Summer 8-2018

Recovering Legacy Geological Data into a Geospatial Database Product: An Example from Baja California Norte, México

Alexander C. Audet
University of Maine

RECOVERING LEGACY GEOLOGICAL DATA INTO A GEOSPATIAL DATABASE PRODUCT: AN EXAMPLE FROM BAJA CALIFORNIA NORTE, MÉXICO

by

Alexander C. Audet

A Thesis Submitted in Partial Fulfillment
of the Requirements for a Degree with Honors
(Earth and Climate Sciences)

The Honors College

University of Maine

August 2018

ABSTRACT

This project develops a workflow for the extraction of legacy geological map data using a case study in the Baja California Norte, México by four workers over forty years. This project is unique from other digitization efforts worldwide because the data were already in an unregistered vector format, instead of a raster format. Thus, the methodology used in this project took advantage of this digital format by writing arcpy scripts for use inside of ArcMap, and using database feature manipulation software, in order to streamline the data extraction process, with the goal being to develop methods for dealing with other similar legacy geological datasets. The project was conditionally successful, with the developed arcpy script extracting strike and dip direction information from the structural geological data, with only minimal manual review required. Additionally, implementation of the FME Workbench software allowed text information describing Dip and Plunge to be extracted and combined with its companion direction and position data; however, project limitations only allowed for a method that required extensive manual review after the automated process. Transferability of the developed workflow is limited by requiring access to FME workbench software in addition to ArcGIS, but as that part of the workflow requires substantial manual work, it could perhaps be replaced by completely manual methods. Additionally, the arcpy scripts might not work properly if used on tightly clustered data, or data constructed differently than the Baja California data. Copies of the arcpy script, FME workflows, and maps can be found on https://drive.google.com/open?id=1yWlCXTGTtm1qSN1uWHy6BaP1ZcIWXyco.

DEDICATION

I would like to dedicate this work to my Granny, Margaret McClenaghan, who has always lived across the world from me, yet been one of my greatest supporters from the age I could not read or spell when most other children could. She has continued to encourage me, and we have had many fascinating conversations up through my college education. I would also like to dedicate this to my mother, who has inspired me with a love of literature and learning, and my father who filled my youth with joy, and always tried to teach me how to survive in the world.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Scott Johnson, for his constant support, willingness to answer my questions, even when peppered with them both in person and by email, and patience with a task that stretched from being a semester project into a year-long honors thesis. I would also like to recognize my computer engineering brother, Sebastian Audet, without whom, I would not have been able to remember the coding syntax I had learned many years ago in high school. He stayed up with me on many nights during the code development, keeping me sane throughout the process, and without him, I would never have been able to do half of what I accomplished. Also vital to the success of this project were the online support communities, such as Stack Exchange GIS, ESRI's GEONET, and Safe Software's FME knowledge center, and those knowledgeable people on them, who helped me solve those errors in my script that stumped me for an exhausting twenty-four hours, or brainstorm ideas for completing some process within the workflow. Thank you to my wonderful committee for agreeing to see this project through with me, and for all their advice on how to go about writing a thesis, as well as comments and edits on the work as it was being brought into fruition! This includes Dr. Christopher Gerbi, my co-advisor, Dr. Alicia Cruz-Uribe, Dr. François G. Amar, and Dr. Martin Yates, as well as my advisor, Dr. Scott Johnson. I would also like to acknowledge support from National Science Foundation grant EAR-1727090.

TABLE OF CONTENTS

INTRODUCTION 1
  Digitizing Legacy Data 1
    Raster Data 1
    Vector Data 2
  What is GIS? 3
  The Future of Legacy Data 4
  Case Study 4
    Initial Data 5
    Geological Summary of the Region 8
METHODS 18
  Part 1 - Applying Nearly-Accurate Coordinates to the Map 19
  Part 2 - Establishing a Reference Latitude & Longitude Grid for Warping 22
  Part 3 - Exporting the Vector Layers to ESRI Shapefiles 23
  Part 4 - Extracting Dip-Direction and Trend Azimuths 24
  Part 5 – Extracting Data from Text Objects in the Canvas Layers 37
  Part 6 – Data Extraction Manual Accuracy Check 44
  Part 7 – Editing Geological Unit Polygons 46
  Part 8 – Final Organization and Formatting of Data 46
  Part 9 – Vector Warp 48
  Part 10 – Transformation from NAD27 to WGS84 49
  Part 11 – Producing the Geological Maps 51
  Part 12 – Archiving the Geological Data 51
DISCUSSION 52
  Suitability for the Arcpy Workflow for Use on Other Datasets and Transfer to other Projects 52
  Error Accumulation and Product Accuracy 54
  FME Workbench Evaluation 57
REFERENCES 60
APPENDICES 63

APPENDIX A: WORKFLOW FOR ASSIGNING A COORDINATE SYSTEM TO AN UNREGISTERED MAP IN CANVAS X GIS 2017 64

APPENDIX B: INSTRUCTIONS FOR ARCPY WORKFLOW FOR ASSIGNING AZIMUTH DIRECTIONS TO THE STRUCTURAL GEOLOICAL DATA..................65

Multi-object Workflow 71

Single-object Workflow..78

APPENDIX C: REGISTRATION SHIFT INSTRUCTIONS..79

APPENDIX D ARCPY SCRIPTS 81

Setting Dipandlength 81

Foli_Prelim_Analysis 82

AngleDifference..83

SelectedNumber 84

WithinDist...85

Line_Prelim_Analysis...86

LineSelectedNumber...87

LineWithinDist 88

Working Code Foliations..89

FoliationModule1 92

FoliationNotes1 97

FoliationModule2...100

FoliationNotes2 103

Working Code Lineations..105

LineModule1 107

LineNotes1 110

LineModule2..112

LineNotes2 113

SymbologyModule...114

Registration Shift 117

AUTHOR’S BIOGRAPHY...118

SUPPLEMENTAL MATERIAL ON CD ....Inside Rear Cover

LIST OF TABLES

Table 1 Important Structural Geological Measurement Terminology 5
Table 2 Steps of the Project Workflow 19
Table 3 Structural Geological Measurement Shapefile Fields 47
Table 4 Geological Units Shapefile Fields 48

LIST OF FIGURES

Figure 1 Geological Map of the Sierra San Pedro Mártir Section of the Peninsular Ranges Batholith in Baja California – Preproject Canvas X Version 7
Figure 2 Example: Gap Between Units in the Canvas X Map 8
Figure 3 Multiple MMt Locations 9
Figure 4 Cerro de Costilla Tonalite Crosscutting the MMt – Conflicting Layers.............10
Figure 5 Example: Dense Foliation Data 11
Figure 6 Extent of Peninsular Ranges Batholith on the Baja California Peninsula 12
Figure 7 Geological Map of the Sierra San Pedro Mártir Section of the Peninsular Ranges Batholith, Baja California México..13
Figure 8 Faults Dividing the PRb into Eastern and Western Terrain...............................15
Figure 9 Base Topographic Raster Map 20
Figure 10 Matching the Vector Object's Scale to the Geospatial Document Scale..........21
Figure 11 Multi-Object versus Single-Object Foliation Symbols....................................26
Figure 12 Arcpy Code Overview Flow Chart 28
Figure 13 Working Code and Foliations Module Multi...30
Figure 14 FoliationModule and Notes 1 31
Figure 15 Correct Foliation Direction 34
Figure 16 Example: Label Far Away From Symbol 39
Figure 17 Example: Too Many Labels 40
Figure 18 FME Workbench Workflow 1 41
Figure 19 FME Workbench Workflow 2 43
Figure 20 Preliminary Analysis Workflow 72
Figure 21 Working Code Foliations and Lineations Multi- & Single-Object Workflow .74

INTRODUCTION

One of the tenets of science is that each scientist will build on the legacy of their predecessors leading to a more complete understanding through time. For future scientists to build upon past discoveries, these discoveries must be recorded and accessible. James Hutton was the founder of modern geology, and in the 220 years since his passing, a vast archive of geological data has been lost or forgotten because it is not readily accessible to present-day geologists. The National Science Foundation considers saving legacy and modern Earth science data to be a high priority, and this is explicit in their policy that no grant proposals will be considered without appropriate data management plans.

Digitizing Legacy Data

Raster Data

This thesis seeks to develop and document an efficient protocol for saving legacy geological data. In particular, legacy geological data can come in at least two formats, as hardcopy geological maps, geological field notes, and digital raster images, or as digital vector maps not connected to any database, or even georeferenced. Raster images are simply pictures composed of pixels, whereas graphics made of vectors are composed of discrete objects or shapes such as points, lines, polylines, or polygons that show the position or shape of features on the map. A lot of work has already been done by, for example, educational institutions and the Maine Geological Survey (Christian Halsted, February 21, 2018), to digitize hardcopy geologic data and raster images. The methods include scanning the map if this has not already been done, using markers on it to georeference and warp the resulting raster image, and then creating vectors by tracing the geological contacts between units and manually placing the structural elements on top of their locations on the raster image. The ultimate goal is to place the data into a format accessible by GIS (geographic information systems). The Geological Survey of Western Australia has implemented its own unique workflow for saving a particularly extensive archive of its own geological data collected over 125 years, and placing it into its own set of databases accessible for querying by the public through their GeoVIEW.WA website (Riganti et al., 2015). For example, they have collected and saved old glass photographic negatives, mineral exploration reports and even Christmas cards, tagging the data for all, when appropriate, with a geographic location.

There is also a growing field of automated map vectorization implementing crowd sourcing, as well as sciences such as image recognition and deep machine learning (Uhl et al., 2018). The goal is for the program to recognize individual objects printed on scanned maps, and create machine-readable, geospatial vector objects that replicate the relevant size and shapes of these objects, with enough attribute data to describe everything that can be determined about the objects by a human eye (Uhl et al., 2018). The program should do so in a way that minimizes manual post-processing (Uhl et al., 2018). This way, computers will be able to automatically digitize thousands of archived historical maps that might otherwise never be manually digitized due to practical limits of funding (Uhl et al., 2018).

Vector Data

This project seeks to streamline a less common scenario where the hardcopy data has already been entered as vector objects into a digital format, but has not been georeferenced, put into a database format, or made accessible to other scientists through

GIS. We use a case example of one such dataset from the northern Sierra San Pedro Mártir region of Baja California, México to develop a workflow and evaluate the feasibility of the implemented solution.

What is GIS?

GIS (geographic information systems) is a database system designed to store, manage, analyze, and view geospatial data. The geospatial data has coordinates, that within a defined coordinate system, are meant to represent the data's relevant position on the surface of the earth, on another planet, or within the solar system. The geospatial data is often viewed and worked with in the form of a map or 3D scene. Additionally, the geospatial data often comes with tabular attribute data describing each object or record within the geospatial database. It is important to note that geographic information systems are not always composed of just software and can include the people and methods used to organize and analyze the geospatial data. In this project, we used the ArcGIS suit of GIS software, and of these, mostly the ArcMap program.

It is important to note that GIS is not just mapping or cartography software (Dempsey, 2018), but instead must be able to spatially analyze and edit its data, producing new data from this analysis. This can be facilitated through either SQL (Structured Query Language) querying (the structured request for the records matching specific criteria from a database), visually from the layers of information placed on the map display, or using other spatial or database analysis tools.

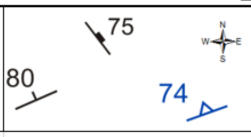
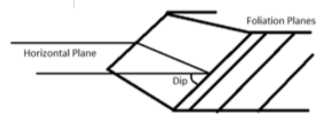
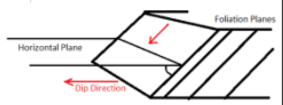
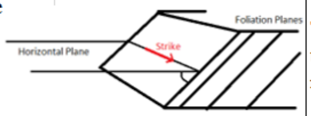
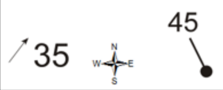
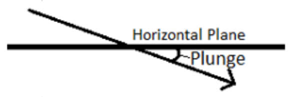
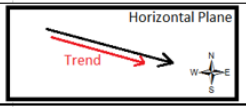
The Future of Legacy Data

An important component of the digitization of legacy data is to make sure that it is put into a format that can be easily accessed, both now, and indefinitely into the future, and then archived where the data will not be lost, but can still be easily accessed by the public. Proprietary software such as ArcGIS might update and phase out past file formats or might disappear entirely fifty or a hundred years from now. However, they have recently given up their proprietary rights to the shapefile format because it has become an industry standard (Christian Halsted, August 01, 2018). Thus, the data can be archived within that format.

Case Study

The geology of approximately 1200 km^2 along the Main Mártir Thrust (MMt) in the northern Sierra San Pedro Mártir region of Baja California, México was mapped, recording rock types, geological contacts, foliations, lineations and bedding (Table 1 explains much of this geological terminology), over a period of 10 years, from 1995-2005 by Drs. Scott Johnson et al. (1999a, 1999b, 2002, 2003, 2004), Erwin Melis (2006), and Gabriel Chávez Cabello (1998). Part of the region had been mapped earlier by Dr. Jay Murray (1978). Together, these workers collected more than 6185 structural measurements, primarily in a portion of the larger area comprising approximately 700 km^2. Prior to this detailed mapping, the area was covered in reconnaissance as part of a major effort by Dr. Gordon Gastil to publish a 3-sheet map set of the northern half of Baja California (Gastil, et al., 1973).

Table 1 Structural geological terms used within this workflow. Terms are described as used in this project and the descriptions don't necessarily match definitions agreed upon within the wider geological community.

Structural Geological Terminology		
Foliations		Foliations are structural planar features that are created from either magmatic flow, as symbolized by the top symbol with a square; the deposition of sediments into distinct layers, the left symbol with a line; or solid-state deformation, the triangular symbol to the right. The dip, dip-direction, and strike of these features is measured. (Image in map view)
Dip		The dip is the downward angle that the plane makes with the horizontal and is shown as the numbers in the foliation symbols above.
Dip Direc-tion		The dip direction is the direction of the dip, or the azimuth that water would flow down the planar surface, and is shown by the object attached to the center of the longer line in the Foliations symbols shown.
Strike		The longer line shows the strike, and is the azimuth direction of a horizontal line perpendicular to the dip. The direction is defined by the right-hand rule as that which puts the dip to its right.
Lineations		Lineations are structural line features that are created from either magmatic flow, solid-state deformation, or the intersection of two planes. The plunge, and trend of these features is measured. (Image in map view).
Plunge		The plunge is the downward angle from the horizontal of the lineation and is shown as the numbers on the lineation symbols above. (Image in a profile view).
Trend		The trend is the horizontal azimuth of the lineation pointing downwards, and is shown by the direction of the arrow or line coming out of a circle in the lineation symbols. (Image in map veiw).

Initial Data

The data from these mapping efforts were in a difficult-to-access, poorly organized state at the beginning of the project. Most of the data were in a Canvas X 2017 GIS file in the form of numerous vector layers as shown in Fig. 1, though some lingered in excel spreadsheets. Canvas is a graphics editing software created by ACD Systems International Inc., similar to Adobe Illustrator, but with limited cartographic capabilities.

These canvas layers came from a variety of sources, including Adobe Illustrator files and ESRI shapefiles, and some of the data were originally spatially offset from the rest of the data. The data were not georeferenced; however, the vector shapes were drawn on top of a raster image of a set of topographic maps, and thus could be assigned a reference frame using the coordinate crosshairs on the background image.

Moreover, the vector data itself was unorganized. The units were not drawn flush with each other on their contacts (Fig. 2) and some units could cover others if the layers were placed in the wrong order, making it difficult to move the layers from one software to another, or open them in a new map. Some of the layers conflicted with each other as there was more than one interpretation of the location of the MMt (Fig. 3). Other polylines existed that had been polygons and now overlapped each other, making the true contact between them vague. Some existing layers were no longer useful, and others did not represent the current accurate understanding of the area’s geology, such as at the point where the Cerro de Costilla tonalite overlaps the MMt, as shown in Fig. 4. Many of the layers needed to be combined as they had been added piece by piece over the years, but represented the same type of measurements. Some of those measurements, however, did not have surviving hardcopy records, and themselves were visually challenging to interpret as so many measurements had been drawn on top of each other that it was nearly impossible to match the structural geology foliation or lineation symbol with the number specifying the measurement’s dip or plunge.

Methods such as writing python ArcGIS (arcpy) scripts to extract geospatial data from vector images (view within Appendix D), and using FME workbench database management software to extract and assign text objects, were used to combine these

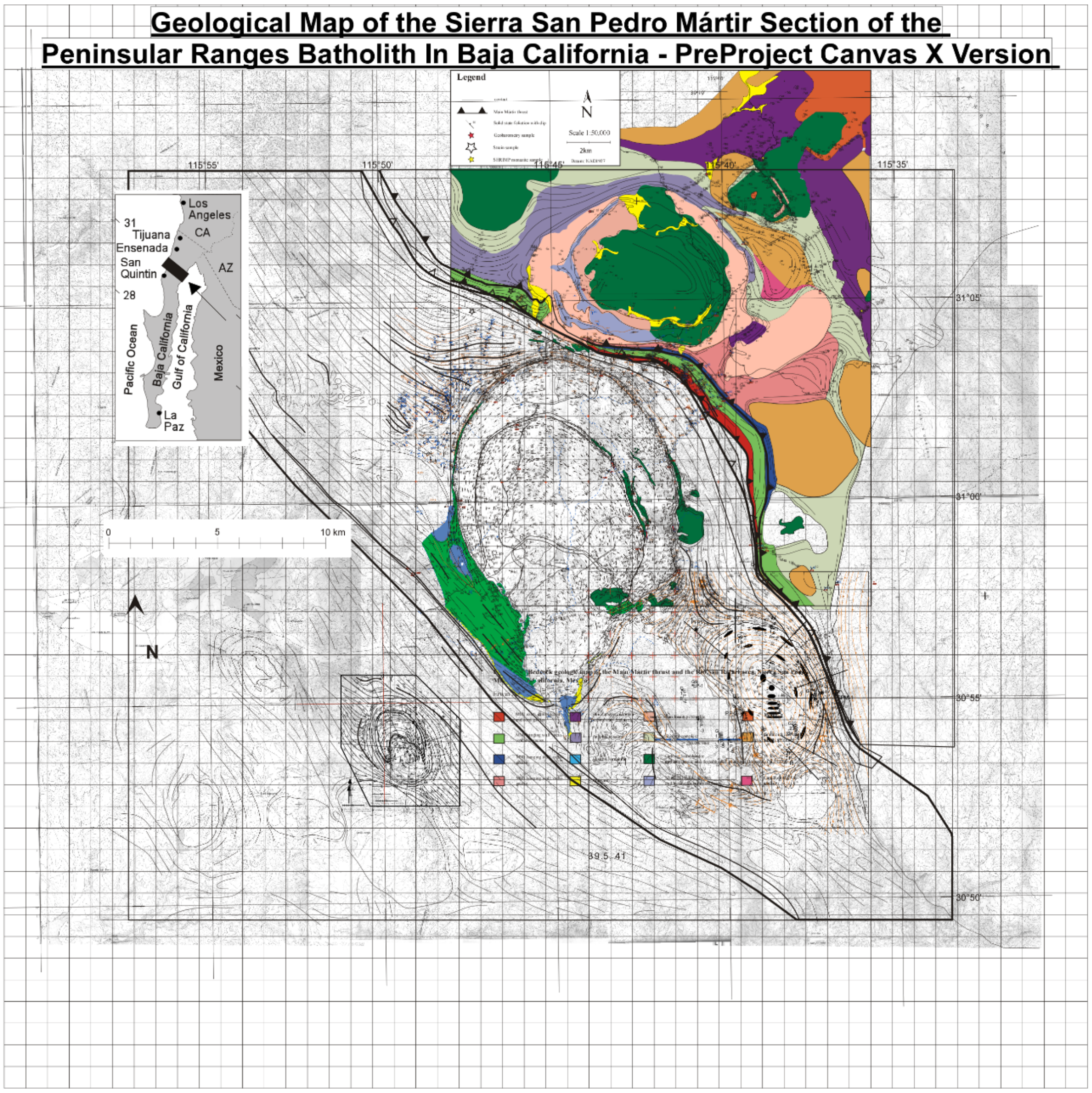


Figure 1 The original Canvas X GIS 2017 map containing all the data from Dr. Jay Murray (1978), Dr. Scott Johnson et al. (1999a, 1999b, 2002, 2003, 2004), Dr. Erwin A. Melis (2006), and Dr. Gabriel Chávez Cabello (1998) compiled into a single document.

disparate elements into an organized format accessible by GIS. Thus, these data will be made not only accessible, but useful for future geologists. Digital 58 by 91 cm PDF geology, foliation, and lineation maps, produced by the project, as well as digital copies of all the scripts and FME workbench workflows, can be found on the CD included in the

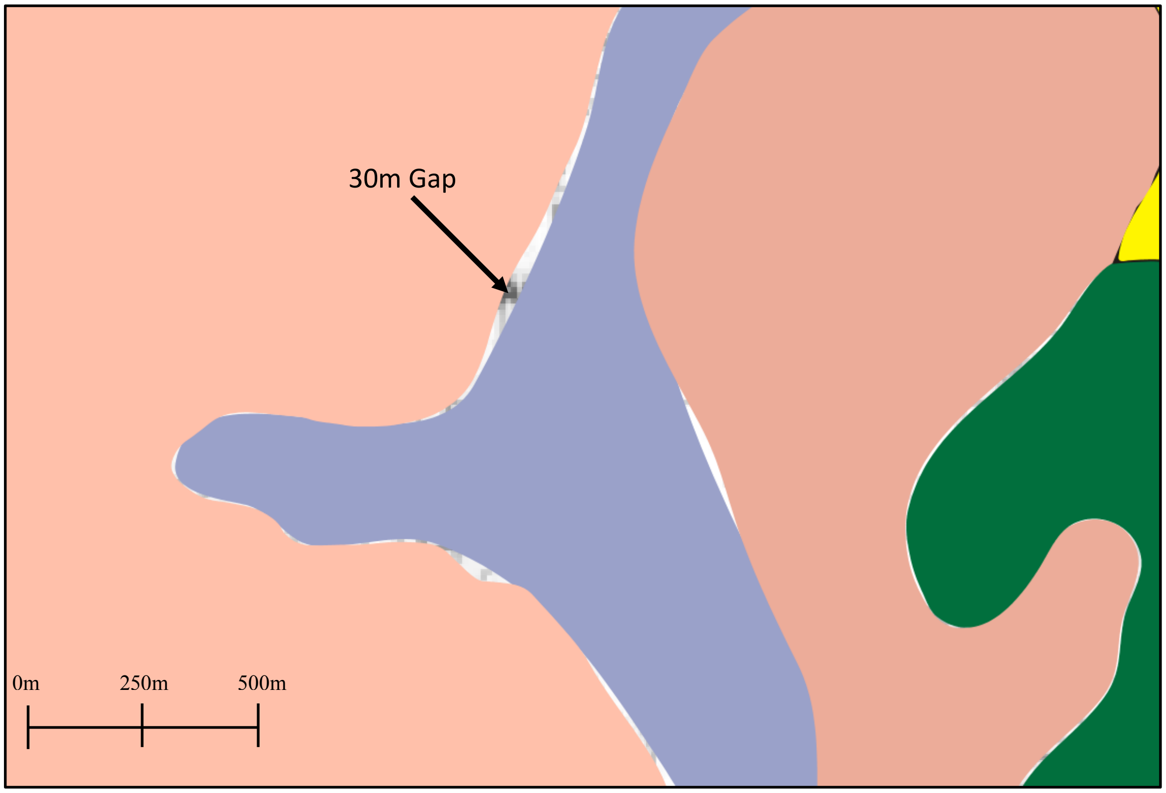


Figure 2 Image illustrating a typical offset found between adjacent polygons representing geological units within the original Canvas X map. In this case, the largest offset is about 34 meters wide. This value approaches the largest offset width within the data.

back of this thesis as well as at:

https://drive.google.com/open?id=1yWlCXTGTtm1qSN1uWHy6BaP1ZcIWXyco.

Geological Summary of the Region

The study site is located in the Mesozoic Peninsular Range Batholith (henceforth known as PRb), on the Baja California peninsula. The PRb comprises a group of mountain ranges that stretch approximately 1600 km from southern California to the southern tip of the Baja California Peninsula (Fig. 6). In the study area, the western portion of the PRb comprises volcanic and volcanogenic rocks of island arc affinity, intruded by a range of plutonic bodies, the largest of which are granitic in composition.

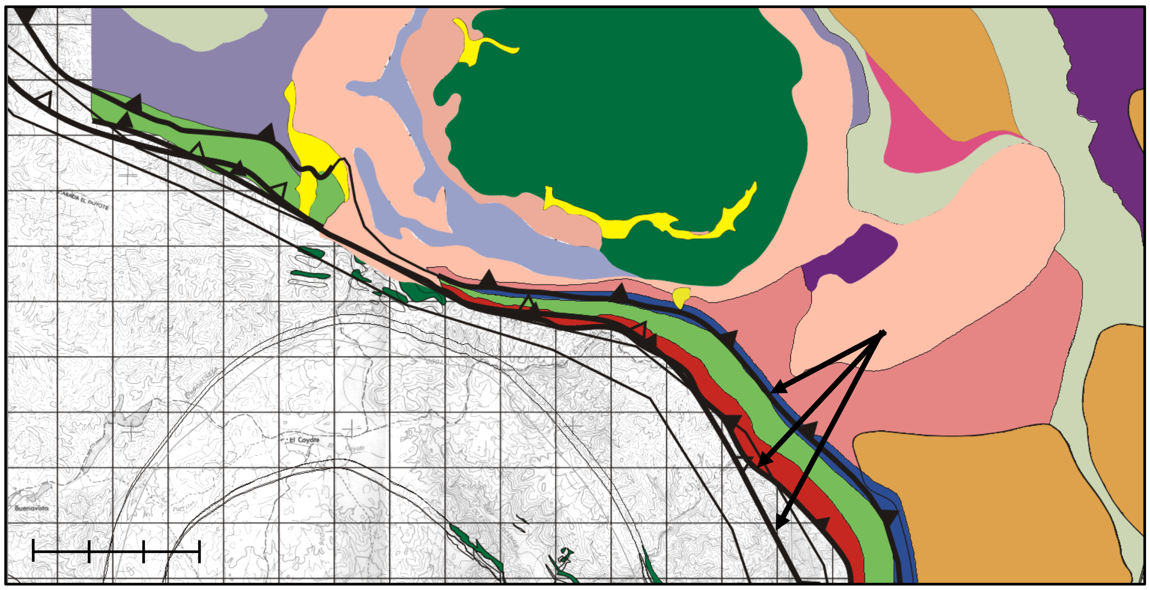


Figure 3 Snapshot of the original Canvas X map of the Main Mártir thrust (MMt), just below the Cerro de Costilla complex, showing that there were originally three different interpretations of where the MMt was located, offset from each other by hundreds to thousands of meters.

Specifically, the study area in the northern Sierra San Pedro Mártir (SSPM), as can be seen in Fig. 7, the final map produced by this project, is centered on a section of the Main Mártir Thrust (MMt), an ancient fault that juxtaposes the western and eastern PRb (Johnson et al., 1999a). In fact, as seen on Fig. 8, a series of sutures like this one run up and down the PRb, separating volcanic island terrain such as the Santiago Peak, and Alisitos arcs from the ancient continental margin.

The MMt juxtaposes upper greenschist/lower amphibolite facies schist of the 115 +/- 1 Ma Alisitos Formation (Johnson et al., 1999a; Carrasco et al., 1995) to the southeast against higher temperature, upper amphibolite facies migmatitic gneisses to the northeast (Johnson et al., 1999b; Melis, 2006) indicating a reverse fault with NE-side-up motion (Melis, 2006). Microstructures, including shear bands showing northeast over southwest

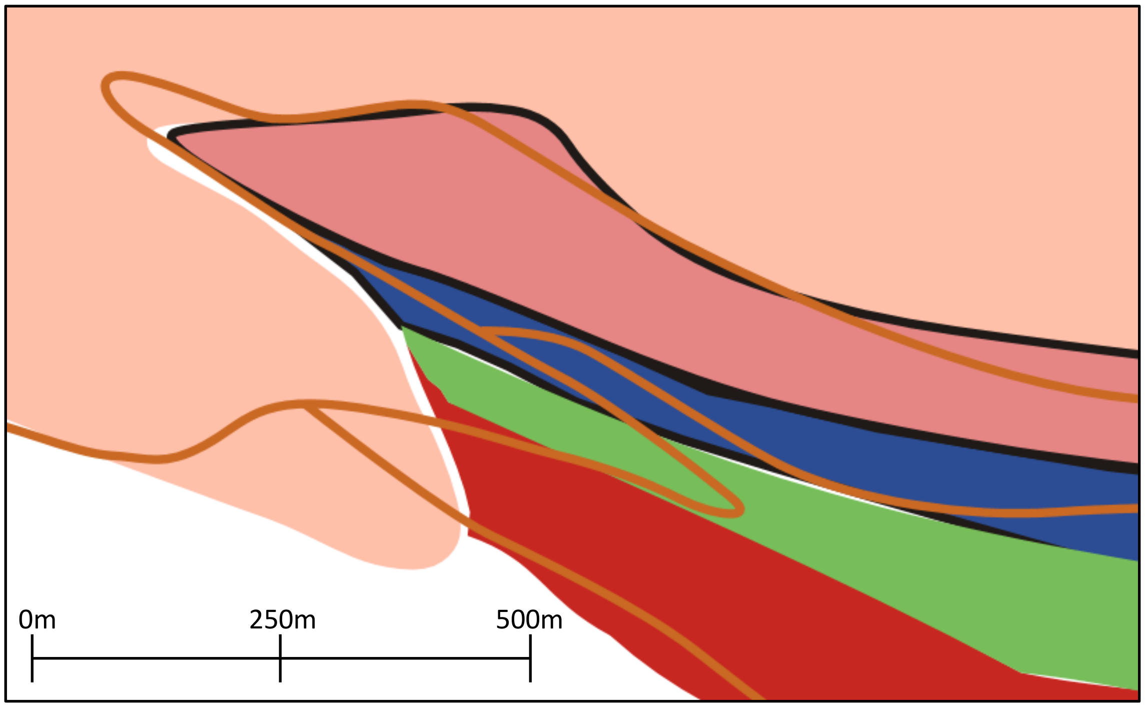


Figure 4 A close-up of the original Canvas X map at the Main Mártir thrust (MMt), where the outer ring of the Cerro de Costilla tonalite, shown in peach, appears to cut across the MMt. Since the MMt cannot be shown due to issues of clarity, the MMt can be imagined to follow the outer edge of the red unit (cordierite gneiss), locally disappearing under the tonalite. The issue shown here is that the polygons do not match the orange lines, which here represent a better understanding of the geology of the region. Thus, the polygons need to be edited to follow these lines.

shear, are consistent with this interpretation. The Alisitos Formation is composed of transitional subaerial to subaqueous volcanic, volcaniclastic, and associated sedimentary deposits (Melis, 2006). Locally, these include reef limestone (Johnson et al., 1999a) where a marine basin probably formed prior to collision. The ages of each terrane moving

The MMt juxtaposes upper greenschist/lower amphibolite facies schist of the 115 +/- 1 Ma Alisitos Formation (Johnson et al., 1999a; Carrasco et al., 1995) to the southeast against higher temperature, upper amphibolite facies migmatitic gneisses to the northeast (Johnson et al., 1999b; Melis, 2006) indicating a reverse fault with NE-side-up motion

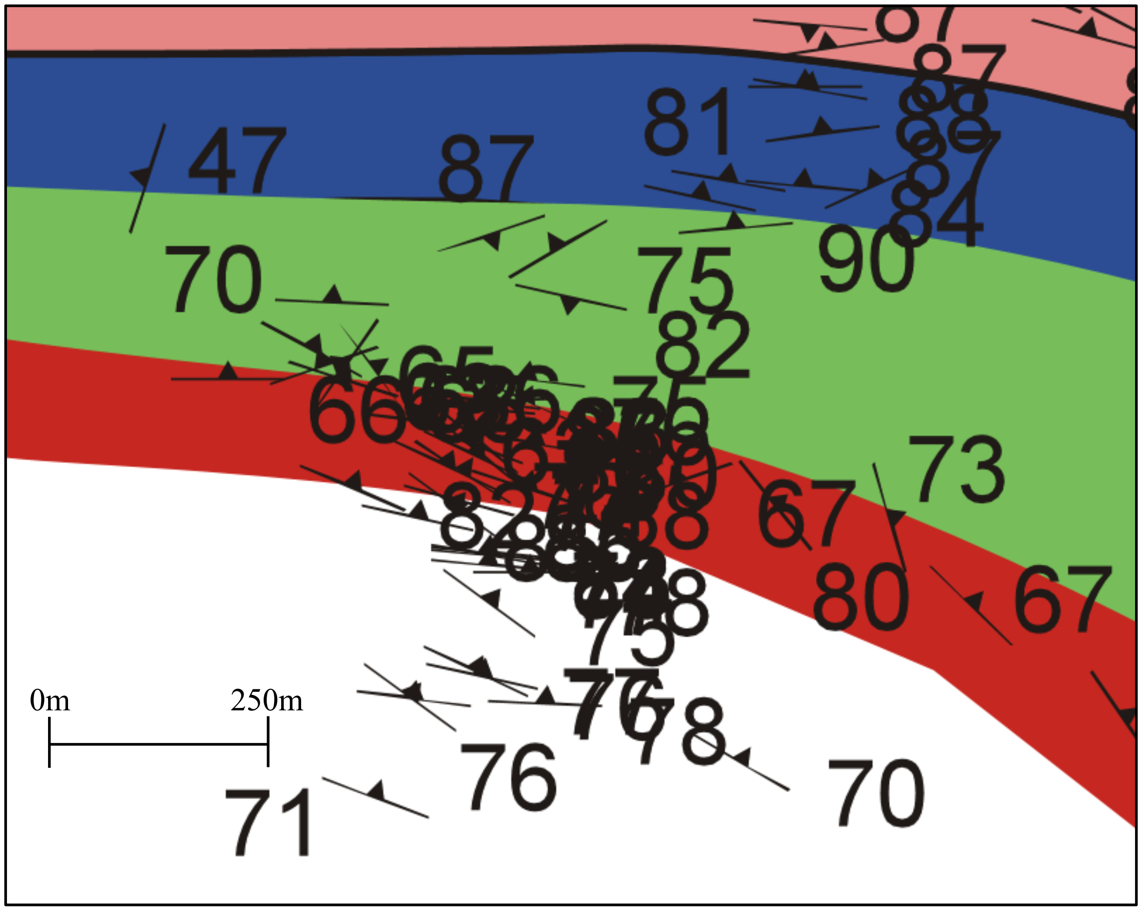


Figure 5 Snapshot of the original Canvas X map showing foliation measurements found within one layer. Note that measurements are so densely spaced that matching the correct number representing the measured dip with the foliation symbol is nearly impossible, and showing the dip orientation is difficult.

