1. Segmentation
   - Running in the Terminal
   - Scaling the Segmentation
   - Calling from within Python

2. Classification: Rule Based
   - Raster Attribute Tables (RAT)
   - Populating the RAT
   - Undertake Classification

3. Change Detection
   - Export 1996
   - Classify 2007
   - Classify 2010

4. Classification: Machine learning
About Us

- **Pete Bunting** (pfb@aber.ac.uk)
  - Senior Lecturer at Aberystwyth University
  - Head of Earth Observation and Ecosystem Dynamics Group at AU.

- **Dan Clewley** (clewley@usc.edu)
  - Post Doctoral Research Associate at the University of Southern California.
  - Working with Mahta Moghaddam in the Microwave Systems, Sensors and Imaging Lab (MiXIL).
  - Ph.D. supervised by Richard Lucas 2008 - 2011
Topics covered

- Some of the utility programs included with GDAL
- Data visualisation using TuiView
- Creating scripts with Python
- The Remote Sensing and GIS Software Library (RSGISLib)
  - Pre-processing
  - Image segmentation
  - Classification
  - Change detection
Series of worked examples.

- We’ll provide an overview of each script then you’ll be given time to run.
- Highlights of scripts covered in slides.
- Complete scripts, with comments (lines starting with #), in notes and with course material.
- Brief description of commands in notes - links to more detailed explanation.
Computing Environment

- Virtual Linux machine running xubuntu
- Data, scripts, worksheet and presentations in RSGISLibCourse, within home folder.
- Copy scripts to data folder and run from within there.
Yesterday we covered:

- Some of the utility programs included with GDAL
- The RSGISLib Python interface and a selection of commands
- Data visualisation using TuiView
Today we will cover:

- Image segmentation
- Classification
- Change detection
- More advanced Python
Script development

- Develop process on single image.
- Test on single image
- Search for images (glob.glob('*.kea'))
- Run on all images
Open a terminal window and change to the directory the data is stored by typing the following:

```bash
cd ~/RSGISLibCourse/Data
```
Run Shepherd et al., Segmentation

- Using RSGISLib, the segmentation algorithm of Shepherd et al. [2014] can either be executed via the terminal or through the python binding.
  - We will do both.
- From the terminal the following command is used:

  rsgislibsegmentation.py
rsgislibsegmentation.py

Run the segmentation

```bash
rsgislibsegmentation.py -i N06W053_96-10_stack_lee_dB.kea \n-o N06W053_96-10_segs.kea -m N06W053_96-10_meansegs.kea \n-t ./tmp/ -k 30 -n 50 -d 1000000
```
Viewing the Segmentation in TuiView

tuiview --separate N06W053_96-10_segs.kea N06W053_96-10_meansegs.kea

- Don’t forget to stretch the mean image!
Shepherd et al., Segmentation

**Segmentation Flowchart**

- **2.1. Seeding (KMeans/ISOData)**
- **2.1. Image Sub-sampling**
- **2.2. Pixel Labelling and Clumping**
- **2.3. Elimination**
- **2.4. Relabelling**

- Seeded using a KMeans clustering \((k)\).
- Iterative eliminates (from small to large) segments which are below a size threshold \((n)\).
Try other parameters for $k$ and $n$

```
rsgislibsegmentation.py -i N06W053_96-10_stack_lee_dB.kea \ 
-o N06W053_96-10_segs.kea -m N06W053_96-10_meansegs.kea \ 
-t ./tmp/ -k 10 -n 100 -d 1000000
```
Scaling the Segmentation

• The Canterbury (New Zealand) regional mosaic comprised of 36 SPOT5 scenes.
  • $36533 \times 35648$ pixels
• Processed using 12 GB of RAM in approximately 3 hours on a 3 GHz x86 processor.
  • producing 1,222,885 segments
Currently working with John Armston (Queensland) to finalise a tiled version of the algorithm.

