
FPAR	15	55842
FPAR	16	68815
FPAR	17	40112
FPAR	18	62851
FPAR	19	42180
FPAR	20	43324
FPAR	21	34354
FPAR	22	20947
FPAR	23	29184
FPAR	24	35510
FPAR	25	32078
FPAR	26	34318
FPAR	27	24998
FPAR	28	28379
FPAR	29	18195
FPAR	30	17813
FPAR	31	15146
FPAR	32	10109
FPAR	33	14734
FPAR	34	17210
FPAR	35	23089
FPAR	36	14491
FPAR	37	15924
FPAR	38	15249
FPAR	39	15657
FPAR	40	9683
FPAR	41	9847

FPAR	42	7833
FPAR	43	11433
FPAR	44	12093
FPAR	45	10148
FPAR	46	9235
FPAR	47	11742
FPAR	48	10464
FPAR	49	8277
FPAR	50	5914
FPAR	51	8763
FPAR	52	6712
FPAR	53	6807
FPAR	54	9616
FPAR	55	7249
FPAR	56	5771
FPAR	57	6052
FPAR	58	5514
FPAR	59	6665
FPAR	60	5731
FPAR	61	5195
FPAR	62	5484
FPAR	63	4754
FPAR	64	4820
FPAR	65	4963
FPAR	66	3775
FPAR	67	4243
FPAR	68	4047
FPAR	69	3461
FPAR	70	2613
FPAR	71	3143
FPAR	72	2812
FPAR	73	2463
FPAR	74	1918
FPAR	75	1894
FPAR	76	1938
FPAR	77	1574
FPAR	78	1329
FPAR	79	1287
FPAR	80	1165
FPAR	81	1210
FPAR	82	1268
FPAR	83	1501
FPAR	84	1562
FPAR	85	1466
FPAR	86	1875
FPAR	87	1865
FPAR	88	2695
FPAR	89	8083
FPAR	90	4573
FPAR	91	800
FPAR	92	830
FPAR	93	723
FPAR	94	446
FPAR	95	146
FPAR	96	59
FPAR	97	52
FPAR	98	26
FPAR	99	6
FPAR	100	1

COMPOSITING LAI FREQUENCIES

LAI	0	135
LAI	1	7017

LAI	2	333047
LAI	3	320752
LAI	4	116468
LAI	5	116636
LAI	6	95497
LAI	7	74899
LAI	8	46317
LAI	9	30182
LAI	10	38262
LAI	11	28509
LAI	12	20955
LAI	13	16270
LAI	14	18420
LAI	15	13305
LAI	16	13099
LAI	17	10446
LAI	18	6287
LAI	19	8236
LAI	20	6428
LAI	21	5986
LAI	22	5851
LAI	23	4651
LAI	24	3958
LAI	25	4149
LAI	26	3228
LAI	27	2795
LAI	28	2712
LAI	29	2032
LAI	30	1961
LAI	31	1927
LAI	32	1557
LAI	33	1448
LAI	34	1445
LAI	35	1123
LAI	36	1135
LAI	37	991
LAI	38	849
LAI	39	748
LAI	40	771
LAI	41	712
LAI	42	641
LAI	43	655
LAI	44	816
LAI	45	947
LAI	46	965
LAI	47	947
LAI	48	934
LAI	49	948
LAI	50	758
LAI	51	677
LAI	52	662
LAI	53	727
LAI	54	651
LAI	55	643
LAI	56	789
LAI	57	790
LAI	58	613
LAI	59	868
LAI	60	676
LAI	61	776
LAI	62	3217
LAI	63	796
LAI	64	608

LAI	65	2298
LAI	66	4179
LAI	67	166
LAI	68	123
LAI	69	5

COMPOSITING DAY INDEX FREQUENCIES

TILE-INDEX	0	Selection Frequency:	177256
TILE-INDEX	1	Selection Frequency:	143267
TILE-INDEX	2	Selection Frequency:	164215
TILE-INDEX	3	Selection Frequency:	128123
TILE-INDEX	4	Selection Frequency:	168637
TILE-INDEX	5	Selection Frequency:	99292
TILE-INDEX	6	Selection Frequency:	110562
TILE-INDEX	7	Selection Frequency:	61898
TILE-INDEX	8	Selection Frequency:	27366
TILE-INDEX	9	Selection Frequency:	89455
TILE-INDEX	10	Selection Frequency:	32747
TILE-INDEX	11	Selection Frequency:	73910
TILE-INDEX	12	Selection Frequency:	43557
TILE-INDEX	13	Selection Frequency:	26380
TILE-INDEX	14	Selection Frequency:	36942
TILE-INDEX	15	Selection Frequency:	14464

LOCALGRANULEID [MCD15A2.A2011185.h09v05.005.2011213154534.hdf]
ECS QC PERCENT N valid (denominator) : 1398071.000000
ECS Total Pixels (denominator) : 1440000

QAPERCENTxx PSA BEFORE CALC
QAPERCENTINTERPOLATEDDATA: 0
QAPERCENTMISSINGDATA : 41929
QAPERCENTOUTOFBOUNDS DATA : 41929
QAPERCENTCLOUDCOVER : 334183
QAPERCENTNOTPRODUCEDCLOUD: 0
QAPERCENTNOTPRODUCEDOTHER: 0
QAPERCENTGOODQUALITY : 1391137
QAPERCENTOTHERQUALITY : 48863
QAPERCENTGOODFPAR : 1391137
QAPERCENTGOODLAI : 1391137
QAPERCENTMAINMETHOD : 1391137
QAPERCENTEMPIRICALMODEL : 6934
QAPERCENTNDAYSCOMPOSITED : 16
QAPERCENTTERRA : 1052272

QAPERCENTxx PSA AFTER CALC
QAPERCENTINTERPOLATEDDATA: 0
QAPERCENTMISSINGDATA : 3
QAPERCENTOUTOFBOUNDS DATA : 3
QAPERCENTCLOUDCOVER : 23
QAPERCENTNOTPRODUCEDCLOUD: 0
QAPERCENTNOTPRODUCEDOTHER: 0
QAPERCENTGOODQUALITY : 100
QAPERCENTOTHERQUALITY : 100
QAPERCENTGOODFPAR : 100
QAPERCENTGOODLAI : 100
QAPERCENTMAINMETHOD : 100
QAPERCENTEMPIRICALMODEL : 0
QAPERCENTNDAYSCOMPOSITED : 16

SESSION ENGINEERING SUMMARY FOR PGE34 8-day FPAR, LAI
UM_VERSION U.MONTANA MODIS PGE34 Vers 5.0.4 Rev 4 Release 10.18.2006 23:59

