MGS Stacked Cross Sections: Check Strat Order and Line Angles

# Summary

This is a quality control tool used to ensure that cross section stratlines were drawn with the correct unit order. It will also check that lines are not drawn on top of each other at an angle. Outputs are: 1. a line file that highlights lines that are incorrectly ordered and 2. a point file that highlights line vertices where a line is drawn on top of itself. Intermediate files are saved in memory workspace and output files are saved in the same gdb as the input stratlines.

The script was written in Python 3 using the Spyder Python IDE. Comments throughout the code explain processes, and additional information is in this document. Throughout the code, some comments begin with “#%%”. The addition of the “%%” turns a chunk of code into a cell that can be run from the Spyder Python IDE. Sample input and output data is saved in the “SampleData” folder. This tool runs the most efficiently if all data is stored on a local drive.

# Code Structure

### #%% 1 Import modules and define functions

This section imports necessary Python modules. Three print statement functions are defined (message, warning, and error), and function are defined to check if a file or attribute field exists.

### #%% 2 Set parameters to work in testing and compiled geoprocessing tool

This section defines parameters that are used in the script. The block within if (len(sys.argv) > 1): will execute if the script is run inside an ArcGIS Pro script tool. The else block is executed if the script is run outside of ArcGIS Pro in a Python editor. To run the script outside of Arc, modify the parameters in the else block. Do not modify the parameters in the if (len(sys.argv) > 1): block unless you are also modifying and updating the script tool. Parameter descriptions are in the code comments.

Parameters are: stratlines (feature class or shapefile should be fine), unit field in stratline file (used to compare with unit list), unitlist (text file with unit order, one unit code per line, must match stratline unit field), and reference polygon (used to isolate one cross section at a time). Temporary directory is set as “in memory”. For information on county relief and vertical exaggeration variables, see documentation on Stacked Cross Sections.

### #%% 3 Set up unit list

This section reads the unit list text file and creates a python list. It uses the “strip” function to remove any extra spaces that are before or after each unit code. It will also remove any blank lines from the list. Finally, it turns the list into a set and checks the length of the set relative to the list to make sure there were no duplicates in the list. If there are duplicates, an error will print.

### #%% 4 Dissolve stratlines by unit

Dissolves stratlines by unit with singlepart features. This will reduce the number of polygons that need to be made later. Output is a temporary stratline file.

### #%% 5 Spatial join with mn\_et\_id polys

Joins mn\_et\_id attribute from the reference polygons. Used to isolate one cross section line at a time. Output is a second temporary stratline file.

### #%% 6 Create empty above polys for each stratline

Creates an empty file for populating with “above polygons” geometry. More on this below, but there will be a polygon for each stratline, and it will include the area directly above the line that is used to determine which stratline units are above it.

### #%% 7 Create empty angle errors point output for geometry creation

Creates an empty point file for storing line angle errors. This file will store places where the geologist has drawn a line on top of itself.