(Melis, 2006). Microstructures, including shear bands showing northeast over southwest shear, are consistent with this interpretation. The Alisitos Formation is composed of transitional subaerial to subaqueous volcanic, volcaniclastic, and associated sedimentary deposits (Melis, 2006). Locally, these include reef limestone (Johnson et al., 1999a) where a marine basin probably formed prior to collision. The ages of each terrane moving outwards from the MMt increases with the Alisitos Formation reaching at minimum 144 Ma in its western extent (Alsleben, 2005), and a significant portion of the PRb to the east

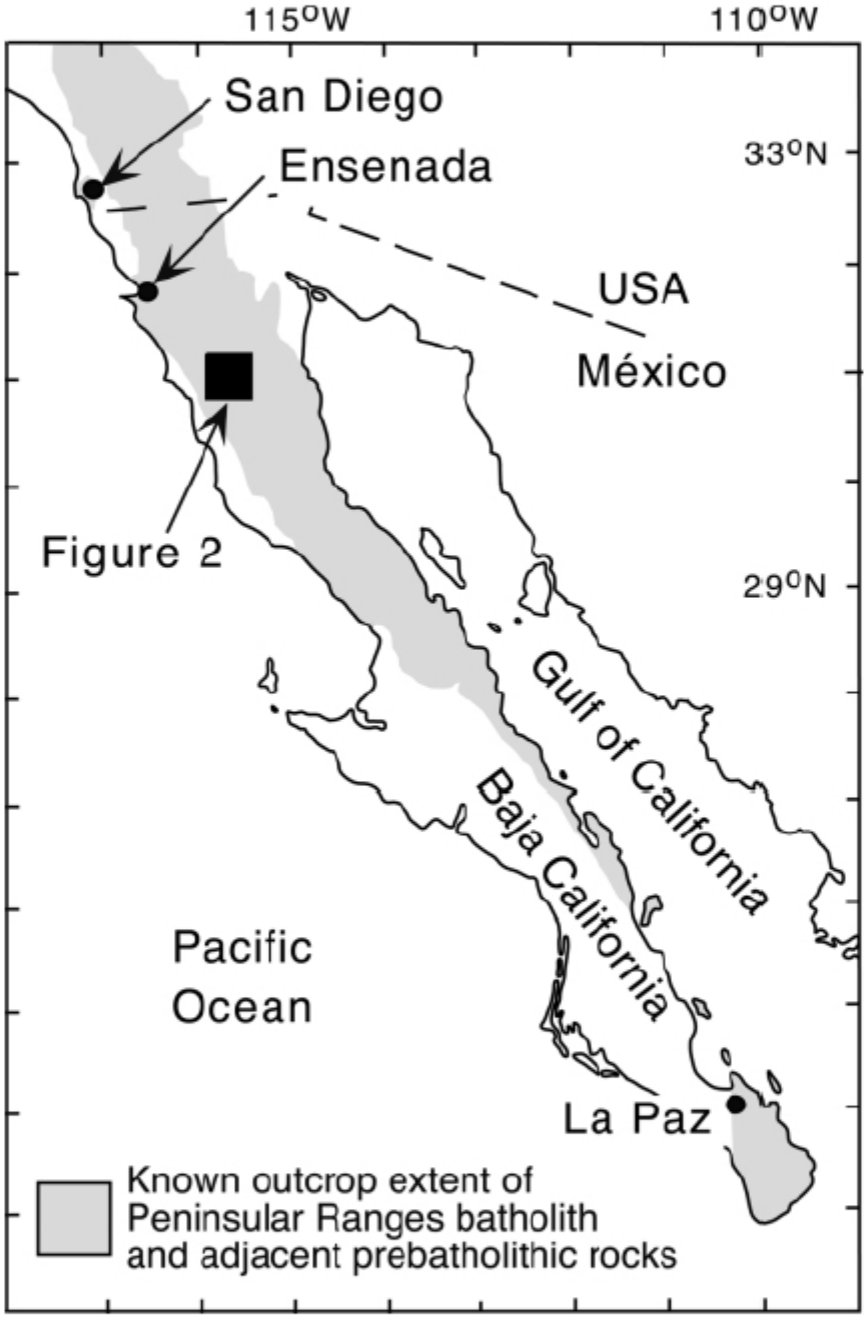
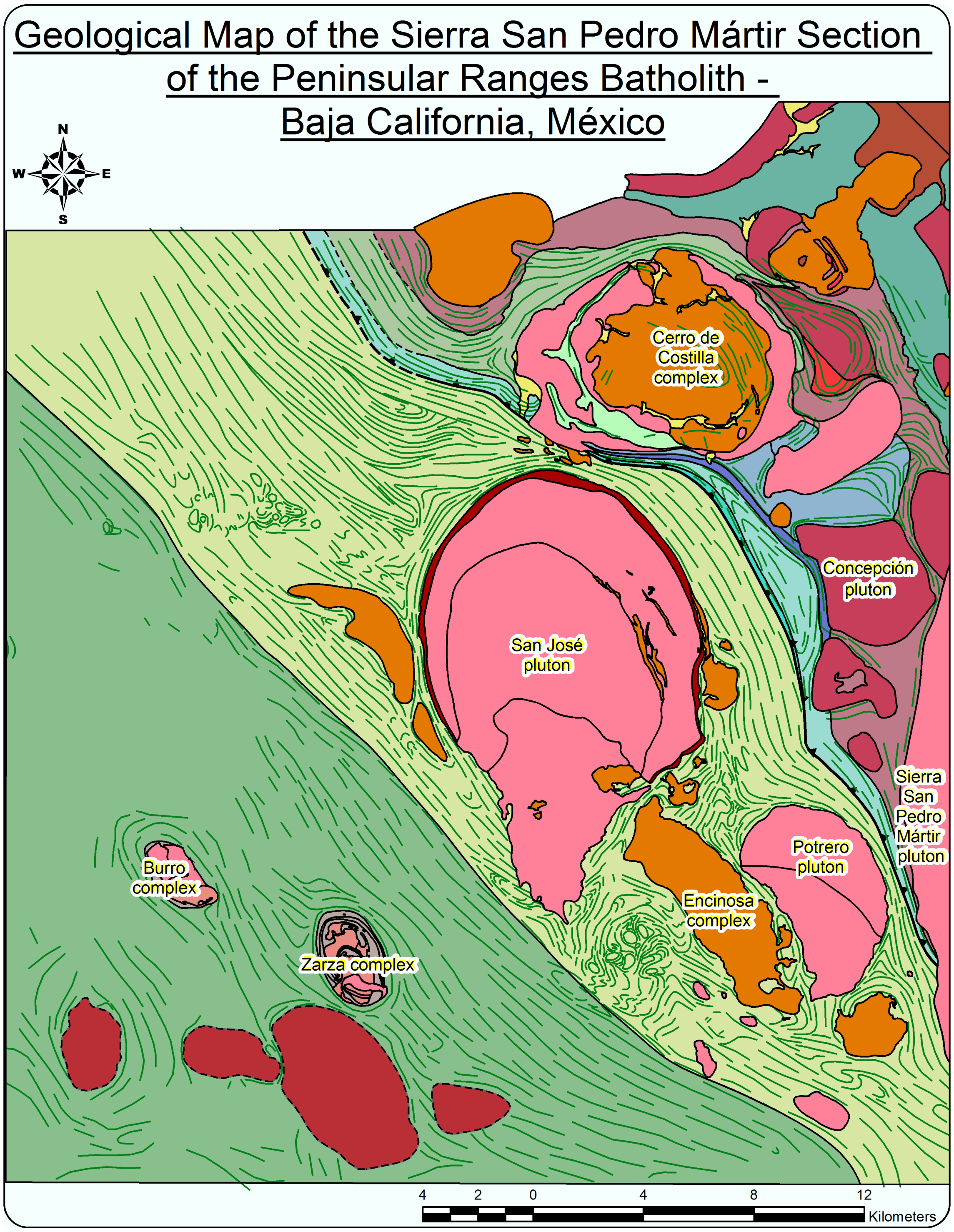


Figure 6 Map showing the entire length of the Peninsular Range Batholith (PRb), extending from Irvine, in southern California, to Cabo St. Lucas, México, the southern tip of the Baja California Peninsula. Image reproduced with permission from Johnson et al. (2003).



Geological Map of the Sierra San Pedro Mártir Section of the Peninsular Ranges Batholith - Baja California, México
N
W
E
S
Cerro de Costilla complex
Concepción pluton
San José pluton
Sierra San Pedro Mártir pluton
Potrero pluton
Encinosa complex
Burro complex
Zarza complex
4
2
0
4
8
12
Kilometers

Figure 7a Geological map of the Sierra San Pedro Mártir region where the data from this project were collected. See Figure 7b below for legend, location map and supplemental information.

Geological Map of the Sierra San Pedro Mártir Section of the Peninsular Ranges Batholith - Baja California, México - Legend

Legend

Geological Units

🟩	Alisitos Formation-Marine Basin
🟩	Alisitos Formation-Volcanic Arc
🟨	Alluvium
🟩	Andalusite phyllite
🟥	Gabbro
🟥	Garnet+/-muscovite alaskite
🟥	Granitoids and tonalites, locally deformed
🟫	Granodiorite-diorite
🟥	Hornblende-biotite tonalite
🟩	Huico gneiss
🟦	MMt hanging wall: metavolcanic gneiss
🟦	MMt hanging wall: orthogneiss 1
🟦	MMt hanging wall: orthogneiss 2
🟦	MMt zone: garnet schist and cord. gneiss
🟫	Mafic conesheet assemblages
🟥	Metasedimentary enclaves
🟩	Mingled tonalite, gabbro and metasedimentary enclaves
🟥	San José Pluton - Solid State Deformed Outer Ring
🟪	Sheeted intrusive complex
🟥	Tonalite enclaves
🟥	Unmapped igneous intrusions
🟧	Variable gabbro tonalite and hypabyssal
—	Trends

Main Martir thrust (MMt)

▼ - -	Inferred
▼ —	Known

Geological Contacts

—	Known
- - - -	Inferred

Prepared by: Alexander Audet,
Dr. Scott Johnson
Preparation Date: 07/20/2018
Projection: UTM zone 11N WGS 1984
Source: United States Census Bureau,
mhoel@ucc.on.ca, Alexander Audet,
(Johnson et al. 1999a, 1999b,
2002, 2003, 2004) (Melis, 2006),
(Chávez-Cabello, 1998) (Murray, 1978).

Location Map

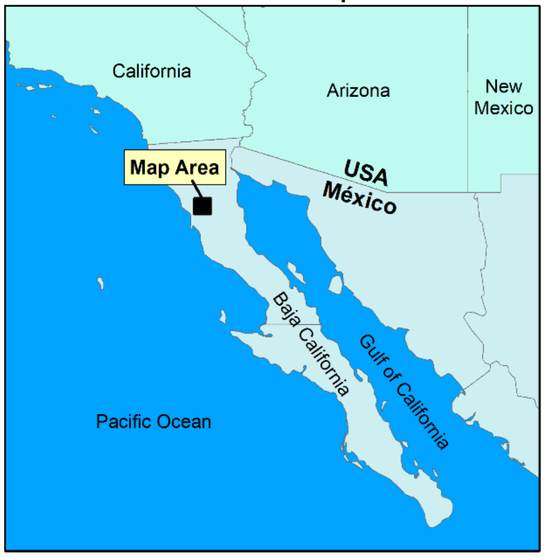


Figure 7b Legend and location map for the geological map of the Sierra San Pedro Mártir region where the data from this project were collected.

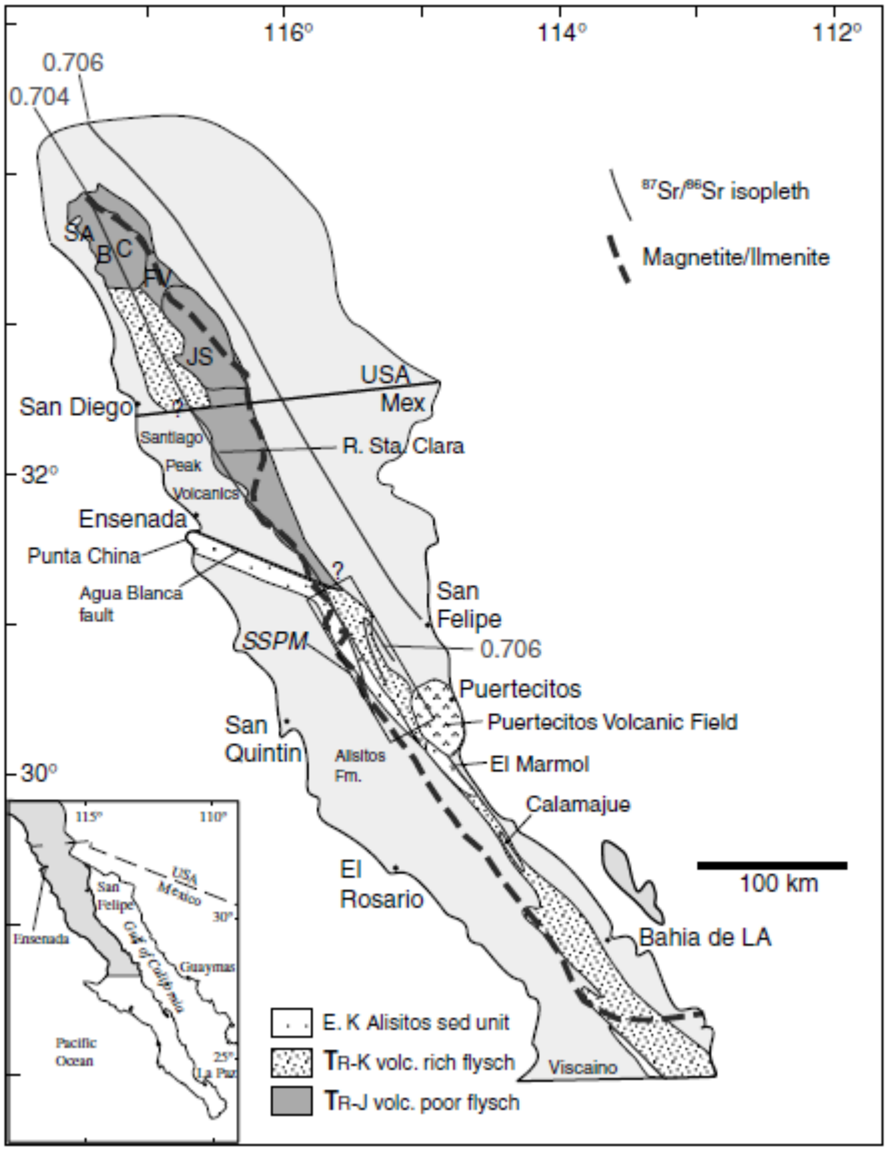


Figure 8 Map illustrating that the Alisitos Formation and Santiago Peak Volcanics abut older continental material, such as volcanic rich flysch formations, thus dividing the Peninsular Range Batholith (PRb) along its length into eastern ancient continental margin, and western accreted arc terrain. Lines marking a shift from ilmenite to magnetite, and the $^{87}Sr/^{86}Sr$ isopleth indicate the approximate position of the suture between these two distinct terrains. Image reproduced from Schmidt et al. (2002).

being composed of a Jurassic arc. In the far east of the field area, the Huico gneiss gives a U-Pb zircon age between 220-230 Ma (Melis, 2006).

The collision and thrusting along the MMt must have occurred sometime between 115 Ma, the depositional age of the Alisitos Formation (Johnson et al., 1999a), and 103 +/- 1 Ma, the age of the Cerro de Costilla tonalite that cuts across the MMt, without showing signs of post-emplacement deformation (Johnson et al., 1999b). If the MMt had continued to move after the Cerro de Costilla tonalite intruded it, the tonalite would have been deformed. Farther south, a reverse fault, possibly equivalent to the MMt shows evidence of remaining active until possibly 85 Ma (Schmidt, 2000). North of the study site, where the MMt might meet the ancestral Agua Blanca Fault, the relative vertical movement across the fault appears to be only a few km between about 105-108 Ma (Wetmore et al., 2003), suggesting it also might have had yet another timeline of movement and creating a still more complex regional tectonic history that requires further work to parse.

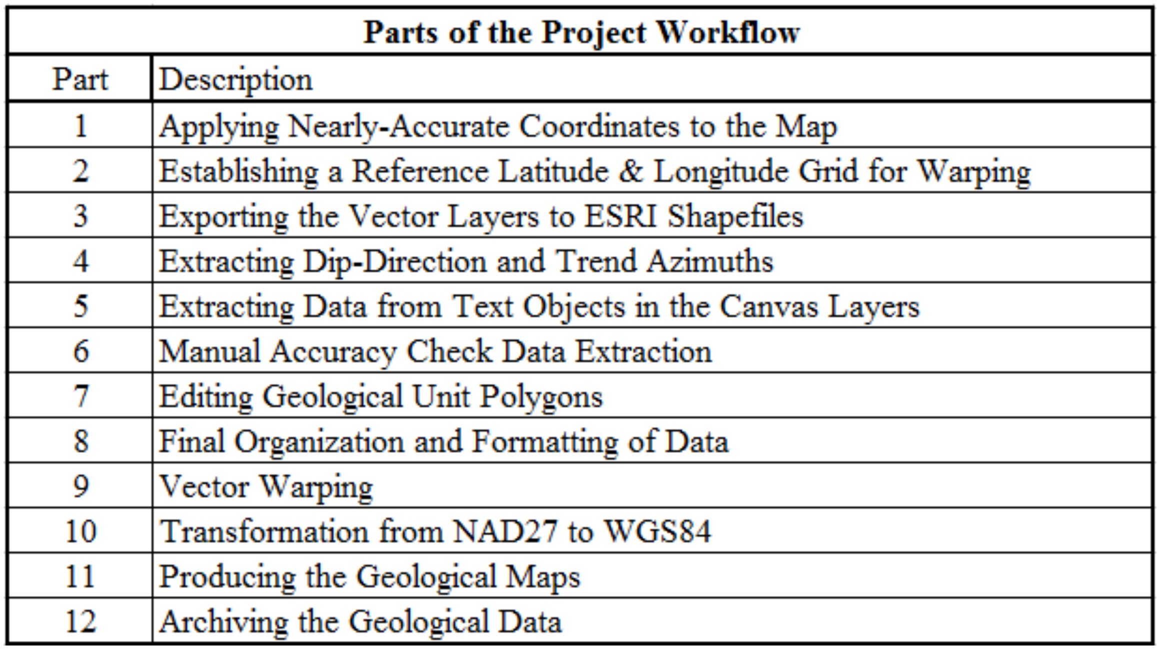
One of the most prominent features of the region are the plutonic intrusions, igneous bodies that solidified within the crust before being exposed at the surface, which vary west to east across the MMt suture (Gastil et al., 1975). Zircon U-Pb isotopic dating gives three age clusters: 135-120 Ma, 110-100 Ma, and 97-92 Ma, found uniformly across the area (Schmidt, 2000; Todd et al., 2003). The ^{40}Ar/^{39}Ar and ^{40}K/^{40}Ar dates on the other hand, which show the age the plutons cooled below particular temperatures, do tend to decrease from west to east (Evernden and Kistler, 1970; Armstrong and Suppe, 1973; Krummenacher and Doupont, 1975). Since the cooling ages provide a proxy for the cooling rate by the calculated difference between these ages and the emplacement age, and the emplacement ages are constant, the ^{40}Ar/^{39}Ar and ^{40}K/^{40}Ar dates indicate the cooling rates decrease from the west, where is it often only takes 5 m.y. to cool to the required temperatures, compared to the east, where it takes between 15-25 m.y., presumably because the greater depth of emplacement makes it so that the eastern plutons were in a deeper, hotter part of Earth's crust for a longer period (Lovera et al., 1999; Schmidt, 2000). Additionally, gabbro with tonalite and diorite are more common west of the MMt, whereas La Posta-type concentrically zoned plutons of hornblende-biotite tonalite to muscovite-biotite grandodiorite dominate east of the MMt (Gastil et al., 1975; Walawender et al., 1990, 1991).

Amongst the remaining puzzles are how the MMt and units in this field area relate to similar features in regions to the north and south, also located along what may be equivalents to the MMt, so that the apparently different tectonic histories found in each area can be reconciled with each other, and the whole of the PRb history better resolved. This will also help sort out uncertainties in the tectonic history of the region such as the amount of displacement that occurred during thrusting on the MMt in different areas. The presence of ring complexes, such as the Cerro de Costilla, to the east side of the MMt, that appear to have formed at depths as great as 18 km, pose another ongoing mystery, as such complexes are typically considered shallow igneous systems at the plutonic/volcanic interface. Also, still uncertain is the cause of the metamorphism either side of the MMt, and how it relates to the tectonic history. Such answers are vital for producing tectonic models of the middle crust during arc-continent collisions that help geologists better understand similar tectonic settings elsewhere. These questions make it more vital that the scientific community have the data presented in this thesis to work with, so that they can continue to investigate this part of the PRb.

METHODS

This thesis largely consists of the creation and implementation of a workflow for the extraction of digital geological legacy data. The workflow consisted of 12 consecutive parts (see Table 2) which can broadly be broken down into accomplishing the following three goals, though not in succession. (1) For example, the first goal was to put the mapped data into a geographic reference frame so that each point on the map had coordinates that, given technical and practical limitations, matches the exact point on the earth it represented. Parts 1, 2, 3 and 8 accomplish this goal, because parts from other two goals required the first few parts of this goal to be completed, but finishing the goal was best accomplished near to the end of the workflow after most of the work creating and editing data was finished. (2) The second goal was to extract structural geological measurements preserved within the vector data, and accompanying text objects, so that these data can be accessed from a database, implemented, queried, and used in a variety of ways in the future. Parts 4 and 5 performed this goal. (3) Finally, a wide range of tasks were required to present the data in an accurate and organized format. This included verifying the accuracy of the data extraction techniques in part 6, cleaning up the data in part 7, adding new information and elements to help organize it in part 9, and reorganizing it into final logical format in parts 10, 11 and 12.

Table 2 Shows a brief description of each of the 12 consecutive parts in the project workflow.



Parts of the Project Workflow	
Part	Description
1	Applying Nearly-Accurate Coordinates to the Map
2	Establishing a Reference Latitude & Longitude Grid for Warping
3	Exporting the Vector Layers to ESRI Shapefiles
4	Extracting Dip-Direction and Trend Azimuths
5	Extracting Data from Text Objects in the Canvas Layers
6	Manual Accuracy Check Data Extraction
7	Editing Geological Unit Polygons
8	Final Organization and Formatting of Data
9	Vector Warping
10	Transformation from NAD27 to WGS84
11	Producing the Geological Maps
12	Archiving the Geological Data

Part 1 - Applying Nearly-Accurate Coordinates to the Map

To accomplish the first objective, the data were assigned a coordinate system that roughly matched the spatial extent represented, and then warped so that, to the best practical degree, the coordinates of each vector object on the map matched the location of the unit or measurement on the earth’s surface. Prior to the start of this project, most of the vectors were placed in the Canvas graphics program, according to a combination of field notes, and scanned topographic maps of the relevant area. Within the Canvas document, a layer was created that stitched together a series of adjacent topographic maps of the field area into a single raster image, shown in Fig. 9, covering the whole area that was mapped. Then units and measurements, drawn on the field maps and recorded in the

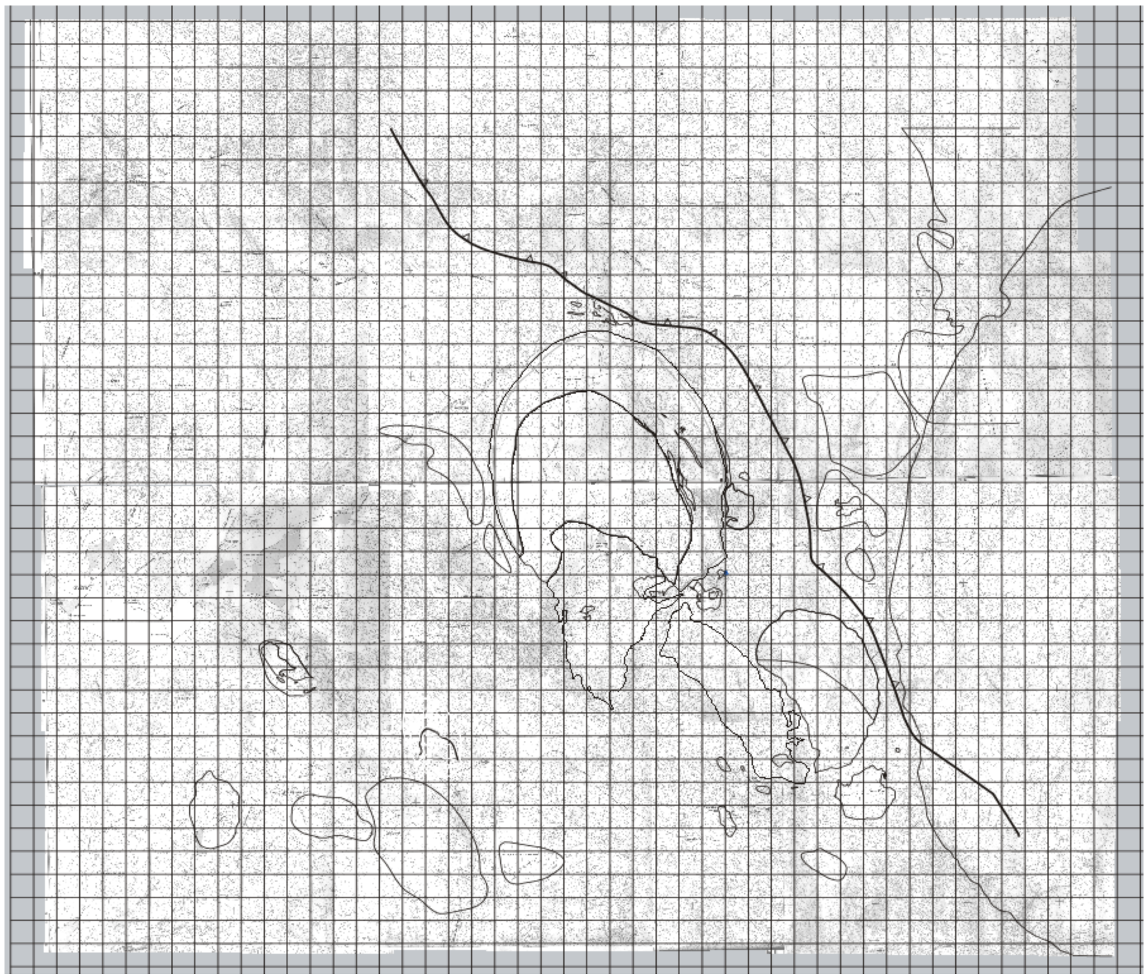


Figure 9 Snapshot of the Canvas X map showing the background raster image of stitched together topographic maps. A couple of raster layers displaying plutons and the MMt are shown for reference. Note how the individual maps do not always appear to line up.

field notes, were placed on top of the topographic raster image, as vector objects, according to the location recorded on the field map and measurements recorded in the field notes. Once this process had been completed, the resulting graphics document showed a map that had no underlying digital geographic reference frame, drawn to match a topographic raster image with regular latitude and longitude cross marks on it. Canvas X GIS 2017 now has the capability of assigning a geographic coordinate system given that you can assign the correct extents that your image document covers on the ground.

Thus, the first step involved assigning such a coordinate system by choosing latitude-longitude cross marks on the map and assigning to them their known coodinates, then adjusting the scale so that the geographic document scale, found on the top and sides of the map frame, fit a scale bar present on the map (Fig. 10) (see Appendix A). The resulting scale was such that 1 n/d unit on the document was set to be equal to 240000 n/d in the coordinate system of choice.

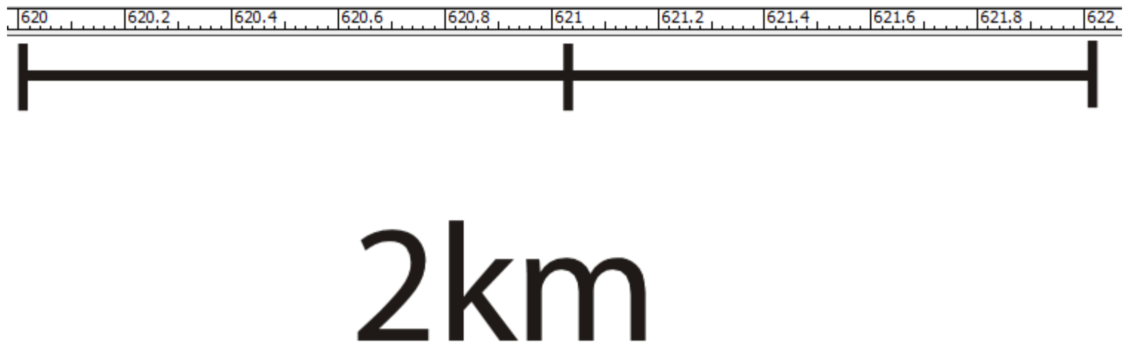


Figure 10 This figure shows how the scale bar on the map, the large black lines, was lined up with the document scale found on the top of the map display, so that different scales could be tried until the most accurate one was found when first assigning coordinates to the map in Canvas X.

Although the coordinate system defined throughout this process was the Universal Transverse Mercator (UTM), zone 11N in the NAD 1927 datum, the grid that was used to warp the data was in radial units, since it followed latitude and longitude cross marks on the reference topographic image. Additionally, all of the interfaces for the tools in Canvas X GIS also use radial global coordinate system units (latitude and longitude). This includes the coordinates used to define the first cross mark, and any subsequent verification of how well the cross marks fit the defined coordinate system. The UTM NAD27 coordinate system was chosen because it was the projection and datum used by the most recent worker when collecting field data (Melis, 2006).

Part 2 - Establishing a Reference Latitude & Longitude Grid for Warping

The results showed good accuracy, but, as expected, were not perfect because the image needed to be warped, and thus this step focused on creating the control points for the warp. The remaining offset is mostly due to the fact that the shape of a flat topographic map will never be able to match a coordinate system describing a curved earth surface, and any error associated with relative mismatch among the individual topographic maps that were tiled together to make the raster base map. Without performing this warp, only one point on the raster image could ever be located at the correct position according to the Canvas document’s geographic coordinate system with the offset, in theory, increasing radially outwards from this point. Thus, in preparation for such a warp, at a later stage in the workflow, within ArcMap, a vector grid of lines of latitude and longitude was created in this part of the workflow that, as closely as possible, matched the cross marks on the topographic map, so that the vector data could be independently warped from the reference raster image.

Originally another strategy was tried, because within Canvas X GIS 2017, it is possible to warp raster~~,~~ but not vector layers, so that, in this case, each of the cross marks that define a certain latitude and longitude would, in theory, be located at that latitude and longitude, according to the document’s defined geographic coordinate system. Thus, a distorted topographic map is warped to match a projected coordinate system. First, the plan was to link the vector data to the topographic map through vector layers that could be matched to precise points on the raster image, such as the latitude-longitude cross marks. Then the raster image could be warped, and the vector layers could be warped later using ArcMap to match the raster image. However, when warping the raster image, only 19 reference points could be created before crashing the software, instead of the ideal 56 reference points that could have been created. Due to this, and perhaps complications consisting of the image being made of several maps, the cross marks were still offset from their true positions, often by only 10 meters, but sometimes by as much as 80 meters, making the warped topographic image a poor reference to later warp the vector layers to.