Candidate Days: 16

```
N. invalid loads      : 0
Pixels failing best day : 41929
Pixels set to fill     : 0
Pixels skipped (disqual): 1118070
Unclassified Pixels    : 41929
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011185.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011186.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011187.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011188.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011189.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011190.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011191.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MOD15A1.A2011192.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011185.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011186.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011187.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011188.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011189.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011190.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011191.h09v05.
MOD15A1 DAILY Input: /MODAPSops8/archive/f5559/running/AMPM_L10m/6913096/MYD15A1.A2011192.h09v05.
```

```
PGE34 Output      : /MODAPSx/archive/f5559/ops8/running/AMPM_L10m/6913096/MCD15A2.1.2011-185T00:
MOD15A2 ANCILLARY : /MODAPSops8/PGE/AMPM/coeff/PGE34/MCD15A2_ANC_RI4.hdf
```

```
QAPERCENTxx PSA AFTER CALC
QAPERCENTINTERPOLATEDDATA: 0
QAPERCENTMISSINGDATA   : 3
QAPERCENTOUTOFBOUNDS DATA: 3
QAPERCENTCLOUDCOVER    : 23
QAPERCENTTERRA         : 75
QAPERCENTGOODQUALITY   : 100
QAPERCENTOTHERQUALITY  : 100
QAPERCENTGOODFPAR     : 100
QAPERCENTGOODLAI       : 100
QAPERCENTMAINMETHOD    : 100
QAPERCENTTEMPIRICALMODEL: 0
QAPERCENTNDAYSCOMPOSITED: 16
```

```
Started Mon Aug  1 11:45:14 2011  Ended Mon Aug  1 11:45:34 2011
Elapsed Time          20 Sec (      0.33 Min)
MOD15A2_FILLVALUE_DOC=MOD15A2 FILL VALUE LEGEND
255 = _Fillvalue, assigned when:
  * the MODAGAGG suf. reflectance for channel VIS, NIR was assigned its _Fillvalue, or
  * land cover pixel itself was assigned _Fillvalue 255 or 254.
254 = land cover assigned as perennial salt or inland fresh water.
253 = land cover assigned as barren, sparse vegetation (rock, tundra, desert.)
252 = land cover assigned as perennial snow, ice.
251 = land cover assigned as "permanent" wetlands/inundated marshlands.
250 = land cover assigned as urban/built-up.
249 = land cover assigned as "unclassified" or not able to determine.
```

```
MOD15A2_FparLai_QC_DOC=
FparLai_QC 5 BITFIELDS IN 8 BITWORD
MODLAND_QC START 0 END 0 VALIDS 2
MODLAND_QC 0 = Good Quality (main algorithm with or without saturation)
MODLAND_QC 1 = Other Quality (back-up algorithm or fill value)
SENSOR START 1 END 1 VALIDS 2
SENSOR      0 = Terra
SENSOR      1 = Aqua
```

```

DEADDETECTOR START 2 END 2 VALIDS 2
DEADDETECTOR 0 = Detectors apparently fine for up to 50% of channels 1,2
DEADDETECTOR 1 = Dead detectors caused >50% adjacent detector retrieval
CLOUDSTATE START 3 END 4 VALIDS 4 (this inherited from Aggregate_QC bits {0,1} cloud state)
CLOUDSTATE 00 = 0 Significant clouds NOT present (clear)
CLOUDSTATE 01 = 1 Significant clouds WERE present
CLOUDSTATE 10 = 2 Mixed cloud present on pixel
CLOUDSTATE 11 = 3 Cloud state not defined, assumed clear
SCF_QC START 5 END 7 VALIDS 5
SCF_QC      000=0 Main (RT) algorithm used, best result possible (no saturation)
SCF_QC      001=1 Main (RT) algorithm used, saturation occurred. Good, very usable.
SCF_QC      010=2 Main algorithm failed due to bad geometry, empirical algorithm used
SCF_QC      011=3 Main algorithm failed due to problems other than geometry, empirical algorithm used
SCF_QC      100=4 Pixel not produced at all, value couldn't be retrieved (possible reasons: bad L

MOD15A2_FparExtra_QC_DOC=
FparExtra_QC 6 BITFIELDS IN 8 BITWORD
LANDSEA PASS-THROUGH START 0 END 1 VALIDS 4
LANDSEA 00 = 0 LAND      AggrQC(3,5)values{001}
LANDSEA 01 = 1 SHORE     AggrQC(3,5)values{000,010,100}
LANDSEA 10 = 2 FRESHWATER AggrQC(3,5)values{011,101}
LANDSEA 11 = 3 OCEAN     AggrQC(3,5)values{110,111}
SNOW_ICE (from Aggregate_QC bits) START 2 END 2 VALIDS 2
SNOW_ICE 0 = No snow/ice detected
SNOW_ICE 1 = Snow/ice were detected
AEROSOL START 3 END 3 VALIDS 2
AEROSOL 0 = No or low atmospheric aerosol levels detected
AEROSOL 1 = Average or high aerosol levels detected
CIRRUS (from Aggregate_QC bits {8,9}) START 4 END 4 VALIDS 2
CIRRUS 0 = No cirrus detected
CIRRUS 1 = Cirrus was detected
INTERNAL_CLOUD_MASK START 5 END 5 VALIDS 2
INTERNAL_CLOUD_MASK 0 = No clouds
INTERNAL_CLOUD_MASK 1 = Clouds were detected
CLOUD_SHADOW START 6 END 6 VALIDS 2
CLOUD_SHADOW 0 = No cloud shadow detected
CLOUD_SHADOW 1 = Cloud shadow detected
SCF_BIOME_MASK START 7 END 7 VALIDS 2
SCF_BIOME_MASK 0 = Biome outside interval <1,4>
SCF_BIOME_MASK 1 = Biome in interval <1,4>

MOD15A2_StdDev_QC_DOC=MOD15A2 STANDARD DEVIATION FILL VALUE LEGEND
255 = _Fillvalue, assigned when:
    * the MODAGAGG suf. reflectance for channel VIS, NIR was assigned its _Fillvalue, or
    * land cover pixel itself was assigned _Fillvalue 255 or 254.
254 = land cover assigned as perennial salt or inland fresh water.
253 = land cover assigned as barren, sparse vegetation (rock, tundra, desert.)
252 = land cover assigned as perennial snow, ice.
251 = land cover assigned as "permanent" wetlands/inundated marshlands.
250 = land cover assigned as urban/built-up.
249 = land cover assigned as "unclassified" or not able to determine.
248 = no standard deviation available, pixel produced using backup method.

```

MOD15A1_ANC_BUILD_CERT=mtAncUtil v. 1.8 Rel. 09.11.2000 17:36 API v. 2.5.6 release 09.14.2000 1

UM_VERSION=U.MONTANA MODIS PGE34 Vers 5.0.4 Rev 4 Release 10.18.2006 23:59

Subdatasets:

```

SUBDATASET_1_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
SUBDATASET_1_DESC=[1200x1200] Fpar_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
SUBDATASET_2_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
SUBDATASET_2_DESC=[1200x1200] Lai_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
SUBDATASET_3_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
SUBDATASET_3_DESC=[1200x1200] FparLai_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)

```

```

SUBDATASET_4_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
SUBDATASET_4_DESC=[1200x1200] FparExtra_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
SUBDATASET_5_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
SUBDATASET_5_DESC=[1200x1200] FparStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
SUBDATASET_6_NAME=HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
SUBDATASET_6_DESC=[1200x1200] LaiStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)

Corner Coordinates:
Upper Left  (    0.0,      0.0)
Lower Left   (    0.0,    512.0)
Upper Right  (  512.0,      0.0)
Lower Right  (  512.0,    512.0)
Center       ( 256.0,    256.0)

# Filter lines that do not have BOUNDINGCOORDINATE in them
!gdalinfo MCD15A2.A2011185.h09v05.005.2011213154534.hdf | grep BOUNDINGCOORDINATE

```

```

NORTHBOUNDINGCOORDINATE=39.9999999964079
SOUTHBOUNDINGCOORDINATE=29.9999999973059
EASTBOUNDINGCOORDINATE=-92.3664205550513
WESTBOUNDINGCOORDINATE=-117.486656023174

```

We can check this against e.g. the [UNH MODIS tile calculator](#), just to confirm that we have interpreted the coordinates correctly.

We can apply other shell GDAL tools, e.g. to perform a reprojection from the native [MODIS sinusoidal](#) projection, to the [Contiguous United States NAD27 Albers Equal Area](#):

```

!gdalwarp -of GTiff \
-t_srs '+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0 +ellps=clrk66 +units=m +no_defs' \
-tr 1000 1000 \
'HDF4_EOS:EOS_GRID:MCD15A2.A2011185.h09v05.005.2011213154534.hdf:MOD_Grid_MOD15A2:Lai_1km' output.tif
Creating output file that is 2152P x 1323L.
Processing input file HDF4_EOS:EOS_GRID:MCD15A2.A2011185.h09v05.005.2011213154534.hdf:MOD_Grid_MOD15A2
Using internal nodata values (eg. 255) for image HDF4_EOS:EOS_GRID:MCD15A2.A2011185.h09v05.005.2011213154534.hdf:MOD_Grid_MOD15A2:Lai_1km
0...10...20...30...40...50...60...70...80...90...100 - done.

```

where `MCD15A2.A2011185.h09v05.005.2011213154534.hdf` is the name of the input HDF file, `MOD_Grid_MOD15A2:Lai_1km` is the data product we want, and the rather menacing string `+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0 +ellps=clrk66 +units=m +no_defs` specifies the projection in Proj4 format. You can typically find the projection you want on [spatialreference.org](#), and just copy and paste the contents of [Proj4 definition](#) (remember to surround it by quotes). The option `-tr xres yres` specifies the desired resolution of the output dataset (1000 by 1000 m in the case above). `-of GTiff` specifies the GeoTiff format to be used as as output.

Having some idea what information is in the hdf file then, we can proceed to read the data in inside Python using the GDAL library:

```

import gdal # Import GDAL library bindings

# The file that we shall be using
# Needs to be on current directory
filename = 'MCD15A2.A2011185.h09v05.005.2011213154534.hdf'

g = gdal.Open(filename)
# g should now be a GDAL dataset, but if the file isn't found
# g will be none. Let's test this:
if g is None:
    print "Problem opening file %s!" % filename
else:
    print "File %s opened fine" % filename

```

```

subdatasets = g.GetSubDatasets()
for fname, name in subdatasets:
    print name
    print "\t", fname

File MCD15A2.A2011185.h09v05.005.2011213154534.hdf opened fine
[1200x1200] Fpar_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Fpar_1km
[1200x1200] Lai_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Lai_1km
[1200x1200] FparLai_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparLai_QC
[1200x1200] FparExtra_QC MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparExtra_QC
[1200x1200] FparStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparStdDev_1km
[1200x1200] LaiStdDev_1km MOD_Grid_MOD15A2 (8-bit unsigned integer)
    HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:LaiStdDev_1km

```

In the previous code snippet we have done a number of different things:

1. Import the GDAL library
2. Open a file with GDAL, storing a handler to the file in `g`
3. Test that `g` is not `None` (as this indicates failure opening the file. Try changing `filename` above to something else)
4. We then use the `GetSubDatasets()` method to read out information on the different subdatasets available from this file (compare to the output of `gdalinfo` on the shelf earlier)
5. Loop over the retrieved subdatasets to print the name (human-readable information) and the GDAL filename. This last item is the filename that you need to use to tell GDAL to open a particular data layer of the 6 layers present in this example

Let's say that we want to access the LAI information. By contrasting the output of the above code (or `gdalinfo`) to the contents of the [LAI/fAPAR product information page](#), we find out that we want the layers for `Lai_1km`, `FparLai_Qc`, `FparExtra_QC` and `LaiStdDev_1km`.

To read these individual datasets, we need to open each of them individually using GDAL, and the GDAL filenames used above:

```

# Let's create a list with the selected layer names
selected_layers = [ "Lai_1km", "FparLai_QC", "FparExtra_QC", "LaiStdDev_1km" ]
# We will store the data in a dictionary
# Initialise an empty dictionary
data = {}
# for convenience, we will use string substitution to create a
# template for GDAL filenames, which we'll substitute on the fly:
file_template = 'HDF4_EOS:EOS_GRID:"%s":MOD_Grid_MOD15A2:%s'
for i, layer in enumerate ( selected_layers ):
    this_file = file_template % ( filename, layer )
    print "Opening Layer %d: %s" % (i+1, this_file )
    g = gdal.Open ( this_file )

    if g is None:
        raise IOError
    data[layer] = g.ReadAsArray()
    print "\t>>> Read %s!" % layer

Opening Layer 1: HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:Lai_1km!
    >>> Read Lai_1km!
Opening Layer 2: HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparLai_QC!
    >>> Read FparLai_QC!
Opening Layer 3: HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2:FparExtra_QC!
    >>> Read FparExtra_QC!

