NumPy Snippets......

Pythonista's do good

Published - 2016-02-01
Dan Patterson

An initial survey of the 3rd order finite difference method of measuring slope and aspect. This method is used by ArcGIS and can be easily programmed using the documentation in a variety of texts (eg. Horn, Burrough). Here are some examples for four of the cardinal aspects. In the examples, the cell size... hence, center-to-center spacing, is denoted by the dx column name. Results are given in both degrees and percentage. The first entry has a cell size and z-value difference of 0.5 units which equates to a 45 degree angle in the downslope direction (towards north). As the cell size increases, for a fixed elevation difference, the angle decreases in a predictable way. The notation for the equations reference the cell positions relative to the middle for a 3x3 window.

a	b	c
d	?	f
g	h	i

```
[dz_dx]   = ((c + 2f + i) - (a + 2d + g)) / (8 * x_cellsize)
[dz_dy]   = ((g + 2h + i) - (a + 2b + c)) / (8 * y_cellsize)
rise_run =  sqrt([dz_dx]**2 + [dz_dy]**2)
         =  sqrt((0.05)**2 + (-3.8)**2)
         =  sqrt(0.0025 + 14.44) = 3.80032
slope_r  =  sqrt( [dz/dx]**2 + [dz/dy]**2)  in radians
slope_d  =  ATAN(slope_r)   in degrees
```

Where **dz_dx**, **dz_dy** are the differences in **z** divided by their respective **x** and **y** differences. The results are returned in radians which are converted to either degree or percentage formats. Aspect is calculated in a similar fashion with modification to produce a direction relative to north. The code sample covers the details.

```
Slope face... North

[[ 0.   0.   0. ]
 [ 0.5  0.5  0.5]
 [ 1.   1.   1. ]]

N      dx    deg.    %
(1 )   0.5  45.00  400.0
(2 )   1.0  26.57  200.0
(3 )   2.0  14.04  100.0
(4 )   4.0   7.13   50.0
(5 )   6.0   4.76   33.3
(6 )   8.0   3.58   25.0
(7 )  10.0   2.86   20.0
(8 )  12.5   2.29   16.0
(9 )  15.0   1.91   13.3
(10)  20.0   1.43   10.0
(11)  25.0   1.15    8.0
(12)  40.0   0.72    5.0
(13)  80.0   0.36    2.5
(14) 100.0   0.29    2.0
```

```
Slope face... West

[[ 0.   0.5  1. ]
 [ 0.   0.5  1. ]
 [ 0.   0.5  1. ]]

N      dx    deg.    %
(1 )   0.5  45.00  400.0
(2 )   1.0  26.57  200.0
(3 )   2.0  14.04  100.0
(4 )   4.0   7.13   50.0
(5 )   6.0   4.76   33.3
(6 )   8.0   3.58   25.0
(7 )  10.0   2.86   20.0
(8 )  12.5   2.29   16.0
(9 )  15.0   1.91   13.3
(10)  20.0   1.43   10.0
(11)  25.0   1.15    8.0
(12)  40.0   0.72    5.0
(13)  80.0   0.36    2.5
(14) 100.0   0.29    2.0
```

```
Slope face... South

[[ 1.   1.   1. ]
 [ 0.5  0.5  0.5]
 [ 0.   0.   0. ]]

N      dx    deg.    %
(1 )   0.5  45.00  400.0
(2 )   1.0  26.57  200.0
(3 )   2.0  14.04  100.0
(4 )   4.0   7.13   50.0
(5 )   6.0   4.76   33.3
(6 )   8.0   3.58   25.0
(7 )  10.0   2.86   20.0
(8 )  12.5   2.29   16.0
(9 )  15.0   1.91   13.3
(10)  20.0   1.43   10.0
(11)  25.0   1.15    8.0
(12)  40.0   0.72    5.0
(13)  80.0   0.36    2.5
(14) 100.0   0.29    2.0
```

```
Slope face... East

[[ 1.   0.5  0. ]
 [ 1.   0.5  0. ]
 [ 1.   0.5  0. ]]

N      dx    deg.    %
(1 )   0.5  45.00  400.0
(2 )   1.0  26.57  200.0
(3 )   2.0  14.04  100.0
(4 )   4.0   7.13   50.0
(5 )   6.0   4.76   33.3
(6 )   8.0   3.58   25.0
(7 )  10.0   2.86   20.0
(8 )  12.5   2.29   16.0
(9 )  15.0   1.91   13.3
(10)  20.0   1.43   10.0
(11)  25.0   1.15    8.0
(12)  40.0   0.72    5.0
(13)  80.0   0.36    2.5
(14) 100.0   0.29    2.0
```