Part 3 - Exporting the Vector Layers to ESRI Shapefiles

The next step involved exporting both the raster and vector data into geospatial files to be worked with inside of ESRI ArcGIS. Canvas X GIS 2017 has the functionality to export each of its layers to separate ESRI files known as shapefiles. When allowed to either give the shapefiles *Geodetic degree* coordinates or the *Current Document (meters)* coordinates, which in this case is UTM zone 11N NAD27, we chose *Current Document* because that was the coordinate system used to map the data in the field. Canvas allows each layer to be exported as point, line, or polygon objects, which was vital for the overall workflow because the arcpy code in part 5 required all the structural measurements, some of which started off as polygons, to be line objects. Raster maps as a geospatial TIFF or GIF also had to be exported to use for verifying that the program correctly analyzed the structural measurements in part 7. Each layer with structural measurements had to be isolated, rendered under Image > Area > Render, and then exported separately, unlike the shapefiles, which were batch exported.

While many of the parts in this workflow could be completed in different orders, the order laid out here follows a particular logic. For example, although the vector data could have been warped as discussed in part 2 and performed in part 8 directly after part

3, doing so would have prevented data from being re-exported from the Canvas X document. Since such backwards retrievals of data occurred several times during the development of the workflow, it was important that this final warp was one of the last steps before the data was used to create the final map and database.

Part 4 - Extracting Dip-Direction and Trend Azimuths

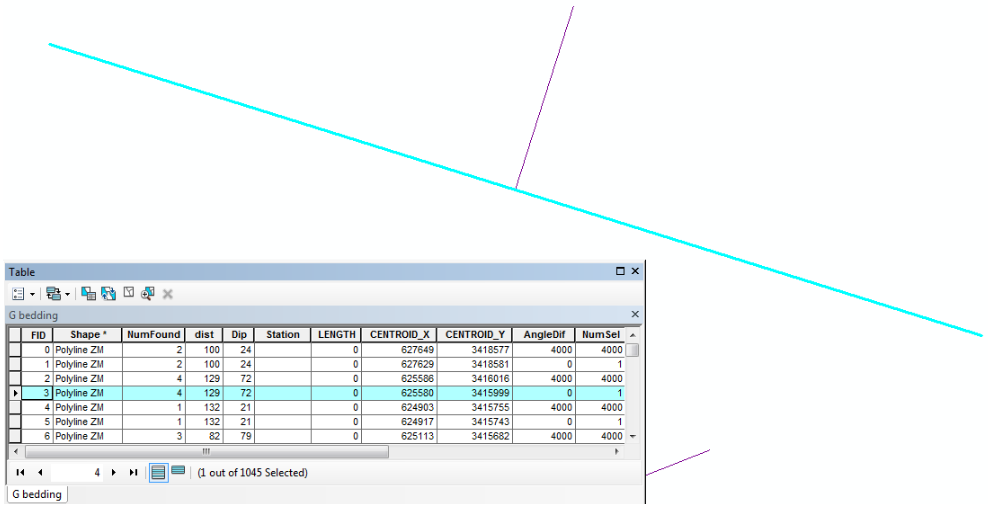
Developing and using arcpy-scripts (viewable within Appendix D, as well as digitally on the included CD) to computationally extract information from the shapes of the vector(s) representing structural measurements was one of the longest and most involved stages of the project. The principle component of the code calculates the strike-azimuth and dip-direction of foliation measurement symbols, the trend-azimuth and plunge-angle of lineation measurements, and determines the type of measurement each symbol represents, such as magmatic, solid-state, and bedding for foliations, and magmatic, regional, and intersection for lineations. This information would be stored in the shapefile's attribute table. From there it was extracted and stored within a geospatial database.

The problem of extracting meaningful directional information was significantly complicated by the variety of situations and problems that came with each different dataset. To accommodate the entirety of different datasets in this project, as well as unknown potential complications in future datasets, we made the code very customizable. The code is therefore very complicated, requiring a skilled user to run. Detailed instructions for running the workflow are included in Appendix B.

The code consists of several lead scripts, each of which calls on a number of modules, scripts opened and read by the main script. This format was used to save time and produce an easier user interface, as each module does not need to be run separately, and one set of user inputted parameters works for all the read modules. Additionally, there is a sequence of lead scripts to use, some of them requiring user inputted parameters, obtained from running previous lead scripts, but the exact sequence of lead scripts depends on the type of symbols being analyzed within the shapefile, namely whether they are foliations or lineations, and single-object or multi-object symbols. The difference between single-object and multi-object symbols is shown in Fig. 11

Fig. 12 shows an overview of the different possible workflow script sequences for each type of symbol. Only the final lead scripts produces measurement data, but the first few scripts and their accompanying modules are required in this workflow in order to populate parameters needed to run the final lead script. Each lead script, except for the first, uses the results of the previous script to populate its required parameters, and it is this interdependence that creates the mutable, but complicated nature of the scripts mentioned above. Such a detailed setup will potentially facilitate fitting these scripts to new datasets in future data recovery projects, thereby making the scripts more useful for the scientific community.

All datasets will first be modified by the settingdipandlength lead script in step 1 of Fig. 12, which creates the Length field, populated by the length of each geometry, the coordinates of the centroid of each object, and an unpopulated field for the dip measurement. This could all have been done manually, but the script automates the process and can run on all the shapefiles simultaneously, saving considerable time.

a. 

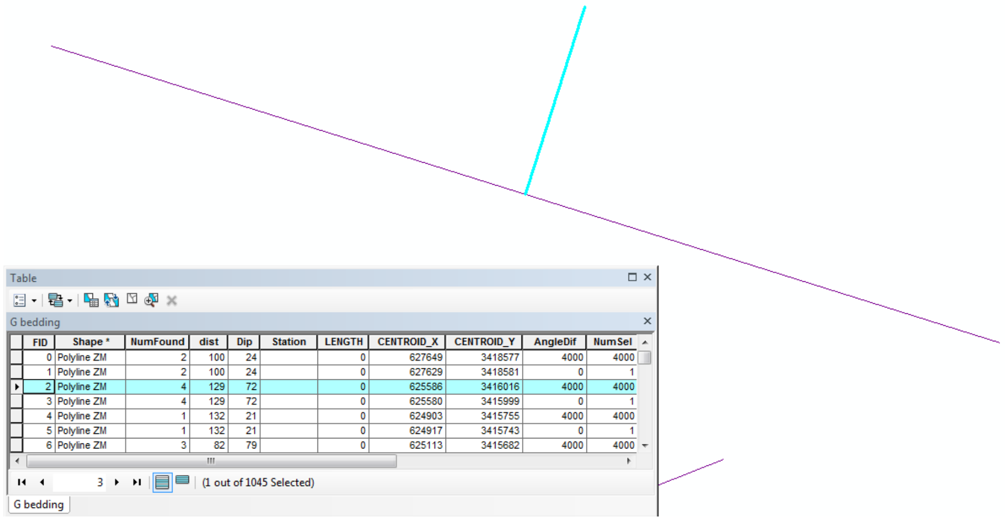
Table

G bedding

FID	Shape *	NumFound	dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel
0	Polyline ZM	2	100	24		0	627649	3418577	4000	4000
1	Polyline ZM	2	100	24		0	627629	3418581	0	1
2	Polyline ZM	4	129	72		0	625586	3416016	4000	4000
3	Polyline ZM	4	129	72		0	625580	3415999	0	1
4	Polyline ZM	1	132	21		0	624903	3415755	4000	4000
5	Polyline ZM	1	132	21		0	624917	3415743	0	1
6	Polyline ZM	3	82	79		0	625113	3415682	4000	4000

4 (1 out of 1045 Selected)

G bedding

b. 

Table

G bedding

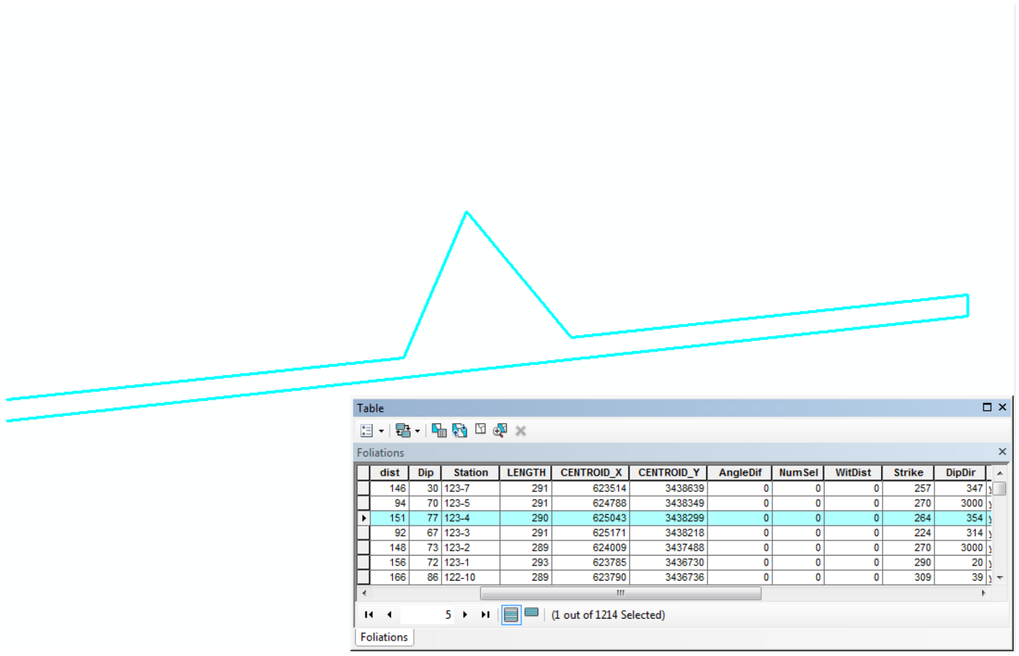
FID	Shape *	NumFound	dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel
0	Polyline ZM	2	100	24		0	627649	3418577	4000	4000
1	Polyline ZM	2	100	24		0	627629	3418581	0	1
2	Polyline ZM	4	129	72		0	625586	3416016	4000	4000
3	Polyline ZM	4	129	72		0	625580	3415999	0	1
4	Polyline ZM	1	132	21		0	624903	3415755	4000	4000
5	Polyline ZM	1	132	21		0	624917	3415743	0	1
6	Polyline ZM	3	82	79		0	625113	3415682	4000	4000

3 (1 out of 1045 Selected)

G bedding

Figure 11 Shows two different measurements with each object composing each symbol being highlighted in separate snapshots. The first symbol is a multi-object symbol shown in (a) and (b) where the measurement is represented by a primary object highlighted in (a) and an accessory object highlighted in (b). The primary object is the longer object because its centroid is the location that future symbols representing this measurement should be centered on. These two objects have no affinity except for their location next to each other on the map. (c) Shows in contrast a single-object symbol where there is only one object representing this measurement.

c.



dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel	WitDist	Strike	DipDir
146	30	123-7	291	623514	3438639	0	0	0	257	347
94	70	123-5	291	624788	3438349	0	0	0	270	3000
151	77	123-4	290	625043	3438299	0	0	0	264	354
92	67	123-3	291	625171	3438218	0	0	0	224	314
148	73	123-2	289	624009	3437488	0	0	0	270	3000
156	72	123-1	293	623785	3436730	0	0	0	290	20
166	86	122-10	289	623790	3436736	0	0	0	309	39

Figure 11 Shows two different measurements with each object composing each symbol being highlighted in separate snapshots. The first symbol is a multi-object symbol shown in (a) and (b) where the measurement is represented by a primary object highlighted in (a) and an accessory object highlighted in (b). The primary object is the longer object because its centroid is the location that future symbols representing this measurement should be centered on. These two objects have no affinity except for their location next to each other on the map. (c) Shows in contrast a single-object symbol where there is only one object representing this measurement.

The next set of lead scripts, the Preliminary Analysis scripts (step 3 of Fig. 12), of which only one that matches the dataset will be used, requires parameters determined from a user analysis of the results of the Length field (see Appendix B and step 2 of Fig. 12), but is only necessary for multipart symbols which require more parameters in the last script. This script produces the AngleDif, NumSel, and WitDist Fields, the uses for which can be found in Appendix B.

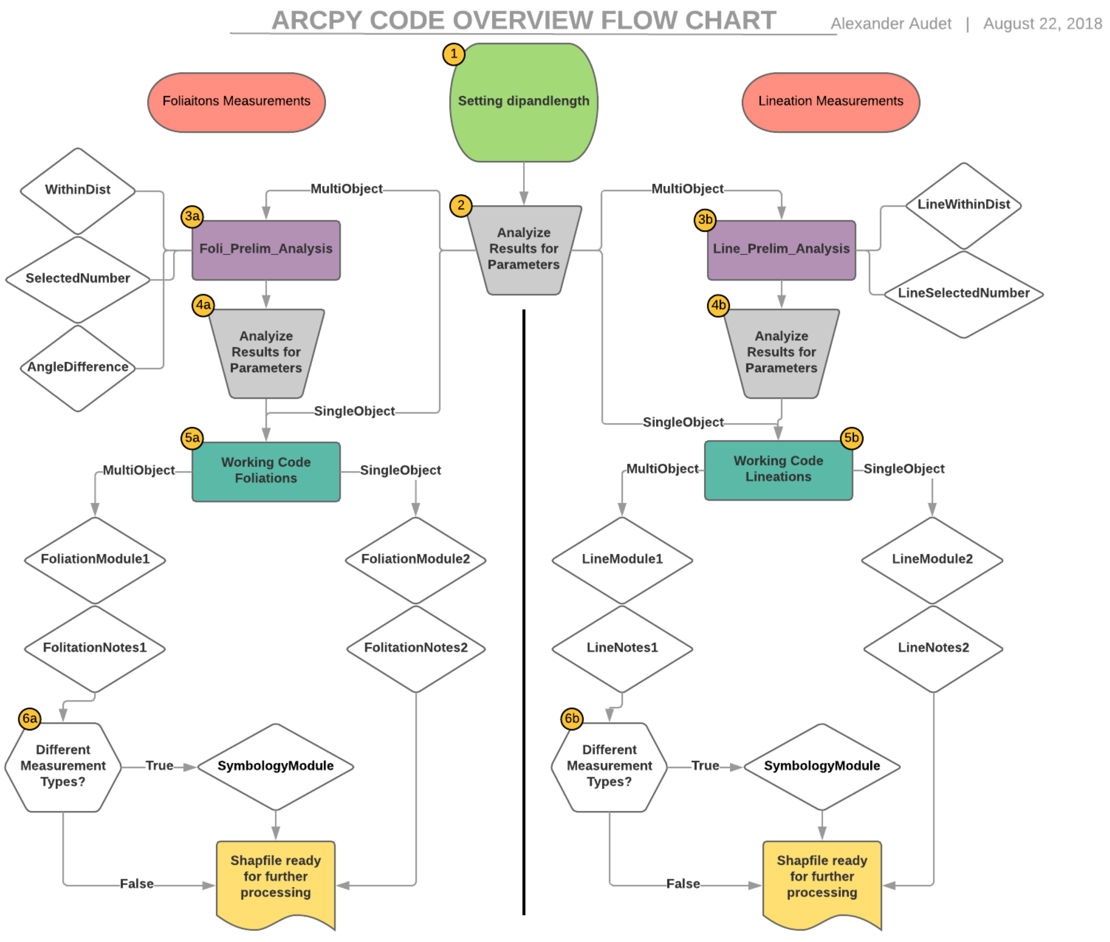


Figure 12 Shows the overview of the arcpy workflow created in this project, including all the lead-scripts and modules used in each of the four different workflow paths. **1.** First the Settingdipandlength lead script code is used on all datasets simultaneously to create the populated Length and empty Dip fields. **2.** The Length field can be analyzed to determine parameters required for the future scripts, such as the LenMin and LenMax, as well as any constraints on vertical measurements. Before moving onto step 3, the user will have to determine which of the four workflows the data matches, either foliation or lineation, and multi-object or single-object. However, the single-object paths skip steps 3 and 4. For multi-part objects, steps **3a & 3b,** for foliations and lineations respectively, involve using the preliminary analysis lead scripts, that run either three or two modules on each dataset or shapefile being worked on at a time. In 3a. the Foli_Prelim_Analysis runs the WithinDist, SelectedNumber and AngleDifference modules on multi-object foliation datasets. In 3b, the Line_Prelim_Analysis runs the LineWithinDist and LineSelectedNumber, the lineation equivalents of the WithinDist and SelectedNumber foliation modules on multi-object lineation datasets. **4a. & 4b.** These modules themselves create fields, one for each module, with values that must be used to populate the parameters of the Working Code lead scripts in step 5. Thus, in step 4 of either of the foliations or lineations workflows, the user must determine the necessary parameter values from the results of the preliminary analysis and input them into the correct working code script. **5a. & 5b.** Depending on whether the current dataset contains multi-object or single-object measurements, the user will specify the either the number 1 or number 2 modules respectively in the working code lead script. Then once the modules and necessary parameters are selected and entered, the lead script can be run, producing the final azimuth measurement data. **6a. & 6b.** Depending on whether there are multiple types of structural geological measurements in the dataset will determine whether or not the SymbologyModule should be used or not. If not, the symbiology values should be manually entered during post script processing.

Finally, the appropriate Working Code script (step 5 & 6 of Fig. 12) must be run using the parameters determined by both previous lead scripts, the second only if applicable, and in this case produces the final results of the arcpy portion of the workflow.

An overview of the functioning of the final lead script, Working Code Foliations is shown in Fig. 13, which shows the code as run on foliations with multipart symbols. The other lead scripts follow similar logic, but are often less complicated, making this script a good representative example. The greyed-out modules are those that would be used on single-object symbols. Each module is ultimately responsible for determining the correct value for one field, and in this case those fields are the DipDir field, the dip direction of the measurement, the Results field, a value used to assess the degree of success with which the code calculated the dip direction, and the Type field, the type of structural measurement the symbol represents, which ultimately is used to determine the measurement's symbolic representation in the final map.

However, the type field was often manually populated, since the module that calculates the type was made to distinguish multiple types of structural measurements, for example magmatic, solid-state, and bedding for foliation. In most cases however, only one type of measurement was in the shapefile, making it easier to leave the parameter symbology set to *False* so that the type module does not run, and instead the type value of all the symbols were populated with one value manually.

An example of how a module works is given in Fig. 14, which illustrates the workings of two modules (Foliation Module 1, and Foliation Notes 1), which return the

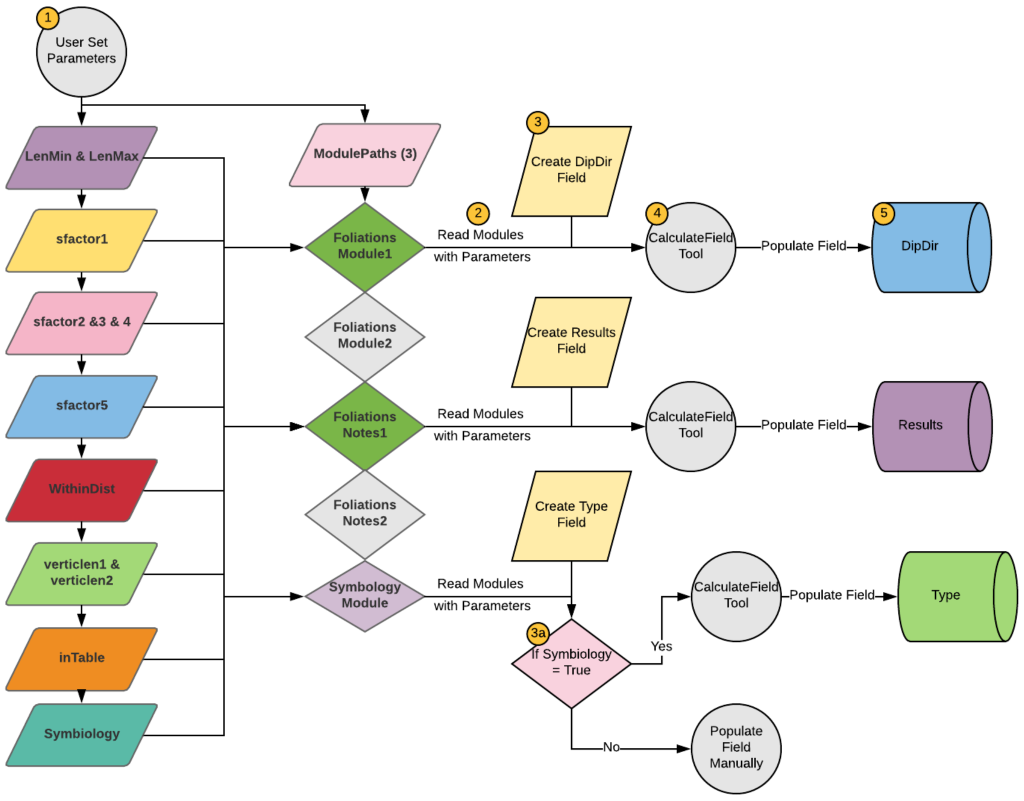


Figure 13 This image illustrates both the inputs, and parts making up the Working Code Foliations script as run with datasets with multi-part measurements. **1.** The user inputs the correct parameter values and gives the correct module path locations. The parameter values are collected from previous lead scripts, as discussed above and in Appendix B, and once the values have been entered, the script is run. **2.** The script opens the modules and reads them along with the other parameters. **3.** Next the script creates the fields in the shapefile currently being worked on. **4.** The calculate field tool runs for each field using the parameters and the correct module of the three for that field. **5.** Thus, at this point, each field should be populated with the correct values, subject to further inspection and processing.

DipDir and Results fields. Again, these modules were selected because they are among the most complex, and thus they broadly illustrate the logic encompassed within all the other modules, including those for the preliminary analysis, which are really just composed of parts of this module.

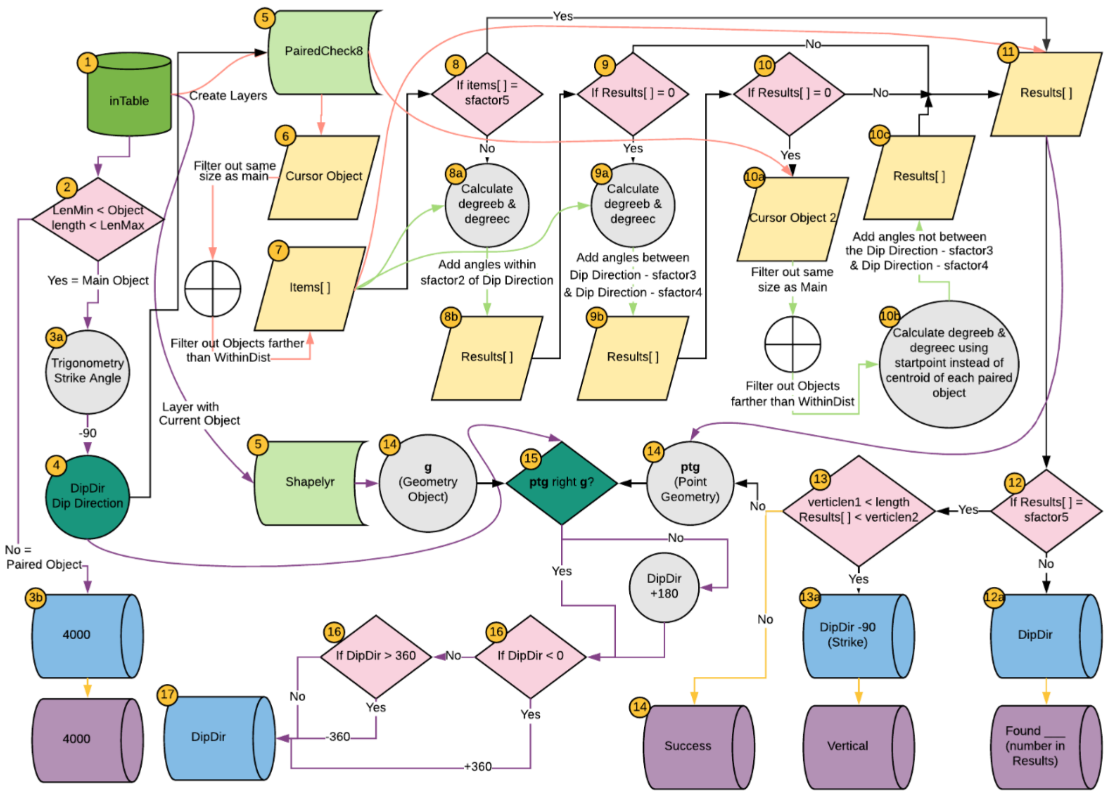

1 inTable
2 LenMin < Object length < LenMax
Yes = Main Object
No = Paired Object
3a Trigonometry Strike Angle
-90
4 DipDir Dip Direction
3b 4000
4000
5 PairedCheck8
Create Layers
6 Cursor Object
Filter out same size as main
Filter out Objects farther than WithinDist
7 Items[ ]
Layer with Current Object
5 Shapelyr
14 g (Geometry Object)
15 ptg right g?
14 ptg (Point Geometry)
8 If items[ ] = sfactor5
8a Calculate degreeb & degreec
Add angles within sfactor2 of Dip Direction
8b Results[ ]
9 If Results[ ] = 0
9a Calculate degreeb & degreec
Add angles between Dip Direction - sfactor3 & Dip Direction - sfactor4
9b Results[ ]
10 If Results[ ] = 0
10a Cursor Object 2
Filter out same size as Main
Filter out Objects farther than WithinDist
10b Calculate degreeb & degreec using startpoint instead of centroid of each paired object
10c Results[ ]
Add angles not between the Dip Direction - sfactor3 & Dip Direction - sfactor4
11 Results[ ]
12 If Results[ ] = sfactor5
12a DipDir
Found ___ (number in Results)
13 verticlen1 < length Results[ ] < verticlen2
13a DipDir -90 (Strike)
Vertical
14 Success
DipDir +180
16 If DipDir < 0
16 If DipDir > 360
+360
-360
17 DipDir
Yes
No

Figure 14 Illustrates the mechanics driving the Foliations Module/Notes 1 arcpy code and breaks up the logic into 17 steps. The rigid lines show progression in the workflow with any data being transferred when the beginning of the line is reached, while curved lines show the movement of data out of sequence in the workflow, where the data is only moved once the destination of the arrow is reached. The purple lines indicate that only a single record is being acted on or moved in that step. Green lines indicate that a batch of data is being processed or moved, one record at a time. Pink lines indicate that a batch of data is being moved altogether. **1.** First the intable, or shapefile currently being processed in the workflow must be open within ArcMap, where the script is to be run, before the code can be run. **2.** Starting in on the script, the non-primary objects in the dataset are filtered by their length, if they are outside the range set by LenMin and LenMax. If they are filtered out, in step **3b.**, they receive a code of 4000 in each module, and no more logic is performed on them. If instead, the objects' length are within the required range set by the user, step **3a.** commences and their directions are calculated by trigonometry. **4.** Ninety degrees are added to the calculated angle to obtain the dip direction, or the angle opposite the dip direction. **5.** Two shape layers are created from the input dataset, PairedCheck8 and Shapelyr, which contain every object but the one currently being iterated over in the calculatefield tool, and only that object respectively. **6.** The script makes a cursor object out of the PairedCheck8 layer, and retrieves the length and true centroid coordinates for the objects inside the cursor object. The script then uses these attributes to filter out all the objects of about the same length as the object that is currently being iterated over, and also filter out all objects whose centorids not within a parameter-specified distance of the centroid of the object being iterated over. **7.** The remaining objects are places into a list, *items[]* ([] indicate a python list). **8.** The length of that list is checked. If the length of the list is not equal to sfactor5, the parameter specifying the correct number of accessory objects with each primary object, then step **8a.** determines the direction and reverse direction, degreeb and degreec, of the centroid of the object being iterated over to the centroid of each matching object. **8b.** With the boundary between 0 and 360 degrees being accounted for by logic that checks if the calculated dip direction is within a meaningful distance of either of those two values, the script check to see if any of the objects have either such angles that are within sfactor2 number of degrees from the previously calculated

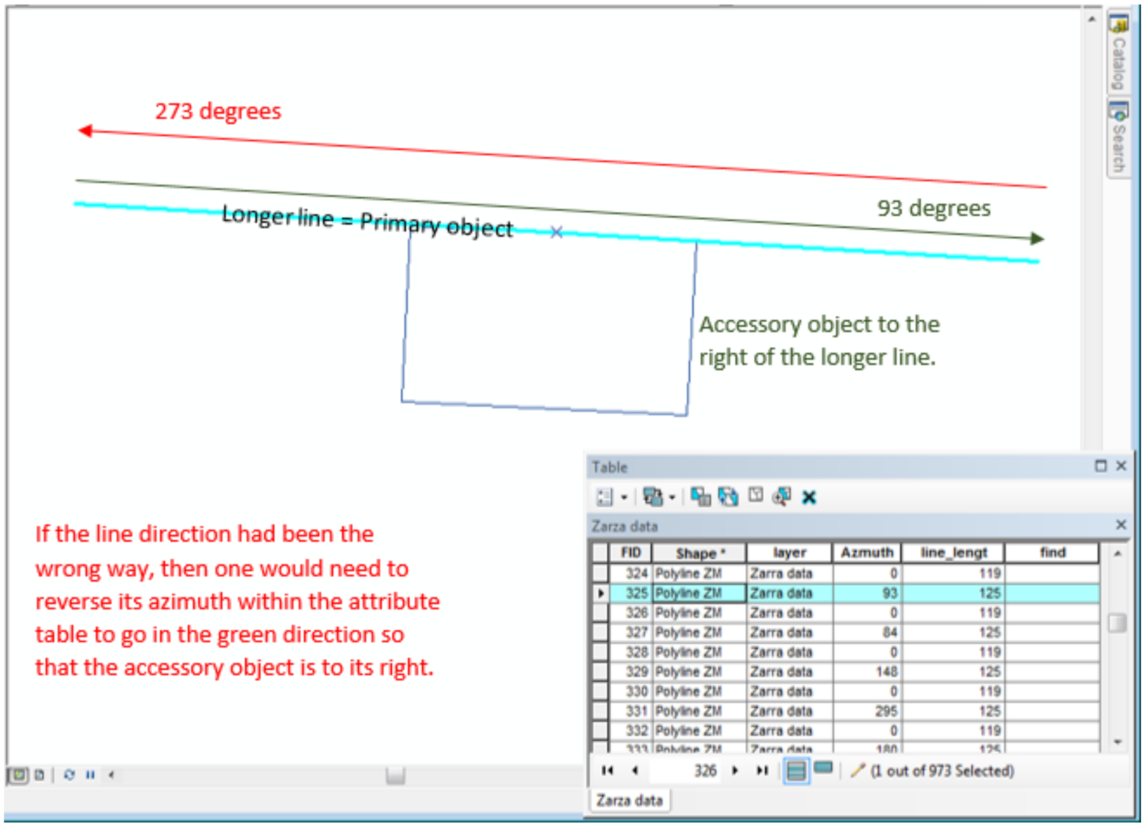
dip direction. If so, then that object is likely a matching object and is placed within a new list, *results[]*. **9.** The length of the list is tested again, and if it still is not the correct length, then the same test as in step 8 is performed, this time looking for angles that are within a second set range of the dip direction. The only time that step 9 was used in this project was when there were two consistent sets of angles between correctly matching objects, those that were within 1 degree of either the calculated dip direction or reverse dip direction, or those that were consistently between 5-6 degree less than one of those azimuths. Thus, for that data set within this project, the parameters for step 9 caused the script to look for angles that were between 5 and 6 degrees less than the calculated dip direction or reverse dip direction. **10.** Finally, in the case where the measurement is showing a vertical dip, the angle for the correct matching objects could be nearly any value, because in that case the centroids are sitting on top of each other. Thus, if the correct number of objects have still not been selected by step 10, the cursor is reproduced using the complete geometry, and the number of objects within it are narrowed down just as in step 6 and 7. Then the angles degreeb and degreec are produced using the endpoint instead of the centroid of the potential matches, that does not have the potential problem of sitting on top of the primary object's centroid in vertical measurements. If the measurement really is vertical, then this procedure will let the script recognize the correct matching objects. **11.** As soon as the correct number of matching objects are found in either *items[]* or *results[]*, or if the end of step 10 is reached without finding the correct number, step 11 makes sure that the selected items are put within results[] if they are not already there. **12.** If the length of results still does not equal the correct number, then the Foliation Module 1 returns the calculated dip direction and the Foliation Notes 1 returns the number of items found in the Results field as a warning message, meant to be used by the user to manually correct the dip direction as needed. **13.** If the number is correct, the script further checks if any of the matching objects are within the length ranges set for vertical measurements. If so, then Foliation Module 1 returns the strike of the measurement as there is no dip direction, and the Foliation Notes 1 returns within the Results field the message "Vertical." **14.** The Notes script returns "Success" in the Results field. Meanwhile, the Module script creates a geometry object out of the Shapelyr object, the layer created in step 5, containing only the object currently being iterated over, and it creates a point geometry object at the position of the centroid of the first object in the *results[]* list. In step **15.**, the script then tests whether the point geometry object is situated to the right of the geometry object using the database defined first and last points for the geometry object to determine its direction. Thus, if the test returns true, the dip direction is correct, if false, then 180 degrees is added to the measurement to correct it. **16.** Finally, the script makes sure that the dip direction is a number between 0 and 360, and if not adds or subtracts 360 as appropriate to make sure that it returns a correct azimuth in step 17. **17.** The final azimuth is returned as the field value for the object being iterated over.