```

```
>>> Read FparExtra_QC!
Opening Layer 4: HDF4_EOS:EOS_GRID:"MCD15A2.A2011185.h09v05.005.2011213154534.hdf":MOD_Grid_MOD15A2
>>> Read LaiStdDev_1km!
```

In the previous code, we have seen a way of neatly creating the filenames required by GDAL to access the independent datasets: a template string that gets substituted with the filename and the layer name. Note that the presence of double quotes in the template requires us to use single quotes around it. The data is now stored in a dictionary, and can be accessed as e.g. `data['Lai_1km']`:

```
print read_data['Lai_1km']

[[ 3  3  2 ...,  6  8 21]
 [ 4  3  6 ...,  8 18 14]
 [ 3 12 11 ..., 12  8  8]
 ...
 [ 2  3  2 ..., 18 11 17]
 [ 2  3  3 ..., 16 19 15]
 [ 3  2  2 ..., 15 16 15]]
```

Now we have to translate the LAI values into meaningful quantities. According to the [LAI](#) webpage, there is a scale factor of 0.1 involved for LAI and SD LAI:

```
lai = data['Lai_1km'] * 0.1
lai_sd = data['LaiStdDev_1km'] * 0.1

print "LAI"
print lai
print "SD"
print lai_sd

LAI
[[ 0.3  0.3  0.2 ...,  0.6  0.8 2.1]
 [ 0.4  0.3  0.6 ...,  0.8  1.8 1.4]
 [ 0.3  1.2  1.1 ...,  1.2  0.8 0.8]
 ...
 [ 0.2  0.3  0.2 ...,  1.8  1.1 1.7]
 [ 0.2  0.3  0.3 ...,  1.6  1.9 1.5]
 [ 0.3  0.2  0.2 ...,  1.5  1.6 1.5]]
SD
[[ 0.2  0.2  0.1 ...,  0.2  0.1 0.3]
 [ 0.2  0.2  0.2 ...,  0.2  0.3 0.2]
 [ 0.  0.1  0.2 ...,  0.1  0.2 0.2]
 ...
 [ 0.1  0.1  0.1 ...,  0.3  0.  0.1]
 [ 0.1  0.1  0.1 ...,  0.2  0.2 0.1]
 [ 0.1  0.1  0.1 ...,  0.1  0.2 0.1]]
```

Some SD values are clearly given as 0.0, which is unlikely to be true. We should then examine the QC (Quality Control) information. The codes for this are also given on the LAI product page. They are described as bit combinations:

Bit No.

Parameter Name	Bit Combination	Explanation
----------------	-----------------	-------------

0

MODLAND_QC bits

0

Good quality (main algorithm with or without saturation)

1

Other Quality (back-up algorithm or fill values)

1

Sensor

0</td><td> TERRA</td>

1

AQUA

2

DeadDetector

0

Detectors apparently fine for up to 50% of channels 1 2

1

Dead detectors caused >50% adjacent detector retrieval

3-4

CloudState

00

Significant clouds NOT present (clear)

01

Significant clouds WERE present

10

Mixed clouds present on pixel

11

Cloud state not defined assumed clear

5-7

CF_QC

000

Main (RT) method used best result possible (no saturation) </td>

001

Main (RT) method used with saturation. Good very usable

010

Main (RT) method failed due to bad geometry empirical algorithm used

011

Main (RT) method failed due to problems other than geometry empirical algorithm used

100

Pixel not produced at all value couldn't be retrieved (possible reasons: bad L1B data unusable MODAGAGG data)

In using this information, it is up to the user which data he/she wants to pass through for any further processing. There are clearly trade-offs: if you look for only the highest quality data, then the number of samples is likely to be lower than if you were more tolerant. But if you are too tolerant, you will get spurious results. You may find useful information on how to convert from actual QA flags to diagnostics in [this page](#) (they focus on NDVI/EVI, but the theory is the same).

But let's just say that we want to use only the highest quality data. From the table above, these data are given by the CF_QC flag being set to 000 or 001, or in other words 0000000 and 0000001, or 0 and 1 in decimal:

```

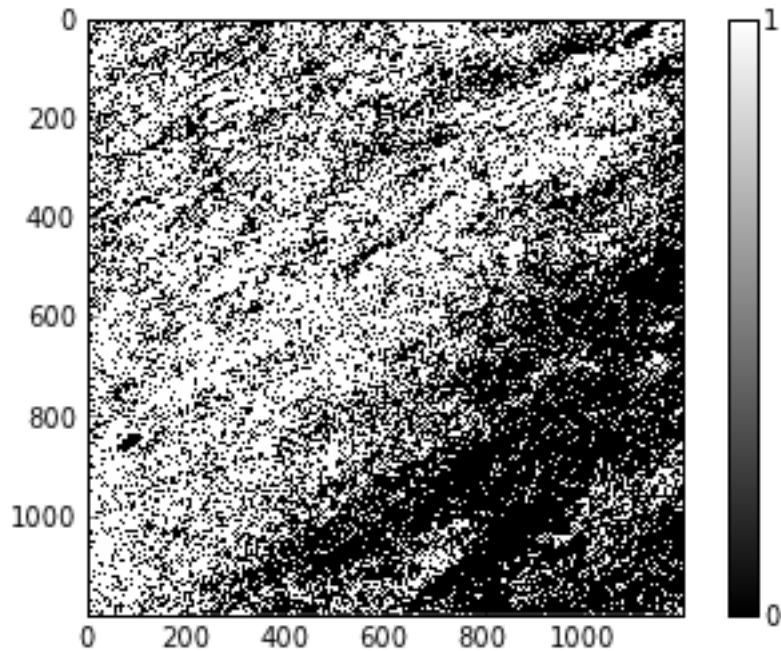
import numpy as np
import pylab as plt

qc = data['FparLai_QC'] # Get the QC data
# Create a mask like the LAI product
mask = np.zeros_like(lai).astype(bool)
# Fill the mask where the conditions are met
mask = np.where( ( qc == 0, True, False ) )
mask = np.where( ( qc == 1, True, mask ) )

# Select a black/white colormap
cmap = plt.cm.gray
# plot the data
plt.imshow(mask, interpolation='nearest', cmap=cmap)
# add a colorbar
plt.colorbar(ticks=[0,1])

<matplotlib.colorbar.Colorbar instance at 0xbb01b00>

```



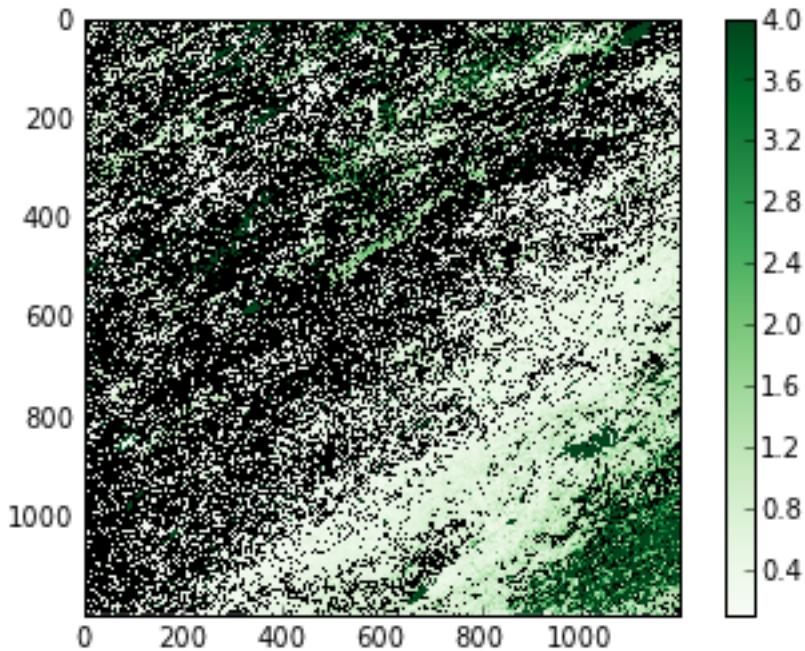
To plot LAI only for the valid pixels (`mask == True`). We can used masked arrays for this. Masked arrays are like normal arrays, but they have an associated mask, which in this case is `mask` define above. We shall also choose another colormap (there are lots to choose from), and set values outside the 0.1 and 4 to be shown as black pixels.

```

cmap = plt.cm.Greens
cmap.set_bad( 'k' )
laim = np.ma.array( lai, mask=mask )
plt.imshow( laim, cmap=cmap, interpolation='nearest', vmin=0.1, vmax=4 )
plt.colorbar()

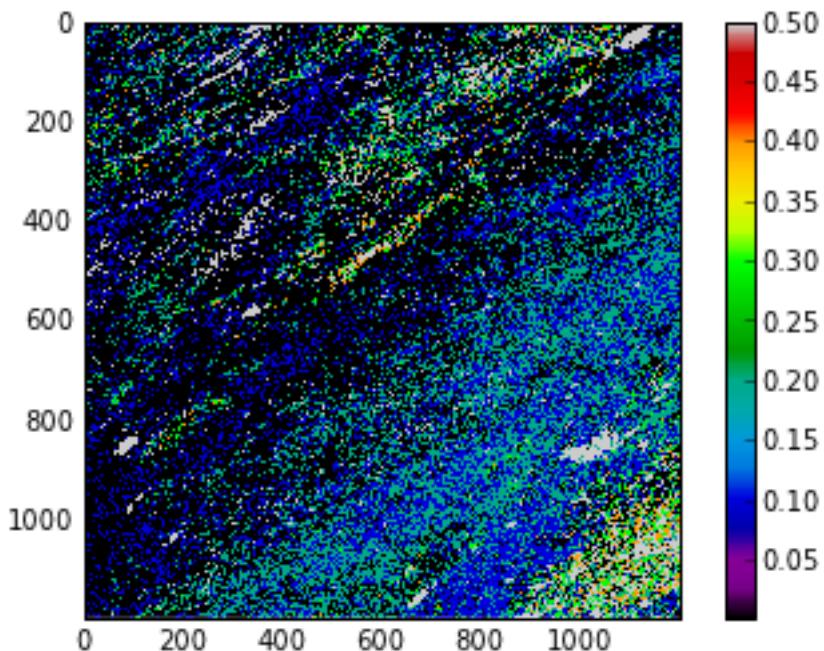
<matplotlib.colorbar.Colorbar instance at 0xd429e18>

