PyRaster - A spatial image processing library

Tomas Holderness

School of Civil Engineering and Geosciences
Newcastle University

Mars Space Flight Facility, Arizona State University - October 2010
Presentation Overview

PyRaster spatial image processing library

- Description
- Key features
- How it works
- Uses
- Demonstration
- Examples
PyRaster

PyRaster is a Python module containing a library of high-level functions for the quantitative manipulation of multiple raster images:

▶ Built on the Geospatial Data Abstraction Library (GDAL) Python bindings
▶ Converts raster images to a standard Numerical Python (Numpy) matrix
▶ Image manipulation routines created using standard matrix math, output written to a spatial raster format
▶ Built from a hybrid combination of existing FOSS modules
▶ Created out of the need for an efficient processing environment for large numbers of Earth observation images
PyRaster - Process Diagram (Write)

Model expressed using Numpy maths

PyRaster reads and writes matrices in memory

GDAL - low level I/O

Raster files on disk
PyRaster - Key features

- Written in Python:
  - A huge selection of pre-written Science and Maths (stats) modules that can be used on matrices and vectors (SciPy, Numpy)
  - Rapid development
  - Accessible to non-specialists to develop domain specific applications
- Fast - as quick as commercial GIS and remote sensing packages
  - Supports NoData values through array masking
At the core is the rasterIO module which reads and writes raster bands to Numpy matrices.

- **rasterIO** contains the following four functions:
  - `opegdalraster` Open a pointer to GDAL supported raster file
  - `readrastermeta` Read raster file metadata
  - `readrasterband` Read the numerical values of a rasterband and store in matrix
  - `writerasterband` Write the values of any numerical matrix to a specified raster file
  - `wkt2epsg` Convert well known text coordinate system representation to an EPSG code
PyRaster – How it works

- `rasterIO.readrasterband()` calling the GDAL function `band.ReadRaster()`
PyRaster - Uses/Demonstration

- Iterating pre-processing steps over a large dataset
- Designing large statistical analysis routines using existing statistical packages
- Creating domain-specific processing functions
- Band ratios expressed in a simple syntax
Getis and Ord Hot-spot detection using Inverse Euclidean Distance Attribute Weighting

```python
def iedw(input_r, i_y, i_x, res):
    """
    Accepts an input raster, origin pixel coordinates and pixel spatial resolution and generates a weighting matrix as a function of inverse Euclidean distance over the surface.
    """
    Note that weights are normalised over enitre matrix, so that sum of weights = 1.
    >>> Gistar.iwd(input_r, 800, 850, 1.1)
    ...  
    # create empty weight matrix for output.
    weights = ma.zeros(input_r.shape, input_r.dtype)
    # iterate over dataset around i.
    for y in range(0, len(input_r[0,:])):
        for x in range(0, len(input_r[y,:])):
            if ma.is_masked(input_r[y,x]):
                weights[y,x] = 9999.0
            elif input_r[y,x] == 9999:
                weights[y,x] = 9999.0
            else:
                # calculate distance to all other pixels
                distance = np.sqrt(np.power(((x-i_x)*spat_res), 2) + np.power(((y-i_y)*spat_res), 2))
                # apply inverse weighting
                weights[y,x] = 1 / distance
                #print weights[y,x]
    # Set local point i to have weight of 1.
    weights[i_y,i_x] = 1
    # apply mask to weights
    weights = ma.masked_values(weights, 9999.0)
    # re-normalise so sum of weights = 1
    weights_sum = ma.sum(weights)
    print "done calc weights sum"
    weights = weights / weights_sum
    # Write Weight files
    #rasterIO.writerasterband(weights, ".\kjj_y\x\il.tif" format(i_y,i_x), driver_short, XSize, YSize)
```
PyRaster - Example scene statistical analysis

Weighting Matrix
PyRaster - Example scene statistical analysis

Temperature Hot-spots - 1997
PyRaster - Example scene statistical analysis

Temperature Hot-spots - 2003
PyRaster - Multi-core batch processing

```python
fname_log = infile+'.log'
log = open(fname_log, 'a')
print >> log, 'Processing log file for: \{0\}'.format(infile)
print >> log, 'Date/Time generated (UTC): \{0\}'.format(str(datetime.utcnow())[:19])
print >> log, 'Script name: \{0\}'.format(sys.argv[0])
print >> log, 'Script version: \{0\}'.format(__version__)
print >> log, 'Scene date (UTC): \{0\}'.format(rdate)
print >> log, 'Start time (UTC): \{0\}'.format(rend)
print >> log, 'End time (UTC): \{0\}'.format(rstart)
print >> log, 'Day/Night Flag: Day'
print >> log, 'NOAA Satellite: \{0\}'.format(satellite)
print >> log, 'EPSG Code: \{0\}'.format(egps)
print >> log, 'Outputs generated: \{0\} \n{1} \n{2} \n{3}'.format(newname_list,
newname_ndvi, newname_snow, newname_cloud)
```

```python
def processRun(q, filename):
    q.put(processfunc(filename))

def main(arg=sys.argv):
    flist = os.listdir(os.getcwd())
    len_flist = len(flist)
    half_len = len_flist / 2
    q = Queue()
    p1 = Process(target=processRun, args=(q, flist[:half_len]))
    p2 = Process(target=processRun, args=(q, flist[half_len:]))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
```

```
if __name__ == '__main__':
    sys.exit(main())
```