Some notable parts of the module are described here. The second step in the script as seen in Fig. 14, and the first piece of logic within every module created for this project, is the logic that filters all the non-primary objects within the dataset by their lengths if they are outside of the set LenMin and LenMax range. The pieces filtered out are either all those that do not represent a measurement, or if part of a group together representing a measurement, are not the primary piece chosen to represent the whole measurement. These objects were given the 4000 code within both their DipDir and Results fields as seen in step 3b to distinguish them from 0-360 degree angles and other results.

In step 3a, for the geometries not filtered out, the azimuth direction, or strike of the object is calculated using trigonometry. 4. Then ninety degrees are added to that direction to obtain the dip direction, which by the convention of the righthand rule is always ninety degrees greater than the strike.

However, at this point, the dip-direction as measured may not be pointing in the correct direction because the direction of the line was determined by which end of the line was recorded as the start, and which end of the line was recorded as the end within the dataset ArcGIS is reading. The resulting direction could be the correct strike direction, or 180 degrees away, as shown on Fig. 15. The rest of the logic within the module seeks the correct strike by identifying the side of the object that is the dip-direction symbol. It should be noted that these accessory object(s) are filtered out from going through this process by step 2, and thus have a 4000 code instead of a dip direction, as discussed previously.

Also, a similar problem arises when searching for the trend of a lineation. This involves searching for the accessory object(s), and determining the correct direction using


273 degrees
93 degrees
Longer line = Primary object
Accessory object to the right of the longer line.
Catalog
Search
Table
Zarza data

If the line direction had been the wrong way, then one would need to reverse its azimuth within the attribute table to go in the green direction so that the accessory object is to its right.

FID	Shape *	layer	Azmuth	line_lengt	find
324	Polyline ZM	Zarra data	0	119	
325	Polyline ZM	Zarra data	93	125	
326	Polyline ZM	Zarra data	0	119	
327	Polyline ZM	Zarra data	84	125	
328	Polyline ZM	Zarra data	0	119	
329	Polyline ZM	Zarra data	148	125	
330	Polyline ZM	Zarra data	0	119	
331	Polyline ZM	Zarra data	295	125	
332	Polyline ZM	Zarra data	0	119	
333	Polyline ZM	Zarra data	180	125	

326 (1 out of 973 Selected)

Figure 15 This image shows the correct (green) and incorrect (red) directions for the selected primary object in light blue, with respect to its accessory object below it in dark blue. The arcpy script must be able to recognize the position of the accessory object in order to choose the correct (green) direction for the primary object.

their position in reference to the primary object's position. The method changes when only a single-object represents each measurement, but the concept is similar, involving checking that the additional mass associated with the direction of the dip in the case of foliations or the trend in the case of lineations is in the predicted position, which if it is not, the measured direction is reversed.

In step 8, after a list of the nearest matching accessory objects has been generated, the length of the list must be checked to see if it contains the expected number of matches. This is because overlapping measurements can sometimes bring unrelated accessory objects closer to the primary object than the correct accessory object. It should be noted that there are often more than one correct matching objects, because, when the vectors were exported from Canvas into ArcMap, sometimes multiple, overlapping accessory objects were placed on top of each other. These objects have identical positions, and usually identical or nearly identical shapes, meaning that which objects is selected as the accessory object is unimportant, yet the presence of multiple objects does have important implications for the logic at several points within this workflow. Thus, if the number of objects in *items[]* (brackets indicate a python list) is equal to the number of objects expected to match as part of the same symbol with the object being iterated over, then the rest of steps 8, 9, and 10, which are meant to further distinguish the correct matching object, are skipped and step 11 is started. Otherwise, if more than the correct number of matching objects are found, then the script attempts to distinguish between them and choose the correct object(s) following the rest of step 8 and possibly 9 and 10. At the beginning of these last two, the same test looking for the correct list length is performed before the step is started to determine if the rest of this part of the logic can be skipped, but with the *results[]* list which is populated by any records that passes the tests in 8, 9, and 10, instead of the *items[]* list.

Notice here that a critical assumption in my code is that if the correct number of objects is selected, then those objects are the correct matching objects for the current object being iterated over, and that unrelated objects have not matched where related objects failed to do so. Within all the datasets of this project, this assumption has so far been found valid, but it is possible that in future high-density datasets this assumption might not hold true. A situation where this could occur is if the centroids of objects in overlapping measurements were arranged such that the centroid of one secondary object was closer to the centroid of an unrelated primary object whose own matching object was just, by chance, outside of the set distance from it. Thus, the script performs as though it has found the correct matching set, when it has in fact mismatched the objects, and thus potentially returned the wrong direction. This situation can usually be avoided by checking for the farthest legitimately matching object when determining the WithinDist parameter as discussed in Appendix B, but might still occur in high density data where the script "misses" the farthest matching objects. Thus, it was important to check suspicious cases where too many accessory objects where found during the preliminary analysis (Appendix B).

One thing to note about step 10, as shown within Fig. 14, is that it is essentially a last resort that is designed for cases where the script cannot distinguish the correct accessory object due to the measurement being vertical with incompatible geometry for prior tests in steps 9 & 8. It remakes the cursor object and calls upon complete geometries for all the records in the dataset (see description of Fig. 14), an operation that is computationally very costly. Within this project, step 10 was used so infrequently that it did not noticeably slow down the script processing time.

Within step 13, the script checks for vertical measurements by searching for accessory object lengths within a range specified by inputted parameters. If the measurement is vertical, the Results field is returned as Vertical, and the DipDir field returns the strike, although the correct direction of the strike in this situation is relative, and is thus just returned as the first direction calculated. There are a few places where horizontal measurements occurred as a circle with a cross in this project. Because these are so uncommon, we did not write a script to recognize them, and so they were manually identified and marked as horizontal.

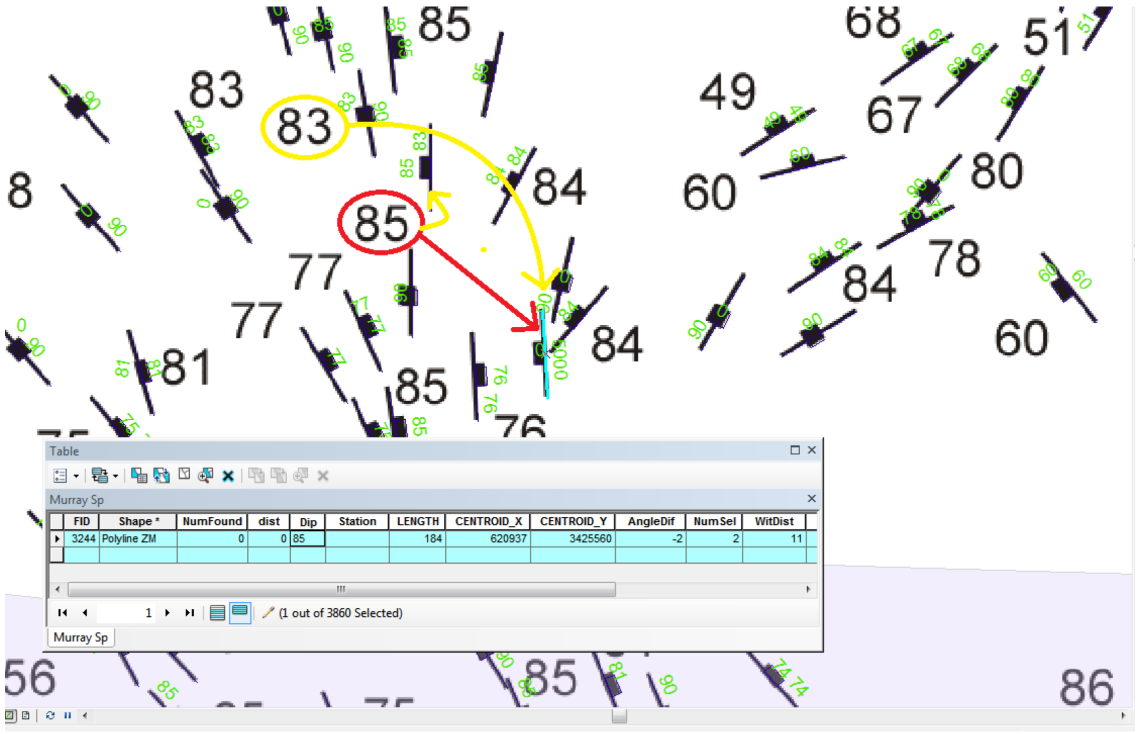
Part 5 – Extracting Data from Text Objects in the Canvas Layers

The next challenge in this project after recognizing the correct strike/dip, and trend/plunge directions was to extract the value of the text objects associated with each measurement given the dip angle in the case of foliations, and the plunge angle in the case of lineations. Some of the foliation and lineation measurements were also associated with certain station numbers. These text objects would not export out of the Canvas X file with the shapefiles, so were exported as GeoPDF files. Thus, the next part of the project involved extracting this information using FME (Feature Manipulation Engine) workbench, a software application useful for transforming geospatial data or databases.

This part of the workflow was split into two separate workflows in the program. The goal of the first was to create a point object with the number attribute(s) attached, and in the second part was to assign the number attribute(s) to either the Dip or Plunge and Station fields in the shapefiles containing line objects with the directional information added by the arcpy code in the previous step. Matching the point objects and the line objects from the previous step was performed by pairing the lines with the point objects closest to their centroids. However, since the data in the GeoPDF was consistently offset either during the export from Canvas X, or during the import into FME workbench by about 76 meters or 76.156 meters East and 9.618 meters South, it had to be affined in step 11 of Fig. 18 to approximately the correct position so that the point objects matched with the correct line objects.

The first workflow consisted of attaching each of the text objects to the correct geometry object. This process was complicated to perform, even manually, as the data density was commonly so high that the text object was placed at some distance from its matching measurement symbol such as shown in Fig. 16. In other cases, there were too many (Fig. 17) or too few numbers for the symbols, and they cannot easily be assigned to a distant unmatched symbol. The only way to figure out the correct matches was to use the process of elimination where all the most obvious matches were made first, and the process continued until the least obvious matches were made. However, while the solution often became obvious using this method, it sometimes involved a limited degree of interpretation and guesswork. The best way to automate this process would have been to implement an optimization of bipartite graphics such as the Hungarian algorithm, also known as the Kuhn-Munkres algorithm. Budig et al. (2016) created their own algorithm to perform the same function. However, this method was outside of the scope of this project, so as a substitute, the workflow implemented a "waterfall approach" shown as steps 6, 7 and 8 of Fig. 18.

The waterfall approach involved matching each text object with its closest symbol object, step 6, but if multiple text objects chose the same symbol object, it only allowed the closest match to process, step 7. These successful matches were removed from the remaining pool of unmatched text and symbol objects, step 8, and then the process would start over, allowing the unmatched symbols to find the next closest text objects and so on, each time only allowing the closest matches through when more than one symbol chose the same text object. This process occurred about five times until the maximum number of matches possible with this method was made. The workflow involved using the text



FID	Shape *	NumFound	dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel	WitDist
3244	Polyline ZM	0	0	85		184	620937	3425560	-2	2	11

(1 out of 3860 Selected)

Figure 16 The above image shows both a raster image of a set of magmatic foliation measurements from the Murray Sp layer, seen as the black symbols and text, and the shapefile vectors on top of this image, of which only the green numbers, showing their assigned dip field values are easily visible. The red circle and arrow points out a case, found a handful of times across the six-thousand measurements, where the dip symbol seems to be placed at some distance from its matching symbol, highlighted in blue. It may not be obvious from this image, but all the other symbols around it, both on and off the image, are matched with their text objects, leaving this unobvious pair unmatched. However, an alternative interpretation, shown in yellow, is possible, where the 85 is matched with the symbol closest to it, and the 83 is instead the missing match.

object to search for their matching geometries, instead of the other way around, due to increases in accuracy with this method. Also, because in preliminary tests, in the case of foliations, matching the text object to the accessory geometry rather than the primary geometry was found to produce much better match results, the primary and accessory parts of the symbols were first matched to one another (step 4 and 5 of Fig. 18), and the text object matched with accessory geometry when possible (see step 6). To filter out multiple accessory parts for the same symbols, the FeatureMerger tool replaced the

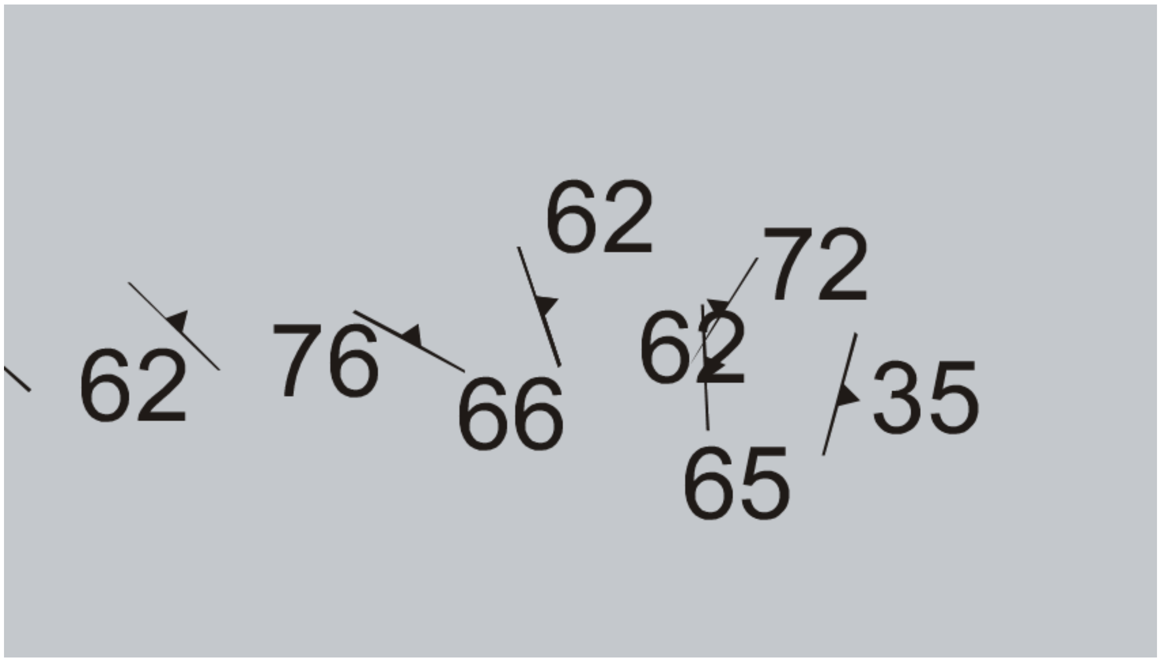


Figure 17 This shows a portion of the raster image from the foliations layer where there is one too many numbers for the number of symbols in range.

geometry of the primary object that had at least one accessory geometry with the geometry of the first accessory object. In single-object cases, without accessory parts, a point geometry representing the symbol's centroid substituted for the accessory geometry.

These successful matches were removed from the remaining pool of unmatched text and symbol objects, step 8, and then the process would start over, allowing the unmatched symbols to find the next closest text objects and so on, each time only allowing the closest matches through when more than one symbol chose the same text object. This process occurred about five times until the maximum number of matches possible with this method was made. The workflow involved using the text object to

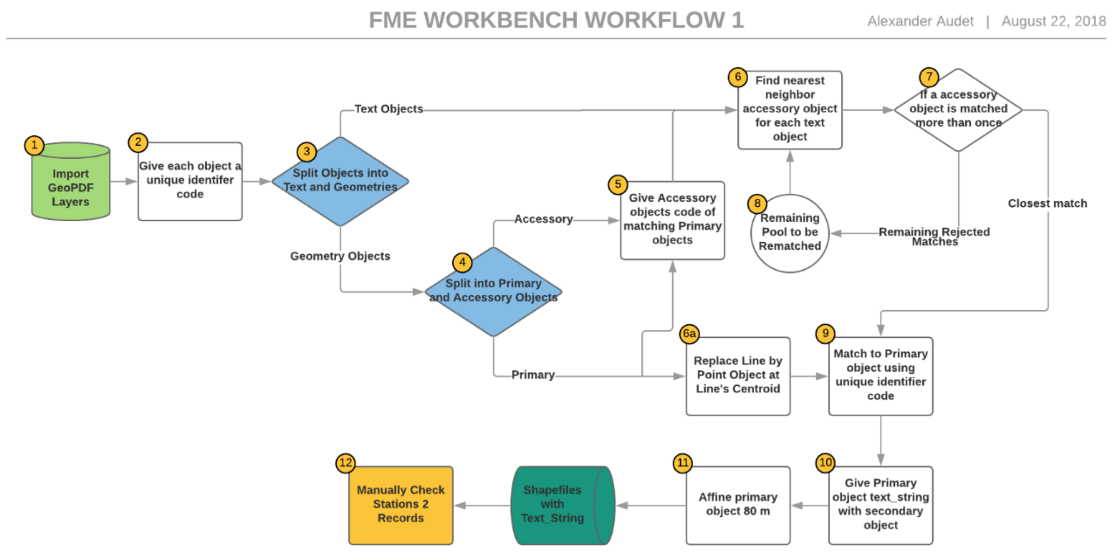

FME WORKBENCH WORKFLOW 1
Alexander Audet | August 22, 2018
1 Import GeoPDF Layers
2 Give each object a unique identifer code
3 Split Objects into Text and Geometries
Text Objects
Geometry Objects
4 Split into Primary and Accessory Objects
Accessory
Primary
5 Give Accessory objects code of matching Primary objects
6 Find nearest neighbor accessory object for each text object
7 If a accessory object is matched more than once
Closest match
Remaining Rejected Matches
8 Remaining Pool to be Rematched
6a Replace Line by Point Object at Line's Centroid
9 Match to Primary object using unique identifier code
10 Give Primary object text_string with secondary object
11 Affine primary object 80 m
Shapefiles with Text_String
12 Manually Check Stations 2 Records

Figure 18 This figure illustrates the general mechanism within the first FME workbench workflow. **1.** The GeoPDF exported out of the canvas document is imported as its separate layers. **2.** Each object is given a unique number by a counter tool, producing its identifying ID. **3.** Then the workflow splits the objects into text objects and geometries, while in **4.**, the geometry objects are themselves split into the primary objects and the accessory objects, if applicable. **5.** If the accessory objects were separated from a primary object, then the accessory objects are matched to the primary objects, and obtain the unique identifier ID of their primary object. A copy of the primary objects are actually given the geometry of their first accessory object with their matching code, so as to effectively only have one accessory object for each primary object in the next part of the workflow. **6.** Then the workflow begins the waterfall process in which the nearest neighbor secondary object for each text object is determined, but since the same accessory object can be matched with more than one text object, in step **7.**, only each accessory object's closest match is allowed forwards in the workflow, and in step **8.**, the rest of the text objects are returned to the pool to be rematched with the next closest text objects. Step 6-8 are repeated until no more matches can be made given a distance constraint on the possible matches. Meanwhile in step **6a.** all primary geometry line objects are transformed into point objects located at the line's centroid. **9.** Then all the pairs matched in the waterfall process are matched again to their primary objects using the unique identifier code obtained from the primary objects when they were matched previously. **10.** Thus, the text string from the text object attached to the secondary object can be used to populate the Dip or Plunge fields of the primary object. **11.** The primary object needed to be affined about 76 meters to match the position of the objects in the shapefiles modified by the arcpy code (see the text). **12.** Finally, the Stations 2 layer was manually checked and corrected before commencing the second FME workflow.

search for their matching geometries, instead of the other way around, due to increases in accuracy with this method. Also, because in preliminary tests, in the case of foliations, matching the text object to the accessory geometry rather than the primary geometry was found to produce much better match results, the primary and accessory parts of the symbols were first matched to one another (step 4 and 5 of Fig. 18), and the text object matched with accessory geometry when possible (see step 6). To filter out multiple accessory parts for the same symbols, the FeatureMerger tool replaced the geometry of the primary object that had at least one accessory geometry with the geometry of the first accessory object. In single-object cases, without accessory parts, a point geometry representing the symbol's centroid substituted for the accessory geometry.

The next few steps are implemented because the primary objects would be able to better match the position of their equivalent line objects from the arcpy modified dataset, and thus pass the dip, plunge, and station measurements on to them when combining the two datasets. First, back in step 6a, the primary line was transformed into a point object at its centroid. Then, before the workflow ended in step 9 and 10, the numbers assigned to the accessory objects representing the dip or plunge measurements were given to the matching primary point objects.

This process of passing on text information is performed within the second workflow shown in Fig. 19. Essentially, the workflow uses the neighbor finder tool to find the nearest matches between the point objects containing the text information, and the centroids of the line objects containing the directional information, and then assigns that text information, which is the dip or plunge to the Dip or Plunge fields for that line object (step 3a, and 4 in Fig. 19). However, in the cases where the measurement was vertical, there was no text object associated with the symbol as the value was understood to be 90 degrees. Thus, within the workflow at step 2 in Fig. 19, before matching the two datasets,

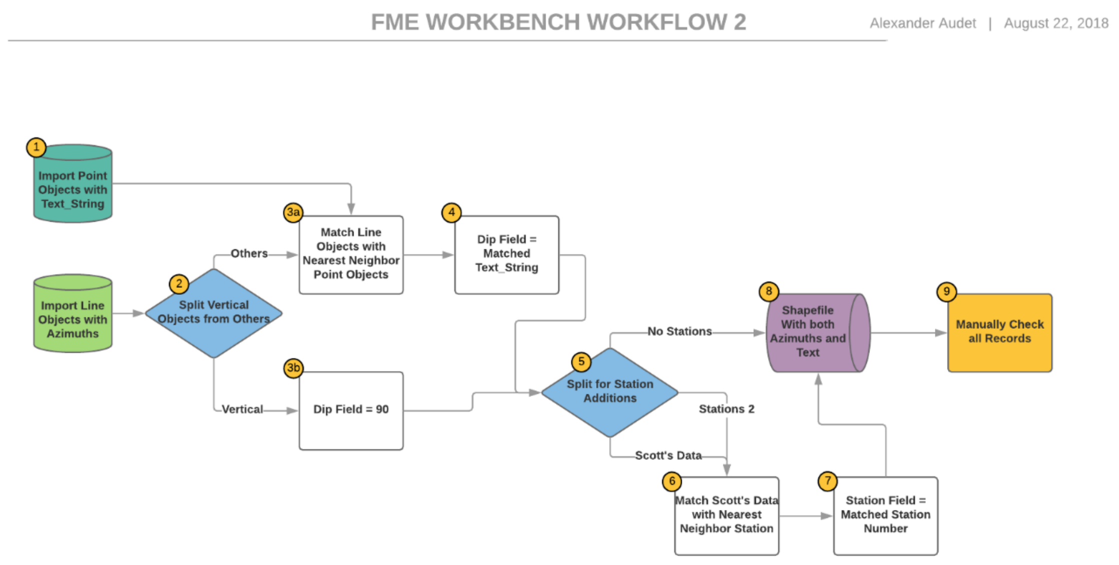

FME WORKBENCH WORKFLOW 2
Alexander Audet | August 22, 2018
1 Import Point Objects with Text_String
Import Line Objects with Azimuths
2 Split Vertical Objects from Others
Others
Vertical
3a Match Line Objects with Nearest Neighbor Point Objects
3b Dip Field = 90
4 Dip Field = Matched Text_String
5 Split for Station Additions
No Stations
Stations 2
Scott's Data
6 Match Scott's Data with Nearest Neighbor Station
7 Station Field = Matched Station Number
8 Shapefile With both Azimuths and Text
9 Manually Check all Records

Figure 19 This figure illustrates the general mechanics found within the second FME workbench workflow. 1. The two datasets to be combined within the workflow are imported, one containing point objects with the text string describing the measured dip and plunge, and the other containing line objects with azimuthal measurement information. 2. The vertical measurements that do not have a matching point object with text from the normal measurements are split. 3a. The neighbor finder tool is used to find the nearest point object neighbor for each line object. Each point object should only match one line object and vice versa. 4. The matched point object's text string is used to populate the dip or plunge field of the line objects. 3b. All vertical measurements are instead assigned a dip or plunge of 90°. 5. Dr. Johnson's measurements, which are associated with stations in the Stations 2 dataset, are separated from the rest of the data along with Stations 2. 6. Each of Dr. Johnson's measurements are associated with their nearest station in Stations 2 using Neighbor Finder, where a station can be matched with more than one measurement. 7. The matched station numbers populate the Station field of Dr. Johnson's data. All the data is combined again, with all the line data now having dip or plunge values and, where appropriate, station numbers. 9. Finally all of the data is manually checked against background georeferenced images of the data from the Canvas document. The dip, strike, and stations numbers are all checked, and some of the azimuths are randomly checked.

I separated the records labeled as vertical within their results field, and then in step 3b, assigned their dip field to be equal to 90.

The same method was used to populate the Station field where appropriate. First however, the Station 2 dataset had gone through all the same steps as the other text-bearing points: its text objects containing the station numbers added to the station points in the first FME workflow shown in Fig. 18 just like the rest of the point objections containing text information. At this point, at step 12 in Fig. 18, the accuracy of the assignments in the stations 2 dataset was checked using a method described below in Part 7, before using the Stations 2 dataset in the second FME workflow. Then, in the second FME workflow of Fig. 19, its station numbers were added to the original Stations 2 shapefile in steps 3a and 4 of Fig. 19 to increase the spatial accuracy of the stations just like all the other geometry points passed their text information to their matching geometries modified by the arcpy scripts. However, at this point, in a process nearly identical to that just described for transferring the dip and plunge, Dr. Johnson's measurements were separated out in the workflow, in step 5 of Fig. 19, so that these stations were not matched with measurements from datasets not associated with the stations. Then the measurement data were associated with their stations in the Station 2 dataset by finding the nearest station point object within a maximum of a 120 meter radius (step 6), and assigning the matched station number to the Station field of the line objects (step 7).

Part 6 – Data Extraction Manual Accuracy Check

Part 6 of the workflow involved checking the accuracy of what has been produced by every previous step of the workflow and contains the last step of Fig. 19, step 9.

After the text information had been assigned to the measurement data in the previous step, we checked all the number assignments manually, by comparing the matched data within ArcMap with underlying raster images of the numbers and symbols, also extracted in part 3, in order to verify that the waterflow method had assigned the correct numbers and station number to the correct measurement symbols, and also that the correct point and line objects were matched between datasets. In most of the datasets, between 80-95% of the numbers were correctly assigned, although in one shapefile or dataset, over 50% of the assignments were incorrect. For reference, the algorithm that Budig et al. (2016) used had a success rate of 96% on the datasets they used it on. The station numbers had a similar, if not slightly higher success rate. The incorrect assignments were manually corrected here, and this comprehensive check of the data provided a good opportunity to verify the accuracy of the arcpy operations.

To perform the check, each individual layer of data within the original Canvas document was exported as a geospatial gif image, including the stations layer. Then in ArcMap, each gif was opened with its associated shapefile dataset and the stations gif, and the dataset was labeled with its dip or plunge and, if applicable, its station number. Then these numbers were checked against the numbers on the gif images, with any applicable changes being made as necessary. The check was comprehensive, covering every single record in the project. At the same time, a number of azimuths where checked from various datasets, and where found to be accurate. Of course, almost all the data had also been sampled in this way when the azimuths were being calculated with the arcpy code, and except for some of the cases caught by the Results field, all the angles were found to be accurate during both stages of checking.

The Results field, populated by the companion code to the Foliation/Lineation 1 Module script, the Foliation/Lineation Notes 1, was designed, amongst other things, to determine if the script could not identify the correct accessory object(s) for the primary object. The few cases where this occurred within the project required manual analysis and correction, and the process was an efficient way of handling exceptions or difficult cases. Of course, this method makes a number of assumptions that would prevent the script from misidentifying the wrong accessory objects as the correct ones, and while such a mistake could not be ruled out, it is highly unlikely for the reasons discussed within the section on the arcpy script.

Another area of concern is the offset in the position of the data when it was exported as a shapefile, versus a GeoPDF. Such a discrepancy raises concern about which, if either, set of exported data is correctly located. However, throughout the workflow, the position of points of the geospatial data were cross referenced with the precise position of those same points in the Canvas file, showing that the shapefiles were correctly located and the GeoPDF had been offset.

Part 7 – Editing Geological Unit Polygons

Because the vectors representing geological units were made in software applications that did not allow for advanced editing abilities, and they had changed between file types a few times before arriving in the Canvas X, many of the units required heavy editing. Part 7, therefore, included aligning the shapes to each other (see Fig. 2), cutting out other shapes so that each unit was accurately displayed, whichever order the shapefiles were opened in, and creating new polygons for missing units. We found a combination of the *Align to Shape* and *Construct Polygon* tools under the advanced editing toolbar to be the most useful methods of performing these tasks.

Part 8 – Final Organization and Formatting of Data

In part 8, FME workbench was used to perform final organization and formatting on the data. First, the data were split into foliations and lineations to be put into two final shapefiles, or datasets, and the attributes, summarized in Table 3, found in each dataset were created and organized. A few notable additions were the Source Lyr field, containing the original layer name that the data was found in within the source Canvas file and then exported as to a shapefile, the Strike field in the foliations shapefile, equal to the dip direction field minus 90 degrees, the Trend and Plunge fields in the lineations shapefile, which replaced the DipDir and Dip fields, and the addition of Horizontal and Vertical to the Type field where applicable. The Strike field was equal to the DipDir value with vertical measurements, and the dip direction field for vertical and both the dip direction and strike for horizontal measurements were not equal to anything and thus were set to the 5000 code. All unnecessary fields for the final dataset product were removed here, including the Results field, and any of the accessory fields used to populate the arcpy script parameters.

Table 3 This table gives descriptions on each of the fields added to the Foliations and Lineations shapefiles. This does not include the FID and Shape fields automatically included with all ESRI shapefiles.

Fields	File	Description
Station	Both	The station number associated with those measurements that are associated with recorded stations.
Dip	Foliations	The dip of the foliation plane found at the measurement site.
Plunge	Lineations	The plunge of the lineation direction at the measurement site.
Strike	Foliations	The azimuth 90 degrees to the left of the direction of dip.
Trend	Lineations	The azimuth direction of the way the lineation is plunging down.
DipDir	Foliations	The azimuth direction of the foliation dip.
Type	Both	The type of geological geometry captured by the measurement, used to determine the measurement's map symbol.
Type2	Both	More in depth information of the measurement type.
Unit	Both	The geological unit the measurement is located on in the map.
Source_L	Both	The original canvas layer the data was found in during the start of the project in October 2017.

The mapped rock units were also combined and organized within FME workbench into one final shapefile, the Geological Units shapefile. The shapefile has a Unit field, and an optional Complex field as shown in Table 4. The unit field was usually populated by their source shapefile name, except where abbreviated. The major exceptions to this were the MMt zone layer, which contained several different units that were entered manually, as well as a few layers that needed more descriptive names. The Complex field was manually populated if the part of a unit belonged to a magmatic complex such as the Zarza Complex, the San José pluton, or the Cerro de Costilla complex.

Table 4 This table gives descriptions on each of the fields added to the Geological Units shapefile. This does not include the FID and Shape fields automatically included with all ESRI shapefiles.

Fields	**Description**
Unit	The name of the geological unit, usually taken from the source layer name
Complex	The name of the igneous complex the unit is part of.

The mapped rock units were also combined and organized within FME workbench into one final shapefile, the Geological Units shapefile. The shapefile has a Unit field, and an optional Complex field as shown in Table 4. The unit field was usually populated by their source shapefile name, except where abbreviated. The major exceptions to this were the MMt zone layer, which contained several different units that were entered manually, as well as a few layers that needed more descriptive names. The Complex field was manually populated if the part of a unit belonged to a magmatic complex such as the Zarza Complex, the San José pluton, or the Cerro de Costilla complex.

<u>Part 9 – Vector Warp</u>

This part involves the final warping of the vector data to move it as close as possible to its correct position as located on the earth. A total of 30 control points placed at the intersections of grid vectors representing lines of latitude and longitude were established. Then control points where placed so that the vector grid intersections were brought closer to their correct locations. Before the transformation, the control points ranged from between 7-105 meters offset from their true positions. For example, starting in the northeast corner and traveling around all the corners clockwise, the measured errors were 7, 105, 103, and 54 meters of offset. The transformation used was the projective transformation, because it performed better than both the affine, and rubbersheet transformations. Likewise, the similarity transformation was inappropriate to use because it preserves geometrical shapes, an unnecessarily, and perhaps undesirable feature for this dataset. A second transformation, using either another projective, or the rubbersheet method was attempted on top of the first, but this seemed to increase the transformation error.

The final transformation error varied from about 32 meters to 5 meters from their true positions. However, some of the low error positions, such as the control point of the northeast corner of the map lost accuracy, going from 7 meters to 20 meters. The program thus seemed unable to find a perfect solution for transforming the data, a fact that was emphasized by the failure of subsequent attempted transformations. It also had the effect of spreading the remaining error more evenly across the map.