### #%% 8 Create empty order errors line file

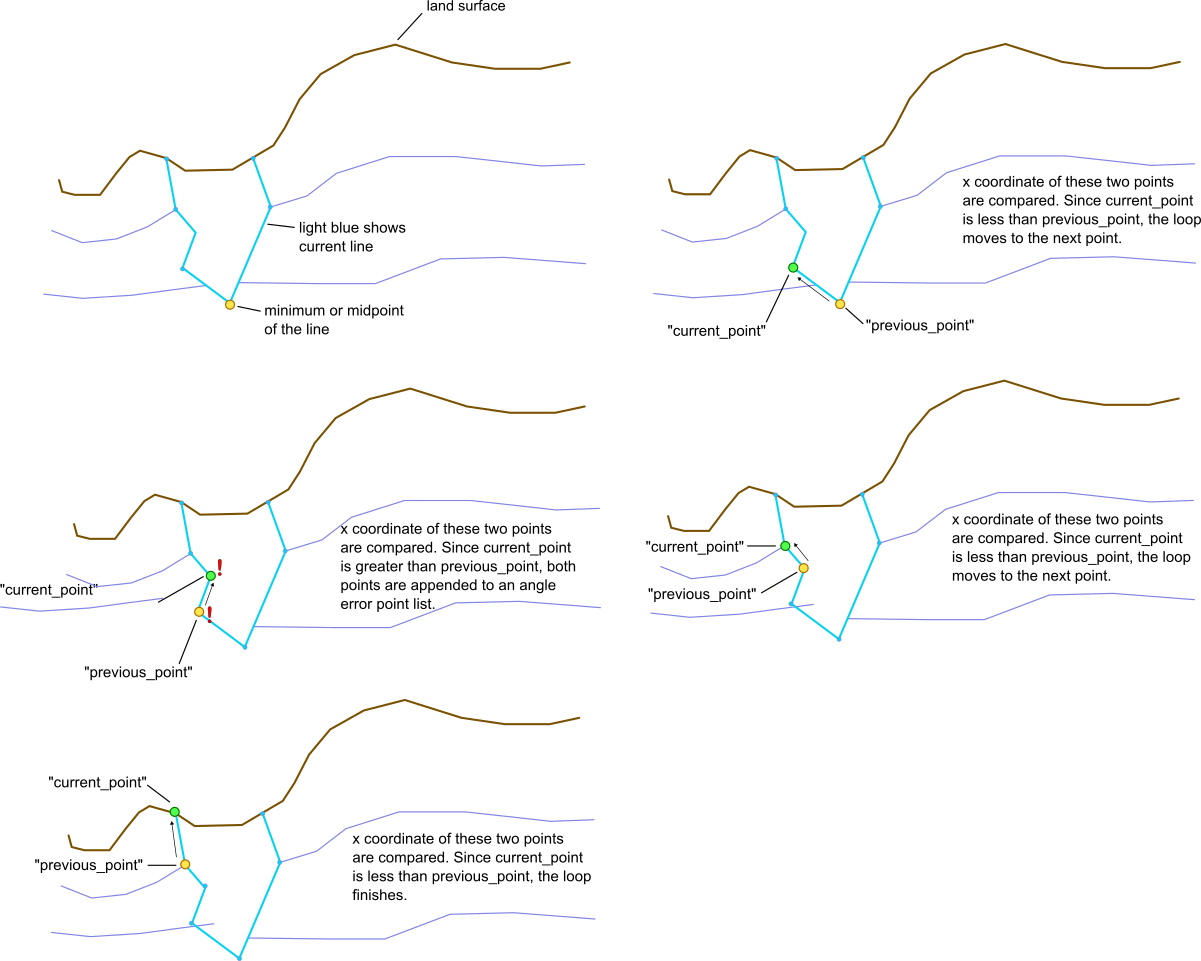
Creates an empty line file for storing line order errors. This file will store lines that are out of order according to the unit list.

### #%% 9 Create geometry for “above polys” and check for angle errors

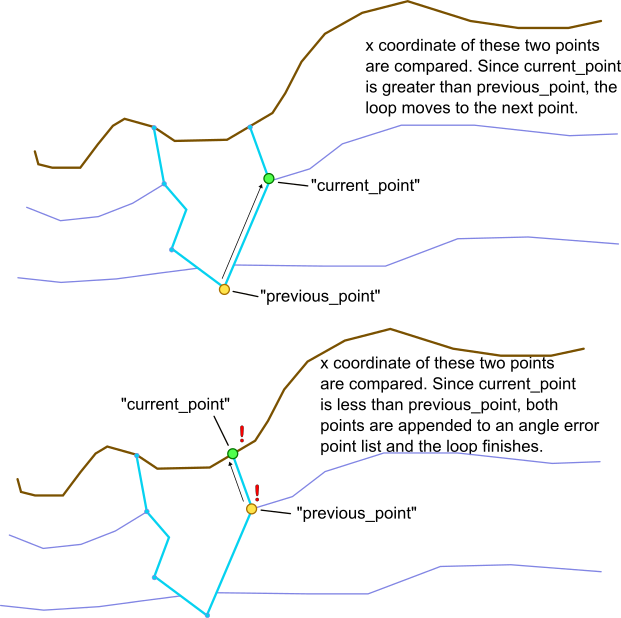
This is the most complex section of the script. It uses SearchCursor to loop through the second temporary stratline file one line at a time. It defines the midpoint/lowpoint of each stratline as the vertex with the lowest elevation, and then checks the points to the left and right of that point to make sure the line wasn’t drawn on top of itself. There are two nearly identical sections to correctly handle the direction the line was drawn in (left to right or right to left), starting with the comment #if line was drawn left to right and #if line was drawn right to left.

First, a list of vertices in the stratline is created (vertex\_list). Then, the index of the list item with the lowest y coordinate (lowest elevation) is found. Since it is possible that there are multiple points drawn at the exact same lowest elevation, this is stored in a list (min\_index\_list). The first and last x coordinates in the line are used to determine the direction that the line was drawn. Drawn left to right is the first scenario handled.

