Hidden Gems
Working with shapefiles and record or structured arrays (recarray)

`recfunctions`

Introduction

I came across a module that I had never heard about. This module, ***recfunctions***, facilitates working with record/structured arrays within the NumPy module. It provides many short cuts which make it easier to work with shapefiles, as well as other data formats. My focus in this report is to show how ArcMap's built-in arcpy functions can serve as links to some of the more powerful capabilities of working with the NumPy module.

See the following links for discussion and documentation:

http://pyopengl.sourceforge.net/pydoc/numpy.lib.recfunctions.html
http://comments.gmane.org/gmane.comp.python.numeric.general/39537
http://stackoverflow.com/questions/5288736/adding-a-field-to-a-structured-numpy-array-2
https://github.com/numpy/numpy/blob/1e22553c37e9112bf2426c1b060275419f906d8d/numpy/lib/recfunctions.py

The function, importing and its use are discussed briefly in the following sections. We will begin by importing the module to see what it offers. The examples here, focus on the **append_fields** function within the module. It is highlighted in the directory list for the module contents.

I have taken the liberty to bold-face sections of code in order to emphasize them. Also, python syntax is highlighted in generally familiar colours.

Importing the module, its contents and help

There are two ways to import the module depending upon the type of namespace you prefer. You can import the module directly or by importing the specific module and changings its namespace, giving it an alias. In the examples that follow, the second method will be used to make the code easier to read.

A listing of the functions in the module can be listed using **dir(function)**. The focus in this document will be on **append_fields**.

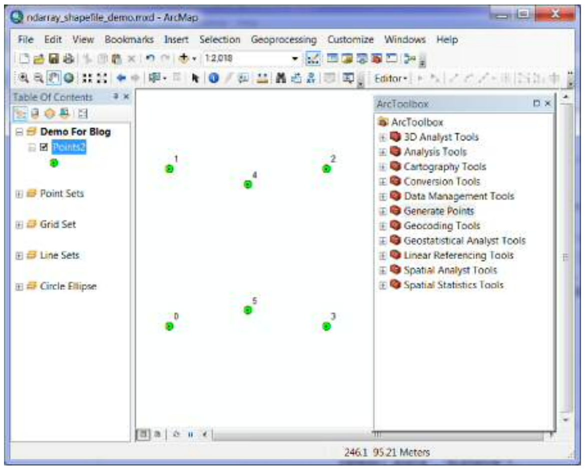
```
>>> import numpy.lib.recfunctions              # general import of recfunctions

>>> from numpy.lib import recfunctions as rfn  # method used in these examples

>>> dir(rfn)  # module contents
['MaskedArray', 'MaskedRecords', '__all__', '__builtins__', '__doc__', '__file__',
'__name__', '__package__', '_check_fill_value', '_fix_defaults', '_fix_output',
'_is_string_like', '_izip_fields', '_izip_fields_flat', 'append_fields',
'drop_fields', 'find_duplicates', 'flatten_descr', 'get_fieldstructure', 'get_names',
'get_names_flat', 'itertools', 'izip_records', 'join_by', 'ma', 'merge_arrays',
'ndarray', 'np', 'rec_append_fields', 'rec_drop_fields', 'rec_join', 'recarray',
'recursive_fill_fields', 'rename_fields', 'stack_arrays', 'sys', 'zip_descr']
>>>
```

Help for a specific module is obtained using the **help(function)** syntax. The requirements for its use are clearly listed as are the defaults. If you wish to use the defaults, you need not include a reference to the option when generating the function requirements.

```
>>> help(rfn.append_fields)
Help on function append_fields in module numpy.lib.recfunctions:

append_fields(base, names, data, dtypes=None, fill_value=-1, usemask=True,
asrecarray=False)
    Add new fields to an existing array.

    The names of the fields are given with the `names` arguments,
    the corresponding values with the `data` arguments.
    If a single field is appended, `names`, `data` and `dtypes` do not have
    to be lists but just values.

    Parameters
    ----------
    base : array
        Input array to extend.
    names : string, sequence
        String or sequence of strings corresponding to the names
        of the new fields.
    data : array or sequence of arrays
        Array or sequence of arrays storing the fields to add to the base.
    dtypes : sequence of datatypes, optional
        Datatype or sequence of datatypes.
        If None, the datatypes are estimated from the `data`.
    fill_value : {float}, optional
        Filling value used to pad missing data on the shorter arrays.
    usemask : {False, True}, optional
        Whether to return a masked array or not.
  asrecarray : {False, True}, optional
      Whether to return a recarray (MaskedRecords) or not.
```