Part 10 – Transformation from NAD27 to WGS84

The final part of editing the data was performing the transformation from the outdated NAD 1927 datum to the more modern WGS 1984 datum. WGS 1984 is a modern, globally used datum, that is also commonly used for Mexico and Baja California Norte. This transformation did not actually move the data, but rather gave it new coordinates under a different definition.

To perform the registration transformation, another script was used, **Registration Shift**. Instruction for running it are found within Appendix C. This script opens all files within the set workplace environment, and first defines their coordinate system to what the user has determined it to be. This step was found necessary because, as Canvas X exported the shapefiles, it defined their coordinate system in such a way that ArcGIS could not recognized the definition. It is important that we determined the correct coordinate system, so that the data received a coordinate system definition recognized by ArcMap, but did not shift on the map. We verified that the data did not shift after the definition by comparing the position of the original data to the data with the defined coordinate system.

Next, a template file is called to easily select the correct target projection without worrying about the syntax. The template file was produced by creating an empty shapefile within ArcCatalog with the selected desired projection and datum: WGS 1984, UTM zone 11N. The correct published transformation between datums should be used. In this case, the

WGS_1984_(ITRF00)_To_NAD_1983 + NAD_1927_To_NAD_1983_NADCON

transformation was determined most appropriate over other simpler transformations such as “NAD_1927_To_WGS_1984_18” because of it was the most accurate transformation available for the lower 48 States and Mexico, the location of this map data (GeoNet, 2018; National Geodetic Survey, 2018).

Part 11 – Producing the Geological Maps

The next to last part was to place the data into a presentable map format. Critical for this was correctly displaying each type of structural measurement and the unit colors for each type of rock. We used documentation and symbols from the FGDC Standard for Geological Map Symbolization, and the GeMS Geologic Map Schema in order to facilitate this process. Labels were added using the dip field of the foliations, and the plunge field of the lineations. The Maplex label engine tool within the label toolbar was used to place the label to the dip direction, and trend azimuth around the symbol, while keeping the label horizontal. Because ArcMap cannot save assigned symbology into its shapefiles, a single final map file was created, all the symbology assigned in that, and then the map was copied to create each separate final map.

Part 12 – Archiving the Geological Data

The final part of the process involves uploading copies of the shapefiles containing data within a system to archive them. Since shapefiles have recently become an acceptable archive format for geospatial data (Christian Halsted, August 01, 2018), we have elected to store our data within 5 files in that format. The data will be archived after the completion of this thesis after the data is first presented in a publication. Once archived, it will be available for future geologist's and the public's use.

DISCUSSION

Suitability for the Arcpy Workflow for Use on Other Datasets and Transfer to other Projects

One of the primary concerns of this thesis project has been developing a workflow for saving legacy geological data in a digital format that could be implemented in other situations. Efforts to make the arcpy code flexible and mutable have given that portion of the workflow some potential for transfer to similar projects. However, constraints in time and project scope have limited this potential. For example, datasets made of heterogeneous symbols, where the primary and accessory parts varied in length across the dataset, might render the scripts in this workflow useless, depending on the severity of the difference. This situation only occurred mildly in one of the datasets in this project, since, for the most part, I assume that during the dataset's original creation, the same symbol was created once, and then copied and rotated for every new measurement. In the dataset where this had not universally occurred, a few symbols had to be recognized and edited manually after being processed by the arcpy script. This was because the worst-case scenario had occurred, and several of the primary lengths were smaller than accessory object lengths, leaving the program unable to distinguish the two. Luckily, most cases such as these can be caught during the first analysis of the object lengths when distinguishing the correct size for the primary and accessory objects (see step 2 of Fig. 12). If there are only a couple cases, their ID numbers can be written down, and they can be edited after running the scripts, otherwise different methods should be sought. If the objects have very variable length, but the primary and accessory lengths do not overlap, then the code should still work on them; however, the ability to recognize vertical measurements and to use the symbiology module, which are both length sensitive, might be compromised.

If the lengths of the parts are variable, the angles calculated in parts 8, 9, and 10 of Fig. 14 might also be variable, making those internal sorting analyses useless. In such a case, the appropriate parameters described in Appendix B should just be set to zero and the rest of the code will function, just with the Results field indicating more cases for the user to come back to.

Other differences in the multi-object code, not discussed here, also exist that would make it hard to transfer this workflow to analyze them. Overall, users should be able to use the multi-objects dataset scripts in more scenarios than the single-object workflow, because the parameters can tailor the first scripts to many different situations, while the later required a much more creative and particular solution within its code. For this reason, the same amount of mutability has not been implemented in the number 2 modules, due to the greater challenge in creating a flexible method that fits multiple possible scenarios.

The mutability of the number 2 or single-object modules could be improved in future work by setting up a better method of implementing a center of gravity analysis. This could perhaps be performed by determining the relative position of the centroid of each object to a line representing the measurement strike, created from two points found within the first 5 % of the line length. New parameters might have to be set up to account for different symbol shapes, such as to adjust the position of the second point used to create the reference line, though it is probably at the starting point and 5 % of the line length would be sufficient in most cases.

The number 2 modules should also, in the future, be given the capacity to recognize vertical and horizontal measurements. During the project, the need to do so never arose. In this case, it would have been hard to do so using the same methodology used in the multi-object code, since the lengths of the symbols were so variable in the single-object datasets, and the previous methodology would have involved looking for lengths distinct from the normal length. A similar discussion arises with implementing a symbology module that worked for the single-object workflow. The need for it did not arise during this project, and it would have involved different logic, so it was not developed.

Error Accumulation and Product Accuracy

Another item worth discussing is the accuracy and uncertainties of the final produced data. There are two broad sources contributing the most error to the data in this project. The first is the error brought with the data at the beginning of the project. It includes not only any of the error accumulated when collecting the data out in the field, and then manually entering the data into the canvas for the first time, but also any error created from moving the various layers through several different software and versions before we digitized it in this project. For example, the part of the data collected by Melis (2006), that was first digitized within Illustrator, and then transferred to ArcGIS, before being put into Canvas, was offset from the rest of the data at the beginning of the project. The image that the data was spatially aligned to consisted of a series of topographic maps of the covered area that had been scanned on a drum roller and then tiled together within

Canvas. However, the individual scanned topographic maps did not always align together perfectly (see Fig. 9), and thus could have generated more error.

An area of preexisting error that has extended over into the error created during this project was the addition of numbers as text objects describing each measurement's dip or plunge. In some datasets, these text objects where placed in such high density that it was sometimes difficult, or in a few rare cases, near impossible, to associate each text object with its correct measurement symbol. Although at the end of the automated matching process in this project, we checked all 6185 records in some detail, making us fairly confident in most matches, the large number of records to check could have produced further mistakes. In the end, there were about two-dozen cases that were difficult to match. We could have referred back to the original field notebooks for a few of these problem measurements taken by Dr. Scott Johnson, but for the majority of the two dozen measurements, we did not have access to the original field notes. However, even if incorrectly matched, the values are mostly within ten to fifteen degree of each other, and in similar locations, minimizing the harm caused by the error. Additionally, during the comprehensive check, eighteen records were found not to even have matching text objects, and were thus discarded, and other places contained too many text objects for the number of symbols locally present. These mistakes raise the question if other mistakes were also made when originally digitizing the data? If so, such unknown cases are likely to have only affected a handful of measurements, like the cases with too few or too many text objects.

The possible error introduced into the data during this project has already started to be discussed above. The first possible area of concern, though, was in editing the contact between the geological units so that they became flush with each other. This had already been done with these units before this project, but the boundaries were again separated when the vertices count of the polygons was subsequently reduced. Thus, the offsets as great as thirty-three meters already encompassed movement that the boundary lines had undergone, and moving the boundary lines that much in this process probably only added minimal error.

Two processes in particular could have been the source of additional error when editing the contacts. One was a smoothing tool; however, the maximum allowable offset could be specified, with only five meters allowed for large polygons, and two meters for smaller polygons. The second was that the tools within ArcMap only allowed a surface to be snapped onto an existing surface. Thus, when there was not preexisting lines separating two contacts, one of the units had to snap onto the location on the other units, instead of the units both meeting in the center of the distance between them. Such a bias potentially moved the contact as much as 30 meters farther from its original position in some cases. However, in many cases, the contacts of the units were not originally able to be placed accurately enough for this introduced variation to make a difference in the final product.

The next step with possible error is during the arcpy script azimuth analysis. However, checks both during the procedure, as well as a comprehensive check after the next procedure where random azimuths were verified suggest that there is very little chance of error. The script is built with the Results field so that any records the script is not confident about, the user manually corrects. The error associated with the next step of assigning dip and plunge numbers is very real but is minimized by a comprehensive check of the values.

The last source for error comes when the vector objects were warped to better fit their position within the coordinate system. The vector data, as discussed in the methods section, can be matched to their true coordinates through a vector grid of latitude and longitude lines. The accuracy of the grid lines are limited by the topographic map that the data, including this grid, was drawn on top of, and indeed the lines could not be drawn so that they remained both straight, and perfectly overlapped their matching crosshairs on the topographic map. This error may have contributed to issues in the current step. The points formed by the intersections between the latitude-longitude gridlines were each variable distances, ranging from 7-105 meters, away from the true positions they represented. However, ArcMap's Spatial Adjustment transformation tool was not able to bring these points to their true positions even after multiple transformations, and a residual error of about 5-30 meters was left on each point. Given the scope of the map, and the errors associated with the original map preparation though, it is unlikely that this error significantly devalues the product.

FME Workbench Evaluation

Another area of discussion is the utility of the FME workbench program. This program was required in this project in order to extract the text objects within the Canvas X layers. However, it could also have performed many of the operations performed by the arcpy script in the previous portion of the workflow. The potentially advantages are that, once the user is familiar with the program, it is more intuitive to work with than the arcpy script. Additionally, we found that there is a very strong and knowledgeable online support community for the product (find in references FME Knowledge Center), more so than for either the Canvas X software, or even the several arcpy communities found online (find in references: GIS Stack Exchange; GeoNet).

The workflow created within FME workbench only correctly assigned between 80-95% of the text objects to their symbols, requiring that the entire data set of 6185 records be manually scanned. It is uncertain if this part of the workflow is time efficient, however the FME workflow substantially decreased the amount of time spent manually editing these numbers, as correct matches quickly become obvious to the practiced eye, and much less time is spent determining the obvious matches before trying to determine the incorrect ones. There are only a lot of errors when the data becomes very dense, so the efficiency of this method is greatest in spatially spread out data and decreases the denser the dataset becomes. An algorithm meant to perform a similar function was recently produced by Budig et al. (2016), which when tested had a success rate of 95% and used a built-in sensitivity analysis to direct the user to only check the assignments of the matches that the algorithm could have gotten wrong, thus vastly reducing the manual processing time. A similar feature could be built into the FME workbench workflow in the future, or future users of this workflow could try attaining the algorithm produced by Budig et al. (2016).

However, we have not been able to produce as clearly a distributable product with the FME workbench workflow because there appear to be no clear ways to input parameters to adjust the workflow to other situations that might be found in different datasets. Instead, it will be necessary for any new user to study the whole workflow included with this project and adjust it themselves to fit their own projects. Additionally, FME workbench does require its own arcpy scripts to access its full range of capabilities, although these were not needed for the relatively simple operations performed within it in this project.

The cost of this additional program could be a deciding factor, which for the permanent license, as of July 2018, ranged at the basic level from $2250 to $8100 for the most comprehensive package. However, in addition to a 30-day free trial, there are situations where a temporary free license will be given to certain individuals or non-for-profit institutions, that can demonstrate that their projects are not economic in nature.

REFERENCES

Alsleben, H., 2005, Changing characteristics of deformation, sedimentation, and magmatism as a result of island arc-continent collision.: University of Southern California, 360 p.

Armstrong, R.L., and Suppe, J., 1973, Potassium-Argon geochronometry of mesozoic igneous rocks in Nevada, Utah, and Southern California: Bulletin of the Geological Society of America, v. 84, p. 1375–1392, doi: 10.1130/0016-7606(1973)84<1375:PGOMIR>2.0.CO;2.

Budig, B., Dijk, T.C.V.A.N., and Wolff, A., 2016, Matching Labels and Markers in Historical Maps : An Algorithm with Interactive Postprocessing r r: v. 2, p. 1–24.

Carrasco, A.P., Kimbrough, D.L., and Herzig, C.T., 1995, Cretaceous arc-volcanic strata of the western Peninsular Ranges: comparison of the Santiago Peak Volcanics and Alisitos Group.: III Peninsular Geological Society International Meeting, p. 19.

Chávez-Cabello, G., 1998, Mecanismos de ascenso, emplazamiento y evolución magmática de varios plutones al oeste de la Sierra San Pedro Mártir, Baja California, México.: CICESE, Ensenada, México, 165 p.

Dempsey, C., 2018, What is GIS? ~ GIS Lounge: GIS LEARNING, https://www.gislounge.com/what-is-gis/ (accessed July 2018).

Evernden, J.F., and Kistler, R.W., 1970, Chronology of emplacement of Mesozoic batholithic complexes in California and western Nevada., *in* U.S. Geological Survey Professional Paper 623, p. 42.

FME Knowledge Center - FME Knowledge Center. https://knowledge.safe.com/index.html. Accessed 30 Aug 2018

Geographic Information Systems Stack Exchange. https://gis.stackexchange.com/. Accessed 30 Aug 2018

Gastil, R.G., Phillips R.P., and Allison, E.C., 1973, Reconnaissance Geologic Map of the State of Baja California: Geological Society of America; accompanies Gastil, R.G., Phillips, R.P., and Allison, E.C., 1975, Reconnaissance geology of the state of Baja California: Geological Society of America Memoir, v. 140, p. 170 p, doi: 10.1130/MEM140.

Gastil, R.G., Phillips, R.P., and Allison, E.C., 1975, Reconnaissance geology of the state of Baja California: Geological Society of America Memoir, v. 140, p. 170 p, doi: 10.1130/MEM140.

Halsted, Christian, Director, Earth Resources Information, Maine Geological Survey, personal communication, February 21, 2018

Halsted, Christian, Director, Earth Resources Information, Maine Geological Survey, personal communication, August 01, 2018

Johnson, S.E., Fletcher, J.M., Fanning, C.M., Vernon, R.H., Paterson, S.R., and Tate, M.C., 2003, Structure, emplacement and lateral expansion of the San José tonalite pluton, Peninsular Ranges batholith, Baja California, México: Journal of Structural Geology, v. 25, p. 1933–1957, doi: 10.1016/S0191-8141(03)00015-4.

Johnson, S.E., Tate, M.C., and Fanning, C.M., 1999a, New geologic mapping and SHRIMP U-Pb zircon data in the Peninsular Ranges batholith, Baja California, Mexico: Evidence for a suture? Geology, v. 27, p. 743–746, doi: 10.1130/0091-7613(1999)027<0743:NGMASU>2.3.CO;2.

Johnson, S.E., Paterson, S.R., and Tate, M.C., 1999b, Structure and emplacement history of a multiple-center, cone-sheet bearing ring comples: The {Zarxa Intrusive Complex, Baja California, Mexico}: Geological Society of America Bulletin, v. 111, p. 607–619, doi: 10.1130/0016-7606(1999)111<0607.

Johnson, S.E., Schmidt, K.L., and Tate, M.C., 2002, Ring complexes in the Peninsular Ranges Batholith, Mexico and the USA: Magma plumbing systems in the middle and upper crust: Lithos, v. 61, p. 187–208, doi: 10.1016/S0024-4937(02)00079-8.

Johnson, S.E., Vernon, R.H., and Upton, P., 2004, Foliation development and progressive strain-rate partitioning in the crystallizing carapace of a tonalite pluton: Microstructural evidence and numerical modeling: Journal of Structural Geology, v. 26, p. 1845–1865, doi: 10.1016/j.jsg.2004.02.006.

Krummenacher, D., and Doupont, J., 1975, K-Ar Apparent Ages, Peninsular Ranges Batholith, Southern California and Baja California: Bulletin of the Geological Society of America, v. 86, p. 760–768, doi: 10.1130/0016-7606(1975)86<760:KAAPRB>2.0.CO;2.

Lovera, O.M., Grove, M., Kimbrough, D.L., and Abbott, P.L., 1999, A method for evaluating basement exhumation histories from closure age distributions of detrital minerals: Journal of Geophysical Research: Solid Earth, v. 104, p. 29419–29438, doi: 10.1029/1999JB900082.

Melis, E.A., 2006, Structural , Temporal and Metamorphic Evolution of the Main Mártir Thrust , Baja California , México: The University of Maine.

Murray, J.D., 1978, The structure and petrology of the San Jose pluton, northern Baja California, México: California Institute of Technology, 709 p.

NADCON - North American Datum Conversion Utility | Tools | National Geodetic Survey, https://geodesy.noaa.gov/TOOLS/Nadcon/Nadcon.shtml (accessed August 2018).

Riganti, A., Farrell, T.R., Ellis, M.J., Irimies, F., Strickland, C.D., Martin, S.K., and Wallace, D.J., 2015, GeoResJ 125 years of legacy data at the Geological Survey of Western Australia : Capture and delivery: GeoResJ, v. 6, p. 175–194, doi: 10.1016/j.grj.2015.02.015.

Schmidt, K.L., 2000, Investigation of arc processes: relationships among deformation, magmatism, mountain building, and the role of crustal anisotropy in the evolution of the Peninsular Ranges batholith, Baja California.: University of Southern California, 310 p.

Todd, V.R., Shaw, S.E., Hammarstrom, J. M. Johnson, S.E., Paterson, S.R., Fletcher, J.M., Girty, G.H., Kimbrough, D.L., and Martin-Barajas, A., 2003, Cretaceous plutons of the Peninsular Ranges batholith, San Diego and westernmost Imperial counties, California: Intrusion across a Late Jurassic continental margin, *in* SPECIAL PAPERS-GEOLOGICAL SOCIETY OF AMERICA 374, p. 185–236.

Uhl, J.H., Leyk, S., and Id, C.A.K., 2018, Map Archive Mining : Visual-Analytical Approaches to Explore Large Historical Map Collections:, doi: 10.3390/ijgi7040148.

Walawender, M.J., Gastil, R.G., Clinkenbeard, J.P., McCormick, W. V., Eastman, B.G., Wemicke, R.S., Wardlaw, M.S., Gunn, S.H., and Smith, B.M., 1990, Origin and evolution of the zoned La Posta-type plutons, eastern Peninsular Ranges batholith, southern and Baja California, *in* TThe Nature and Origin of Cordilleran Magmatism: Boulder, Colorado, Issue 174, Geological Society of America, p. 1–18.

Walawender, M.J., Girty, G.H., Lombardi, M.R., Kimbrough, D., Girty, M.S., Anderson, C., Walawender, M.J., and Hanan, B.B., 1991, A synthesis of recent work in the Peninsular Ranges batholith, *in* Geological Excursions in Southern California and México, p. 297–318.

Welcome | GeoNet. https://community.esri.com/. Accessed 30 Aug 2018

Wetmore, P., Herzig, C., Alsleben, H., Sutherland, M., Schmidt, K.L., Schultz, P.W., Paterson, S.R., Johnson, S.E., Fletcher, J.M., Girty, G.H., Kimbrough, D.L., and Martin-Barajas, A., 2003, Mesozoic tectonic evolution of the Peninsular Ranges of southern and Baja California., *in* SPECIAL PAPERS-GEOLOGICAL SOCIETY OF AMERICA 374, p. 93–116.

WGS_1984_(ITRF00)_To_NAD_1983 - when to use? | GeoNet, https://community.esri.com/thread/37607 (accessed August 2018).

APPENDICES

APPENDIX A: WORKFLOW FOR ASSIGNING A COORDINATE SYSTEM TO AN UNREGISTERED MAP IN CANVAS X GIS 2017

We enabled GIS and chose my projection as UTM zone 11N and datum as NAD27 within GIS > GIS Document Settings. Next we chose GIS > Choose Reference Point, selecting the Latitude-Longitude cross mark zoomed in so the crosshairs took the entire screen, and entered the correct location under New Location, and kept *Preserve objects positions* unselected. Then under the Units toolbar, I found Drawing Scale > Define custom scale, selected *Do not scale existing objects*, and then defined 1 n/d unit on paper to be 240000 n/d in world. The relative accuracy of this process was verified by checking the document coordinates using GIS > GIS Positioning at various other hashmarks throughout the map.

APPENDIX B: INSTRUCTIONS FOR ARCPY WORKFLOW FOR ASSIGNING AZIMUTH DIRECTIONS TO THE STRUCTURAL GEOLOICAL DATA

The next few paragraphs walk through the workflow, summarized in Fig. 12, for using the arcpy script built for this project in enough detail that future users will be able to implement it.

Before beginning, all the modules should be placed in the same folder, unless they have been divided, so that the module paths should only differ by the module name at the end of the path. Similarly, one should place all shapefiles of interest in a folder identified to be the workspace.

The first step is to run the **Setting dipandlength** code within the ArcMap terminal. This code requires only one user input on line 9, the workspace path where all the shapefiles containing the geological measurements are contained. The code can be opened for editing within Notepad++, or a similar text editing software, and then after providing the workspace path, copied whole into the python terminal within ArcMap where the user will press *enter* one or a few time to run it. It will add attribute columns containing empty dip and station fields, and the calculated length of each shape in meters to each of the files within the workspace.

At this point it is important for the user to examine the objects in the shapefile and determine, among other things, whether each measurement is represented by one solid object, or two or more objects, as illustrated in Fig. 11. If it is only composed of one object, the Prelim_Analysis code can be skipped, and the Working Code will use the number 2 modules as will be discussed below. Simply record the range of lengths that

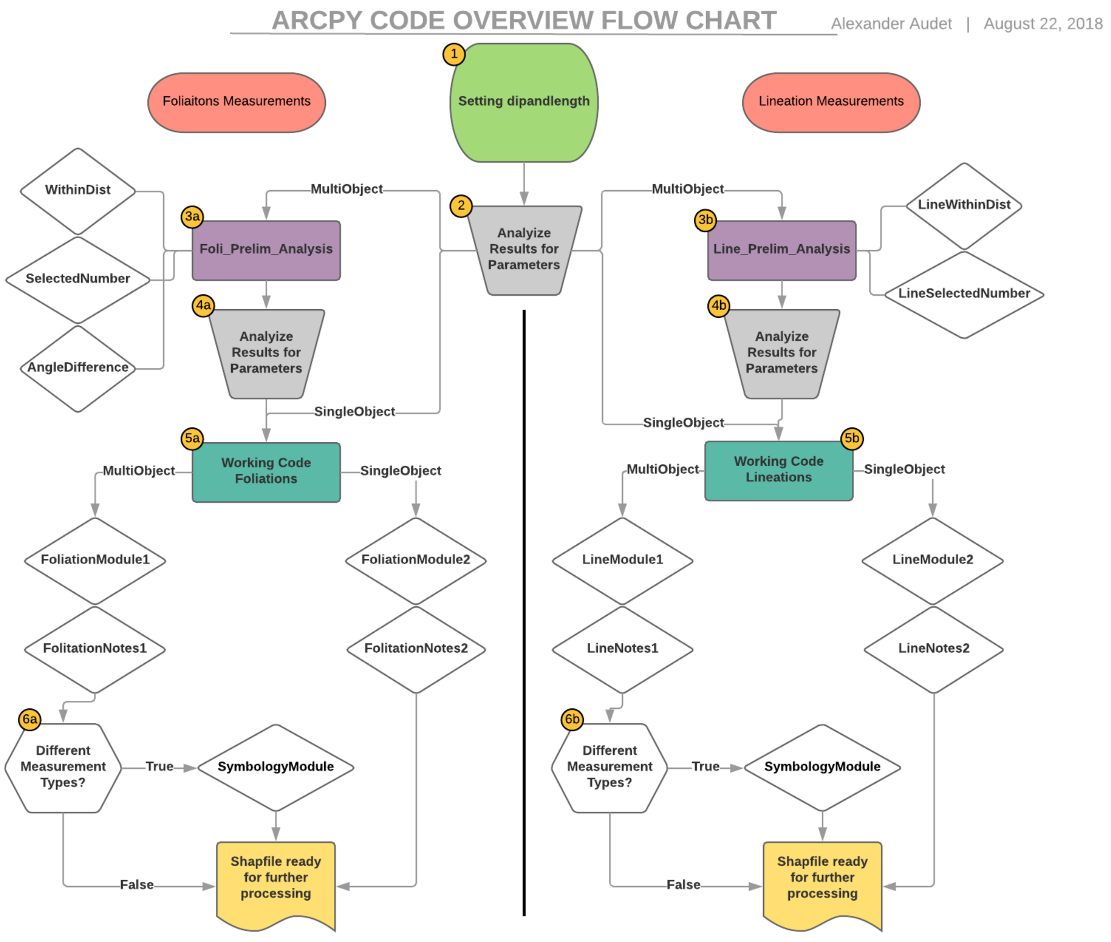
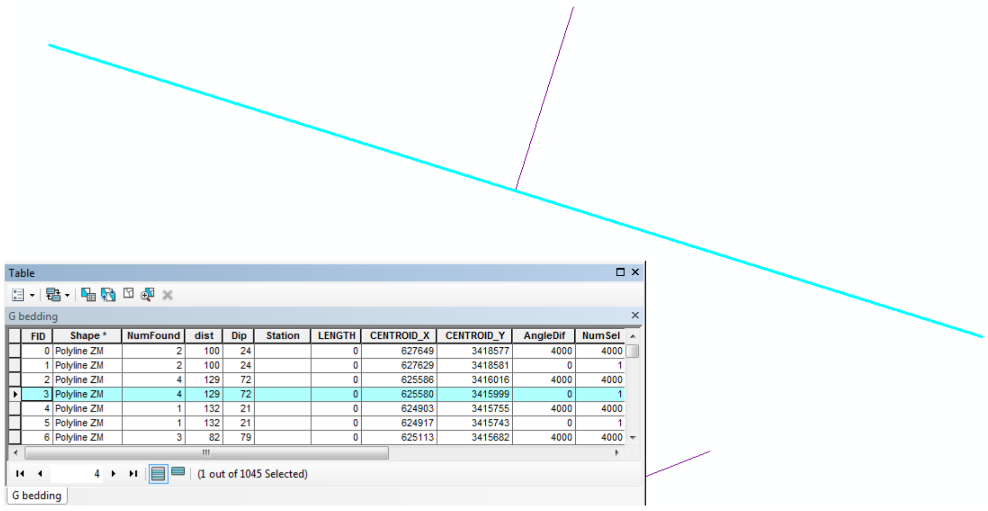


Figure 12 Shows the overview of the arcpy workflow created in this project, including all the lead-scripts and modules used in each of the four different workflow paths. **1.** First the Settingdipandlength lead script code is used on all datasets simultaneously to create the populated Length and empty Dip fields. **2.** The Length field can be analyzed to determine parameters required for the future scripts, such as the LenMin and LenMax, as well as any constraints on vertical measurements. Before moving onto step 3, the user will have to determine which of the four workflows the data matches, either foliation or lineation, and multi-object or single-object. However, the single-object paths skip steps 3 and 4. For multi-part objects, steps **3a & 3b,** for foliations and lineations respectively, involve using the preliminary analysis lead scripts, that run either three or two modules on each dataset or shapefile being worked on at a time. In 3a. the Foli_Prelim_Analysis runs the WithinDist, SelectedNumber and AngleDifference modules on multi-object foliation datasets. In 3b, the Line_Prelim_Analysis runs the LineWithinDist and LineSelectedNumber, the lineation equivalents of the WithinDist and SelectedNumber foliation modules on multi-object lineation datasets. **4a. & 4b.** These modules themselves create fields, one for each module, with values that must be used to populate the parameters of the Working Code lead scripts in step 5. Thus, in step 4 of either of the foliations or lineations workflows, the user must determine the necessary parameter values from the results of the preliminary analysis and input them into the correct working code script. **5a. & 5b.** Depending on whether the current dataset contains multi-object or single-object measurements, the user will specify the either the number 1 or number 2 modules respectively in the working code lead script. Then once the modules and necessary parameters are selected and entered, the lead script can be run, producing the final azimuth measurement data. **6a. & 6b.** Depending on whether there are multiple types of structural geological measurements in the dataset will determine whether or not the SymbologyModule should be used or not. If not, the symbiology values should be manually entered during post script processing.

encompasses all measurements in these files. This range will become the parameters LenMin and LenMax used below. A good way to do this is to open the shapefile's attribute table, right click the head of the length column, and click either of the sort options. Then browse quickly from top to bottom, treating unusually long or short lengths with suspicion. Check that they are actually measurements by selecting them, and clicking *zoom to selected*.

At this point it is important for the user to examine the objects in the shapefile and determine, among other things, whether each measurement is represented by one solid object, or two or more objects, as illustrated in Fig. 11. If it is only composed of one object, the Prelim_Analysis code can be skipped, and the Working Code will use the number 2 modules as will be discussed below. Simply record the range of lengths that encompasses all measurements in these files. This range will become the parameters LenMin and LenMax used below. A good way to do this is to open the shapefile's attribute table, right click the head of the length column, and click either of the sort options. Then browse quickly from top to bottom, treating unusually long or short lengths with suspicion. Check that they are actually measurements by selecting them, and clicking *zoom to selected*.

For shapefiles that contain more than one object per measurement, the user will need to choose which shape to use in order to best characterize the feature. In this project that object was always the longest line because the centroid of that line probably lies closest to the center of where the symbols representing the measurements where supposed to be centered on. Then, by sorting the lengths again as above, the user should be able to determine a range of lengths that include all the chosen pieces of each

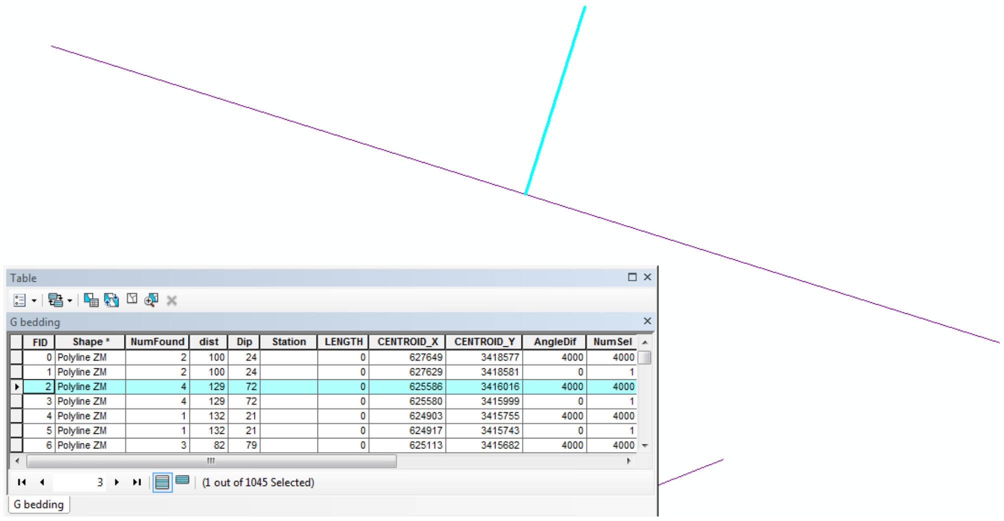
a. 
Table

G bedding

FID	Shape *	NumFound	dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel
0	Polyline ZM	2	100	24		0	627649	3418577	4000	4000
1	Polyline ZM	2	100	24		0	627629	3418581	0	1
2	Polyline ZM	4	129	72		0	625586	3416016	4000	4000
3	Polyline ZM	4	129	72		0	625580	3415999	0	1
4	Polyline ZM	1	132	21		0	624903	3415755	4000	4000
5	Polyline ZM	1	132	21		0	624917	3415743	0	1
6	Polyline ZM	3	82	79		0	625113	3415682	4000	4000

4 (1 out of 1045 Selected)

G bedding

b. 
Table

G bedding

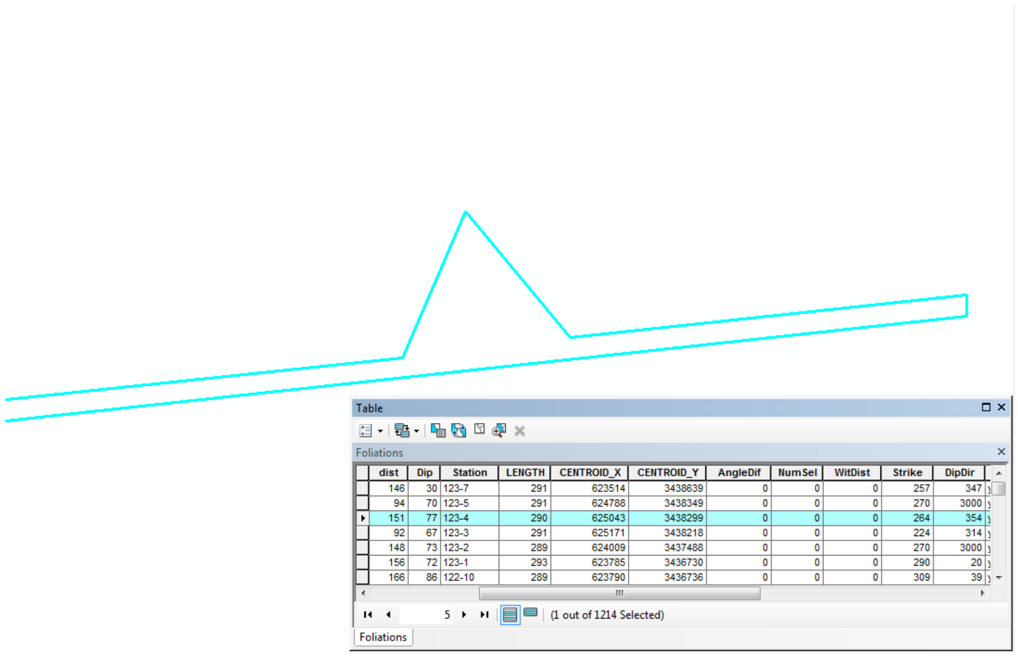
FID	Shape *	NumFound	dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel
0	Polyline ZM	2	100	24		0	627649	3418577	4000	4000
1	Polyline ZM	2	100	24		0	627629	3418581	0	1
2	Polyline ZM	4	129	72		0	625586	3416016	4000	4000
3	Polyline ZM	4	129	72		0	625580	3415999	0	1
4	Polyline ZM	1	132	21		0	624903	3415755	4000	4000
5	Polyline ZM	1	132	21		0	624917	3415743	0	1
6	Polyline ZM	3	82	79		0	625113	3415682	4000	4000

3 (1 out of 1045 Selected)

G bedding

Figure 11 Shows two different measurements with each object composing each symbol being highlighted in separate snapshots. The first symbol is a multi-object symbol shown in (a) and (b) where the measurement is represented by a primary object highlighted in blue in (a) and an accessory object highlighted in (b). The primary object is the longer object because its centroid is the location that future symbols representing this measurement should be centered on. These two objects have no affinity except for their location next to each other on the map. (c) Shows in contrast a single-object symbol where there is only one object representing this measurement.

c.



dist	Dip	Station	LENGTH	CENTROID_X	CENTROID_Y	AngleDif	NumSel	WitDist	Strike	DipDir
146	30	123-7	291	623514	3438639	0	0	0	257	347
94	70	123-5	291	624788	3438349	0	0	0	270	3000
151	77	123-4	290	625043	3438299	0	0	0	264	354
92	67	123-3	291	625171	3438218	0	0	0	224	314
148	73	123-2	289	624009	3437488	0	0	0	270	3000
156	72	123-1	293	623785	3436730	0	0	0	290	20
166	86	122-10	289	623790	3436736	0	0	0	309	39

5 (1 out of 1214 Selected)

Figure 11 Shows two different measurements with each object composing each symbol being highlighted in separate snapshots. The first symbol is a multi-object symbol shown in (a) and (b) where the measurement is represented by a primary object highlighted in blue in (a) and an accessory object highlighted in (b). The primary object is the longer object because its centroid is the location that future symbols representing this measurement should be centered on. These two objects have no affinity except for their location next to each other on the map. (c) Shows in contrast a single-object symbol where there is only one object representing this measurement.

measurement in the shapefile and excludes all the other pieces. This range will become the parameters LenMin and LenMax used below. Hopefully, there will only be as many lengths in the shapefile as there are pieces in each measurement, give or take a few meters variation. All outliers between these standard lengths, found in the sorted attribute table, should be examined. Again, check that they are actually measurements by selecting them, and clicking *zoom to selected*.

