MGS Stacked Cross Sections: Create Lixpys

# Summary

This script creates polyline files that are used to visualize well lithologies along defined cross sections. Outputs are: a well stick diagram file in 3D with true x, y, and z coordinates, well stick diagram files (line and polygon format) in 2D stacked cross-section space to use for cross-section creation and editing, and a well point file in 2D stacked cross-section space. Data is retrieved from a well point location feature class and a corresponding stratigraphy table with lithology and depth information. The two tables have a one-to-many relationship.

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

This section imports necessary Python modules.

### #%% 2 Define functions

Two print statement functions are defined (message and error), and a function is defined to check if a file exists.

### #%% 3 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.

### #%% 4 set county relief and vertical exaggeration variables

Parameters vertical\_exaggeration and county\_relief should not be modified except in special cases. county\_relief is an arbitrary constant that is used to vertically space cross sections. The numbers used for these variables must be consistent for all cross section data in order for everything to line up.

### #%% 5 Set 3d spatial reference based on xsln file

Spatial reference variable is set based on map view cross section line file. If the file has an undefined spatial reference, an error message prints.

### #%% 6 Add mn\_et\_id field to wwpt file if it doesn't exist

The statewide cross section ID field is necessary for plotting data in the stacked system. Y coordinates are calculated based on this statewide cross section ID number. This way, all cross sections are plotted in the same system and cross sections in adjacent study areas will line up automatically. This section checks that cross section line file has mn\_et\_id field. Then, it checks that the well point file also has the mn\_et\_id field. If it doesn’t, the code will join the mn\_et\_id field to the well point file based on the matching et\_id in the cross section file.

### #%% 7 Data QC

This section ensures that input data is of the correct type, and that necessary attribute fields exist. It checks to ensure that the mapview cross section lines are not multipart, since the code will not work with multipart features. It also checks to make sure input files are not empty.

### #%% 8 Check that strat table, well point file, and cross section file match

This section checks that cross section ID values and well ID values match between files in the input data. If they do not match, the files will not be able to relate to each other and lixpys will not be drawn. It makes four lists: well id values in the strat table, well id values in the well point file, et\_id values in the well point file, and et\_id values in the cross section line files. It compares the two well id lists to each other, and makes a list of well id values that are not in both lists. When it is finished comparing the lists, it prints a warning with the length of the list of well id values that are not in both lists. It does the same with the two et\_id lists.

et\_id values in the strat table are *not* used in this script. Later in the processing, the strat table will relate to the well point file, which already has the associated cross section line attribute.

This section also checks that the data type for the well id fields are the same in the strat table and well point file. Depending on which attribute field the user chooses to function as the well id/relate id, it may be a numeric or text field. The script stores the data type as a variable, which is important for correctly writing SQL queries in upcoming sections.

### # %% 9 List fields that are used in 3d line, 2d line, and 2d point

This section makes lists of fields that are necessary in the output files. First, it sets the well id field as numeric or text, based on the result in the previous section. There are three output files: lixpy lines in 3D, lixpy lines in 2D, and well point files in 2D (which the user can choose not to create in the input parameters if desired). (Note: the lixpy polygons are created at the very end of the processing by buffering the lixpy lines in 2D, so it is not necessary to define its fields here.) fields\_base is a list of attribute fields that are needed in all output files. Note that the xy coordinates of the wells will be saved in the “x\_coord” and “y\_coord” fields in the output files. fields\_strat lists the stratigraphy fields, which include the strat record number and top and bottom elevation. These fields are needed for all outputs with stratigraphy information, which is all files except the 2D point file. fields\_2d lists the fields that are needed for 2d files, which includes distance from the cross section line and percent distance from the cross section line. Finally, fields\_2d\_point lists the elevation field that is necessary for the 2D point file.

### # %% 10 Create empty 3d polyline file

Creates empty 3D polyline file and adds necessary attribute fields. arcpy.env.overwriteOutput = True allows the script to be re-run without having to manually delete previous files.

### # %% 11 Create empty 2d polyline file

Creates empty 2D polyline file and adds necessary attribute fields. arcpy.env.overwriteOutput = True allows the script to be re-run without having to manually delete previous files.

### # %% 12 Create empty 2d point file, to show well locations in cross section space

Creates empty 2D point file and adds necessary attribute fields. If the user chooses not to create this file, nothing will execute in this section. arcpy.env.overwriteOutput = True allows the script to be re-run without having to manually delete previous files.