The code is documented on the next page. It consists of a main calling function (slope_dem) which has a nested function (cal_slope) inside.

```
import numpy as np
from numpy.lib.stride_tricks import as_strided
np.set_printoptions(edgeitems=3,linewidth=80,precision=2,suppress=True,threshold=100)
import matplotlib.pyplot as plt

def slide_a(a, block=(3,3)):
    """Provide a 2D sliding/moving array view.  There is no edge
    correction for outputs.
    """
    r, c = block  # 3x3 block default
    a = np.ascontiguousarray(a)
    shape = (a.shape[0] - r + 1, a.shape[1] - c + 1) + block
    strides = a.strides * 2
    s_a = as_strided(a, shape=shape, strides=strides)
    return s_a

def aspect_dem(a):
    """Return aspect relative to north """
    a = np.asarray(a)
    dzdx_a = (a[:,2] - a[:,0]).sum()/8.0    # aspect: col2 - col0
    dzdy_a = (a[2] - a[0]).sum()/8.0        # aspect: row2 - row0
    s = np.arctan2(dzdy_a, -dzdx_a)
    s = np.rad2deg(s)
    aspect = np.where(s<0, 90.-s, np.where(s>90, 360.0-s+90.0, 90.0-s))
    return aspect

def slope_dem(a, cell_size=1):
    """Return slope in degrees for an input array using 3nd order
       finite difference method
    """
    def cal_slope(win, cell_size):
        """Calculate the slope for the window"""
        dzdx_s = (win[:,2] - win[:,0]).sum()/cell_size # slope: col2 - col0
        dzdy_s = (win[2] - win[0]).sum()/cell_size     # slope: row2 - row0
        s1ope = np.sqrt(dzdx_s**2 + dzdy_s**2)
        slope = np.rad2deg(np.arctan(s1ope))
        return slope
    # ---- read array and parse to calculate slope
    a = np.ascontiguousarray(a)
    ndim = a.ndim
    shp = a.shape
    f_dxyz = np.array([[1,2,1],[2,0,2],[1,2,1]], dtype="float64") # factor
    cell_size = (8.0*cell_size)   # cell size
    a = a*f_dxyz                  # apply slope filter to array
    if ndim == 2:
        slope = cal_slope(a,cell_size)
    elif ndim == 3:
        slope = [ cal_slope(a[i],cell_size)
                  for i in range(a.shape[0])]  # shape (0,x)
        slope = np.asarray(slope)
    elif ndim == 4:
        s0, s1, s2, s3 = shp
        slope = [ cal_slope(a[i][j],cell_size)
```

```
                  for i in range(a.shape[0])  # shape (0,x,x,x)
                  for j in range(a.shape[1])] # shape (x,1,x,x)
        slope = np.asarray(slope).reshape((s0,s1))
    return slope**2 + dzdy_s**2)
        slope = np.rad2deg(np.arctan(s1ope))
        return slope
    # ---- read array and parse to calculate slope
    a = np.ascontiguousarray(a)
    ndim = a.ndim
    shp = a.shape
    f_dxyz = np.array([[1,2,1],[2,0,2],[1,2,1]], dtype="float64") # factor
    cell_size = (8.0*cell_size)   # cell size
    a = a*f_dxyz                  # apply slope filter to array
    if ndim == 2:
        slope = cal_slope(a,cell_size)
    elif ndim == 3:
        slope = [ cal_slope(a[i],cell_size)
                  for i in range(a.shape[0])]  # shape (0,x)
        slope = np.asarray(slope)
    elif ndim == 4:
        s0, s1, s2, s3 = shp
        slope = [ cal_slope(a[i][j],cell_size)
                  for i in range(a.shape[0])   # shape (0,x,x,x)
                  for j in range(a.shape[1])   # shape (x,1,x,x)
                 ]
        slope = np.asarray(slope).reshape((s0,s1))
    return slope

# ---------------------------------------------------------------------
if __name__=="__main__":
    """run sample for slope and aspect determinations for dem data"""
    #
    za =np.array([[0, 1, 2, 3, 2, 1, 0],
                  [1, 2, 3, 4, 3, 2, 1],
                  [2, 3, 4, 5, 4, 3, 2],
                  [3, 4, 5, 5, 5, 4, 3],
                  [2, 3, 4, 5, 4, 3, 2],
                  [1, 2, 3, 4, 3, 2, 1],
                  [0, 1, 2, 3, 2, 1, 0]])
    a = aspect_demo()               # 1
    a = slope_demo()                # 2
```