Sfactor1 is used for both single-object and multi-object (where multiple lines compose the measurement symbol) scripts, though in the first case, it is a generic buffering number used to take into account the slight variations in how different tools in

ArcGIS measure distances, and therefore angles. For example, length measurements taken with two different tools during this project were found to be as much as half a meter different, and thus in this case a value of 1 will almost always suffice. Within the multi-object workflow, the variation between lengths of the primary components, or the difference between LenMin and LenMax will determine the value of sfactor1. Among other uses, the sfactor1 parameter is used when pairing other accessory objects representing the same measurement to the primary object. Since overlapping measurements might lead to incorrect pairing, sfactor1 aids the process by excluding object of the same effective size as the main set of objects, since these main objects should only ever be paired with smaller objects making up the rest of the measurement symbol. Ultimately, this means that sfactor1 should be large enough to cover the range of the main object's lengths', but not so large as to overlap with and thus exclude any of the accompanying smaller objects. If this is a possibility, it is essential to reduce the sfactor1 value so as to make sure than no smaller accessory objects are excluded, even if this might allow for more incorrect pairing.

The final parameters that need to be determined at this point are the Verticlen1, and Verticlen2. Hopefully, as the user has been browsing the sorted lengths in the dataset being examined, any measurements representing vertical measurements would shown lines with a distinct set of accessory line lengths from the accessory lines representing the normal measurements. It is important to cover the range of lengths encompassing these measurements in two parameters, the lower bound by Verticlen1 and the upper bound by Verticlen2. Notice that this method only works on multi-object datasets, and the lengths encompassed by these parameters need only describe one of the accessory objects consistently present with vertical measurements, even if more than one accessory object is present with each measurement symbol.

Multi-object Workflow

These instructions will now walk through the rest of the steps required for datasets with multiple objects representing each measurement (Fig. 12), and then will briefly cover the same instructions for datasets with only one object per measurement in the following section. The rest of the code, in both cases, uses a variety of modules, can only handle shape files consisting of entirely lineation or foliation measurements (Fig. 12), and unlike the last script, can work on only one shapefile at a time. Since now we are dealing with multi-objects datasets, the appropriate preliminary analysis code, either **Foli_Prelim_Analysis** or **Line_Prelim_Analysis,** must be run, depending on whether the measurements being worked on are foliations or lineations. Fig. 20 give an overview of this portion of the workflow for both of the preliminary analysis workflows. Before they can be run, they require input parameters to function correctly. There are a total of 7/8 parameters on lines 7-18/19. The first three LenMin and LenMax, and sfactor1 are determined by examining the results of the length calculations in the previous step as discussed above. The fourth parameter WithinDist should be a broad overestimate of the maximum distance between the centroid of the selected main object making up each measurement and the centroid(s) of the companion objects, and can be determined by both visual estimates and using ArcMap's *measure* tool to measure some of the farther cases. If the different shapefiles are not widely different from each other this value could stay the same across processing multiple shapefiles. For example, in this project, although the final WithinDist values ranged from 10-30 meter, setting the WithinDist to 40 for all the shapefiles was more than adequate for the Preliminary Analysis code.

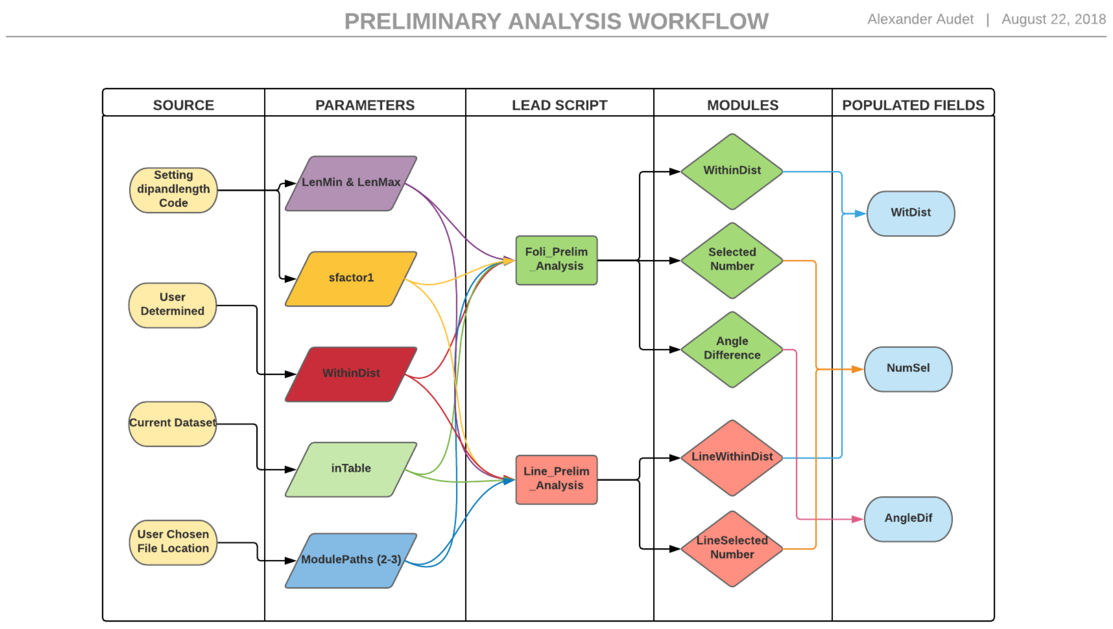


Figure 20 This diagram shows the major elements composing the preliminary analysis portion of the workflow for both foliations and lineations. The elements are organized in columns by category, and the flow of information through the workflow progress from left to right in the figure. On the far left, are the sources for the parameters which are discussed in depth in the text. The parameters, similarly discussed, are in the next column, and are composed of LenMin, LenMax, sfactor1, WithinDist, the inTable, and 2-3 module paths. Notice that a preliminary overestimate of the WithinDist value is required as an input, but also is produced in the output as WitDist, in order to determine a more accurate WithinDist value. Each of the parameters are used in both the Foli_Prelim_Analysis and the Line_Prelim_Analysis, except for the fact that three module paths are required for the first of these scripts, while only two are required for the second one, because the first uses three modules, as can be seen under the module column, and the second uses two modules. Finally, in the products of the process, under the populated fields column, on the far right of the diagram, are the three fields created and populated by the process, WitDist, NumSel, and AngleDif, the last of which is only created within the foliation workflow. WitDist is populated by both the WithinDist and the LineWithinDist modules, and provides the distance of the nearest neighbor accessory object's centroid from its primary object's centroid. NumSel, populated by the SelectedNumber and LineSelectedNumber modules, shows the number of accessory objects found within a radius, equal to the preliminary WithinDist value, from each primary object's centroid. The AngleDif is populated by the Angle Difference module, and shows the difference between the azimuth of the centroid of the primary object, to the centroid of one of the accessory objects, and the forward or the back azimuth of the perpendicular to the primary line, whichever yields a smaller angle.

The fifth parameter, inTable is set as a string of the actual name of the shapefile the user is currently working on within double quotations. The shapefile at this point must be open within ArcMap. The last two or three parameters, which need only be filled out once, are the path names to the modules that the Preliminary Analysis uses. The user placed all the modules in a folder at the beginning of these instructions and must now modify the pathname for each to the filename in that location so that the program can open the correct modules.

The three modules produce three fields that can be used to determine the parameters required if analyzing multi-object datasets with the next set of scripts, either **Working Code Foliations** or **Working Code Lineations**. The components for this part of the workflow are outlined in Fig. 21a and 21b, where 21a shows the scripts functioning for multi-object datasets. The parameters for these scripts are inputted over lines 9-36/11-33. The first module WithinDist or LineWithinDist creates a field WithinDist within the attribute table that shows the distance between the centroid of the main objects representing the measurement and the centroid of one of its potential matches. There is no reason that the distance shown is for the correct matched object, but in all cases in this project, the majority of the distances were correct. Using the sort option, the user can again individually check a few outliers and record the WithinDist parameter value for that particular shapefile to be slightly greater than that of the greatest distance between legitimately matched objects recorded in the table. If the user misses a few outliers, a later portion of the workflow should catch them and they can be manually fixed.

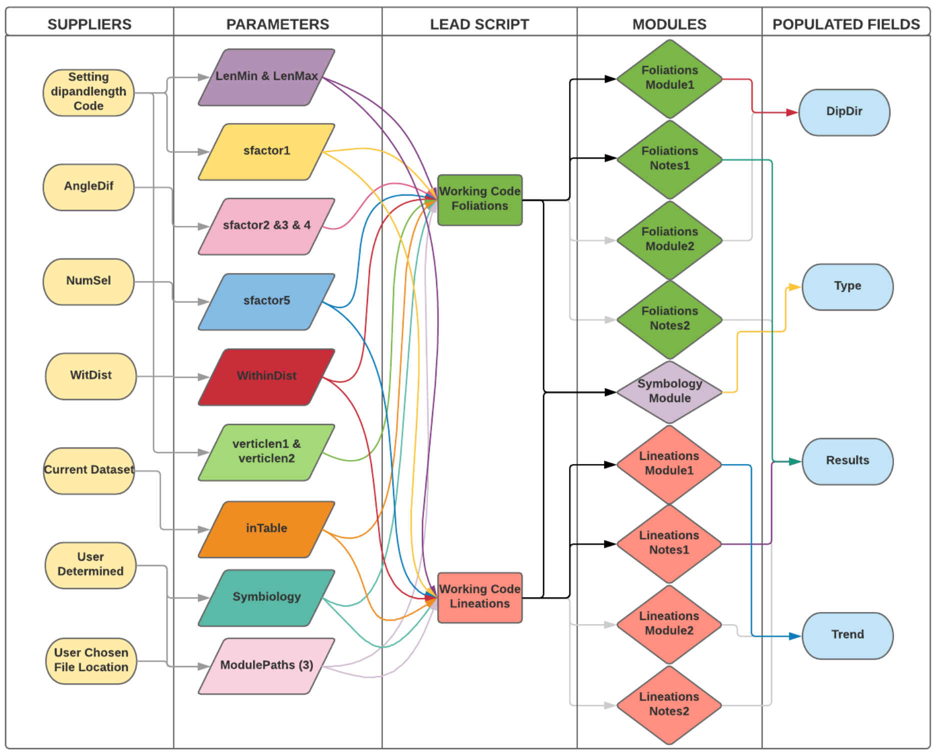


Figure 21a These diagrams show the major elements composing the Working Code portion of the workflow for both foliations and lineations in both multi-object datasets (a) and single-object datasets (b). Greyed-out elements are unused in that specific workflow. The elements are organized in columns by category, and the flow of information through the workflow progress from left to right in the figures. On the far left, are the sources for the parameters which are discussed in depth in the text. The parameters, similarly discussed, are in the next column, and are composed of LenMin, LenMax, sfactor1, sfactor 2, 3, &4, sfactor5, WithinDist, verticlen1, verticlen2 the inTable, Symbiology, and 2-3 module paths. Less parameters are required by the single-object workflow as shown in (b) where the unused ones are greyed-out, however, grey paths coming out of some of these unused parameters mark potential for future functionality in the single-object scripts. Moving to the third column, Lead Script, the parameters are used in both the Working Code Foliations and the Working Code Lineations lead scripts in order for them to correctly run their modules, shown in column 4, with the sfactor2, 3, &4 and verticlen1 & 2 not being used for the lineations multi-object scripts in (a). The lead script functions to pass on the required combination of parameters to each of its currently active modules, and then run the modules to populate the output fields shown in column 5. Thus, the products of the process, under the populated fields column on the far right of the diagram, are the four fields, DipDir, Type, Results, and Trend, the first of which is only created within the foliation workflow, and the last only within the lineation workflow. DipDir is the azimuth direction of the downdip measurement of a foliation plane that the symbol is representing. The Trend also gives an azimuth direction, but the direction of a lineation instead of a foliation dip. The Results field shows whether the structural measurement was vertical, and if the module was or was not able to correctly determine the direction of the measurement. The Type is the type of structural geological measurement the symbol represents, such as magmatic foliation, bedding, or solid-state foliation, and regional lineation, magmatic lineation, or intersection lineation.

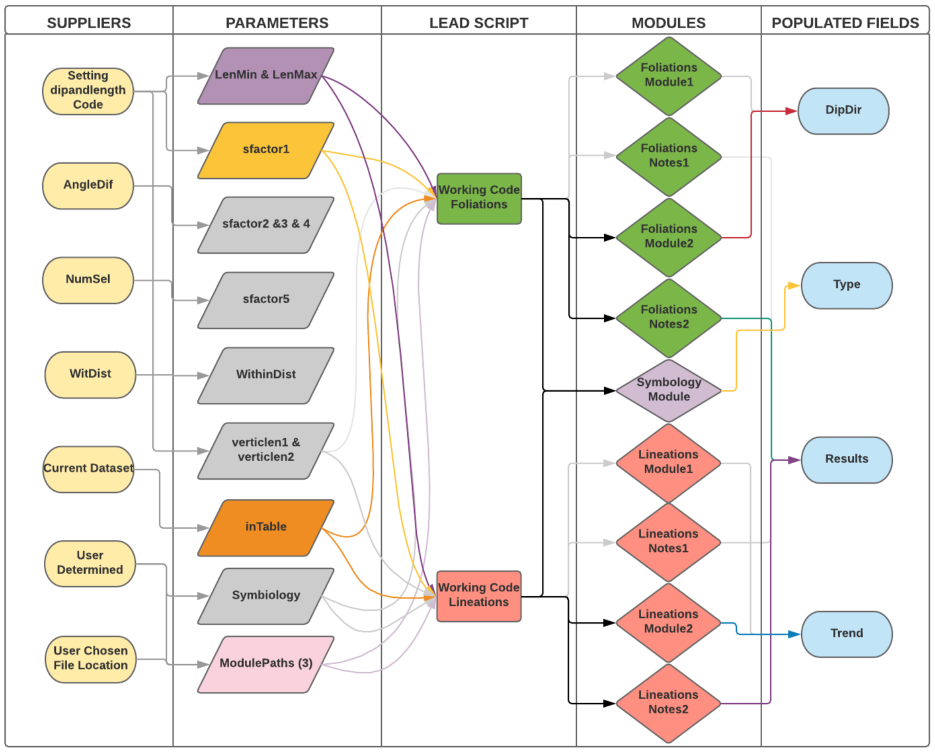


Figure 21b These diagrams show the major elements composing the Working Code portion of the workflow for both foliations and lineations in both multi-object datasets (a) and single-object datasets (b). Greyed-out elements are unused in that specific workflow. The elements are organized in columns by category, and the flow of information through the workflow progress from left to right in the figures. On the far left, are the sources for the parameters which are discussed in depth in the text. The parameters, similarly discussed, are in the next column, and are composed of LenMin, LenMax, sfactor1, sfactor 2, 3, &4, sfactor5, WithinDist, verticlen1, verticlen2 the inTable, Symbiology, and 2-3 module paths. Less parameters are required by the single-object workflow as shown in (b) where the unused ones are greyed-out, however, grey paths coming out of some of these unused parameters mark potential for future functionality in the single-object scripts. Moving to the third column, Lead Script, the parameters are used in both the Working Code Foliations and the Working Code Lineations lead scripts in order for them to correctly run their modules, shown in column 4, with the sfactor2, 3, &4 and verticlen1 & 2 not being used for the lineations multi-object scripts in (a). The lead script functions to pass on the required combination of parameters to each of its currently active modules, and then run the modules to populate the output fields shown in column 5. Thus, the products of the process, under the populated fields column on the far right of the diagram, are the four fields, DipDir, Type, Results, and Trend, the first of which is only created within the foliation workflow, and the last only within the lineation workflow. DipDir is the azimuth direction of the downdip measurement of a foliation plane that the symbol is representing. The Trend also gives an azimuth direction, but the direction of a lineation instead of a foliation dip. The Results field shows whether the structural measurement was vertical, and if the module was or was not able to correctly determine the direction of the measurement. The Type is the type of structural geological measurement the symbol represents, such as magmatic foliation, bedding, or solid-state foliation, and regional lineation, magmatic lineation, or intersection lineation.

The second module, SelectedNumber or LineSelectedNumber, creates a field SelectedNum within the attribute table that shows the number of objects within the first WithinDist value, in this case 40 meters of each main object. Again, some of the main objects will find too many potential pairs, but in this project, the majority found the correct number of associated objects. That number will become the parameter sfactor5.

The third module, AngleDifference, only occurs with the Foliation code, and creates the AngleDif field within the attribute table, which shows the angle difference of the primary object's centroid to one of the paired object's centroids from the closest angle perpendicular to the strike of the primary object. Again, in this case, the majority of the results in the field are correct, and thus when consistent, the parameter sfactor2 can be set as that resulting value.

However, sometime there is more than one common angle from the centroid of the primary object to that of the accessory object, so this code has functionality built in it to handle two different correct angles. The first parameter given above, sfactor2, is used in a test that checks for angles the value of sfacotr2 degrees to either side of the closest angle perpendicular to the strike of the main object. However, the next test looks for an angle between sfactor3 and sfactor4 from the closest angle perpendicular to the strike of the main object, where sfactor3 gives the lower limit of the range, and sfactor4 the upper limit. For example, an sfactor3 of -6 and sfactor4 of -5 would look for angles between 6-5 degrees less than the angle of the closest perpendicular to the strike. The user must take care when assigning these two parameters that they will cover the correct range to select the target angles shown in the AngleDif field.

Most of the remaining parameters Sfactor1, LenMin, LenMax are all the same as used for the Preliminary Analysis scripts, unless the user believes that the values they used for that part of the workflow were selected wrong and need to be adjusted. Similarly, the inTable value should be the same, and the Verticlen1 and Verticlen2 if set at the beginning of the workflow should also be entered. The Module path names need to be set to use the new modules required by the Working Code script. In the case that we are currently following for multi-objects, the user would use the number1 set of modules: FoliationModule1, FoliationNotes1, LineModule1, and LineNotes1.

Finally, if there is more than one type of lineation or foliation measurement in a single-object dataset, then the Symbiology parameter must be set from False to True. Additionally, the length of characteristic accessory object for each of the types of measurements must be inputted as the parameters typeA1, typeA2, typeB1, etc between lines 95-107 and 75-86. Here the first of the two parameters such as A1 and B1 should specify the lower limit of the length, and the second of the two parameters, A2 and B2 should specify the upper limit of the length.

Once all these parameter have been entered into the **Working Code Foliations** or **Lineations** script, the script can be run on the dataset currently being worked on. It produces three fields, the DipDir, Results, and Type field, and populates either only the first two, or all three if the Symbiology module is used. Then as discussed in the text, the Results module can be used to evaluate the success of the script, or if any of the records need to be manually analyzed and edited. A good way to do this is right click at the top of the Results field within the attributes table, and select sort, causing any records without either 4000, or Success to be grouped apart from these two scenarios. Then each of these records can be viewed by selecting one and clicking the Zoom to Selected. For records that could not find the correct accessory match(es), the DipDir and Strike or trend will either be pointing the correct way, or 180 degrees from the correct way. If the user determines that the second outcome is the case, then they will simply have to reverse the angle, by putting the layer into Edit mode, and retyping the angle(s) with the correct adjustment into the attribute table.

Single-object Workflow

The above procedure is far simpler, yet similar for single-object datasets. The Preliminary Analysis script is completely skipped, and the only parameters required for the Working Code scripts, again found on lines 12-38/33, are the LenMin, LenMax, and sfactor1 from the first part of the analysis, and the inTable and Module paths as shown in Fig. 21b. The module paths have to open the number 2 modules, such as the FoliationModule2, FoliationNotes2, LineModule2, and LineNotes2 instead of the number 1 modules for the Multipart datasets. The remaining parameters can be left equal to zero or else any other number, but the symbiology parameter should equal False.

The Results Field is left here for consistency, and because later additions to the code, as discussed in the discussion section of the thesis, could utilize it to record which measurements are vertical or horizontal, however, as the code currently functions, it has no practical use for the single-object measurements. It cannot be used to troubleshoot, or check the successful operation of the script.

In most cases where the Type field is not automatically filled, it was designed to be manually filled.

APPENDIX C: REGISTRATION SHIFT INSTRUCTIONS

The third appendix walks through the instructions for filling in the required parameters for using the **Registration Shift** arcpy script in Part 10 of this project.

Under the *Set Workspace* area at lines 10-12, the input workspace and output workspace addresses must be put within double quotations. Separate addresses should be chosen for the input and the output workspace so as not to corrupt any of the files. The input workspace should contain only those shapefiles that need to have their projection defined and then a copied, re-projected version made, as the program will act on all the shape files in that folder.

Next, the definition of the projection the files start in must be entered with correct syntax within the double parenthesis of line 22. It is first required that the correct spatial reference that the data is already be chosen, and this can be determined either from the source of the data or sometimes by right clicking the dataset under the *Table of Contents* within ArcMap, and going to Properties > Source, then reading the information under *Data Source*. To find the correct syntax, one should go to the pro.arcgis.com spatial reference page, http://pro.arcgis.com/en/pro-app/arcpy/classes/spatialreference.htm, and find the documentation on the coordinate system names that ArcGIS uses. Then, all the underscores must be replaced by spaces. For example, NAD_1927_UTM_Zone_11N becomes “NAD 1927 UTM Zone 11N”.

In line 33, a template file must be referenced. Once the user has determined which projection and datum they wish to use, they should create a new shapefile within ArcCatalog, which will be empty, and set the coordinate system of that to the desired coordinate system and datum. Then that shapefile's folder location and file name must be referenced within the quotes. Note that it is possible to directly reference a coordinate system definition, like was done for the starting coordinate system, instead of a template file, but that doing so would require the code on line 42 to be changed to reflect this. See the pro.arcgis.com documentation on BatchProject for more details.

The next step is to find the correct transformation to the datum desired, and place it in the parenthesis in line 38. This will require some research, however, some of the available transformations are found in a document on the Geographic Transformation Page of pro.arcgis.com again, http://pro.arcgis.com/en/pro-app/help/mapping/properties/geographic-coordinate-system-transformation.htm. In this project, we determined the best transformation to be a concatenation of two different transformations connected by a "+". Thus, the transformation used was "WGS_1984_(ITRF00)_To_NAD_1983+NAD_1927_To_NAD_1983_NADCON." Notice the syntax which does not remove the underscores in this case.

Finally, the script is ready to be run within the ArcMap terminal found under Geoprocessing > Python. Copy the whole script into there from NotePad++ or your script editing software of choice and press enter a few times until it starts running.

APPENDIX D ARCPY SCRIPTS

Setting Dipandlength

```python
#  Setting Dipandlength
#  Alexander Audet  - University of Maine
#  05/30/2018

# Import system modules
import arcpy

# Set environment settings to current folder containing the shapefiles
arcpy.env.workspace = "D:/AlexandersWork/AllData/NAD27/Numbers_Workflow"

#allowing overwriting
arcpy.env.overwriteOutput = True

# Here you set all the field names containing nonGeometry attributes to be added to all your shapefiles.
fieldName1 = "Dip"
fieldName2 = "Station"

# The script creates the file list from all the feature class files (effectively shapefiles) in the workspace set above.
files = arcpy.ListFeatureClasses()
# Then for each of those files, the script tries to add whichever fields specified, and uses AddGeometryAttributes to
# add prepopulated length, X coordinate and Y coordinate fields. If the field already exists or otherwise fails to
# be created, the script skips that portion of the script.
for inTable in files:
    try:
        arcpy.AddField_management(inTable, fieldName1, "SHORT")
    except:
        pass
    try:
        arcpy.AddField_management(inTable, fieldName2, "TEXT")
    except:
        pass
    try:
        arcpy.AddGeometryAttributes_management(inTable, "LENGTH", "METERS")
    except:
        pass
    try:
        arcpy.AddGeometryAttributes_management(inTable, "CENTROID")
    except:
        pass
```

Foli_Prelim_Analysis

```python
#  Foli_Prelim_Analysis
#  Alexander Audet
#  05/30/2018

#See Working Code Foliations for more complete notations on the functioning of the lead scripts.

######## User Inputs ########
LenMin = 175
LenMax = 195
sfactor1 = 1
WithinDist = 40

# Tables: S0, G wr foliation, Sa, Sr, Murray Sp, Sr_1999_1, SoSa, S0Sr, S0 add, S0 1999_1, G bedding, G extra beddings, Murray extra So, Murray extra Sp, G pluton Foliation
inTable = "G pluton Foliation"

f = open("D:/AlexandersWork/PythonModules/AngleDifference.py",'r')
f2 = open("D:/AlexandersWork/PythonModules/SelectedNumber.py",'r')
f3 = open("D:/AlexandersWork/PythonModules/WithinDist.py",'r')
######## User Inputs ########


fieldName = "AngleDif"
expression = "Strike( !Shape!, !FID! )"
codeblock = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax = ' +str(LenMax) +'\r'+'\n'+ f.read()

fieldName2 = "NumSel"
codeblock2 = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax = ' +str(LenMax) +'\r'+'\n'+ f2.read()

fieldName3 = "WitDist"
codeblock3 = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax = ' +str(LenMax) +'\r'+'\n'+ f3.read()

try:
    arcpy.AddField_management(inTable, fieldName, "SHORT")
except:
    pass

arcpy.CalculateField_management(inTable, fieldName, expression, "PYTHON_9.3", codeblock)

try:
    arcpy.AddField_management(inTable, fieldName2, "SHORT")
except:
    pass
arcpy.CalculateField_management(inTable, fieldName2, expression, "PYTHON_9.3", codeblock2)

try:
    arcpy.AddField_management(inTable, fieldName3, "SHORT")
except:
    pass
arcpy.CalculateField_management(inTable, fieldName3, expression, "PYTHON_9.3", codeblock3)
```

AngleDifference

```python
#  AngleDifference
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid):  # See FoliationModule1 for notation explaining this fuction. Only info unique to
# this module will be included here.

    length = shape.length

    if length == 0:
        return 7000

    if length >= LenMin and length <= LenMax:

        xPoint = (shape.firstPoint.X - shape.lastPoint.X)
        yPoint = (shape.firstPoint.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        degreeCorrection = math.atan2(xPoint, yPoint) * conversionFactor

        degreeBearing = degreeCorrection + 180 + 90

        if degreeBearing >= 360:
            degreeBearing -= 360


        arcpy.MakeFeatureLayer_management(inTable, 'shapelyr', '"FID" ='+str(fid))
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))
        
        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])

        items = []
        for item in cursor:

            if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
                distcheck = math.hypot(shape.trueCentroid.X - item[0][0], shape.trueCentroid.Y - item[0][1])
                if distcheck <= WithinDist:
                    items.append(item)

        for result in items:
            xPointb = (result[0][0] - shape.trueCentroid.X)
            yPointb = (result[0][1] - shape.trueCentroid.Y)
            degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
            degreeb = degreeCorrb + 180
            degreec = degreeb + 180
            if degreec >= 360:
                degreec -= 360


            # The script calculates the difference between the angle measured to the first object in the
            # cursor from the centroid of the objects currently being iterated over by CalculateField shown
            # as both degreeb and degreec and the dip direction (degreeBearing) calculated above. This will
            # help the user determine appropriate sfactor1-4 values to use. The closest difference
            # (degreeDifference1 or degreedifference2) is returned to take into account back azimuths.

            degreedifference1 = degreeb - degreeBearing
            degreedifference2 = degreec - degreeBearing
            if math.fabs(degreedifference2) > math.fabs(degreedifference1):
                return str(degreedifference1)
            else:
                return str(degreedifference2)
    else:
        return 4000
```

SelectedNumber

```
#  SelectedNumber
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid): # See FoliationModule1 for notation explaining this fuction. Only info unique to
# this module will be included here.

    length = shape.length

    if length == 0:
        return 7000

    if length >= LenMin and length <= LenMax:

        arcpy.MakeFeatureLayer_management(inTable, 'shapelyr', '"FID" ='+str(fid))
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))
        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])

        items = []
        for item in cursor:
            if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
                distcheck = math.hypot(shape.trueCentroid.X - item[0][0], shape.trueCentroid.Y - item[0][1])
                if math.fabs(distcheck) <= WithinDist:
                    items.append(item)

        # returns the number of objects potentially associated with the current strike line (primary object).
        return len(items)
    else:
        return str(4000)
```

<u>WithinDist</u>

```
#··WithinDistCRLF
#··Alexander·Audet··-·University·of·MaineCRLF
#··05/30/2018CRLF
CRLF
#·This·module·is·for·multi-object·datasetsCRLF
CRLF
def·Strike(shape,·fid):·#·See·FoliationModule1·for·notation·explaining·this·fuction.·Only·info·unique·toCRLF
#·this·module·will·be·included·here.CRLF
CRLF
····length·=·shape.lengthCRLF
CRLF
····if·length·==·0:CRLF
········return·7000CRLF
CRLF
····if·length·>=·LenMin·and·length·<=·LenMax:CRLF
CRLF
········arcpy.MakeFeatureLayer management(inTable,·'shapelyr',·'"FID"·='+str(fid))CRLF
········arcpy.MakeFeatureLayer_management(inTable,·'PairedCheck8',·'"FID"·<>'+str(fid))CRLF
CRLF
········arcpy.SelectLayerByLocation management('PairedCheck8',·'WITHIN A DISTANCE',·'shapelyr',·'12·METERS')CRLF
CRLF
········cursor·=·arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID',·'SHAPE@LENGTH'])CRLF
CRLF
········for·result·in·cursor:CRLF
············if·result[1]·<=·length·-·sfactor1·or·result[1]·>=·length·+·sfactor1:CRLF
················distcheck·=·math.hypot(shape.trueCentroid.X·-·result[0][0],·shape.trueCentroid.Y·-·result[0][1])CRLF
················#·returns·the·distance·of·the·objects·potentially·associated·with·the·current·strike·lineCRLF
················#·(primary·object)·to·the·centroid·of·that·strike·line.CRLF
················return·distcheckCRLF
CRLF
····else:CRLF
········return·4000
```