The section if first\_x < last\_x: contains two main blocks of code. The first block handles all vertices plotted to the *left* of the minimum/low point of the line, and the second block handles all vertices plotted to the *right* of the minimum/low point of the line. To check the vertices to the left of the midpoint, the code steps through all vertices with list indices that are *less* than the list index of the midpoint. The i in range section handles this. The first previous\_point is set as the midpoint, and then each following point is set as previous\_point as it is compared to the next. The x coordinates are compared, and if the x coordinate of one point is greater than the x coordinate of the previous point, the vertex will be added to an angle error point list. The graphic below illustrates the concept of how this works:



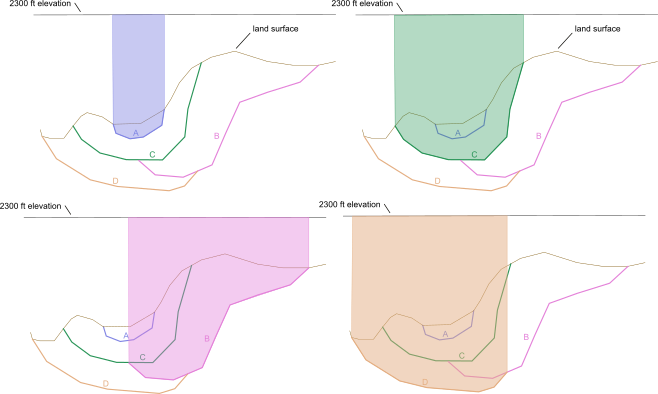
The next block checks the points to the right of the midpoint. The function is the exact same, just stepping forwards in the line instead of backwards:



The section if first\_x > last\_x: functions exactly the same, just with opposite list indices since the line was drawn in the opposite direction.

If any points were added to the angle error point list, InsertCursor is used to add their geometry to the output angle error point feature class.

Next, a polygon is created for the stratline that uses all line vertices as the bottom of the polygon, and stretches the polygon up to the maximum elevation (2300 is used since that is the maximum elevation in reference polygons, and the maximum elevation in the state of MN). This “above polygon” is later used to define all of the area above a given stratline to see if there are any units out of order. The graphic below illustrates what these polygons will look like for hypothetical units A, B, C, and D:



### #%% 10 Make temp feature layer of stratlines

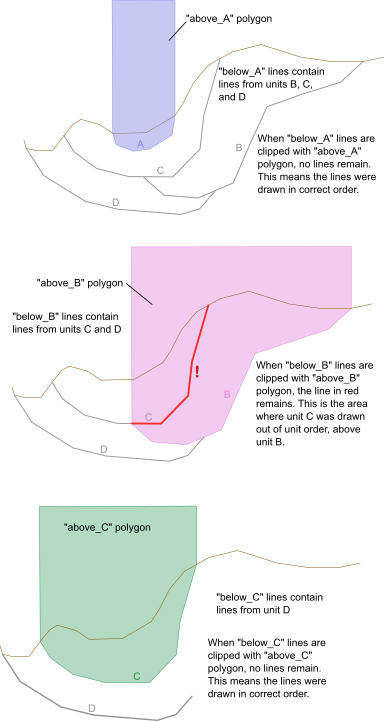
Does exactly what it says. Used for next section.

### #%% 11 Make temp stratline files

Loops through units in order from top to bottom. Deletes all lines from stratline feature layer that have the current unit attribute, and then saves a temporary feature class with all remaining lines with the current unit name in the filename. For example: for units A, B, C, and D (in order): “below\_A” line file will contain lines from units B, C, and D. “below\_B” line file will contain lines from units C and D. “below\_C” line file will contain lines from unit D. And “below\_D” will contain no lines, since it is the lowest unit. These “below line” files contain lines that *should* be below each unit based on the unitlist, NOT based on how they are drawn. Checking for how they are drawn comes next.

### #%% 12 Clip below unit files using polygon feature classes

This section will match the temporary stratline files created in section 11 with the “above polys” created in section 9. Each “below\_unit” line file will be clipped with the corresponding “above poly” file. The result is that any lines that remain in the clipped area should *not* be there. Any lines that are in the clip area are lines that are *below* the unit in unit order, but are drawn *above* the line on the cross section. Any lines that remain are appended to the output unit order error line file. Graphic below illustrates how this works:



### #%% 13 Delete tiny features

This section deletes tiny line segments that are likely just topology errors. Arbitrary length of 1 meter is chosen.

### #%% 14 Record and print tool end time

Finally, the script calculates processing time and prints the elapsed time.