```



Similarly, we can do a similar thing for Standard Deviation

```
cmap = plt.cm.spectral  
cmap.set_bad( 'k' )  
stdm = np.ma.array( lai_sd, mask=mask )  
plt.imshow( stdm, cmap=cmap, interpolation='nearest', vmin=0.001, vmax=0.5)  
plt.colorbar()  
  
<matplotlib.colorbar.Colorbar instance at 0x2b8ee20374d0>
```



50.1 Exercise 1

For the moment, we will suppose this data masking to be sufficient. However, closer inspection of the [product data](#) page would show us that the data can take on various Fill Values which any data user should check for.

Modify the code we have developed above to also check that the data are not unwanted ‘fill values’ and use this to modify the data mask.

Hint Note that these fill values are applied to Lai_1km, not the QC information, so you have to check values in that dataset and modify the mask accordingly.

50.2 Exercise 2

We also have access to FparExtra_QC

```
qc1 = data['FparExtra_QC']
```

The data values in FparExtra_QC are:

Bit No.

```
<th>Parameter Name</th><th> Bit Combination</th><th>Explanation</th>
```

0-1

LandSea

00

Land

01

Shore

10

Freshwater

11

Ocean

2

Snow/Ice

0

No snow/ice detected

1

Snow/ice detected

3

Aerosol

0

No or low atmospheric aerosol levels detected

1

Average or high atmospheric aerosol levels detected

4

Cirrus

```
0</td><td>No cirrus detected</td>
```

```
1  
Cirrus detected  
5  
Cloud  
0  
No cloud detected  
1  
Cloud detected  
6  
Cloud shadow  
0  
No cloud shadow detected  
1  
Cloud shadowdetected  
7  
Biome_mask  
0</td><td>Biome outside interval 1-4</td>  
1  
Biome in interval 1-4
```

Use this QC dataset to make sure that **only** Land pixels are passed in the mask, and apply other data quality measures as appropriate.

Hint this is much the same as the exercise above, but looking at a different QC dataset. The key to doing this is to identify the bit codes for combinations that you want to set or unset.

You can find an example of how to do this [here](#)[DEADLINK]. This filtering shouldn't make much difference in this case, as the tile is mostly land pixels. However, consider tile h17v03 (part of the UK) for the month of June (e.g. MCD15A2.A2011185.h17v03.005.2011213154608.hdf)

TODO as NASA website is down!!!!

Figure 50.1: UK LAI map

```
!pwd
```

```
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory  
pwd: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory  
pwd: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
```

```
def css_styling():  
    from IPython.display import display, HTML  
    styles = "https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter5/mcmc_ipynb.css"  
    return HTML(styles)  
css_styling()  
  
import json  
import matplotlib  
fp = open("../bmh_matplotlibrc.json", 'r')  
s = json.load(fp)  
fp.close()
```

```
matplotlib.rcParams.update(s)
%pylab inline
%config InlineBackend.figure_format = 'svg'
figsize( 7,7 )

Populating the interactive namespace from numpy and matplotlib
```


GDAL AND OGR LIBRARIES

In the previous session, we used the GDAL library to open HDF files. GDAL is not limited to a single file format, but can actually cope with many different raster file formats seamlessly. For *vector* data (i.e., data that is stored by features, each being made up of fields containing different types of information, one of them being a *geometry*, such as polygon, line or point), GDAL has a sister library called OGR. The usefulness of these two libraries is that they allow the user to deal with many of the different file formats in a consistent way.

It is important to note that both GDAL and OGR come with a suite of command line tools that let you do many complex tasks on the command line directly. A listing of the GDAL command line tools is available [here](#), but bear in mind that many blogs etc. carry out examples of using GDAL tools in practice. For OGR, most of the library can be accessed using [ogr2ogr](#), but as usual, you might find more useful information on [blogs](#) etc.

51.1 GDAL data type

GDAL provides a very handy way of dealing with raster data in many different formats, not only by making access to the data easy, but also abstracting the nuances and complications of different file formats. In GDAL, a raster file is made up of the actual raster data (i.e., the values of each pixel of LAI that we saw before), and of some *metadata*. Metadata is data that describes something, and in this case it could be the projection, the location of corners and pixel spacing, etc.

51.1.1 The GeoTransform

GDAL stores information about the location of each pixel using the GeoTransform. The GeoTransform contains the coordinates (in some projection) of the upper left (UL) corner of the image (taken to be the **borders of the pixel** in the UL corner, not the center), the pixel spacing and an additional rotation. By knowing these figures, we can calculate the location of each pixel in the image easily. Let's see how this works with an easy example. We have prepared a GeoTIFF file (GeoTIFF is the more ubiquitous file format for EO data) of the MODIS landcover product for the UK. The data has been extracted from the HDF-EOS files that are similar to the LAI product that we have seen before, and converted. The file is 'lc_h17v03.tif <https://raw.github.com/jgomezdans/geogg122/master/ChapterX_GDAL/lc_h17v03.tif>'. We will open the file in Python, and have a look at finding a particular location.

Assume we are interested in locating Kinder Scout, a moorland in the Peak District National Park. Its coordinates are 1.871417W, 53.384726N. In the MODIS integerised sinusoidal projection, the coordinates are (-124114.3, 5936117.4) (you can use the [MODLAND tile calculator website](#) to do this calculation yourself).

```
import gdal # Import GDAL library
g = gdal.Open ( "lc_h17v03.tif" ) # Open the file
if g is None:
    print "Could not open the file!"
geo_transform = g.GetGeoTransform ()
print geo_transform
print g.RasterXSize, g.RasterYSize
```

```
(-1111950.519667, 463.3127165279167, 0.0, 6671703.118, 0.0, -463.3127165279165)
2400 2400
```

In the previous code, we open the file (we just use the filename), and then query the object for its GeoTransform, which we then print out. The 6-element tuple comprises

1. The Upper Left *easting* coordinate (i.e., *horizontal*)
2. The E-W pixel spacing
3. The rotation (0 degrees if image is “North Up”)
4. The Upper left *northing* coordinate (i.e., *vertical*)
5. The rotation (0 degrees)
6. The N-S pixel spacing, negative as we will be counting from the UL corner

We have also seen that the dataset is of size 2400x2400, using RasterXSize and RasterYSize. The goal is to find the pixel number (i, j) , $0 \leq i, j < 2400$ that corresponds to Kinder Scout. To do this, we use the following calculations:

```
pixel_x = (-124114.3 - geo_transform[0])/geo_transform[1] \
    # The difference in distance between the UL corner (geot[0] \
    #and point of interest. Scaled by geot[1] to get pixel number

pixel_y = (5936117.4 - geo_transform[3])/(geo_transform[5]) # Like for pixel_x, \
    #but in vertical direction. Note the different elements of geot \
    #being used

print pixel_x, pixel_y

2132.11549009 1587.66572071
```

So the pixel number is a floating point number, which we might need to round off as an integer. Let's plot the entire raster map (with minimum value 0 to ignore the ocean) using `'plt.imshow'` and plot the location of Kinder Scout using `'plt.plot'`. We will also use `'plt.annotate'` to add a label with an arrow:

```
lc = g.ReadAsArray() # Read raster data
# Now plot the raster data using gist_earth palette
plt.imshow ( lc, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
# Plot location of our area of interest as a red dot ('ro')
plt.plot ( pixel_x, pixel_y, 'ro' )
# Annotate
plt.annotate('Kinder Scout', xy=(pixel_x, pixel_y), \
    xycoords='data', xytext=(-150, -60), \
    textcoords='offset points', size=12, \
    bbox=dict(boxstyle="round4,pad=.5", fc="0.8"), \
    arrowprops=dict(arrowstyle="->", \
    connectionstyle="angle,angleA=0,angleB=-90,rad=10", \
    color='w'), )
# Remove vertical and horizontal ticks
plt.xticks([])
plt.yticks([])

([], <a list of 0 Text yticklabel objects>)
```

Try it out in some other places!

Find the longitude and latitude of some places of interest in the British isles (West of Greenwich!) and using the MODLAND MODIS tile calculator and the geotransform, repeat the above experiment. Note that the MODIS

calculator calculates both the projected coordinates in the MODIS sinusoidal projection, as well as the pixel number, so it is a helpful way to check whether you got the right result.