Application to Australia, ALOS PALSAR and Landsat FPC composite.
  - $228354 \times 116735$ pixels (54 Gb image, compressed)

Commands in RSGISLib but still work in progress...
Scaling the Segmentation - Tiling

Segmentation Tiling stitching

Pete Bunting and Daniel Clewley
The Remote Sensing and GIS Software Library
Scaling the Segmentation - Tiling

ALOS PALSAR and Landsat FPC mosaic
Scaling the Segmentation - Tiling

Segmentation of Australia

- 33,778,634 clumps $k = 30$ $n = 100$ pixels
Scaling the Segmentation - Tiling

Validating the segmentation parameters
Using Python as ‘the glue’ allows many algorithms to be joined together and run on multiple datasets.

Python Segmentation Function

```python
segutils.runShepherdSegmentation(inputImage, segmentClumps, outputMeanSegments, tmpPath, "KEA", False, False, False, numClusters, minObjectSize, distThres, None)
```
Run Segmentation in Python

Copy the python script to the data directory

```
cp ../Scripts/5_segmentation.py ./.
```

- Open `5_segmentation.py` and review the code.
  - Questions?

Run the segmentation script

```
python 5_segmentation.py
```
Classification Process

Classification to be Produced

![Classification Image]
Classification Process

1. Segmentation (from above)
2. Populating backscatter statistics.
3. Calibrate backscatter statistics to dB.
4. Classify water (including a minimum region size).
5. Calculate a per-pixel proximity to the classify water regions.
6. Populate the segment clumps with water proximity.
7. Classify the scene into broad categories (Water (Ocean), Coastal Strip, Other)
8. Classify the coastal zone to identify mangroves.
Classification Process

Rule based Classification Workflow

1. Segmentation
2. Populate Attribute Table
3. Apply Classifier
4. Clumping

Pete Bunting and Daniel Clewley
The Remote Sensing and GIS Software Library
Classification: paradigm

- Create a large table (matrix) of data associated with segments
  - Each row is associated with a segment / clump / object / feature.
- Using the row of information answer: ‘which class does this feature belong to?’
- The table of information is stored within the image file.
  - A raster attribute table (RAT).
What is a raster attribute table?

- Raster clumps are a method of representing non-overlapping polygons.
  - Each polygon has unique pixel value which identifies it.
- The unique pixel value is used to index the attribute table rows.
- Attribute tables support 3 data types:
  - Integer
  - Float
  - String
- Rasters are very scalable as they are inherently spatially index.
- ‘Point in Polygon’ problem is also simple.
What is a raster attribute table?

RAT

<table>
<thead>
<tr>
<th>FID</th>
<th>Var1</th>
<th>Var2</th>
<th>Var3</th>
<th>Var4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>54.3</td>
<td>0.1</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>56.4</td>
<td>0.25</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>52.1</td>
<td>0.01</td>
<td>220</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>51.3</td>
<td>0.6</td>
<td>330</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>58.6</td>
<td>0.466</td>
<td>280</td>
</tr>
<tr>
<td>6</td>
<td>34</td>
<td>62.3</td>
<td>0.2</td>
<td>230</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>51.3</td>
<td>0.3</td>
<td>150</td>
</tr>
</tbody>
</table>
How to create a RAT?

- The image file format you are using needs to support RATs.
  - Either KEA or Erdas Imagine HFA.
- When using RSGISLib a RAT is built when it’s needed so doesn’t need to explicitly created.
- However, the image needs to be defined as a ‘thematic’ image.
  - The segmentation code in rsgislib automatically does this.
How to add information columns to the RAT?