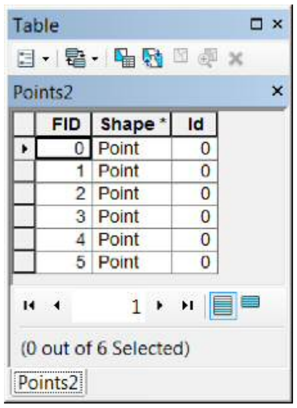
Sample project...

A sample point shapefile is shown in the figure to the right. It consists of six points. The first four point form both the exterior of a rectangle or the convex hull for the six points, depending upon your interests.



The spatial reference for the object was defined as a variant of UTM zone 18N...so technically, these points are just north of the equator within the zone. I wanted to make sure that real geometry was being used and that includes a valid spatial reference.

The purpose of the exercise is to add two fields to the table and calculate values for them.



FID	Shape *	Id
0	Point	0
1	Point	0
2	Point	0
3	Point	0
4	Point	0
5	Point	0

(0 out of 6 Selected)

Points2

The angles formed by the points with respect to the center point will be added to the first field. The point distances to the center will be calculated for the second field. Finally, the output array will be sorted using a radial sort on the angles previously derived.

A new shapefile is created containing the original input data, the new fields and their calculated values. The actual process is contained in a script, but the whole procedure can be done using the command line. This document uses that approach so that the inputs and outputs can be readily identified.

The sample command line instructions are as follows:

Basic imports and inputs

```
>>> import numpy as np
>>> from numpy.lib import recfunctions as rfn
>>> input_shp = 'C:\\temp\\Shapefiles\\Points2.shp'
>>> SR =arcpy.Describe(input_shp).spatialReference
>>> print SR.name
NAD_1983_CSRS_UTM_Zone_18N
>>>
```

Create the array from the existing shapefile

```
>>> fields = arcpy.ListFields(input_shp)
>>> field_names = [f.name for f in fields]
>>> print field_names
[u'FID', u'Shape', u'Id']
>>> data = arcpy.da.FeatureClassToNumPyArray(input_shp,field_names,"",SR)
>>> np_array = np.array(data)
>>> np_array
array([(0, [0.0, 0.0], 0), (1, [0.0, 100.0], 0), (2, [100.0, 100.0], 0),
       (3, [100.0, 0.0], 0), (4, [50.0, 90.0], 0), (5, [50.0, 10.0], 0)],
      dtype=[('FID', '<i4'), ('Shape', '<f8', (2,)), ('Id', '<i4')])
>>> temp_array = np_array[:]
```

Calculate the new values to add to the array

```
>>> #....snip ... methods to calculate angles and distance omitted to
>>> #    simplify the presentation, so the following values will be
>>> #....used from the results
>>> angle = np.array([-135.,  135.,   45.,  -45.,   90.,  -90.])
>>> length = np.array([ 70.71067812, 70.71067812, 70.71067812, 70.71067812, 40. ,  40.
])
>>> add_flds = [angle, length]
>>>
>>> NOTE: I have wrapped the following to facilitate reading
>>>
>>> out_array= rfn.append_fields(temp_array,
                                 names=('angle','distance'),
                                 data=add_flds,
                                 dtypes=None,
                                 usemask=False, asrecarray=False)
>>>
>>> out_array
array([(0, [0.0, 0.0], 0, -135.0, 70.71067812),
       (1, [0.0, 100.0], 0, 135.0, 70.71067812),
       (2, [100.0, 100.0], 0, 45.0, 70.71067812),
       (3, [100.0, 0.0], 0, -45.0, 70.71067812),
       (4, [50.0, 90.0], 0, 90.0, 40.0), (5, [50.0, 10.0], 0, -90.0, 40.0)],
      dtype=[('FID', '<i4'), ('Shape', '<f8', (2,)), ('Id', '<i4'), ('angle', '<f8'),
('distance', '<f8')])
>>>
```