Park name

Longitude [deg]

Latitude [deg]

Dartmoor national park

-3.904

50.58

New forest national park

-1.595

50.86

Exmoor national park

-3.651

51.14

Pembrokeshire coast national park

-4.694

51.64

Brecon beacons national park

-3.432

51.88</td>

Pembrokeshire coast national park

-4.79

51.99

Norfolk and suffolk broads

1.569

52.62

Snowdonia national park

-3.898 </td><td>52.9</td>

Peak district national park

-1.802

53.3

Yorkshire dales national park

-2.157

54.23

North yorkshire moors national park

-0.8855

54.37

Lake district national park

-3.084

54.47
Galloway forest park
-4.171
54.87
Northumberland national park
-2.228
55.28
Loch lomond and the trossachs national park
-4.593
56.24
Tay forest park
-4.025
56.59</td>
Cairngorms national park
-3.545
57.08

51.2 The projection

Projections in GDAL objects are stored can be accessed by querying the dataset using the `GetProjection()` method. If we do that on the currently opened dataset (stored in variable `g`), we get:

```
print g.GetProjection()
```

```
PROJCS["unnamed",GEOGCS["Unknown datum based upon the custom spheroid",DATUM["Not_specified_based_
```

The above is the description of the projection (in this case, MODIS sinusoidal) in WKT (well-known text) format. There are a number of different ways of specifying projections, the most important being

- WKT
- Proj4
- EPSG codes

The site spatialreference.org allows you to search a large collection of projections, and get the representation that you want to use.

51.3 Saving files

So far, we have read data from files, but lets see how we can save raster data **to** a new file. We will use the previous landcover map as an example. We will write a method to save the data in a format provided by the user. The procedure is fairly straightforward: we get a handler to a driver (e.g. a GeoTIFF or Erdas Imagine format), we create the output file (giving a filename, number of rows, columns, bands, the data type), and then add the relevant metadata (projection, geotransform, ...). We then select a band from the output and copy the array that we want to write to that band.

```

g = gdal.Open ( "lc_h17v03.tif" ) # Open original file
# Get the x, y and number of bands from the original file
x_size, y_size, n_bands = g.RasterXSize, g.RasterYSize, g.RasterCount
data = g.ReadAsArray ()
driver = gdal.GetDriverByName ( "HFA" ) # Get a handler to a driver
# Maybe try "GeoTIFF" here
# Next line creates the output dataset with
# 1. The filename ("test_lc_h17v03.img")
# 2. The raster size (x_size, y_size)
# 3. The number of bands
# 4. The data type (in this case, Byte.
#     Other typical values might be gdal.GDT_Int16
#     or gdal.GDT_Float32)

dataset_out = driver.Create ( "test_lc_h17v03.img", x_size, y_size, n_bands, \
                             gdal.GDT_Byte )
# Set the output geotransform by reading the input one
dataset_out.SetGeoTransform ( g.GetGeoTransform() )
# Set the output projection by reading the input one
dataset_out.SetProjection ( g.GetProjectionRef() )
# Now, get band # 1, and write our data array.
# Note that the data array needs to have the same type
# as the one specified for dataset_out
dataset_out.GetRasterBand ( 1 ).WriteArray ( data )
# This bit forces GDAL to close the file and write to it
dataset_out = None

```

The output file should hopefully exist in this directory. Let's use `gdalinfo` <<http://www.gdal.org/gdalinfo.html>> to find out about it

```
!gdalinfo test_lc_h17v03.img
```

```

Driver: HFA/Erdas Imagine Images (.img)
Files: test_lc_h17v03.img
Size is 2400, 2400
Coordinate System is:
PROJCS["Sinusoidal",
    GEOGCS["GCS_Unknown datum based upon the custom spheroid",
        DATUM["Not_specified_based_on_custom_spheroid",
            SPHEROID["Custom_spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,6671703.117999999783933)
Pixel Size = (463.312716527916677,-463.312716527916507)
Corner Coordinates:
Upper Left  (-1111950.520, 6671703.118) ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-1111950.520, 5559752.598) ( 15d33'26.06"W, 50d 0' 0.00"N)
Upper Right  (      0.000, 6671703.118) (  0d 0' 0.01"E, 60d 0' 0.00"N)
Lower Right  (      0.000, 5559752.598) (  0d 0' 0.01"E, 50d 0' 0.00"N)
Center       (-555975.260, 6115727.858) ( 8d43' 2.04"W, 55d 0' 0.00"N)
Band 1 Block=64x64 Type=Byte, ColorInterp=Undefined
    Description = Layer_1
    Metadata:
        LAYER_TYPE=athematic

```

So the previous code works. Since this is something we typically do (read some data from one or more files, manipulate it and save the result in output files), it makes a lot of sense to try to put this code in a method that is more or less generic, that we can test and then re-use. Here's a first attempt at it:

```
def save_raster ( output_name, raster_data, dataset, driver="GTiff" ):
    """
    A function to save a 1-band raster using GDAL to the file indicated
    by ``output_name``. It requires a GDAL-accessible dataset to collect
    the projection and geotransform.

    Parameters
    -----
    output_name: str
        The output filename, with full path and extension if required
    raster_data: array
        The array that we want to save
    dataset: str
        Filename of a GDAL-friendly dataset that we want to use to
        read geotransform & projection information
    driver: str
        A GDAL driver string, like GTiff or HFA.
    """

    # Open the reference dataset
    g = gdal.Open ( dataset )
    # Get the Geotransform vector
    geo_transform = g.GetGeoTransform ()
    x_size = g.RasterXSize # Raster xsize
    y_size = g.RasterYSize # Raster ysize
    srs = g.GetProjectionRef () # Projection
    # Need a driver object. By default, we use GeoTIFF
    driver = gdal.GetDriverByName ( driver )
    dataset_out = driver.Create ( output_name, x_size, y_size, 1, \
                                  gdal.GDT_Float32 )
    dataset_out.SetGeoTransform ( geo_transform )
    dataset_out.SetProjection ( srs )
    dataset_out.GetRasterBand ( 1 ).WriteArray ( \
        raster_data.astype(np.float32) )
    dataset_out = None
```

Now try modifying that method so that you can

1. Select the output data type different to Float32
2. Provide a given projection and geotransform (e.g. if you don't have a GDAL filename)

51.4 Reprojecting things

Previously, we have used the [MODLAND](#) grid converter site to go from latitude/longitude pairs to MODIS projection. However, in practice, we might want to use a range of different projections, and convert many points at the same time, so how do we go about that?

In GDAL/OGR, most projection-related tools are in the `osr` package, which needs to be imported like e.g. `gdal` itself. The main tools are the `osr.SpatialReference` object, which defines a projection object (with no projection to start with), and the `osr.CoordinateTransformation` object.

Once you instantiate `osr.SpatialReference`, it holds no projection information. You need to use methods to set it up, using EPSG codes, Proj4 strings, or whatever. These methods typically start by `ImportFrom` (e.g. `ImportFromEPSG`, `ImportFromProj4`, ...).

The `CoordinateTransformation` requires a source and destination spatial references that have been configured. Once this is done, it exposes the method `TransformPoint` to convert coordinates from the source to the destination projection.

Let's see how this works by converting some latitude/longitude pairs to the Ordnance Survey's [National Grid](#) projection. The projection is also available in [spatialreference.org](#), where we can gleam its EPSG code (27700).

The EPSG code for longitude latitude is 4326. Let's see this in practice:

```
from osgeo import osr

# Define the source projection, WGS84 lat/lon.
wgs84 = osr.SpatialReference() # Define a SpatialReference object
wgs84.ImportFromEPSG( 4326 ) # And set it to WGS84 using the EPSG code

# Now for the target projection, Ordnance Survey's British National Grid
osng = osr.SpatialReference() # define the SpatialReference object
# In this case, we get the projection from a Proj4 string
osng.ImportFromEPSG( 27700 )
# or, if using the proj4 representation
osng.ImportFromProj4 ( "+proj=tmerc +lat_0=49 +lon_0=-2 " + \
                      "+k=0.9996012717 +x_0=400000 +y_0=-100000 " + \
                      "+ellps=airy +datum=OSGB36 +units=m +no_defs" )

# Now, we define a coordinate transformation object, *from* wgs84 *to* OSNG
tx = osr.CoordinateTransformation( wgs84, osng )
# We loop over the lines of park_data,
#           using the split method to split by newline characters
park_name, lon, lat = "Snowdonia national park", -3.898,      52.9

# Actually do the transformation using the TransformPoint method
osng_x, osng_y, osng_z = tx.TransformPoint ( lon, lat )
# Print out
print park_name, lon, lat, osng_x, osng_y

Snowdonia national park -3.898 52.9 272430.180112 335304.936823
```

You can test the result is reasonable by feeding the data for `osng_x` and `osng_y` in the OS own coordinate conversion website and making sure that the calculated longitude latitude pair is the same as the one you started with.