- RSGISLib providing a number of functions for populating the attribute tables with information.
  - [http://www.rsgislib.org/rsgislib_rastergis.html#attribute-segments](http://www.rsgislib.org/rsgislib_rastergis.html#attribute-segments)
- Options include values from other images (e.g., backscatter), shape and relationships to neighbours
- `populateRATWithStats` is the function used in this worksheet.
  - It can provide columns for min, max, mean and standard deviation of an input image (e.g., backscatter) for each segment.
  - Think zonal statistics (but not the Zonal Stats module)
  - The input image needs to be the same pixel resolution as the clumps.
How to add information columns to the RAT?

```python
from rsgislib import rastergis

def populateSegmentsMeanStdDev(inputImage, clumpsImg, colNamePrefix):
    bandStats = []  # Create an empty list
    bandStats.append(
        rastergis.BandAttStats(band=1,
                                meanField=colNamePrefix+'Mean',
                                stdDevField=colNamePrefix+'StdDev'))

    rastergis.populateRATWithStats(inputImage, clumpsImg, bandStats)
```
How to add information columns to the RAT?

```python
segClumps = "N06W053_96-10_segs.kea"
imgs = ["N06W053_JERS1_96_HH_utm_sub.kea",
       "N06W053_PALSAR_07_HH_utm_sub.kea",
       "N06W053_PALSAR_10_HH_utm_sub.kea"]
namesPrefix = ["HH96", "HH07", "HH10"]
umImgs = 3

for i in range(numImgs):
    populateSegmentsMeanStdDev(imgs[i], segClumps, namesPrefix[i])
```
Copy the python script to the data directory

```bash
cp ../Scripts/6_populatestats.py ./.
```

- Open `6_populatestats.py` and review the code.
- Questions?

Run the populate statistics script

```bash
python 6_populatestats.py
```
Due to the \textit{log} transform in the calibration this is applied after the mean and standard deviation have been calculated.

The RIOS library is used to read the columns and the numpy library used to undertake the calculation.

```python
ratDataset = gdal.Open(clumpsImg, gdal.GA_Update)
col = rat.readColumn(ratDataset, inColName)
# Apply the calibration formula (20 * log10(DN) + \textit{c})
col = numpy.where(col > 0, 20 * numpy.log10(col) + \textit{const}, 0)
rat.writeColumn(ratDataset, outColName, col)
ratDataset = None
```
Loop through the columns

dnNames = ["HH96Mean", "HH07Mean", "HH10Mean",
            "HH96StdDev", "HH07StdDev", "HH10StdDev"]

dBNames = ["HH96MeandB", "HH07MeandB", "HH10MeandB",
            "HH96StdDevdB", "HH07StdDevdB", "HH10StdDevdB"]

dBConst = [-84.66, -83.0, -83.0, -84.66, -83.0, -83.0]

numCols = 6

for i in range(numCols):
    convertColumnTodB(segmentClumps, dnNames[i],
                        dBNames[i], dBConst[i])
Run Calibrate Statistics

Copy the python script to the data directory

```
cp ../Scripts/7_calibrateTodBs.py ./.
```

- Open `7_calibrateTodBs.py` and review the code.
- Questions?

Run the calibration script

```
python 7_calibrateTodBs.py
```

Pete Bunting and Daniel Clewley
The Remote Sensing and GIS Software Library
Open in TuiView and Query the RAT
Classifying Water from 1996 JERS-1.

The classification of water is undertaken in 3 steps:

1. Classify segments with a dB $< -12$ as ‘Water’
2. Clump and remove small regions of ‘Water’ $< 1000$ pixels
3. Populate the RAT with the final classification