Line_Prelim_Analysis

```python
#  Line_Prelim_Analysis
#  Alexander Audet
#  05/30/2018

#See Working Code Foliations for more complete notations on the functioning of the lead scripts.

######## User Inputs ########
LenMin = 180
LenMax = 190
sfactor1 = 1
WithinDist = 40

# Tables: La, LLo, LaLo, Lp, G lineations
inTable = "G lineations"

f2 = open("D:/AlexandersWork/PythonModules/LineSelectedNumber.py",'r')
f3 = open("D:/AlexandersWork/PythonModules/LineWithinDist.py",'r')
######## User Inputs ########



fieldName2 = "NumSel"
expression2 = "Strike( !Shape!, !FID! )"
codeblock2 = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax = ' +str(LenMax) +'\r'+'\n'+ f2.read()

fieldName3 = "WitDist"
expression3 = "Strike( !Shape!, !FID! )"
codeblock3 = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax = ' +str(LenMax) +'\r'+'\n'+ f3.read()

try:
    arcpy.AddField_management(inTable, fieldName2, "SHORT")
except:
    pass
arcpy.CalculateField_management(inTable, fieldName2, expression2, "PYTHON_9.3", codeblock2)

try:
    arcpy.AddField_management(inTable, fieldName3, "SHORT")
except:
    pass
arcpy.CalculateField_management(inTable, fieldName3, expression3, "PYTHON_9.3", codeblock3)
```

LineSelectedNumber

```python
#  LineSelectedNumber
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid): # See LineModule1 for notation explaining this fuction. Only info unique to
# this module will be included here.

    length = shape.length

    if length == 0:
        return 7000

    if length >= LenMin and length <= LenMax:

        arcpy.MakeFeatureLayer_management(inTable, 'shapelyr', '"FID" ='+str(fid))
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))
        
        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])

        items = []
        for item in cursor:
            if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
                distcheck1 = math.hypot(shape.firstPoint.X - item[0][0], shape.firstPoint.Y - item[0][1])
                distcheck2 = math.hypot(shape.lastPoint.X - item[0][0], shape.lastPoint.Y - item[0][1])
                if distcheck1 <= WithinDist or distcheck2 <= WithinDist:
                    items.append(item)

        # returns the number of objects potentially associated with the current lineation line
        # (primary object).
        return len(items)
    else:
        return str(4000)
```

LineWithinDist

```python
#  LineWithinDist
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid): # See LineModule1 for notation explaining this fuction. Only info unique to
# this module will be included here.

    length = shape.length

    if length == 0:
        return 7000

    if length >= LenMin and length <= LenMax:

        arcpy.MakeFeatureLayer management(inTable, 'shapelyr', '"FID" ='+str(fid))
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))

        arcpy.SelectLayerByLocation management('PairedCheck8', 'WITHIN A DISTANCE', 'shapelyr', '1 METERS')

        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])

        for result in cursor:
            if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:
                distcheck1 = math.hypot(shape.firstPoint.X - result[0][0], shape.firstPoint.Y - result[0][1])
                distcheck2 = math.hypot(shape.lastPoint.X - result[0][0], shape.lastPoint.Y - result[0][1])
                # returns the distance of the objects potentially associated with the current lineation line
                # (primary object) to the closest end of that lineation line. 
                if distcheck1 > distcheck2:
                    return distcheck2
                else:
                    return distcheck1
        return 8000

    else:
        return 4000
```

<u>Working Code Foliations</u>

```
1   #  Working Code FoliationsCRLF
2   #  Alexander Audet  - University of MaineCRLF
3   #  05/30/2018CRLF
4   CRLF
5   # Import system modulesCRLF
6   import arcpyCRLF
7   CRLF
8   CRLF
9   ######## User Inputs ########CRLF
10  CRLF
11  # Select the inTable (the shapefile full of data you are currently working on)CRLF
12  # Tables: Zarza data, Murray Sp, Foliations, Murray S0, S0, G wr foliation, Sa, Sr, Sr 
    1999_1, SoSa, S0Sr, S0 add, S0 1999_1, G bedding, G extra beddings, Murray extra So, 
    Murray extra Sp, G pluton Foliation, Sp, Sp 1999 1CRLF
13  inTable = "Murray S0"CRLF
14  CRLF
15  # Variable to set for your current job. See instructions for running workflow for 
    parameter descriptions and how to obtain the correct values.CRLF
16  sfactor1 = 1 #This value will only change if the degree of variability of the length of 
    similarly sized objects becomes greater than 1.CRLF
17  sfactor2 = 3CRLF
18  sfactor3 = 0 #This and sfactor4 are added to the degreebearing. Make these values 
    negative and/or positive accordingly.CRLF
19  sfactor4 = 0CRLF
20  sfactor5 = 1CRLF
21  WithinDist = 30CRLF
22  LenMin = 160CRLF
23  LenMax = 200CRLF
24  horzlen1 = 135CRLF
25  horzlen2 = 145CRLF
26  Symbology = False #If Symbology is set to true, parameters on lines 95-105 below should 
    be filled in.CRLF
27  CRLF
28  # list of modules: FoliationModule1.py, FoliationModule2.pyCRLF
29  # Pick the appropriate module for the intable set above and add it to the end of the 
    file address below. File address should point to where the modules have been stored.CRLF
30  f = open("D:/AlexandersWork/PythonModules/FoliationModule1.py",'r')CRLF
31  CRLF
32  # list of modules: FoliationNotes1.py, FoliationNotes2.pyCRLF
33  # Pick the appropriate module for the intable set above and add it to the end of the 
    file address below. File address should point to where the modules have been stored.CRLF
34  f2 = open("D:/AlexandersWork/PythonModules/FoliationNotes1.py",'r')CRLF
35  CRLF
36  ######## User Inputs ########CRLF
37  CRLF
38  CRLF
39  ##### DipDir FieldCRLF
40  CRLF
41  # This sets the field name (for the dip direction) to be created and then populated by 
    the dip direction azimuth values when the CalculateField fuction is run on this field 
    below. CRLF
42  fieldName = "DipDir"CRLF
43  # This sets the expression used within the CalculateField function with the correct 
    syntax. The function Strike is defined within the module called within the codeblock 
    (line 48 below).CRLF
44  # The function Strike is run by CalculateField for each object in the inTable and 
    returns the values CalculateField assigns as the field values for each object. The 
    Shape and FID of CRLF
45  # each object iterated over are set as the parameters the Strike function requires each 
    time CalculateField runs it.CRLF
46  expression = "Strike( !Shape!, !FID! )"CRLF
47  # The codeblock calls the necessary moldule for calculating the field defined above: 
    "fieldname". It also passes all the parameters set above required in the module into 
    that module.CRLF
48  codeblock = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) 
    +'\r'+'\n'+ 'sfactor2 = ' +str(sfactor2) +'\r'+'\n'+ 'sfactor3 =' +str(sfactor3) 
    +'\r'+'\n'+ 'sfactor4 =' +str(sfactor4) +'\r'+'\n'+ 'sfactor5 =' +str(sfactor5) 
```

```
+'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax =' +str(LenMax) +'\r'+'\n'+ 'horzlen1 =' +str(horzlen1) +'\r'+'\n'+ 'horzlen2 =' +str(horzlen2) +'\r'+'\n'+ f.read()CRLF
CRLF
##### Result Field CRLF
CRLF
# The same set of variables (a fieldname, expression and codeblock) are defined here for the Results field as were defined above for the DipDir field. Refer to the DipDir for a CRLF
#complete explanantion of these elements.CRLF
CRLF
# This sets the field name (for the result field) to be created and then populated by an indication of whether the CalculateField fuction could correctly find each primaryCRLF
# object's accessory objects, and thus correctly calculate its dip direction azimuths.CRLF
fieldName2 = "Result"CRLF
# Because the codeblock2 below calls a different module than the codeblock above, this Strike function here will be different from that used to calculate the DipDir field. CRLF
expression2 = "Strike( !Shape!, !FID! )"CRLF
codeblock2 = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'sfactor2 = ' +str(sfactor2) +'\r'+'\n'+ 'sfactor3 =' +str(sfactor3) +'\r'+'\n'+ 'sfactor4 =' +str(sfactor4) +'\r'+'\n'+ 'sfactor5 =' +str(sfactor5) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax =' +str(LenMax) +'\r'+'\n'+ 'horzlen1 =' +str(horzlen1) +'\r'+'\n'+ 'horzlen2 =' +str(horzlen2) +'\r'+'\n'+ f2.read()CRLF
CRLF
CRLF
# Adds the dip direction (DipDir) field that will be populated by the dip direction azimuth, to the inTable .CRLF
try:CRLF
    arcpy.AddField_management(inTable, fieldName, "SHORT")CRLF
except:CRLF
    passCRLF
CRLF
# Adds the Result field that will be populated by the success the script had properly analyizing each object to the inTable.CRLF
# Error and instances that need to be manually entered will be noted here.CRLF
try:CRLF
    arcpy.AddField_management(inTable, fieldName2, "TEXT")CRLF
except:CRLF
    passCRLF
CRLF
# The Type fieldname is set here instead of below with the rest of the Type information under Symbology so that the Type field can be created regardless of whether the script calculatesCRLF
# its values. This is because, the user will calculate the values manually if the script is not used by setting Symbology to True in the parameters at the top.CRLF
fieldName3 = "Type"CRLF
CRLF
# Adds the Type field that will be populated by the structural geological measurement type of each object to the inTable.CRLF
try:CRLF
    arcpy.AddField_management(inTable, fieldName3, "TEXT")CRLF
except:CRLF
    passCRLF
CRLF
# Calcualtes the values for the DipDir field just created using the appropriate expression and codeblock defined above.CRLF
arcpy.CalculateField_management(inTable, fieldName, expression, "PYTHON_9.3", codeblock)CRLF
CRLF
# Calcualtes the values for the Result field just created using the appropriate expression and codeblock defined above.CRLF
arcpy.CalculateField_management(inTable, fieldName2, expression2, "PYTHON_9.3", codeblock2)CRLF
CRLF
##### Symbol TypeCRLF
```

```

# The Type field is only populated if Symbology is set to True.
if Symbology:
    # The following module retuns the geological measurement type based on the shape (determined here by length) of the dip symbol.
    typeA = "Magmatic Foliation"
    typeA1 = 118
    typeA2 = 120
    typeB = "Bedding"
    typeB1 = 20
    typeB2 = 60
    typeC = "Solid State Foliation"
    typeC1 = 65
    typeC2 = 96
    # File address should point to where the modules have been stored.
    f3 = open("D:/AlexandersWork/PythonModules/SymbologyModule.py",'r')

# The same set of variables (an expression and codeblock) are defined here for the Results field as were defined above for the DipDir field. Refer to the DipDir for a 
#complete explanantion of these elements.

    expression3 = "Strike( !Shape!, !FID! )"
    codeblock3 = 'typeA = "' +typeA +'"'+'\r'+'\n'+ 'typeB = "' +typeB +'"'+'\r'+'\n'+ 'typeC = "' +typeC +'"'+'\r'+'\n'+ 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'sfactor2 = ' +str(sfactor2) +'\r'+'\n'+ 'sfactor3 =' +str(sfactor3) +'\r'+'\n'+ 'sfactor4 =' +str(sfactor4) +'\r'+'\n'+ 'sfactor5 =' +str(sfactor5) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax =' +str(LenMax) +'\r'+'\n'+ 'typeA1 =' +str(typeA1) +'\r'+'\n'+ 'typeA2 =' +str(typeA2) +'\r'+'\n'+ 'typeB1 =' +str(typeB1) +'\r'+'\n'+ 'typeB2 =' +str(typeB2) +'\r'+'\n'+ 'typeC2 =' +str(typeC2) +'\r'+'\n'+ 'typeC1 =' +str(typeC1) +'\r'+'\n'+ f3.read()

# Calcualtes the values for the Type field created using the appropriate expression and codeblock defined above.
    arcpy.CalculateField_management(inTable, fieldName3, expression3, "PYTHON_9.3", codeblock3)
```

FoliationModule1

```python
#  FoliationModule1
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid): # The function is defined and the parameter set as the "shape" and "fid" of the object being
# iterated over so that the script can call upon this information of each object that passes through this fucntion by way
# of the CalculateField function within the lead script: Working Code Foliations.

    # Calculate the length of the object being iterated over by CalculateField
    length = shape.length

    # This checks that the program correctly calculated the length. Otherwise the script returns an error code.
    if length == 0:
        return 7000

    # The following if statement is to differentiate the strike symbols (or primary objects) based on its length and must
    # thus use length limits that include the strike length and exclude all other shapes in the shapefile including accessory
    # objects. This ensures that the azimuth is only calculated on strike symbols, with all other objects returning the
    # error code 4000 by the else clause at the bottom of the script.
    if length >= LenMin and length <= LenMax:

        # The script calculates the strike azimuth based on Trig and what the program has recorded as the first and last point of each
        # polyline to be. However, it takes the first point and subtract the last point, instead of the other way around because
        # this calculates the back azimuth, so it can simplily add 180 degrees to the calculated angle instead of having to
        # change all angles between -180 to 0 degrees to their corresponding angles between 180 to 360 degrees.
        xPoint = (shape.firstPoint.X - shape.lastPoint.X)
        yPoint = (shape.firstPoint.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        # The X, and Y term are switched within the atan2() fuction so as to calulate the angle from North, or the positive X-axis
        # instead of from the trigonometric x-axis.
        degreeCorrection = math.atan2(xPoint, yPoint) * conversionFactor
        # When finnished, the angle is a backwards azimuth between -180 to 180 degrees. So the script adds 180 to that angle and get the positive 
        # azimuth between 0-360 degrees. The script then adds an addtional 90 degrees to it to calcualte the angle of the right perpendicular to the azimuth.
        # In other words the dip direciton is calculated from the strike direction.
        degreeBearing = degreeCorrection + 180 + 90
        # Here the script fixes the angle if it is greater than 360 degrees
        if degreeBearing >= 360:
            degreeBearing -= 360


        # The shapelyr1 feature layer out of the current intable set within the lead script, but only select the current object being
        # calculated by using the where-statement to select only the object with the current FID (the "fid" is one of the two parameters of the
        # function set on line 5). This layer will be used to create a geometry object down near the bottom of the code.
        arcpy.MakeFeatureLayer_management(inTable, 'shapelyr1', '"FID" ='+str(fid))
        # The PairedCheck8 feature layer out of my current intable set within the lead script, but excluding the current object being
        # calculated by using the where statement to select only objects that do not match the current FID. This layer will be used 
        # to select the matching dip symbol (or accessory object) that belongs with the 
```

```
            strike symbol (or primary object) for which the azimuth isCRLF
50          # being calculated.CRLF
51          arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" 
            <>'+str(fid))CRLF
52  CRLF
53          # Next the script creates a curser object which it uses to find the 
            TrueCentroid and Length of each of the features in PairedCheck8.CRLF
54          cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 
            'SHAPE@LENGTH'])CRLF
55  CRLF
56  CRLF
57          # Create items, an empty list to be filled by only the cursor objects, composed 
            of centroid/length pairs, within a certian set distanceCRLF
58          # (WithinDist) from and not of the same length (+/- sfactor1) of the object the 
            CalculateField function in the lead script: Working CodeCRLF
59          # Foliations is currently iterating over.CRLF
60          items = []CRLF
61          for item in cursor:CRLF
62              # As mentioned above, this excludes objects of the same length as the 
                objects being iterated over since its matching accessoryCRLF
63              # object will be a different length. Note that since the cursor calulates a 
                different geometry length from the process at theCRLF
64              # begining of this code, I put a 1 unit buffer on the interated object 
                length for which the cursor objects cannot overlap to beCRLF
65              # added to the items[] list.CRLF
66              if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:CRLF
67                  distcheck = math.hypot(shape.trueCentroid.X - item[0][0], 
                    shape.trueCentroid.Y - item[0][1])CRLF
68                  # This part of the code, as mentioned above, checks each object in turn 
                    that passed the length test to see if the distance betweenCRLF
69                  # each object in the cursor and the object currently being interated 
                    over by the CalculateField function in the lead script, distcheckCRLF
70                  # is smaller than the set WithinDist parameter, and only adds the 
                    object to items[] if distcheck is smaller. Thus, objects farther than 
                    CRLF
71                  # WithinDist are excuded from the list of potential accessory objects. 
                    CRLF
72                  if math.fabs(distcheck) <= WithinDist:CRLF
73                      items.append(item)CRLF
74  CRLF
75          # Create results[], an empty list to be filled by the best candidates for the 
            correct accessory objects from the list (items[]) of potentiallyCRLF
76          # matching cursor objects (composed of only the centroid/length pairs of each 
            candidate).CRLF
77          results = []CRLF
78          # First however I check whether the number of selected items[] matches the 
            number that should be found, set by the parameter sfactor5, in whichCRLF
79          # case I assume that these items[] are all correct accessory object(s). Thus I 
            skip the logic on lines 80-176.CRLF
80          if len(items) <> sfactor5:CRLF
81  CRLF
82              # The following three sets of for-in statements, the second and third one 
                nested inside an if statement, narrow down the centroid/length pairs soCRLF
83              # that the only one(s) added to the results[] list are the centroid/length 
                of the dip symbol(s) (accessory object(s)) that correspond to the strike 
                symbolCRLF
84              # CalculateField is currently determining the dip direction of.CRLF
85  CRLF
86              for result in items:CRLF
87                  # Here I determine the adjacent and opposite lengths of a right 
                    triangle whose hypotenuse extends from the iterated object's midpoint 
                    to each candidateCRLF
88                  # accessory object's centroid in turn, and use the calculated values to 
                    return the angle from the iterated object's midpoint to each accessory 
                    object'sCRLF
89                  # centroid using the atan trigonometric function.CRLF
90                  xPointb = (result[0][0] - shape.trueCentroid.X)CRLF
```

```
                yPointb = (result[0][1] - shape.trueCentroid.Y)
                degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
                # Then I add 180 twice to find two angles in opposite directions from one another. The first time corrects for the instances when the
                # above calculation gives negative angles (it returns angles between -180 to 180) following the procedure used in the first set of angle calculations. 
                degreeb = degreeCorrb + 180
                degreec = degreeb + 180
                # Here I fix the angle if it is greater than 360 degrees
                if degreec >= 360:
                    degreec -= 360

                #Set range of angles based on inputted sfactor2 for the above calculated angle or its reverse azimuth to be within in order to compute as a match.
                LC = degreeBearing - sfactor2
                HC = degreeBearing + sfactor2
                # This script makes sure that when comparing angles near the 360-0 degree border, it properly checks the angle similarity
                if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):
                    degreeb -= 360
                    degreec -= 360
                if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):
                    degreeb += 360
                    degreec += 360

                # If the computed angle for the current cursor object does lie within the set range of the calcualted dip direction or its reverse 
                # angle, then I append that angle to the results list.
                if LC <= degreeb <= HC or LC <= degreec <= HC:
                    results.append(result)


            # I repeat the above procedure if the results are still zero because in some cases, the angle to the correct paired object is between
            # 5-6 degrees less than the calculated dip direction.
            if len(results) == 0:
                for result in items:
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:
                        xPointb = (result[0][0] - shape.trueCentroid.X)
                        yPointb = (result[0][1] - shape.trueCentroid.Y)
                        degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
                        degreeb = 180 + degreeCorrb
                        degreec = degreeb + 180
                        if degreec >= 360:
                            degreec -= 360

                        # Set range of angles this time based on inputted sfactor3 and sfactor4 values so that only items[] objects making an angle between sfacotr3
                        # and sfacotr4 (or an angle with a back-azimuth between those values) can be selected as correct accessory objects.
                        LC = degreeBearing + sfactor3
                        HC = degreeBearing + sfactor4
                        # This script makes sure that when comparing angles near the 360-0 degree border, it properly checks the angle similarity
                        if degreeBearing <= sfactor3 and (degreeb >= 360 - sfactor3 or degreec >= 360 - sfactor3):
                            degreeb -= 360
                            degreec -= 360

                        if LC <= degreeb <= HC or LC <= degreec <= HC:
                            results.append(result)
```

```
CRLF
            # I again repeat the above procedure if no other items have still been added to results but this time create a new cursor object to check the angle a different way.CRLF
            # This part of the code is suppose to catch horizontal measurements where the accessory objects centroid nearly sits on top of the primary object's centroid, making theCRLF
            # above logic insufficient to recognize the correct accessory object.CRLF
            if len(results) == 0:CRLF
                # Produce a new cursor with complete SHAPE@ attributes. The SHAPE@ geometry info slows down the program so is only used as a last resort of a sort.CRLF
                cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH', 'SHAPE@'])CRLF
                for result in cursor:CRLF
                    # Repeat previous logic from above the above two secitons to narrow the cursor items used.CRLF
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:CRLF
                        distcheck = math.hypot(shape.trueCentroid.X - result[0][0], shape.trueCentroid.Y - result[0][1])CRLF
                        if math.fabs(distcheck) <= WithinDist:CRLF
                            # Repeat the above two sections except use the firstPoint coordinates from added SHAPE@ geometry info instead of centroid coordinates.CRLF
                            xPointb = (result[2].firstPoint.X - shape.trueCentroid.X)CRLF
                            yPointb = (result[2].firstPoint.Y - shape.trueCentroid.Y)CRLF
                            degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactorCRLF
                            degreeb = 180 + degreeCorrbCRLF
                            degreec = degreeb + 180CRLF
                            if degreec >= 360:CRLF
                                degreec -= 360CRLF
                            CRLF
                            #Set range of angles based on inputted sfactor2 for the above calculated angle or its reverse angle to be within in order to compute as a match.CRLF
                            LC = degreeBearing - sfactor2CRLF
                            HC = degreeBearing + sfactor2CRLF
                            # This script makes sure that when comparing angles near the 360-0 degree border, it properly checks the angle similarityCRLF
                            if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):CRLF
                                degreeb -= 360CRLF
                                degreec -= 360CRLF
                            if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):CRLF
                                degreeb += 360CRLF
                                degreec += 360CRLF
                            CRLF
                            # If the computed angle for the current cursor object does lie within the set range of the calcualted dip direction or its reverse CRLF
                            # angle, then the script appends that centroid/length pair object to the results list.CRLF
                            if LC <= degreeb <= HC or LC <= degreec <= HC:CRLF
                                results.append(result)CRLF
CRLF
        # If there are the right number of objects (sfactor5) in the cursor, the script skips to adding them all to the results list.CRLF
        else:CRLF
            for result in items:CRLF
                results.append(result)CRLF
CRLF
CRLF
        # I create my centroid object which I will then populate with the TrueCentroid
```

```python
        of one of the items in the results
        # if there is only sfactor5 number of items in the results
        centroid = None
        # Checks if there is sfactor5 numbers of items in the results:
        if len(results) == sfactor5:

            # Next I determine by checking the lengths of the correctly selected objects with the set parameters horzlen1 and horzlen2 if the
            # symbol represents a horizontal measurement. If so, there is no dip direciton and thus I return a strike
            for item in results:
                if horzlen1 <= item[1] <= horzlen2:
                    degreeBearing -= 90
                    if degreeBearing < 0:
                        degreeBearing += 360
                    return degreeBearing


            # Sets the first result item's centroid coordinates as the centroid object
            centroid = results[0][0]

            #I then have to turn the coordinates within the centroid into a point object so that it can be used as an input in queryPointAndDistance function below.
            ptg = arcpy.Point(*centroid)

            # Create an internal geometry object from the object belonging to the record CalculateField is currently iterating over (The current feature
            # for which the dip-direction is being calculated)
            # This is so that a queryPointAndDistance can be performed from this geometry object

            # first I define the geometry object template
            g = arcpy.Geometry()
            # Next I copy my single shapelyr1 object (the object currently being iterated over by CalculateField) and copy it into the geometryList while 
            # defining it as a geometry using the template set above.
            geometryList = arcpy.CopyFeatures_management('shapelyr1', g)
            # I take the geometry object out of list form by setting shapeG to the first (and only) list item.
            shapeG = geometryList[0]

        # Return the the calulated angle degreeBearing at this point if the results do not have the specified number of objects set by sfactor 5. The correct
        # diretion will have to be determined manually at this point.
        else:
            return degreeBearing

        # Next perform a queryPointAndDistance to determine if the dip symbol is to the right of the strike line using the direction of the strike symbol
        # from the starting point to the end point as forward.
        (pointGeometry, distance, minDistance, isCentroidRight) = shapeG.queryPointAndDistance( ptg, False)

        # The follwoing if statement checks if the dip symbol is to the left of the strick symbol 
        # If it is to the left, then 180 degrees are added to the dip direction (named degreeBearing)
        if not isCentroidRight:
            degreeBearing += 180
        # This if statement makes sure that the degreeBearing value is between 0 and 360 degrees.
        if degreeBearing >= 360:
            degreeBearing -= 360

      #returns the degreebearing value as the dip direction



        return degreeBearing
    # For the objects CalculateField iterates over that are not of the specified length to be strike symbols (including the accessory objects that are associated
    # with the strike symbols to shows dip direciton) the code 4000 is applied instead of a dip direction azimuth value for identification purposes.
    else:
        return 4000
```

FoliationNotes1

```python
#  FoliationNotes1
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid): # See FoliationModule1 for notation explaining this fuction. Only info unique to the Notes Module will be included here.

    length = shape.length

    if length == 0:
        return 7000

    if length >= LenMin and length <= LenMax:

        xPoint = (shape.firstPoint.X - shape.lastPoint.X)
        yPoint = (shape.firstPoint.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        degreeCorrection = math.atan2(xPoint, yPoint) * conversionFactor

        degreeBearing = degreeCorrection + 180 + 90

        if degreeBearing >= 360:
            degreeBearing -= 360

        arcpy.MakeFeatureLayer_management(inTable, 'shapelyr', '"FID" ='+str(fid))
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))

        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])

        items = []
        for item in cursor:

            if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
                distcheck = math.hypot(shape.trueCentroid.X - item[0][0], shape.trueCentroid.Y - item[0][1])
                if math.fabs(distcheck) <= WithinDist:
                    items.append(item)

        results = []

        if len(items) <> sfactor5:

            for result in items:

                xPointb = (result[0][0] - shape.trueCentroid.X)
                yPointb = (result[0][1] - shape.trueCentroid.Y)
                degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor

                degreeb = degreeCorrb + 180
                degreec = degreeb + 180
                if degreec >= 360:
                    degreec -= 360

                LC = degreeBearing - sfactor2
                HC = degreeBearing + sfactor2
                if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):
                    degreeb -= 360
                    degreec -= 360
                if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):
                    degreeb += 360
                    degreec += 360
```

```
CRLF
                if LC <= degreeb <= HC or LC <= degreec <= HC:CRLF
                    results.append(result)CRLF
CRLF
            if len(results) == 0:CRLF
                for result in items:CRLF
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:CRLF
                        xPointb = (result[0][0] - shape.trueCentroid.X)CRLF
                        yPointb = (result[0][1] - shape.trueCentroid.Y)CRLF
                        degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactorCRLF
                        degreeb = 180 + degreeCorrbCRLF
                        degreec = degreeb + 180CRLF
                        if degreec >= 360:CRLF
                            degreec -= 360CRLF
CRLF
                        LC = degreeBearing + sfactor3CRLF
                        HC = degreeBearing + sfactor4CRLF
                        if degreeBearing <= sfactor3 and (degreeb >= 360 - sfactor3 or degreec >= 360 - sfactor3):CRLF
                            degreeb -= 360CRLF
                            degreec -= 360CRLF
CRLF
                        if LC <= degreeb <= HC or LC <= degreec <= HC:CRLF
                            results.append(result)CRLF
CRLF
            if len(results) == 0:CRLF
                cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH', 'SHAPE@'])CRLF
                for result in cursor:CRLF
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:CRLF
                        distcheck = math.hypot(shape.trueCentroid.X - result[0][0], shape.trueCentroid.Y - result[0][1])CRLF
                        if math.fabs(distcheck) <= WithinDist:CRLF
                            xPointb = (result[2].firstPoint.X - shape.trueCentroid.X)CRLF
                            yPointb = (result[2].firstPoint.Y - shape.trueCentroid.Y)CRLF
                            degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactorCRLF
                            degreeb = 180 + degreeCorrbCRLF
                            degreec = degreeb + 180CRLF
                            if degreec >= 360:CRLF
                                degreec -= 360CRLF
CRLF
                            LC = degreeBearing - sfactor2CRLF
                            HC = degreeBearing + sfactor2CRLF
                            if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):CRLF
                                degreeb -= 360CRLF
                                degreec -= 360CRLF
                            if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):CRLF
                                degreeb += 360CRLF
                                degreec += 360CRLF
                            CRLF
                            if LC <= degreeb <= HC or LC <= degreec <= HC:CRLF
                                results.append(result)CRLF
CRLF
CRLF
        else:CRLF
            for result in items:CRLF
                results.append(result)CRLF
CRLF
        # Checks if the script has found the correct number of accessory objects, so that the results field can tell the user if the FoliationModule1CRLF
        # was able to compute the correct azimuth or not for the current record being interated over.CRLF
```

```
········if·len(results)·==·sfactor5:CRLF
CRLF
············#·If·the·symbol·represents·a·horizontal·measurement·(See·FoliationModule1)·the·script·returns·the·string·"Horizontal".CRLF
············for·item·in·results:CRLF
················if·horzlen1·<=·item[1]·<=·horzlen2:CRLF
····················degreeBearing·-=·90CRLF
····················return·"Horizontal"CRLF
CRLF
············#·Otherwise·I·return·the·string·"sucess"·to·indicate·that·the·code·found·the·correct·number·of·objects·and·will·compute·the·correctCRLF
············#·dip·direction.CRLF
············else:CRLF
················return·"Success"CRLF
········#·If·the·script·was·not·able·to·find·the·correct·number·of·accessory·objects·it·gives·the·number·of·objects·it·has·found,·in·order·toCRLF
········#·alert·the·user·that·the·script·could·not·determine·the·correct·angle·for·the·record·and·that·the·user·must·determine·the·correctCRLF
········#·azimuth·manually·after·the·script·has·run.·CRLF
········else:CRLF
············return·str('selected·'·+·str(len(results)))CRLF
CRLF
····else:CRLF
········return·str(4000)
```