51.5 Reprojecting whole rasters

51.5.1 Using command line tools

The easiest way to reproject a raster file is to use GDAL's '`gdalwarp <http://www.gdal.org/gdalwarp.html>`' tool. As an example, let's say we want to reproject the landcover file from earlier on into latitude/longitude (WGS84):

```
!gdalwarp -of GTiff -t_srs "EPSG:4326" -ts 2400 2400 test_lc_h17v03.img lc_h17v03_wgs84.tif

Output dataset lc_h17v03_wgs84.tif exists,
but some commandline options were provided indicating a new dataset
should be created. Please delete existing dataset and run again.
```

We see here that the command takes a number of arguments:

1. `-of GTiff` is the output format (in this case GeoTIFF)
2. `-t_srs "EPSG:4326"` is the **to** projection (the **from** projection is already specified in the source dataset), in this case, lat/long WGS84, known by its **EPSG code**
3. `-ts 2400 2400` instructs `gdalwarp` to use an output of size 2400*2400.
4. `test_lc_h17v03.img` is the **input dataset**
5. `lc_h17v03_wgs84.tif` is the **output dataset**

Note that `gdalwarp` will reproject the data, and decide on the pixel size based on some considerations. This can result in the size of the raster changing. If you wanted to still keep the same raster size, we use the `-ts 2400 2400` option, or select an appropriate pixel size using `-tr xres yres` (note it has to be in the target projection, so degrees in this case). We can use `gdalinfo` to see what we've done.

```
!gdalinfo test_lc_h17v03.img
```

```
Driver: HFA/Erdas Imagine Images (.img)
Files: test_lc_h17v03.img
Size is 2400, 2400
Coordinate System is:
PROJCS["Sinusoidal",
    GEOGCS["GCS_Unknown datum based upon the custom spheroid",
        DATUM["Not_specified_based_on_custom_spheroid",
            SPHEROID["Custom_spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,6671703.117999999783933)
Pixel Size = (463.312716527916677,-463.312716527916507)
Corner Coordinates:
Upper Left  (-1111950.520, 6671703.118)  ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-1111950.520, 5559752.598)  ( 15d33'26.06"W, 50d 0' 0.00"N)
Upper Right  (      0.000, 6671703.118)  ( 0d 0' 0.01"E, 60d 0' 0.00"N)
Lower Right  (      0.000, 5559752.598)  ( 0d 0' 0.01"E, 50d 0' 0.00"N)
Center       ( -555975.260, 6115727.858)  ( 8d43' 2.04"W, 55d 0' 0.00"N)
Band 1 Block=64x64 Type=Byte, ColorInterp=Undefined
    Description = Layer_1
    Metadata:
        LAYER_TYPE=athematic
```

```
!gdalinfo lc_h17v03_wgs84.tif
```

```
Driver: GTiff/GeoTIFF
Files: lc_h17v03_wgs84.tif
Size is 2400, 2400
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (-19.99999994952233,59.99999994611805)
Pixel Size = (0.008333919248404,-0.004166959624202)
Metadata:
    AREA_OR_POINT=Area
Image Structure Metadata:
    INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (-20.0000000, 60.0000000)  ( 20d 0' 0.00"W, 60d 0' 0.00"N)
Lower Left   (-20.0000000, 49.9992969)  ( 20d 0' 0.00"W, 49d59'57.47"N)
Upper Right  ( 0.0014062, 60.0000000)  ( 0d 0' 5.06"E, 60d 0' 0.00"N)
Lower Right  ( 0.0014062, 49.9992969)  ( 0d 0' 5.06"E, 49d59'57.47"N)
Center       ( -9.9992969, 54.9996484)  ( 9d59'57.47"W, 54d59'58.73"N)
Band 1 Block=2400x3 Type=Byte, ColorInterp=Gray
    Description = Layer_1
```

```
Metadata:
  LAYER_TYPE=athematic
```

Let's see how different these two datasets are:

```
g = gdal.Open( "lc_h17v03_wgs84.tif" )
wgs84 = g.ReadAsArray()
g = gdal.Open("test_lc_h17v03.img")
modis = g.ReadAsArray()
plt.subplot( 1, 2, 1 )
plt.imshow( modis, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.subplot( 1, 2, 2 )
plt.imshow( wgs84, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )

<matplotlib.image.AxesImage at 0xe566950>
```

51.5.2 Reprojecting using the Python bindings

The previous section demonstrated how you can reproject raster files using command line tools. Sometimes, you might want to do this from inside a Python script. Ideally, you would have a python method that would perform the projection for you. GDAL allows this by defining in-memory raster files. These are normal GDAL datasets, but that don't exist on the filesystem, only in the computer's memory. They are a convenient "scratchpad" for quick intermediate calculations. GDAL also makes available a function, `gdal.ReprojectImage` that exposes most of the abilities of `gdalwarp`. We shall combine these two tricks to carry out the reprojection. As an example, we shall look at the case where the landcover data for the British Isles mentioned in the previous section needs to be reprojected to the Ordnance Survey National Grid, an appropriate projection for the UK.

The main complication comes from the need of `gdal.ReprojectImage` to operate on GDAL datasets. In the previous section, we saved the data to a GeoTIFF file, so this gives us a starting dataset. We still need to create the output dataset. This means that we need to define the geotransform and size of the output dataset before the projection is made. This entails gathering information on the extent of the original dataset, projecting it to the destination projection, and calculating the number of pixels and geotransform parameters from there. This is a (heavily commented) function that performs just that task:

```
def reproject_dataset( dataset, \
                      pixel_spacing=463., epsg_from=4326, epsg_to=27700 ):
    """
    A sample function to reproject and resample a GDAL dataset from within
    Python. The idea here is to reproject from one system to another, as well
    as to change the pixel size. The procedure is slightly long-winded, but
    goes like this:

    1. Set up the two Spatial Reference systems.
    2. Open the original dataset, and get the geotransform
    3. Calculate bounds of new geotransform by projecting the UL corners
    4. Calculate the number of pixels with the new projection & spacing
    5. Create an in-memory raster dataset
    6. Perform the projection
    """

    # Define the UK OSNG, see <http://spatialreference.org/ref/epsg/27700/>
    osng = osr.SpatialReference()
    osng.ImportFromEPSG(epsg_to)
    wgs84 = osr.SpatialReference()
    wgs84.ImportFromEPSG(epsg_from)
    tx = osr.CoordinateTransformation(wgs84, osng)
    # Up to here, all the projection have been defined, as well as a
    # transformation from the from to the to :
    # We now open the dataset
    g = gdal.Open(dataset)

    # Get the Geotransform vector
```

```
geo_t = g.GetGeoTransform ()
x_size = g.RasterXSize # Raster xsize
y_size = g.RasterYSize # Raster ysize

# Work out the boundaries of the new dataset in the target projection
(ulx, uly, ulz) = tx.TransformPoint( geo_t[0], geo_t[3])
(lrx, lry, lrz) = tx.TransformPoint( geo_t[0] + geo_t[1]*x_size, \
                                     geo_t[3] + geo_t[5]*y_size )
print ulx, uly, ulz
print lrx, lry, lrz
# See how using 27700 and WGS84 introduces a z-value!
# Now, we create an in-memory raster
mem_drv = gdal.GetDriverByName( 'MEM' )
# The size of the raster is given the new projection and pixel spacing
# Using the values we calculated above. Also, setting it to store one band
# and to use Float32 data type.
dest = mem_drv.Create('', int((lrx - ulx)/pixel_spacing), \
                      int((uly - lry)/pixel_spacing), 1, gdal.GDT_Float32)
# Calculate the new geotransform
new_geo = ( ulx, pixel_spacing, geo_t[2], \
            uly, geo_t[4], -pixel_spacing )
# Set the geotransform
dest.SetGeoTransform( new_geo )
dest.SetProjection( osng.ExportToWkt() )
# Perform the projection/resampling
res = gdal.ReprojectImage( g, dest, \
                           wgs84.ExportToWkt(), osng.ExportToWkt(), \
                           gdal.GRA_Bilinear )
return dest
```

The function returns a GDAL in-memory file object, where you can `ReadAsArray` etc. As it stands, `reproject_dataset` does not write to disk. However, we can save the in-memory raster to any format supported by GDAL very conveniently by making a copy of the dataset. This takes a few lines of code:

CHAPTER
FIFTYTWO

WE NEED A LATLONG DATASET!!! SHOW GDALWARP USAGE FOR THIS

```
# Do in memory reprojection
reprojected_dataset = reproject_dataset ( "lc_h17v03_wgs84.tif" )
# Output driver, as before
driver = gdal.GetDriverByName ( "GTiff" )
# Create a copy of the in memory dataset `reprojected_dataset`, and save it
dst_ds = driver.CreateCopy( "test_lc_h17v03_OSNG.tif", reprojected_dataset, 0 )
dst_ds = None # Flush the dataset to disk

-595472.202548 1261034.77555 -55.2326775854
543532.18509 12933.1712342 -43.993001678
```

Let's see how the different projections look like by plotting them side by side

```
plt.subplot ( 1, 3, 1 )
plt.title ("MODIS sinusoidal")
plt.imshow ( modis, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.subplot ( 1, 3, 2 )
plt.imshow ( wgs84, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
g = gdal.Open("test_lc_h17v03_OSNG.tif" )
osng = g.ReadAsArray()
plt.title ("WGS84, Lat/Long")
plt.subplot ( 1, 3, 3 )
plt.imshow ( osng, interpolation='nearest', vmin=0, cmap=plt.cm.gist_earth )
plt.title("OSNG")

<matplotlib.text.Text at 0x2b9eac0b4ed0>

def css_styling():
    from IPython.display import display, HTML
    styles = "https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/notebooks/02%20-%20Probabilistic%20Inference.ipynb?raw=True"
    #return HTML(styles)
css_styling()
```


WORKING WITH VECTOR DATA: OGR

```
import json
import matplotlib
import os
import ogr
import osr
import gdal

fp = open("../bmh_matplotlibrc.json", 'r' )
s = json.load ( fp )
fp.close()
matplotlib.rcParams.update(s)
%pylab inline
%config InlineBackend.figure_format = 'svg'
figsize( 7,7 )

Populating the interactive namespace from numpy and matplotlib
```

53.1 Vector data

Sometimes, geospatial data is acquired and recorded for particular geometric objects such as polygons or lines. An example is a road layout, where each road is represented as a geometric object (a line, with points given in a geographical projection), with a number of added *features* associated with it, such as the road name, whether it is a toll road, or whether it is dual-carriageway, etc. This data is quite different to a raster, where the entire scene is tessellated into pixels, and each pixel holds a value (or an array of value in the case of multiband rasterfiles).