Finally, the classified segments are coloured dark blue to identify them as water.
1) Classify segments with a dB $<-12$ as ‘Water’

```python
import osgeo.gdal as gdal
from rios import rat
import numpy

ratDataset = gdal.Open( clumpsImg, gdal.GA_Update )
HH96MeandB = rat.readColumn(ratDataset, "HH96MeandB")
Water96 = numpy.zeros_like(HH96MeandB, dtype=numpy.int8)
Water96 = numpy.where((HH96MeandB < -12), 1, Water96)
Water96[0] = 0
rat.writeColumn(ratDataset, "Water96", Water96)
ratDataset = None
```
2) Clump and remove regions of ‘Water’ < 1000 pixels

```python
# Export water mask as binary image
rastergis.exportCols2GDALImage(clumpsImg,
    "N06W053WaterMask.kea",
    "KEA", rsgislib.TYPE_8UINT,
    ["Water96"])

# Clump mask to find connected regions
segmentation.clump("N06W053WaterMask.kea",
    "N06W053WaterMaskClumps.kea",
    "KEA", False, 0)

# Populate clumps with stats - for Histogram
rastergis.populateStats("N06W053WaterMaskClumps.kea", True, False)

# Remove clumps with a size less than 1000 pixels.
segmentation.rmSmallClumps("N06W053WaterMaskClumps.kea",
    "N06W053WaterMaskClumpsNoSmall.kea",
    1000, "KEA")
```
Removal of Features < 1000 pixels
3) Populate the RAT with the final classification

```python
bandStats = []
bandStats.append(rastergis.BandAttStats(band=1,
                                        minField='WaterClpsMin'))
rastergis.populateRATWithStats("N06W053WaterMaskClumpsNoSmall.kea", clumpsImg, bandStats)
ratDataset = gdal.Open( clumpsImg, gdal.GA_Update )
# Open the column of clump ID's
WaterClpsMin = rat.readColumn(ratDataset, "WaterClpsMin")
Water96[...] = 0
# Classify segments which are within the 1000 pxl clumps as water
Water96 = numpy.where((WaterClpsMin > 0), 1, Water96)
Water96[0] = 0
rat.writeColumn(ratDataset, "Water96", Water96)
```
Run Water Classification

Copy the python script to the data directory

```
cp ../Scripts/8_classifyWater.py ./.
```

- Open `8_classifyWater.py` and review the code.
  - Questions?

Run the classify water script

```
python 8_classifyWater.py
```
Resulting Water Classification
Calculate Proximity to Water

- A key component of classify mangroves is context, e.g., proximity of segments to the classified water mask.
- The GDAL proximity function will be used for this.

Create new image

```python
# Export the water mask to an image.
rastergis.exportCols2GDALImage(clumpsImg, waterMask, "KEA",
                                rsgislib.TYPE_8UINT, ["Water96"])

# Use RSGISLib to create a blank image for the proximity functions output.
imageutils.createCopyImage(waterMask, prox2Water, 1, 0.0, "KEA",
                           rsgislib.TYPE_32FLOAT)
```
Calculate Proximity to Water

```python
waterMaskDS = gdal.Open(waterMask, gdal.GA_ReadOnly)
waterMaskBand = waterMaskDS.GetRasterBand(1)
waterProxDS = gdal.Open(prox2Water, gdal.GA_Update)
waterProxBand = waterProxDS.GetRasterBand(1)
options = []
options.append( 'MAXDIST=30000' )
options.append( 'VALUES=1' )
options.append( 'DISTUNITS=GEO' )
gdal.ComputeProximity(waterMaskBand, waterProxBand, options, callback=gdal.TermProgress)

waterMaskDS = None
waterProxDS = None
```
Run Calculate Proximity to Water

Copy the python script to the data directory

```bash
cp ../Scripts/9_calcWaterProximity.py ./
```

- Open `9_calcWaterProximity.py` and review the code.
- Questions?

Run the proximity to water script

```bash
python 9_calcWaterProximity.py
```
Resulting Proximity to Water Surface
Run Populate Distance to Water Statistics

Copy the python script to the data directory

cp ..:/Scripts/10_populateClumpWaterProximity.py ./.

- Open 10_populateClumpWaterProximity.py and review the code.
  - Questions?

Run the populate proximity to water statistics script

python 10_populateClumpWaterProximity.py
Classify Broad Categories

A useful step is to build a hierarchical classification scheme where context or other information is used to define a set of broad categories or classes.