Sort the array using the angle field

```
>>> indices = (angle.argsort().tolist())
>>> indices.reverse()
>>> sorted_array = np.array( [out_array[i] for i in indices] )
>>> sorted_array
array([(1, [0.0, 100.0], 0, 135.0, 70.71067812),
       (4, [50.0, 90.0], 0, 90.0, 40.0),
       (2, [100.0, 100.0], 0, 45.0, 70.71067812),
       (3, [100.0, 0.0], 0, -45.0, 70.71067812),
       (5, [50.0, 10.0], 0, -90.0, 40.0),
       (0, [0.0, 0.0], 0, -135.0, 70.71067812)],
      dtype=[('FID', '<i4'), ('Shape', '<f8', (2,)), ('Id', '<i4'), ('angle', '<f8'),
('distance', '<f8')])
```

Create the output shapefile using the new data

```
>>> field_names = data_types.names
>>> field_names
('FID', 'Shape', 'Id', 'angle', 'distance')
>>> SR.name
u'NAD_1983_CSRS_UTM_Zone_18N'
>>>
>>> output_shp = "c:/temp/Points2_sorted.shp"
>>> arcpy.da.NumPyArrayToFeatureClass(sorted_array, output_shp, field_names, SR)
>>> output_shp = "c:/temp/Points2_sorted.shp"
>>> geom_fields = ('Shape')
>>> arcpy.da.NumPyArrayToFeatureClass(sorted_array, output_shp, geom_fields, SR)
```

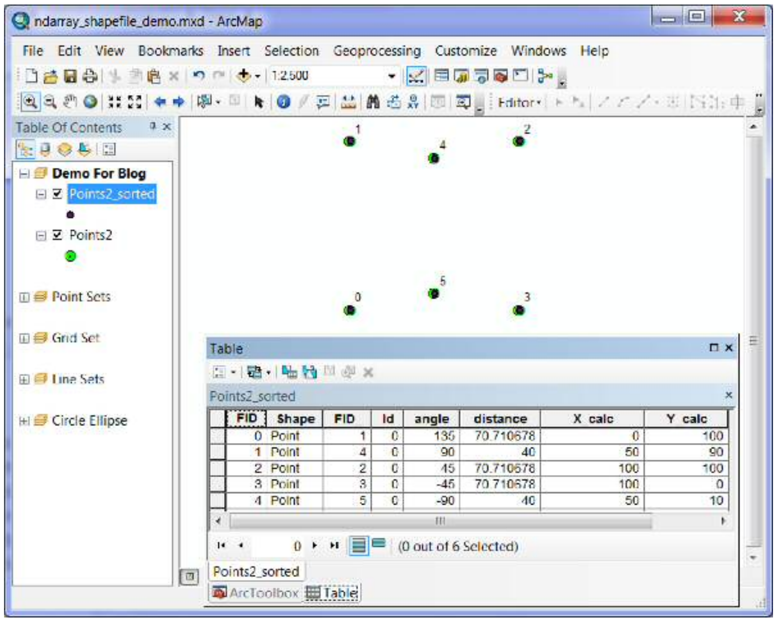
The results are shown in the figure to the right.

The coordinate geometry values were also calculated within ArcMap and added to the table, as a check on the input values.

You should note that the old FID field (feature ID), has been retained but renamed to FID_ so one can identify the initial and final point order resulting from the sorting process.

This example only scratches the surface of what can be done with recfunctions and its tools.

More examples to follow.



FID	Shape	FID	Id	angle	distance	X calc	Y calc
0	Point	1	0	135	70.710678	0	100
1	Point	4	0	90	40	50	90
2	Point	2	0	45	70.710678	100	100
3	Point	3	0	-45	70.710678	100	0
4	Point	5	0	-90	40	50	10

(0 out of 6 Selected)