If you are familiar with databases, vector files are effectively a database, where one of the fields is a geometry object (a line in our previous road example, or a polygon if you consider a cadastral system). We can thus select different records by writing queries on the features. Some of these queries might be spatial (e.g. check whether a point is inside a particular country polygon).

The most common format for vector data is the **ESRI Shapfile**, which is a multifile format (i.e., several files are needed in order to access the data). We'll start by getting hold of a shapefile that contains the countries of the world as polygons, together with information on country name, capital name, population, etc. The file is available [here](#).

Figure 53.1: World

We will download the file with `wget` (or `curl` if you want to), and uncompress it using `unzip` in the shell:

```
# Downloads the data using wget
!wget http://aprsworld.net/gisdata/world/world.zip
# or if you want to use curl...
#! curl http://aprsworld.net/gisdata/world/world.zip -o world.zip
!unzip -o -x world.zip
```

We need to import `ogr`, and then open the file. As with GDAL, we get a handler to the file, (`g` in this case). OGR files can have different layers, although Shapefiles only have one. We need to select the layer using `GetLayer(0)` (selecting the first layer).

```
from osgeo import ogr

g = ogr.Open( "world.shp" )
layer = g.GetLayer( 0 )
```

In order to see a field (the field `NAME`) we can loop over the features in the layer, and use the `GetField('NAME')` method. We'll only do ten features here:

```
n_feat = 0
for feat in layer:

    print feat.GetField('NAME')

    n_feat += 1
    if n_feat == 10:
        break

-----
NameError                                         Traceback (most recent call last)

<ipython-input-2-f4731880d34d> in <module>()
      1
      2 n_feat = 0
----> 3 for feat in layer:
      4
      5     print feat.GetField('NAME')

NameError: name 'layer' is not defined
```

If you wanted to see the different layers, we could do this using:

```
layerDefinition = layer.GetLayerDefn()

for i in range(layerDefinition.GetFieldCount()):
    print "Field %d: %s" % ( i+1, layerDefinition.GetFieldDefn(i).GetName() )
```

Each feature, in addition to the fields shown above, will have a `Geometry` field. We get a handle to this using the `GetGeometryRef()` method. Geometries have many methods, such as `ExportToKML()` to export to KML (Google Maps/Earth format):

```
the_geometry = feat.GetGeometryRef()
the_geometry.ExportToKML()
```

Many of the methods that don't start with `__` are interesting. Let's see what these are. typically, the interesting methods start with an upper case letter, so we'll only show those:

```
for m in dir( the_geometry ):
    if m[0].isupper():
        print m
```

You'll notice that many of these mechanisms e.g. `Overlaps` or `Touches` are effectively geoprocessing operations (they operate on geometries and return True if one geometry overlaps or touches, respectively, the other). Other operations, such as `Buffer` return a buffered version of the same geometry. This allows you to actually do fairly complicated geoprocessing operations with OGR. However, if you want to do geoprocessing in earnest, you should really be using `Shapely`.

A particularly useful webpage for this section is [available in the OGR cookbook](#). Have a look through that if you want more in depth information.

53.2 Selecting attributes and/or data extents

OGR provides an easy way to select attributes on a given layer. This is done using a SQL-like syntax (you can read more on OGR's SQL subset [here](#). The main point is that the *attribute filter* is applied to a complete layer. For example, let's say that we want to select only countries with a population (field APPROX) larger than 90 000 000 inhabitants:

```
g = ogr.Open( "world.shp" )
lyr = g.GetLayer( 0 )
lyr.SetAttributeFilter( "APPROX > 90000000" )
for feat in lyr:
    print feat.GetFieldAsString( "NAME" ) + " has %d inhabitants" % \
        feat.GetFieldAsInteger("APPROX")
```

```
PAKISTAN has 123490000 inhabitants
JAPAN has 124710000 inhabitants
RUSSIAN FEDERATION has 150500000 inhabitants
INDIA has 873850000 inhabitants
BANGLADESH has 120850000 inhabitants
BRAZIL has 159630000 inhabitants
NIGERIA has 91700000 inhabitants
CHINA has 1179030000 inhabitants
INDONESIA has 186180000 inhabitants
JOHNSTON ATOLL has 256420000 inhabitants
KINGMAN REEF - PALMYRA ATOLL has 256420000 inhabitants
UNITED STATES has 256420000 inhabitants
```

So we get a list of populous countries (note that Johnston Atoll and Palmyra are part of the US, and report the sample population as the US!)

An additional way to filter the data is by geographical extent. Let's say we wanted a list of all the countries in (broadly speaking) Europe, *i.e.* a geographical extent in longitude from 14W to 37E, and in latitude from 72N to 38N. We can use SetSpatialFilterRect to do this:

```
g = ogr.Open( "world.shp" )
lyr = g.GetLayer( 0 )
lyr.SetSpatialFilterRect( -14, 37, 38, 72)
for feat in lyr:
    print feat.GetFieldAsString( "NAME" ) + " ---- " + feat.GetFieldAsString( "CAPITAL" )

ALGERIA ---- ALGIERS
BELGIUM ---- BRUSSELS
LUXEMBOURG ---- LUXEMBOURG
SAN MARINO ---- SAN MARINO
AUSTRIA ---- VIENNA
CZECH REPUBLIC ---- PRAGUE
SLOVENIA ---- LJUBLJANA
HUNGARY ---- BUDAPEST
SLOVAKIA ---- BRATISLAVA
YUGOSLAVIA ---- BELGRADE [BEOGRADE]
BOSNIA AND HERZEGOVINA ---- SARAJEVO
ALBANIA ---- TIRANE
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC ---- SKOPJE
LITHUANIA ---- VILNIUS
LATVIA ---- RIGA
BULGARIA ---- SOFIA
BELARUS ---- MINSK
MOLDOVA, REPUBLIC OF ---- KISHINEV
IRELAND ---- DUBLIN
ICELAND ---- REYKJAVIK
SPAIN ---- MADRID
SWEDEN ---- STOCKHOLM
FINLAND ---- HELSINKI
```

TURKEY ---- ANKARA
RUSSIAN FEDERATION ---- MOSCOW
GREECE ---- ATHENS
PORTUGAL ---- LISBON
POLAND ---- WARSAW
NORWAY ---- OSLO
GERMANY ---- BERLIN
ESTONIA ---- TALLINN
TUNISIA ---- TUNIS
CROATIA ---- ZAGREB
ROMANIA ---- BUCURESTI
UKRAINE ---- KIEV
NETHERLANDS ---- AMSTERDAM
JERSEY ---- SAINT HELIER
GUERNSEY ---- SAINT PETER PORT
FAROE ISLANDS ---- TORSHAVN
DENMARK ---- COPENHAGEN
MONACO ---- MONACO
ANDORRA ---- ANDORRA LA VELLA
LIECHTENSTEIN ---- VADUZ
SWITZERLAND ---- BERN
ISLE OF MAN ---- DOUGLAS
UNITED KINGDOM ---- LONDON
FRANCE ---- PARIS
VATICAN CITY (HOLY SEE) ---- VATICAN CITY
ITALY ---- ROME

53.3 Saving a vector file

Saving a vector file using OGR requires a number of steps:

1. Definition of the format
2. Definition of the layer projection and geometry type (e.g. lines, polygons...)
3. Definition of the data type of the different fields
4. Creation of a feature, population of the different fields, and setting a geometry
5. Addition of the feature to the layer
6. Destruction of the feature

This appears quite involved, but let's see how this works. Note that when you generate a new vector file, OGR will fail if the file already exists. You might want to use `os.remove()` to get rid of the file if it exists.

Let's see how this is done with an example which is a snippet that creates a GeoJSON file with the location of the different national parks. GeoJSON is a nice geographic format, and [github](#) allows you to display it easily as a map.

```
# National park information, separated by TABs

parks = """Dartmoor national park\t-3.904\t50.58
New forest national park\t-1.595\t50.86
Exmoor national park\t-3.651\t51.14
Pembrokeshire coast national park\t-4.694\t51.64
Brecon beacons national park\t-3.432\t51.88
Pembrokeshire coast national park\t-4.79\t51.99
Norfolk and suffolk broads\t1.569\t52.62
Snowdonia national park\t-3.898\t52.9
Peak district national park\t-1.802\t53.3
Yorkshire dales national park\t-2.157\t54.23
North yorkshire moors national park\t-0.8855\t54.37
```

```

Lake district national park\t-3.084\t54.47
Galloway forest park\t-4.171\t54.87
Northumberland national park\t-2.228\t55.28
Loch lomond and the trossachs national park\t-4.593\t56.24
Tay forest park\t-4.025\t56.59
Cairngorms national park\t-3.545\t57.08"""

# See if the file exists from a previous run of this snippet
if os.path.exists ( "parks.json" ):
    # It does exist, so remove it
    os.remove ( "parks.json" )

# We need the output projection to be set to Lat/Long
latlong = osr.SpatialReference()
latlong.ImportFromEPSG( 4326 )

# Invoke the GeoJSON driver
drv = ogr.GetDriverByName( 'GeoJSON' )
# This is the output filename
dst_ds = drv.CreateDataSource( 'parks.json' )
# This is a single layer dataset. The layer needs to be of points
# and needs to have the WGS84 projection, which we defined above
dst_layer = dst_ds.CreateLayer('', srs =latlong , \
                                geom_type=ogr.wkbPoint )

# We just need a field with the Park's name, and its type is a String
field_defn=ogr.FieldDefn( 'name', ogr.OFTString )
dst_layer.CreateField( field_defn )

# Algorithm is as follows:
# 1. Loop over lines
# 2. Split line into park name, longitude, latitude
# 3. Create WKT of the point
# 4. Set the attribute name to name of park
# 5. Clean up

for park_id, line in enumerate( parks.split( "\n" ) ):
    # Get the relevant information
    park_name, lon, lat = line.split("\t")
    # Create a geographical representation of the current park
    wkt = "POINT ( %f %f )" % ( float(lon), float(lat) )
    # Create a feature, using the attributes/fields that are
    # required for this layer
    feat = ogr.Feature(feature_def=dst_layer.GetLayerDefn())
    # Feed the WKT into a geometry
    p = ogr.CreateGeometryFromWkt( wkt )
    # Feed the geometry into a WKT
    feat.SetGeometryDirectly( p )
    # Set the name field to its value
    feat.SetField ( "name", park_name )
    # Attach the feature to the layer
    dst_layer.CreateFeature( feat )
    # Clean up
    feat.Destroy()

# Close file
dst_ds = None