- Water
- Coastal Strip (< 3 km from coast)
- Other
Classify Broad Categories: Using strings as classes column

```python
Water96ProxMin = rat.readColumn(ratDataset, "Water96ProxMin")
Water96 = rat.readColumn(ratDataset, "Water96")
Category = numpy.empty_like(Water96ProxMin, dtype=numpy.dtype('a255'))
# Initialise the column to all have the value 'NA'.
Category[:] = "NA"
# Use the water mask to assign the water region
Category = numpy.where((Water96 == 1), "Water", Category)
# Define a simple coastal strip using a 3 km threshold
Category = numpy.where(((Water96 == 0) & (Water96ProxMin<3000)), "Coastal Strip", Category)
Category = numpy.where((Category == "NA".encode()), "Other", Category)
# Assign the no data region back to 'NA'
Category[0] = "NA"
rat.writeColumn(ratDataset, "Category", Category)
```
Run Broad Categories Classification

Copy the python script to the data directory

```bash
cp ../Scripts/11_classifyBroadCategories.py ./.
```

- Open `11_classifyBroadCategories.py` and review the code.
  - Questions?

Run the broad categories classification script

```bash
python 11_classifyBroadCategories.py
```
Resulting Broad Categories Classification
To identify the mangrove regions the coastal strip will be classified.

classes = numpy.empty_like(HH96MeandB, dtype=numpy.dtype('a255'))
classes[...] = "NA"
classes = numpy.where(((Category == "Coastal Strip".encode()) &
(HH96MeandB < -10) & (Water96ProxMin < 100)),
"Water", classes)
classes = numpy.where(((Category == "Coastal Strip".encode()) &
(HH96MeandB > -10) & (Water96ProxMin < 1200)),
"Mangroves", classes)
classes = numpy.where(((Category == "Coastal Strip".encode()) &
(HH96MeandB > -8)), "Mangroves", classes)
rat.writeColumn(ratDataset, "classes", classes)
Run Coastal Strip Classification

Copy the python script to the data directory

```
cp ../Scripts/12_ClassifyCoastalZone.py ./
```

- Open `12_ClassifyCoastalZone.py` and review the code.
- Questions?

Run the segmentation script

```
python 12_ClassifyCoastalZone.py
```
Final Classification: TuiView
Final Classification: Export to Google Earth

Using `gdal_translate`

```
gdal_translate -of KMLSUPEROVERLAY -expand rgba \ 
N06W053_96-10_segs.kea N06W053_96-10_class.kmz
```
Final Classification: Export to Google Earth
Combining as a Single Script

The individual scripts which have been used to undertake this classification can easily be combined into a single python script which run through all the steps.

Script with the process combined

```
../Scripts/13_rb_classification.py
```
Change Detection

Change detection in this context is the process of:

1. Identifying segments in the existing segmentation/classification which are change candidates when compared to a new image.

2. Classifying the change candidates using the new image.

The advantage of this method is that it is taking advantage of the existing classification and simply updating the classification rather than attempting to reclassify the whole scene.
Change Detection: Identifying change candidates

Using the new image data and existing classification the following steps are undertaken:

1. Calculate the mean and standard deviation of the image values for each class of interest from the new data.

2. Use a number of standard deviations from the mean to define thresholds used to identify change features.

3. Apply the thresholds to the segments identifying the candidate change features.
Change Detection: Classifying change candidates

To classify the candidate change features a number of assumptions can be made, as the previous class is known then it is likely that the change feature will only have a small number of trajectories to other classes. For example, in this processing we are only considering 2 change types:

- Mangroves to Water
- Water to Mangroves
Before running the change process to generate the 2007 classification the 1996 classification is exported to allow comparison and visualisation at a later point.

RSGISLib contains a function to collapse an attribute table of segments to a ‘classification’, where each row in the attribute refers to a single class and all pixels classified as a class have the same unique value.
Export 1996 the Classification

# Export the 1996 classification as an independent image
# with each class a single row in the attribute table
classification.collapseClasses("N06W053_96-10_segs.kea",
    "N06W053_1996Classification.kea",
    "KEA", "classes")

# Calculate statistics for the classification and
# populate with a histogram.
rastergis.populateStats("N06W053_1996Classification.kea",
    False, True)
Run Export 1996 Classification

Copy the python script to the data directory

```
cp ../Scripts/14_Collapse1996Class.py ./.
```

- Open `14_Collapse1996Class.py` and review the code.
- Questions?