### #%% 13 Create feature dataset to store wwpt files by xs

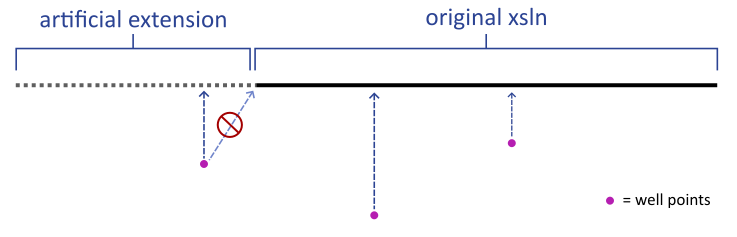
In order to ensure that each well references the correct cross section line, the well point file will be split into multiple files based on their cross section ID number. This section creates a feature dataset for temporary storage of these files, and makes a copy of the well point file in this feature dataset. Creating a copy of the well point file means that the original data is *not* modified in case of errors. arcpy.env.overwriteOutput = True allows the script to be re-run without having to manually delete previous temporary files.

### #%% 14 Add fields to temporary wwpt point feature class

This section adds fields that are later populated by the near analysis tool in section 16. “NEAR\_DIST” will store the distance from the cross section line (in meters), and “NEAR\_X” and “NEAR\_Y” will store the UTM coordinates for the point along the cross section line that is closest to the well. (Note that cross section lines will be artificially extended to the east and west in section 15 to correctly handle wells that are past the extent of the original cross section line.)

### #%% 15 Create a temporary xsln file and extend the lines equal to xsln spacing

This section will create a copy of the mapview cross section line file, and then artificially extend the line to the east and west. This ensures that wells beyond the extent of the cross section line will be plotted correctly in cross section view. The lines are extended equal to the cross section spacing variable defined in section 3. The exact length of the extension is not important; it only matters that it extends far enough to capture all wells associated with the cross section. Below is a graphic schematic of how this extension will work:



First, the code creates a blank polyline feature class and adds cross section ID fields. The search cursor loop goes through each original cross section line one at a time. It defines the coordinates for each vertex of the line. It uses angleAndDistanceTo to define the angle of the first and last segments. Then, it uses pointFromAngleAndDistance to define a point that is a specified distance (equal to cross section spacing) from the start and end point along the same angle as the segment. It redefines the first and last vertex based on these new calculations, and creates a line in the temporary cross section line file based on the new calculated vertices. Note: this does NOT modify the original cross section line file. It only reads the geometry of the original line and then creates the extended lines in a new feature class.

### #%% 16 Populate near analysis fields in wwpt file

This section will split the copied well point file (create in section 13) into individual files based on cross section number, and then populate the near analysis fields (added in section 14) based on the temporary extended cross section line file. It does NOT yet create geometry for well stick diagrams, it simply calculates where each well should be plotted along the cross section line.

SearchCursor is used to loop through the temporary extended cross section lines, one line at a time, and creates a geometry object for the line. (The geometry object is necessary for the near analysis function to work.) Then, it uses an SQL where clause to select (from the well point file in the temporary feature dataset) all well points with the current cross section id number. It creates a feature class in the temporary feature dataset with only wells for the current line. arcpy.analysis.Near uses the cross section line geometry object and the new well point feature class to populate near analysis fields. Four fields are populated: NEAR\_FID, NEAR\_DIST, NEAR\_X, and NEAR\_Y. NEAR\_X and NEAR\_Y are the UTM coordinates of the nearest point on the line. NEAR\_DIST is the distance between the original well point and the point on the line.

### #%% 17 Delete wwpt\_temp from feature dataset

Since there are now individual well point files for each cross section in the feature dataset, the temporary well point file is deleted from the feature dataset. This file contains all of the wells, but the near analysis fields are empty.

### #%% 18 Merge together wwpt by xs files into one file

All well point files in the temporary feature dataset are merged together. You now have only one well point file, with attributes that show where to plot the wells along the cross section line.

### #%% 19 Create 2d well point geometry from merged wwpt file

This section only executes if the user chose to create a 2D well point file.

Uses SearchCursor to loop through the merged well point file (section 18) and reads the xy geometry and the following fields: well id field (parameter from section 3), cross section id field (parameter from section 3, 'NEAR\_DIST', 'elevation', and 'mn\_et\_id'. If the elevation field is empty, an error message prints and it skips the well. The index variable is used to keep track of the number of loop iterations. With every 5,000 iterations, a print statement executes so that the user can keep track of progress. Since the x coordinate in the stacked system matches true x coordinate, the UTM x coordinate for the well is the same as the x coordinate for the well in stacked cross section view. The y coordinate for the well in stacked cross section view is calculated with the value in the elevation field (in feet) and the stacked cross section equation. The stacked cross section equation factors in conversion from feet to meters, vertical exaggeration factor, and statewide cross section id number. (To learn more about the stacked cross section equation, see *About Stacked Cross Section Display.docx* in the Stacked Cross Sections *Docs* folder.*)* InsertCursor is used to insert a point into the point file for every well, storing necessary information in attributes.

