Polyline Distance/length Calculations With... Numpy and Arcpy

This is a simple example of calculating distance/length related values using points that are the vertices of a polyline. The usual methods would include using arcpy methods associated with the point class and various cursors in the arcpy module.

We will begin by a simple 2D shapefile with 10 points. Import our two main modules and create an array using the built-in FeatureClassToNumPyArray method. All that is required is a featureclass in a geodatabase or ashapefile and the fields that you want to import. Details are presented in the help topics given in the reference section.

I prefer to use my own dtype names when working with arrays, so I just create my own (line 9) and assign them to the input array (line 11). This is possible since the data types have not changed, just the names.

Note: i4 is an integer data type and f8 is a double/float... details elsewhere, but that is effectively what you will use except for text (U5 for unicode string 5 characters wide) and maybe datetime (addressed much later).

```
In [1]: import numpy as np
        import arcpy

        # ---- load a shapefile into an array and simplify the dtype ----
        in_fc = r'C:\Git_Dan\JupyterNoteBooks\Data\Shapefiles\test.shp'

        a = arcpy.da.FeatureClassToNumPyArray(in_fc,
                                              ["OID@", "SHAPE@X", "SHAPE@Y"])

        dt = [('ID', '<i4'), ('X', '<f8'), ('Y', '<f8')]
        # ---------------------------------------------------------------------
        a.dtype = dt

        frmt = """
        (1) Structured array, reshaped to facilitate viewing\n{!r:}
        """
        print(frmt.format(a.reshape(a.shape[0], -1)))
```

```
(1) Structured array, reshaped to facilitate viewing
array([[(0, 7.0, 2.0)],
       [(1, 8.0, 1.0)],
       [(2, 2.0, 1.0)],
       [(3, 4.0, 9.0)],
       [(4, 0.0, 7.0)],
       [(5, 1.0, 8.0)],
       [(6, 5.0, 7.0)],
       [(7, 7.0, 4.0)],
       [(8, 1.0, 1.0)],
       [(9, 2.0, 2.0)]],
      dtype=[('ID', '<i4'), ('X', '<f8'), ('Y', '<f8')])
```

At this point, a structured array has been created and ready for use. Calculating distance between points can be accomplished using a variety of means. They are all valid, but I have my preference since it scales well to arrays with dimensions greater than 2D. The details of the calculation are not really pertinent to the discussion.

At this point, we need to pull out the X, Y coordinates to make it easier and faster to complete the task. Remember, this demonstration is showing all the steps... there are helper functions to do these repetitive tasks for you...

```
In [2]: # ---------------------------------------------------------------------
        #  Create an array to hold the X, Y values and our result, and
        #  fill it appropriately ----
        #  # np.zeros(shape=(rows, cols), dtype=datatype)
        #  [:, 0] means every row, the first column [:, 1] is the second column
        # compute the sequential differences between the points
        xy = np.zeros((len(a), 3), dtype='float')

        xy[:, 0] = a['X']
        xy[:, 1] = a['Y']

        diff = xy[1:] - xy[:-1]

        dist_sq = np.einsum('ij,ij->i', diff, diff)  # magic happens here

        e_dist = np.sqrt(dist_sq).squeeze()

        h_dist = np.hypot(diff[:, 1], diff[:, 0])    # Pythagoras to the rescue

        # now fill the xy array's third column with the distances skipping the first
        #
        xy[1:, 2] = e_dist

        print("(2) e_dist = h_dist?... {}\n".format(np.all(e_dist==h_dist)))
        print("(3) the X, Y and Distances\n{}".format(xy))

(2) e_dist = h_dist?... True

(3) the X, Y and Distances
[[ 7.          2.          0.        ]
 [ 8.          1.          1.41421356]
 [ 2.          1.          6.        ]
 [ 4.          9.          8.24621125]
 [ 0.          7.          4.47213595]
 [ 1.          8.          1.41421356]
 [ 5.          7.          4.12310563]
 [ 7.          4.          3.60555128]
 [ 1.          1.          6.70820393]
 [ 2.          2.          1.41421356]]
```

The results can be sent out to a featureclass either by appending to an existing one, or creating a new one. The following shows how to accomplish this. The first step is to create a new dtype and an output array to fill. The results are created and send to an output featureclass. The previous dtype will be reused and added to.

```
In [3]: # ---------------------------------------------------------------------
        # prepare a structured array to fill
        # notice that there is no 'ID' field!!!
        # fill the array
        # off to a shapefile or featureclass, uncomment line 14 obviously

        dt = [('X', '<f8'), ('Y', '<f8')] + [('Dist', '<f8')]

        out = np.zeros((len(xy),), dtype=dt)
        names = out.dtype.names
        for i in range(xy.shape[1]):
            out[names[i]] = xy[:, i]

        # arcpy.da.NumPyArrayToFeatureClass(out, r'c:\temp\test.shp', ['X', 'Y'])
```

You may have noticed that no ID field was needed to create the output array since ArcMap or ArcGIS Pro will create the necessary identification field.

In the above example, the interpoint distances were calculated. It should come as no surprise that one could calculate:

1. the total length
2. the distance between any two points on the polyline (ie first and last, first and all others, etcetera)
3. a distance matrix of distances of all points to all other points

The options are endless, and only slight modifications to the base code is required.

```
In [4]: # ---------------------------------------------------------------------
        # begin with xy array, form a cloned version with a different shape
        # this is needed for 'array broadcasting'
        # subtract the two arrays
        # return euclidean distance and clean up

        a = xy.reshape(xy.shape[0], 1, xy.shape[1])
        diff = a - xy
        e_matrix = np.einsum('ijk,ijk->ij', diff, diff)  # 'Magic'
        out = np.sqrt(e_matrix).squeeze()

        # ---- set up some formatting options and print away ---- (homework)

        np.set_printoptions(edgeitems=10, linewidth=80, precision=1,
                            suppress=True, threshold=100)

        print("\nDistance matrix between all points...\n{!r:}".format(out))
```

```
Distance matrix between all points...
array([[  0. ,   2. ,   7.9,  11.2,   9.7,   8.6,   6.8,   4.1,   9.1,   5.2],
       [  2. ,   0. ,   7.6,  11.3,  10.5,   9.9,   7.2,   3.8,   8.8,   6.1],
       [  7.9,   7.6,   0. ,   8.5,   6.5,   8.4,   7. ,   6.3,   1.2,   4.7],
       [ 11.2,  11.3,   8.5,   0. ,   5.9,   7.5,   4.7,   7.5,   8.7,  10. ],
       [  9.7,  10.5,   6.5,   5.9,   0. ,   3.4,   5. ,   7.7,   6.5,   6.2],
       [  8.6,   9.9,   8.4,   7.5,   3.4,   0. ,   4.9,   7.5,   8.8,   6.1],
       [  6.8,   7.2,   7. ,   4.7,   5. ,   4.9,   0. ,   3.6,   7.7,   6.4],
       [  4.1,   3.8,   6.3,   7.5,   7.7,   7.5,   3.6,   0. ,   7.4,   5.8],
       [  9.1,   8.8,   1.2,   8.7,   6.5,   8.8,   7.7,   7.4,   0. ,   5.5],
       [  5.2,   6.1,   4.7,  10. ,   6.2,   6.1,   6.4,   5.8,   5.5,   0. ]])
```

References

Point class

http://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-classes/point.htm

Point Geometry class

http://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-classes/pointgeometry.htm

FeatureClassToNumPyArray method

http://desktop.arcgis.com/search/?q=featureclasstonumpyarray&collection=help&product=arcgis-d