FoliationModule2

```python
#  FoliationModule2
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for single-object datasets

def Strike(shape, fid): # The function is defined and the parameter set as the "shape" and "fid" of the object being
# iterated over so that the script can call upon this information of each object that passes through this fucntion by way
# of the CalculateField function within the lead script: Working Code Foliations.

    # Calculate the length of the object being iterated over by CalculateField
    length = shape.length

    # This checks that the program correctly calculated the length. Otherwise the script returns an error code.
    if length == 0:
        return 7000

    # The following if statement is to differentiate the strike and dip symbols based on its length and must thus use length limits
    # that include the strike length and exclude all other shapes in the shapefile. This ensures that the azimuth is only calculated
    # on strike symbols, with all other objects returning the error code 4000 by the else clause at the bottom of the script.
    if length >= LenMin and length <= LenMax:
        # The script calculates the strike azimuth based on Trig and what the program has recorded as the last point and the point at
        # 95% of the length of the line of each polyline to be. However, it takea the 95% point and subtract the last point, instead of
        # the other way around because this calculates the back azimuth so it can simplily add 180 degrees to the calculated angle instead
        # of having to change all angles between -180 to 0 degrees to their corresponding angles between 180 to 360 degrees.
        xcomponent = (shape.positionAlongLine(0.95,True).centroid.X - shape.lastPoint.X)
        ycomponent = (shape.positionAlongLine(0.95,True).centroid.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        # In addition to the two points used above for calculating components for calculating an angle, the last point and the point
        # at 75% of the length of the line are used to calculate another angle. The angles should be the same in order to indicate
        # that all three points are along a straight line representing the strike and that none of the points are on the part of the
        # line representing the dip portion of the symbol.
        xcheckcomponent = (shape.positionAlongLine(0.75,True).centroid.X - shape.lastPoint.X)
        ycheckcomponent = (shape.positionAlongLine(0.75,True).centroid.Y - shape.lastPoint.Y)
        # The atan functions are used to calculate the two angles, degree, and Checkdegree using each pair of x, y components, and are
        # reversed so as to calculate geometric angles with 0 degrees pointing north and the angles increasing clockwise, instead of trigonometric angles.
        degree = math.atan2(xcomponent, ycomponent) * conversionFactor
        Checkdegree = math.atan2(xcheckcomponent, ycheckcomponent) * conversionFactor
        # The script checks that the two angles are essentially equal, within the sfactor1 buffer.
        if degree - sfactor1 < Checkdegree < degree + sfactor1:
            # If equal, the angle is a backwards azimuth between -180 to 180 degrees. So the script adds 180 to that angle and get the positive 
            # azimuth between 0-360 degrees. The script then adds an addtional 90 degrees to it to calcualte the angle of the right perpendicular to the azimuth.
            # In other words the dip direciton is calculated from the strike direction.
```

```
44              degreeBearing = degree + 180 + 90
45              # Here the script fixes the angle if it is greater than 360 degrees
46              if degreeBearing >= 360:
47                  degreeBearing -= 360
48              # Then the script creates an array of coordinates defining two points so that a line object can be created later on in the script.
49              PforGeometry = arcpy.Array([arcpy.Point(shape.positionAlongLine(0.75,True).centroid.X, shape.positionAlongLine(0.75,True).centroid.Y), arcpy.Point(shape.positionAlongLine(0.95,True).centroid.X, shape.positionAlongLine(0.95,True).centroid.Y)])
50          # If the two angles are not equal, the above procedure of calculating two angles is repeated, this time using three points at the start of the polyline:
51          # the starting point, the point at 5% the line's length, and the point at 25% of the line's length.
52          else:
53              xcomponent = (shape.positionAlongLine(0.05,True).centroid.X - shape.firstPoint.X)
54              ycomponent = (shape.positionAlongLine(0.05,True).centroid.Y - shape.firstPoint.Y)
55              xcheckcomponent = (shape.positionAlongLine(0.25,True).centroid.X - shape.firstPoint.X)
56              ycheckcomponent = (shape.positionAlongLine(0.25,True).centroid.Y - shape.firstPoint.Y)
57              degree = math.atan2(xcomponent, ycomponent) * conversionFactor
58              Checkdegree = math.atan2(xcheckcomponent, ycheckcomponent) * conversionFactor
59          # The angles are again checked to make sure they are equal.
60              if degree - sfactor1 < Checkdegree < degree + sfactor1:
61                  degreeBearing = degree + 180 + 90
62                  if degreeBearing >= 360:
63                      degreeBearing -= 360
64                  # Here the direction of the PforGeometry points is reversed compared to origional line direction to keep the line direction consistent with the
65                  # calculated azimuth. In reality, the direction would be roughly the same as that of the other instance on line 50, in the dataset this script was
66                  # constructed for, had the other instance worked.
67                  PforGeometry = arcpy.Array([arcpy.Point(shape.positionAlongLine(0.25,True).centroid.X, shape.positionAlongLine(0.25,True).centroid.Y), arcpy.Point(shape.positionAlongLine(0.05,True).centroid.X, shape.positionAlongLine(0.05,True).centroid.Y)])
68              # If neither set of angles are equal at this point, then the error code 50000 is returned.
69              else:
70                  return 50000
71
72          # A centroid[] list is created that contains the centroid coordinates of the object currently being iterated over by CalculateField in the Working Code
73          # Folaitions lead script. This centroid should be offset from the strike line in the direction of the dip, and thus will be used to determine the correct
74          # dip direction.
75          centroid = [shape.trueCentroid.X, shape.trueCentroid.Y]
76          # The centroid list is turned into a point object so it can be used in queryPointAndDistance().
77          ptg = arcpy.Point(*centroid)
78          # Next a line (polyline) is created from the array of two points (PforGeometry) created above. This line should reflect the strike line, and thus can be
79          # compared with the point object using queryPointAndDistance.
80          ShapeG = arcpy.Polyline(PforGeometry)
81
82          # Then queryPointAndDistance is used to test whether the ptg point object is to the right of the strike line object, ShapeG, with the test depending on
83          # the starting point and ending point of the strike line. Thus if the point is 
```

```
to·the·right·of·the·line,·then·the·calculated·angle·is·correct.CRLF
········(pointGeometry,·distance,·minDistance,·isCentroidRight)·=·ShapeG.queryPointAndDistance(·ptg,·False)CRLF
CRLF
········#·Here·it·is·tested·whether·the·centroid·is·to·the·left·of·the·line.·If·so,·180·degrees·is·added·to·the·angle·to·correct·it.CRLF
········if·not·isCentroidRight:CRLF
············degreeBearing·+=·180CRLF
········#·This·if·statement·makes·sure·that·the·degreeBearing·value·is·between·0·and·360·degrees.·CRLF
········if·degreeBearing·>=·360:CRLF
············degreeBearing·-=·360CRLF
·······#returns·the·degreebearing·value·as·the·dip·direction.CRLF
········return·degreeBearingCRLF
CRLF
····#·For·the·objects·CalculateField·iterates·over·that·are·not·of·the·specified·length·to·be·foliation·symbols,·the·code·4000·is·applied·instead·of·a·dipCRLF
····#·direction·azimuth·value·for·identification·purposes.CRLF
····else:CRLF
········return·4000
```

FoliationNotes2

```python
#  FoliationNotes2
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for single-object datasets

def Strike(shape, fid): # See FoliationModule2 for notation explaining this fuction.
    length = shape.length

    if length == 0:
        return 7000


    if length >= LenMin and length <= LenMax:

        distcheck1 = math.hypot(shape.positionAlongLine(0.50,True).centroid.X - shape.firstPoint.X, shape.positionAlongLine(0.50,True).centroid.Y - shape.firstPoint.Y)
        distcheck2 = math.hypot(shape.positionAlongLine(0.95,True).centroid.X - shape.lastPoint.X, shape.positionAlongLine(0.95,True).centroid.Y - shape.lastPoint.Y)

        xcomponent = (shape.positionAlongLine(0.95,True).centroid.X - shape.lastPoint.X)
        ycomponent = (shape.positionAlongLine(0.95,True).centroid.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        xcheckcomponent = (shape.positionAlongLine(0.75,True).centroid.X - shape.lastPoint.X)
        ycheckcomponent = (shape.positionAlongLine(0.75,True).centroid.Y - shape.lastPoint.Y)
        degree = math.atan2(xcomponent, ycomponent) * conversionFactor
        Checkdegree = math.atan2(xcheckcomponent, ycheckcomponent) * conversionFactor
        if degree - sfactor1 < Checkdegree < degree + sfactor1:
            degreeBearing = degree + 180 + 90
            if degreeBearing >= 360:
                degreeBearing -= 360
            PforGeometry = arcpy.Array([arcpy.Point(shape.positionAlongLine(0.75,True).centroid.X, shape.positionAlongLine(0.75,True).centroid.Y), arcpy.Point(shape.positionAlongLine(0.95,True).centroid.X, shape.positionAlongLine(0.95,True).centroid.Y)])
        else:
            xcomponent = (shape.positionAlongLine(0.05,True).centroid.X - shape.firstPoint.X)
            ycomponent = (shape.positionAlongLine(0.05,True).centroid.Y - shape.firstPoint.Y)
            xcheckcomponent = (shape.positionAlongLine(0.25,True).centroid.X - shape.firstPoint.X)
            ycheckcomponent = (shape.positionAlongLine(0.25,True).centroid.Y - shape.firstPoint.Y)
            degree = math.atan2(xcomponent, ycomponent) * conversionFactor
            Checkdegree = math.atan2(xcheckcomponent, ycheckcomponent) * conversionFactor
            if degree - sfactor1 < Checkdegree < degree + sfactor1:
                degreeBearing = degree + 180 + 90
                if degreeBearing >= 360:
                    degreeBearing -= 360
                PforGeometry = arcpy.Array([arcpy.Point(shape.positionAlongLine(0.25,True).centroid.X, shape.positionAlongLine(0.25,True).centroid.Y), arcpy.Point(shape.positionAlongLine(0.05,True).centroid.X, shape.positionAlongLine(0.05,True).centroid.Y)])
            else:
                return 50000

        centroid = [shape.trueCentroid.X, shape.trueCentroid.Y]
        ShapeG = arcpy.Polyline(PforGeometry)
        ptg = arcpy.Point(*centroid)

        (pointGeometry, distance, minDistance, isCentroidRight) = ShapeG.queryPointAndDistance( ptg, False)

        if isCentroidRight:
            return "yes"
        else:
            return "no"

########### Unused code
        midwayXPoint = shape.firstPoint.X + (shape.positionAlongLine(100).firstPoint.X - shape.firstPoint.X)/2
        midwayYPoint = shape.firstPoint.Y + (shape.lastPoint.Y - shape.firstPoint.Y)/2
        if midwayXPoint <= shape.trueCentroid.X:
            isCentroidRight = True

        
        #I create my centroid object which I will then populate with the TrueCentroid of the single item in the results

        # The follwoing if statement checks if the dip symbol is to the left of the strick symbol usinge the direction of the strike symbol from the 
        # beginnign point to the end point as up. If it is to the left, then 180 degrees are added to the dip  direction (named degreeBearing at the moment)
    else:
        return 4000
```

Working Code Lineations

```
1  #  Working Code Lineations
2  #  Alexander Audet  - University of Maine
3  #  05/30/2018
4  
5  #See Working Code Foliations for more complete notations on the functioning of the lead scripts.
6  
7  # Import system modules
8  import arcpy
9  
10 
11 ######## User Inputs ########
12 
13 # Select the inTable (the shapefile full of data you are currently working on)
14 #Tables: La, LLo, Lineations, LaLo, Lp, Lp19991, Lr 19991, Lr, LrLo, Murray Lp, Murray extra Lp, G lineations
15 inTable = "Lineations"
16 
17 # Variable to set for your current job.
18 sfactor1 = 1 #This value should not change as it puts a 1 unit buffer on the shape.length caluclated below when comparing it to nearby object lengths. 
19 sfactor5 = 2
20 WithinDist = 14
21 LenMin = 180
22 LenMax = 190
23 Symbology = False #If Symbology is set to true, parameters on lines 75-84 below should be filled in.
24 
25 # list of modules: LineModule1.py, LineModule2.py
26 # Pick the appropriate module for the intable set above and add it to the end of the file address below. File address should point to where the modules have been stored.
27 f = open("D:/AlexandersWork/PythonModules/LineModule2.py",'r')
28 
29 # list of modules: LineNotes1.py, LineNotes2.py
30 # Pick the appropriate module for the intable set above and add it to the end of the file address below. File address should point to where the modules have been stored.
31 f2 = open("D:/AlexandersWork/PythonModules/LineNotes2.py",'r')
32 
33 ######## User Inputs ########
34 
35 
36 ##### Trend Field
37 
38 fieldName = "Trend"
39 expression = "Strike( !Shape!, !FID! )"
40 codeblock = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'sfactor5 =' +str(sfactor5) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax =' +str(LenMax) +'\r'+'\n'+ f.read()
41 
42 ##### Result Field
43 
44 fieldName2 = "Result"
45 expression2 = "Strike( !Shape!, !FID! )"
46 codeblock2 = 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'sfactor5 =' +str(sfactor5) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax =' +str(LenMax) +'\r'+'\n'+ f2.read()
47 
48 # Adds the Trend field to the table that will be populated by the trend azimuth.
49 try:
50     arcpy.AddField_management(inTable, fieldName, "SHORT")
51 except:
52     pass
53 
54 # Adds the Result field to the table that will be populated by the ability for the code to properly analyze the data.
```

```python
# Error and instances that need to be manually entered will be noted here.
try:
    arcpy.AddField_management(inTable, fieldName2, "TEXT")
except:
    pass

fieldName3 = "Type"

try:
    arcpy.AddField_management(inTable, fieldName3, "TEXT")
except:
    pass

# Calcualtes the values for the Trend field just created using the expression and codeblock defined above.
arcpy.CalculateField_management(inTable, fieldName, expression, "PYTHON_9.3", codeblock)

# Calcualtes the values for the Result field just created using the expression and codeblock defined above.
arcpy.CalculateField_management(inTable, fieldName2, expression2, "PYTHON_9.3", codeblock2)

##### Symbol Type
if Symbology:
    # The following module retuns the geological measurement type based on the shape (determined here by length) of the accessory symbol.
    typeA = ""
    typeA1 = 0
    typeA2 = 0
    typeB = ""
    typeB1 = 0
    typeB2 = 0
    typeC = ""
    typeC1 = 0
    typeC2 = 0
    f3 = open("D:/AlexandersWork/PythonModules/SymbologyModule.py",'r')
    expression3 = "Strike( !Shape!, !FID! )"
    codeblock3 = 'typeA = "' +typeA +'"'+'\r'+'\n'+ 'typeB = "' +typeB +'"'+'\r'+'\n'+ 'typeC = "' +typeC +'"'+'\r'+'\n'+ 'inTable = "' +inTable +'"'+'\r'+'\n'+ 'sfactor1 =' +str(sfactor1) +'\r'+'\n'+ 'sfactor2 = ' +str(sfactor2) +'\r'+'\n'+ 'sfactor3 =' +str(sfactor3) +'\r'+'\n'+ 'sfactor4 =' +str(sfactor4) +'\r'+'\n'+ 'sfactor5 =' +str(sfactor5) +'\r'+'\n'+ 'WithinDist =' +str(WithinDist) +'\r'+'\n'+ 'LenMin =' +str(LenMin) +'\r'+'\n'+ 'LenMax =' +str(LenMax) +'\r'+'\n'+ 'typeA1 =' +str(typeA1) +'\r'+'\n'+ 'typeA2 =' +str(typeA2) +'\r'+'\n'+ 'typeB1 =' +str(typeB1) +'\r'+'\n'+ 'typeB2 =' +str(typeB2) +'\r'+'\n'+ 'typeC2 =' +str(typeC2) +'\r'+'\n'+ 'typeC1 =' +str(typeC1) +'\r'+'\n'+ f3.read()

    # Calcualtes the values for the Type field created using the appropriate expression and codeblock defined above
    arcpy.CalculateField_management(inTable, fieldName3, expression3, "PYTHON_9.3", codeblock3)
```

LineModule1

```
#  LineModule1
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for multi-object datasets

def Strike(shape, fid): # The function is defined and the parameter set as the "shape" and "fid" of the object being
# iterated over so that the script can call upon this information of each object that passes through this fucntion by way
# of the CalculateField function within the lead script: Working Code Lineations.

    #Calcualte the length of the object being iterated over by CalculateField
    length = shape.length

    # This checks that the program correctly calculated the length. Otherwise the script returns an error code.
    if length == 0:
        return 7000

    # The following if statement is to differentiate the primary object based on its length and must thus use length
    # limits that include the primary object length and exclude all other shapes in the shapefile including accessory
    # objects. This ensures that the azimuth is only calculated on primary objects, with all other objects returning the
    # error code 4000 by the else clause at the bottom of the script.
    if length >= LenMin and length <= LenMax:

        # The script calculates the azimuth based on Trig and what the program has recorded as the first and last point of each
        # polyline to be. However, it takes the first point and subtract the last point, instead of the other way around because
        # this calculates the back azimuth, so it can simplily add 180 degrees to the calculated angle instead of having to
        # change all angles between -180 to 0 degrees to their corresponding angles between 180 to 360 degrees.
        xPoint = (shape.firstPoint.X - shape.lastPoint.X)
        yPoint = (shape.firstPoint.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        # The X, and Y term are switched within the atan2() fuction so as to calulate the angle from North, or the positive X-axis
        # instead of from the trigonometric x-axis.
        degreeCorrection = math.atan2(xPoint, yPoint) * conversionFactor
        # When finnished, the angle is a backwards azimuth between -180 to 180 degrees. So the script adds 180 to that angle and get the positive 
        # azimuth between 0-360 degrees.
        degreeBearing = degreeCorrection + 180

        # The PairedCheck8 feature layer out of my current intable set within the lead script, but excluding the current object being
        # calculated by using the where statement to select only objects that do not match the current FID. This layer will be used 
        # to select the matching accessory objects that belongs with the primary object for which the azimuth is
        # being calculated.
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))

        # Next the script creates a curser object which it uses to find the TrueCentroid and Length of each of the features in PairedCheck8.
        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])


        # Create results, an empty list to be filled by only the cursor objects, composed of centroid/length pairs, within a certian set distance
        # (WithinDist) from and not of the same length (+/- sfactor1) of the object the CalculateField function in the lead script: Working Code
        # Lineations is currently iterating over.
        results = []
        for item in cursor:
            # As mentioned above, this excludes objects of the same length as the objects being iterated over since its matching accessory
            # object will be a different length. Note that since the cursor calculates a different geometry length from the process at the
            # begining of this code, I put a 1 unit buffer on the interated object length for which the cursor objects cannot overlap to be
            # added to the results[] list.
            if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
                distcheck1 = math.hypot(shape.firstPoint.X - item[0][0], shape.firstPoint.Y - item[0][1])
                distcheck2 = math.hypot(shape.lastPoint.X - item[0][0], shape.lastPoint.Y - item[0][1])
                # This part of the code, as mentioned above, checks each object in turn that passed the length test to see if the distances between
                # each object in the cursor and one or other other end points of the object currently being interated over by the CalculateField
                # function in the lead script, distcheck1 or distcheck2 are smaller than the set WithinDist parameter, and only adds the object to
                # results[] if one of these distcheck distances is smaller. Thus, accessory objects farther than WithinDist from both ends of the
                # primary objects are excuded from the list of potential accessory objects. 
                if distcheck1 <= WithinDist or distcheck2 <= WithinDist:
                    results.append(item)

        # I create my centroid object which I will then populate with the TrueCentroid of one of the items in the results
        # if there is the correct number of matching objects, set as sfactor5, in the results
        centroid = None
        # Checks if there is sfactor5 numbers of items in the results:
        if len(results) == sfactor5:

            # Sets the first result item's centroid coordinates as the centroid object
            centroid = results[0][0]

            # These next set distance checks, distcheck1 and distcheck2 below, determine which end of the primary object being iterated over its accessory
            # objects are at by comparing which end is closer to the accessory. If the accessory object is closer to the last point of the primary object,
            # then the trend azimuth (degreeBearing) must be reversed by adding 180 degrees to it. This is because in the datasets this script was developed
            # for, the secondary objects rested at the origin of the lineation symbol, with the primary object pointing out from that secondary object in the
            # direciton of the trend, and the trigonometric caclulations above assumed that the secondary object was on the starting point. 
            distcheck1 = math.hypot(shape.firstPoint.X - centroid[0], shape.firstPoint.Y - centroid[1])
            distcheck2 = math.hypot(shape.lastPoint.X - centroid[0], shape.lastPoint.Y - centroid[1])
            if distcheck1 > distcheck2:
                degreeBearing += 180
        # If the accessory object's centorid is closer to the firstpoint, the azimuth is returned unmodified.
        else:
            return degreeBearing

        # This if statement makes sure that the degreeBearing value is between 0 and 360 degrees.
        if degreeBearing >= 360:
```

```
············degreeBearing·-=·360CRLF
CRLF
·······#returns·the·degreebearing·value·as·the·trendCRLF
········return·degreeBearingCRLF
····#·For·the·objects·CalculateField·iterates·over·that·are·not·of·the·specified·length·
    to·be·lineation·symbols·(including·the·accessory·objects·that·are·associatedCRLF
····#·with·the·primary·object·in·order·to·show·trend·and·plunge·direction)·the·code·
    4000·is·applied·instead·of·a·trend·azimuth·value·for·identification·purposes.CRLF
····else:CRLF
········return·4000
```

LineNotes1

```
1   #  LineNotes1
2   #  Alexander Audet  - University of Maine
3   #  05/30/2018
4   
5   # This module is for multi-object datasets
6   
7   def Strike(shape, fid): # See LineModule1 for notation explaining this fuction. Only 
    info unique to the Notes
8   # Module will be included here.
9   
10      length = shape.length
11  
12      if length == 0:
13          return 7000
14  
15  
16      if length >= LenMin and length <= LenMax:
17  
18  
19          xPoint = (shape.firstPoint.X - shape.lastPoint.X)
20          yPoint = (shape.firstPoint.Y - shape.lastPoint.Y)
21          conversionFactor = ( 180 / math.pi )
22  
23          degreeCorrection = math.atan2(xPoint, yPoint) * conversionFactor
24  
25          degreeBearing = degreeCorrection + 180
26  
27  
28          arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" 
            <>'+str(fid))
29  
30          cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 
            'SHAPE@LENGTH'])
31  
32  
33          results = []
34          for item in cursor:
35              if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
36                  distcheck1 = math.hypot(shape.firstPoint.X - item[0][0], 
                    shape.firstPoint.Y - item[0][1])
37                  distcheck2 = math.hypot(shape.lastPoint.X - item[0][0], 
                    shape.lastPoint.Y - item[0][1])
38                  if distcheck1 <= WithinDist or distcheck2 <= WithinDist:
39                      results.append(item)
40  
41          centroid = None
42  
43          if len(results) == sfactor5:
44              # Returns "Success" If the correct number of accessory objects are found, 
                and thus the script
45              # should have been able to correctly calculate the trend azimuth in 
                LineModule1.
46              return "Success"
47              # If the above line is commented out, the below code, taken from the 
                LineModule1 can be used to
48              # further analize the dataset by determining which whether the accessory 
                object is closer to
49              # the first or last point for each primary object. (Lines 50-58)
50              centroid = results[0][0]
51  
52              distcheck1 = math.hypot(shape.firstPoint.X - centroid[0], 
                shape.firstPoint.Y - centroid[1])
53              distcheck2 = math.hypot(shape.lastPoint.X - centroid[0], shape.lastPoint.Y 
                - centroid[1])
54              if distcheck1 > distcheck2:
55                  degreeBearing += 180
56                  return "last"
```

```

                if LC <= degreeb <= HC or LC <= degreec <= HC:
                    results.append(result)

            if len(results) == 0:
                for result in items:
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:
                        xPointb = (result[0][0] - shape.trueCentroid.X)
                        yPointb = (result[0][1] - shape.trueCentroid.Y)
                        degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
                        degreeb = 180 + degreeCorrb
                        degreec = degreeb + 180
                        if degreec >= 360:
                            degreec -= 360

                        LC = degreeBearing + sfactor3
                        HC = degreeBearing + sfactor4
                        if degreeBearing <= sfactor3 and (degreeb >= 360 - sfactor3 or degreec >= 360 - sfactor3):
                            degreeb -= 360
                            degreec -= 360

                        if LC <= degreeb <= HC or LC <= degreec <= HC:
                            results.append(result)

            if len(results) == 0:
                cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH', 'SHAPE@'])
                for result in cursor:
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:
                        distcheck = math.hypot(shape.trueCentroid.X - result[0][0], shape.trueCentroid.Y - result[0][1])
                        if math.fabs(distcheck) <= WithinDist:
                            xPointb = (result[2].firstPoint.X - shape.trueCentroid.X)
                            yPointb = (result[2].firstPoint.Y - shape.trueCentroid.Y)
                            degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
                            degreeb = 180 + degreeCorrb
                            degreec = degreeb + 180
                            if degreec >= 360:
                                degreec -= 360

                            LC = degreeBearing - sfactor2
                            HC = degreeBearing + sfactor2
                            if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):
                                degreeb -= 360
                                degreec -= 360
                            if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):
                                degreeb += 360
                                degreec += 360
                            
                            if LC <= degreeb <= HC or LC <= degreec <= HC:
                                results.append(result)


        else:
            for result in items:
                results.append(result)

        # Checks if the script has found the correct number of accessory objects, so that the results field can tell the user if the FoliationModule1
        # was able to compute the correct azimuth or not for the current record being interated over.
```

LineModule2

```python
#  LineModule2
#  Alexander Audet  - University of Maine
#  05/30/2018

# This module is for single-object datasets

def Strike(shape, fid): # The function is defined and the parameter set as the "shape" and "fid" of the object being
# iterated over so that the script can call upon this information of each object that passes through this fucntion by way
# of the CalculateField function within the lead script: Working Code Lineations.

    #Calcualte the length of the object being iterated over by CalculateField
    length = shape.length

    # This checks that the program correctly calculated the length. Otherwise the script returns an error code.
    if length == 0:
        return 7000

    # The following if statement is to differentiate the primary object based on its length and must thus use length
    # limits that include the primary object length and exclude all other shapes in the shapefile including accessory
    # objects. This ensures that the azimuth is only calculated on primary objects, with all other objects returning the
    # error code 4000 by the else clause at the bottom of the script.
    if length >= LenMin and length <= LenMax:

        # The script calculates the azimuth based on Trig and what the program has recorded as the first and last point of each
        # polyline to be. However, it takes the first point and subtract the last point, instead of the other way around because
        # this calculates the back azimuth, so it can simplily add 180 degrees to the calculated angle instead of having to
        # change all angles between -180 to 0 degrees to their corresponding angles between 180 to 360 degrees.
        xcomponent = (shape.positionAlongLine(0.27,True).centroid.X - shape.positionAlongLine(0.25,True).centroid.X)
        ycomponent = (shape.positionAlongLine(0.27,True).centroid.Y - shape.positionAlongLine(0.25,True).centroid.Y)
        conversionFactor = ( 180 / math.pi )
        # The X, and Y term are switched within the atan2() fuction so as to calulate the angle from North, or the positive X-axis
        # instead of from the trigonometric x-axis.
        degree = math.atan2(xcomponent, ycomponent) * conversionFactor
        # When finnished, the angle is a backwards azimuth between -180 to 180 degrees. So the script adds 180 to that angle and get the positive 
        # azimuth between 0-360 degrees.
        degreeBearing = degree + 180
        return degreeBearing
        # This code was enough to correctly identify the direction of trend in the datasets this script was developed for because in those
		# datasets the trend always pointed in the direction from the last point to the first point. This makes the code less applicable to
		# new datasets, so more logic like that found in FoliationModule2 should be implimented here in the future in order to expand this
		# script's fucntionality.

    # For the objects CalculateField iterates over that are not of the specified length to be lineation symbols the code 4000 is applied
    # instead of a trend azimuth value for identification purposes.
    else:
        return 4000
```

```
#··LineNotes2CRLF
#··Alexander·Audet··-·University·of·MaineCRLF
#··05/30/2018CRLF
CRLF
#·This·module·is·for·multi-object·datasetsCRLF
CRLF
def·Strike(shape,·fid):·#·This·script·onlyCRLF
#·distinguishes·measurements·from·non-measurementsCRLF
#·in·the·dataset·using·each·object's·length·andCRLF
#·the·set·length·limits.·See·LineModule2·for·furtherCRLF
#·notation.CRLF
CRLF
····length·=·shape.lengthCRLF
CRLF
····if·length·==·0:CRLF
········return·str(7000)CRLF
CRLF
····if·length·>=·LenMin·and·length·<=·LenMax:CRLF
CRLF
········return·"Success"CRLF
CRLF
····else:CRLF
········return·str(4000)
```

SymbologyModule

```python
#  SymbologyModule
#  Alexander Audet  - University of Maine
#  05/30/2018

def Strike(shape, fid): # See FoliationModule1 for notation explaining this fuction. Only info unique to this Module will be included here.

    length = shape.length

    if length == 0:
        return 7000

    if length >= LenMin and length <= LenMax:

        xPoint = (shape.firstPoint.X - shape.lastPoint.X)
        yPoint = (shape.firstPoint.Y - shape.lastPoint.Y)
        conversionFactor = ( 180 / math.pi )
        degreeCorrection = math.atan2(xPoint, yPoint) * conversionFactor

        degreeBearing = degreeCorrection + 180 + 90

        if degreeBearing >= 360:
            degreeBearing -= 360

        arcpy.MakeFeatureLayer_management(inTable, 'shapelyr', '"FID" ='+str(fid))
        arcpy.MakeFeatureLayer_management(inTable, 'PairedCheck8', '"FID" <>'+str(fid))

        cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH'])

        items = []
        for item in cursor:

            if item[1] <= length - sfactor1 or item[1] >= length + sfactor1:
                distcheck = math.hypot(shape.trueCentroid.X - item[0][0], shape.trueCentroid.Y - item[0][1])
                if math.fabs(distcheck) <= WithinDist:
                    items.append(item)

        results = []

        if len(items) <> sfactor5:

            for result in items:

                xPointb = (result[0][0] - shape.trueCentroid.X)
                yPointb = (result[0][1] - shape.trueCentroid.Y)
                degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor

                degreeb = degreeCorrb + 180
                degreec = degreeb + 180
                if degreec >= 360:
                    degreec -= 360

                LC = degreeBearing - sfactor2
                HC = degreeBearing + sfactor2
                if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):
                    degreeb -= 360
                    degreec -= 360
                if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):
                    degreeb += 360
                    degreec += 360

                if LC <= degreeb <= HC or LC <= degreec <= HC:
                    results.append(result)

            if len(results) == 0:
                for result in items:
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:
                        xPointb = (result[0][0] - shape.trueCentroid.X)
                        yPointb = (result[0][1] - shape.trueCentroid.Y)
                        degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
                        degreeb = 180 + degreeCorrb
                        degreec = degreeb + 180
                        if degreec >= 360:
                            degreec -= 360

                        LC = degreeBearing - sfactor3
                        HC = degreeBearing - sfactor4
                        if degreeBearing <= sfactor3 and (degreeb >= 360 - sfactor3 or degreec >= 360 - sfactor3):
                            degreeb -= 360
                            degreec -= 360

                        if LC <= degreeb <= HC or LC <= degreec <= HC:
                            results.append(result)

            if len(results) == 0:
                cursor = arcpy.da.SearchCursor('PairedCheck8',['SHAPE@TRUECENTROID', 'SHAPE@LENGTH', 'SHAPE@'])
                for result in cursor:
                    if result[1] <= length - sfactor1 or result[1] >= length + sfactor1:
                        distcheck = math.hypot(shape.trueCentroid.X - result[0][0], shape.trueCentroid.Y - result[0][1])
                        if math.fabs(distcheck) <= WithinDist:
                            xPointb = (result[2].firstPoint.X - shape.trueCentroid.X)
                            yPointb = (result[2].firstPoint.Y - shape.trueCentroid.Y)
                            degreeCorrb = math.atan2(xPointb, yPointb) * conversionFactor
                            degreeb = 180 + degreeCorrb
                            degreec = degreeb + 180
                            if degreec >= 360:
                                degreec -= 360

                            LC = degreeBearing - sfactor2
                            HC = degreeBearing + sfactor2
                            if degreeBearing <= sfactor2 and (degreeb >= 360 - sfactor2 or degreec >= 360 - sfactor2):
                                degreeb -= 360
                                degreec -= 360
                            if degreeBearing >= 360 - sfactor2 and (degreeb <= sfactor2 or degreec <= sfactor2):
                                degreeb += 360
                                degreec += 360
                            
                            if LC <= degreeb <= HC or LC <= degreec <= HC:
                                results.append(result)

        else:
            for result in items:
                results.append(result)


        if len(results) == sfactor5:

            # If the paired object length is within a defined size then the script calls it a particular type of measurement and return that.
            if typeA1 <= results[0][1] <= typeA2:
                return typeA
```

```
············if·typeB1·<=·results[0][1]·<=·typeB2:CRLF
················return·typeBCRLF
············if·typeC1·<=·results[0][1]·<=·typeC2:CRLF
················return·typeCCRLF
············#·Otherwise·the·paired·object·is·not·of·a·recognized·length·and·will·need·
            to·be·manually·entered.CRLF
············else:CRLF
················return·"Unknown?"CRLF
CRLF
CRLF
········else:CRLF
············#·If·an·incorrect·number·of·objects·are·selected·the·type·will·need·to·be·
            manually·entered.CRLF
············return·"?"CRLF
····else:CRLF
········return·str(4000)
```

Registration Shift

```
#  Registration Shift
#  Alexander Audet - University of Maine
#  07/17/2018

# Import system modules
import arcpy, os
from arcpy import env

# Set workspace environments
env.workspace = "D:/AlexandersWork/AllData/NAD27"   # Location of input dataset.

out workspace = "D:/AlexandersWork/AllData/WGS84"   # Location output datasets will be placed.

# The script creates the file list from all the feature class files (effectively shapefiles)
# in the input workspace set above.
input features = arcpy.ListFeatureClasses()

# Sets the sr variable to the spatial reference object (pretty much a fancy definition) that
# matches the spatial reference the data should initially be in. Consult instructions for
# determining how to insert the correct coordinate system name for your data into the 
# SpatialReference function below.
sr = arcpy.SpatialReference("NAD 1927 UTM Zone 11N")
# Defines the projection of the files as that sr to make sure that the BatchProject function
# will know what the file's projections are if they have not already been properly defined
# for some reason. This does not change the dataset's coordinate system, it just simply gives
# it a name, but it is important that the name is correct.
for file in input_features:
    arcpy.DefineProjection management(file, sr)

# Set the template dataset for defining the output coordinate system - This one had the
# UTM Zone 11N, GCS WGS 1984 coordinate system for example, and thus sets that as the
# coordinate system BatchProject will reproject the input datasets into.
template = "D:/AlexandersWork/Proj Dat Definition.shp"

# Set the Geographic transformation - This needs to be the valid name of a predefined
# transformation between the initial datum and the output datum, so that BatchProject can
# correctly reproject the datasets. 
transformation = "WGS 1984 (ITRF00) To NAD 1983+NAD 1927 To NAD 1983 NADCON" #NAD_1927_To_WGS_1984_18" #NAD_1927_To_NAD_1983_NADCON NAD_1983_To_WGS_1984_1 WGS 1984 (ITRF00) To NAD 1983 HARN

# Finally, BatchProject is run, using the above inputs. 
try:
   res = arcpy.BatchProject management(input features, out workspace, Template_dataset=template, Transformation=transformation)
   if res.maxSeverity == 0:
     print "projection of all datasets successful"
   else:
     print "failed to project one or more datasets"
except:
   print res.getMessages()
```

AUTHOR'S BIOGRAPHY

Alexander C. Audet was born in Monrovia, Indiana on July 27 1995. Since then his family has moved to Maine where when he isn't hard at work studying, reading books, and playing the cello, he can usually be found paddling for miles and bagging peaks, or hanging out with the University of Maine Sailing Team on the coast. Additionally, he is part of the Mapping Club, Xi Sigma Pi, and Navigators groups at the University of Maine. He is currently graduating with an Earth Science concentration within the Earth and Climate Sciences major.

Upon graduation, he will immediately enter a Forestry Graduate Program, in the fall of 2018, where he will finish his forestry studies, and then search for additional Earth Science graduate work.