### #%% 20 Create 3D and 2D polyline geometry from strat and wwpt tables

This section creates geometry in the 3D and 2D polyline files. It does both of them in the same section to save on processing time, and to avoid repeating the same operations. A blank list, “nomatch\_list” is defined that will store any well ID values from the strat table that do not have a matching well point. It uses SearchCursor to loop through the stratigraphy table one row at a time. It reads the following attribute fields: “elev\_top”, “elev\_bot”, well id field (user-defined parameter), and OID (ESRI geodatabase table ID number). The “elev\_top” and “elev\_bot” values are used to define the true elevation values (in feet) of the top and bottom of the strat record. If either of these fields are blank, an error message prints and it skips to the next strat record.

Next, the well id value is used to define a where clause. Two different where clauses are defined with slightly different SQL queries so that it will work with either numeric or text values. The index\_int variable is used to keep track of progress, and prints a progress statement every 1000 records. Using the where clause, SearchCursor is used to read attribute values from the associated well point. It uses the merged well point file created in section 18 that has all of the near analysis fields populated. SearchCursor will read the x and y geometry of the well point, the “NEAR\_DIST” field (distance between the well and the cross section line), cross section ID number (user-defined parameter), and the “mn\_et\_id” field. A for loop is used to read these values, but there will only be one well point to loop through. The variables real\_x and real\_y define the true x and y coordinates of the well. The variable x\_coord defines the x coordinate of the well in the 2D stacked display, which is the same as the true x coordinate. Distance and percent distance from the cross section line are defined to be stored as attributes in the output. The i variable is used to determine if there is no matching well point. Variable i is initially defined to equal zero. If the where clause finds an associated well point, it adds 1 to the value. If i is still equal to zero after searching the well point, the well point is added to nomatch\_list and it continues to the next strat record.

The variables z\_top and z\_bot define the top and bottom of the line segment in the 2D stacked display. (For more information on the equation, see document “About Stacked Cross Section Display.docx”) Variables real\_pointA and real\_pointB define xyz coordinates for the 3D line segment. These coordinates are turned into a point list and array to create a geometry object. The geometry object is used to insert a line segment into the output 3D line file using InsertCursor. The well id, xy coordinates of the well, strat record number, top and bottom elevation, and mn\_et\_id values are all stored as attributes in the output attribute table.

Next, pointA and pointB variable define the xy coordinates for the top and bottom of the line segment in the 2D stacked display. These points are converted to a list and an array, and finally a geometry object. This geometry object is used to create a line segment in the output 2D line file, storing attribute values identical to the 3D attributes, plus distance and percent distance.

Once all stratigraphy records are finished, the script will print a message with the number of strat records that were skipped because a matching well point could not be found. It also prints elapsed time for the geometry creation.

### #%% 21 Create list of stratigraphy fields based on which fields exist and which are relevant

This section lists the attribute fields in the stratigraphy table. The list fields\_not\_to\_join lists out redundant or unnecessary fields that do not need to be added to the output. This list can be edited if needed; however, it will join fields by default if the field is not in the fields\_not\_to\_join list. Redundant or unnecessary fields can always be manually deleted once the tool is complete. A for loop is used to remove fields from the list of relevant strat fields that is used for the join in the next section.

### #%% 22 Join stratigraphy fields to 2d and 3d polyline feature classes

This section joins stratigraphy information to 2D and 3D polyline outputs based on the strat record number.

### #%% 23 Create 2d polygon lixpys from 2d lines

This section creates a 2D polygon file based on the 2D line file using the Buffer tool. The bufferdist variable scales relative to vertical exaggeration, in case the user chooses to create lixpys with different vertical exaggeration. The equation was chosen by trial and error to create a polygon width that looks good at 10x – 250x. The 2D line file is buffered based on this number, and the output is sorted by wellid. This is to ensure that layer drawing order makes sense, so all sections of each well will draw at the same level.

### #%% 24 Delete temporary files/fields

This section deletes all temporary files, including the well point feature dataset, the merged well point file, and the extended cross section line file.

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

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