```

You can see the result of this on [github](#).

Additionally, note that if we had defined a coordinate transformation as in the raster session, we could apply this transformation to an OGR geometry entity (in the snippet above, `p` would be such), and it would be reprojected.

Exercise Modify the above snippet to output a GeoJSON file for the Peak District National Park, whose UTM30N (EPSG code: 32630) co-ordinates are 577659, 5911841.

53.4 Rasterising

A very frequent problem one finds is how to mask out an area in a raster file that is defined as polygon in a shapefile. For example, if you have a raster of the world's population density, and you want to extract all the pixels that belong to one particular country, how do you go about that? One way around this is to *rasterise* the polygon(s), which translates into “burning” pixels that fall within the polygon with a number, resulting in a mask.

The way to do this is to use GDAL's `RasterizeLayer` method. The method takes a handle to a GDAL dataset (one that you create yourself, with the right projection and geotransform, as you've seen above), and a OGR layer. The syntax for `RasterizeLayer` is

```
err = gdal.RasterizeLayer ( raster_ds, [raster_band_no], ogr_layer, burn_values=[burn_val] )
```

where `raster_ds` is the GDAL raster datasource (note that it needs to be georeferenced, *i.e.* it requires projection and geotransform), `raster_band_no` is the band of the GDAL dataset where we want to burn pixels, `ogr_layer` is the vector layer object, and `burn_val` is the value that we want to burn.

Let's use `gdal.RasterizeLayer` in conjunction with all that we have covered above. Say we want to create a mask that only selects the UK or Ireland in `world.shp`, and we want to apply this mask to the MODIS landcover product that we used in the GDAL session (`h17v03.tif`), file `lc_h17v03.tif`. We find that in this case, `world.shp` is in longitude/latitude, and the MODIS data is in the MODIS projection, so we will reproject the vector data to match the MODIS data (so the latter is not interpolated and artifacts introduced). To make this efficient and avoid saving to disk, we shall use *in-memory vector and rasters*, and we will output a numpy array as our mask. Note then the steps:

1. Create the projection conversion object (as for GDAL before)
2. Create an in memory **raster** dataset to store the mask, using `lc_h17v03.tif` as a reference for geotransforms, array size and projection.
3. Create an in memory **vector** dataset to hold the features that will be reprojected
4. Open `world.shp` and apply an `AttributeFilter` to select a country
5. Select a geometry from `world.shp`, project it and store it in the destination in memory vector layer
6. Once this is done, use `gdal.RasterizeLayer` with both in-memory raster and vector datasets
7. Read the in memory raster into an array

This is a particularly good exercise that will stress all that we have learned so far.

```
reference_filename = "lc_h17v03.tif"
target_vector_file = "world.shp"
attribute_filter = "NAME = 'IRELAND'"
burn_value = 1

# First, open the file that we'll be taking as a reference
# We will need to gleam the size in pixels, as well as projection
# and geotransform.

g = gdal.Open( reference_filename )

# We now create an in-memory raster, with the appropriate dimensions
drv = gdal.GetDriverByName('MEM')
target_ds = drv.Create('', g.RasterXSize, g.RasterYSize, 1, gdal.GDT_Byte)
target_ds.SetGeoTransform( g.GetGeoTransform() )

# We set up a transform object as we saw in the previous notebook.
# This goes from WGS84 to the projection in the reference datasets
```

```
wgs84 = osr.SpatialReference( ) # Define a SpatialReference object
wgs84.ImportFromEPSG( 4326 ) # And set it to WGS84 using the EPSG code

# Now for the target projection, Ordnance Survey's British National Grid
to_proj = osr.SpatialReference() # define the SpatialReference object
# In this case, we get the projection from a Proj4 string

# or, if using the proj4 representation
to_proj.ImportFromWkt( g.GetProjectionRef() )
target_ds.SetProjection ( to_proj.ExportToWkt() )
# Now, we define a coordinate transformation object, *from* wgs84 *to* OSNG
tx = osr.CoordinateTransformation( wgs84, to_proj )

# We define an output in-memory OGR dataset
# You could also do select a driver for an eg "ESRI Shapefile" here
# and give it a sexier name than out!

drv = ogr.GetDriverByName( 'Memory' )
dst_ds = drv.CreateDataSource( 'out' )
# This is a single layer dataset. The layer needs to be of polygons
# and needs to have the target files' projection
dst_layer = dst_ds.CreateLayer('', srs = to_proj, geom_type=ogr.wkbPolygon )

# Open the original shapefile, get the first layer, and filter by attribute
vector_ds = ogr.Open( target_vector_file )
lyr = vector_ds.GetLayer ( 0 )
lyr.SetAttributeFilter( attribute_filter )

# Get a field definition from the original vector file.
# We don't need much more detail here
feature = lyr.GetFeature(0)
field = feature.GetFieldDefnRef( 0 )
# Apply the field definition from the original to the output
dst_layer.CreateField( field )
feature_defn = dst_layer.GetLayerDefn()
# Reset the original layer so we can read all features
lyr.ResetReading()
for feat in lyr:
    # For each feature, get the geometry
    geom = feat.GetGeometryRef()
    # transform it to the reference projection
    geom.Transform ( tx )
    # Create an output feature
    out_geom = ogr.Feature ( feature_defn )
    # Set the geometry to be the reprojected/transformed geometry
    out_geom.SetGeometry ( geom )
    # Add the feature with its geometry to the output yaer
    dst_layer.CreateFeature(out_geom)
    # Clear things up
    out_geom.Destroy
    geom.Destroy
# Done adding geometries
# Reset the output layer to the 0th geometry
dst_layer.ResetReading()

# Now, we rasterize the output vector in-memory file
# into the in-memory output raster file

err = gdal.RasterizeLayer(target_ds, [1], dst_layer,
                          burn_values=[burn_value])
if err != 0:
    print("error:", err)
```

```
# Read the data from the raster, this is your mask
data = target_ds.ReadAsArray()

# Plotting to see whether this makes sense.

ndata = g.ReadAsArray()
plt.imshow ( ndata, interpolation='nearest', cmap=plt.cm.gray, vmin=0, vmax=1, alpha=0.3 )
plt.hold ( True )

plt.imshow ( data, interpolation='nearest', cmap=plt.cm.gray, alpha=0.7 )
plt.grid ( False )
plt.show()
```

53.5 Using matplotlib to plot geometries

Using matplotlib to plot geometries from OGR can be quite tedious. Here's an example of plotting a map of Angola from the `world.shp`. In the same vein of recommending Shapely and Fiona above for serious geoprocessing of vector data, you are encouraged to use `descartes` for plotting vector data!

```
import matplotlib.path as mpath
import matplotlib.patches as mpatches

# Extract first layer of features from shapefile using OGR
ds = ogr.Open('world.shp')
lyr = ds.GetLayer(0)

# Prepare figure
plt.ioff()
plt.subplot(1,1,1)
ax = plt.gca()

paths = []
lyr.ResetReading()

lyr.SetAttributeFilter ( " NAME = 'ANGOLA' " )
ax.set_xlim(11, 24.5)
ax.set_ylim(-20, -2)
# Read all features in layer and store as paths

for feat in lyr:

    for geom in feat.GetGeometryRef():
        envelope = np.array( geom.GetEnvelope() )
        # check if geom is polygon
        if geom.GetGeometryType() == ogr.wkbPolygon:
            codes = []
            all_x = []
            all_y = []
            for i in range(geom.GetGeometryCount()):
                # Read ring geometry and create path
                r = geom.GetGeometryRef(i)
                x = [r.GetX(j) for j in range(r.GetPointCount())]
                y = [r.GetY(j) for j in range(r.GetPointCount())]
                # skip boundary between individual rings
                codes += [mpath.Path.MOVETO] + \
```

```
(len(x)-1) * [mpath.Path.LINETO]
all_x += x
all_y += y
path = mpath.Path(np.column_stack((all_x, all_y)), codes)
paths.append(path)

# Add paths as patches to axes
for path in paths:
    patch = mpatches.PathPatch(path, \
        facecolor='0.8', edgecolor='black')
    ax.add_patch(patch)

ax.set_aspect(1.0)
plt.show()
```

CHAPTER
FIFTYFOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*