Run the export 1996 classification script

```
python 14_Collapse1996Class.py
```
1996-2007 Change Detection: Identify the Change Candidates

```python
ChangeFeat = collections.namedtuple('ChangeFeats', ['name', 'outName', 'threshold'])
changeFeatVals = []
# Threshold of 3 Standard Deviations
changeFeatVals.append(ChangeFeat(name="Water", outName=1, threshold=3.0))
# Threshold of 3 Standard Deviations
changeFeatVals.append(ChangeFeat(name="Mangroves", outName=2, threshold=3.0))
# Run change detection
rastergis.findChangeClumpsFromStdDev(clumpsImage, "classes", "ChangeFeats9607", ["HH07MeandB"], changeFeatVals)
```
1996-2007 Change Detection: Classify the Change Candidates

```python
ChangeFeats9607 = rat.readColumn(ratDataset, "ChangeFeats9607")
classes = rat.readColumn(ratDataset, "classes")
HH07MeandB = rat.readColumn(ratDataset, "HH07MeandB")
Classes2007 = numpy.empty_like(classes, dtype=numpy.dtype('a255'))
Classes2007 = classes
# Classify the change features within 1996 water mask as mangroves
Classes2007 = numpy.where(((ChangeFeats9607 == 1)&(HH07MeandB > -11)),
                           "Mangroves", Classes2007)
# Classify the change features within the 1996 mangroves mask as water
Classes2007 = numpy.where(((ChangeFeats9607 == 2)&(HH07MeandB < -10)),
                           "Water", Classes2007)
rat.writeColumn(ratDataset, "Classes2007", Classes2007)
```
Run 1996-2007 Change Detection

Copy the python script to the data directory

cp ../Scripts/15_classifyChange9607.py ./.

- Open 15_classifyChange9607.py and review the code.
  - Questions?

Run the change detection script

python 15_classifyChange9607.py
Final Classification: 1996
Final Classification: 2007
Run 2007-2010 Change Detection

Copy the python script to the data directory

cp ../Scripts/16_classifyChange0710.py ./.

- Open 16_classifyChange0710.py and review the code.
  - Questions?

Run the change detection script

python 16_classifyChange0710.py
Final Classification: 2007
Final Classification: 2010
Final Classification: 1996
Whilst the classification utilised has utilised a simple rule-based approach more advanced methods are available. The Scikit-learn Python library provides a number of algorithms for machine learning that can be applied to the RAT.

Aim: Provide a very brief introduction to how the library can be applied to the segments generated, using Hierarchical clustering as an example.

Very new area for us - lots of potential!
Data Representation

- Represent columns in RAT as single matrix.
- Will be subsetting rows, need to include ID so rows get written back to the correct location.
- Only consider data in ‘Coastal Strip’ Category defined earlier

\[
\mathbf{X} = \begin{pmatrix}
\text{ID} & \sigma^0_{96} & \sigma^0_{07} & \sigma^0_{10} & \text{Coastal Strip} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\end{pmatrix}
\]  

(1)
# Import Ward clustering
from sklearn.cluster import Ward

# Run clustering
clusterer = Ward(n_clusters=8)
clusterer.fit(X[:,1:-1])

# Get cluster labels
labels = clusterer.labels_

- Don’t pass in the first and last column X[:,1:-1]
Clustering in Scikit-learn

Copy the python script to the data directory

```
cp ../Scripts/17_sklearn_clustering.py ./.
```

- Open `17_sklearn_clustering.py` and review the code.
  - Questions?

Run the change detection script

```
python 17_sklearn_clustering.py
```
Clustering Output

Note: colours will be different
Scikit-learn

• If ground truth data are available algorithms for supervised classification are available such as Random Forests and Support Vector Machines
• Scikit-learn is one of many libraries - anything that will run in Python can be applied as part of an object based classification.
• Not limited to classification - could apply regression / inversion approaches.