USGS
science for a changing world

Laharz_py: GIS Tools for Automated Mapping of Lahar Inundation Hazard Zones

By Steve P. Schilling

Open-File Report 2014–1073

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
SALLY JEWELL, Secretary

U.S. Geological Survey
Suzette M. Kimball, Acting Director

U.S. Geological Survey, Reston, Virginia: 2014

Suggested citation:
Schilling, S.P., 2014, Laharz_py—GIS tools for automated mapping of lahar inundation hazard zones: U.S. Geological Survey Open-File Report 2014-1073, 78 p., http://dx.doi.org/10.3133/ofr20141073.

ISSN 2331-1258 (online)

Contents

Figures

Conversion Factors

SI to Inch/Pound

Multiply	By	To obtain
	Length	
meter (m)	3.281	foot (ft)
meter (m)	1.094	yard (yd)
kilometer (km)	0.6214	mile (mi)
kilometer (km)	0.5400	mile, nautical (nmi)
kilometer per hour (km/h)	0.6214	mile per hour (mi/h)
	Volume	
cubic meter (m^3)	35.31	cubic foot (ft^3)
cubic meter (m^3)	264.2	gallon (gal)
cubic meter (m^3)	0.0002642	million gallons (Mgal)
cubic meter (m^3)	1.308	cubic yard (yd^3)
cubic meter (m^3)	0.0008107	acre-foot (acre-ft)

Laharz_py: GIS Tools for Automated Mapping of Lahar Inundation Hazard Zones

By Steve P. Schilling

Introduction

"Lahar" is an Indonesian term for large debris flows that originate on volcano flanks; contain primarily water, mud and rock debris; and have a density often compared to wet concrete. To form a lahar, there must be adequate water and abundant unconsolidated debris along with steep relief and some type of triggering mechanism (Vallance, 2000). Lahars surge down steep volcano flanks into and along river channels as gravity-driven flows that may reach speeds up to 65 km/h (Newhall and others, 1997) and can reach tens or even hundreds of kilometers downstream. As lahars flow along the upper reaches of the stream path, they often erode and scour loose volcanic debris and vegetation and may incorporate snow, ice, or streamflow, increasing their size by several times. As lahars continue downstream to lower elevations they lose coarser components, become dilute, and decrease in size.

Lahars can flow during eruption where water is released from the interaction of hot rock on snow and ice (Nevado del Ruiz, Colombia, 1985), at the onset of volcanic activity as groundwater is rapidly expelled ahead of a rising magma intrusion (Nevado del Huila, Colombia, 2007), after an eruption where heavy rain can remobilize fresh fragemental deposits (after the large eruption of Mount Pinatubo, Philippines, 1991), or without any eruption where torrential rains cause ground failure that generate lahars (Casita Volcano, Nicaragua, 1998). Landslides may transition into lahars (Mount St. Helens, Washington, 1980) if sufficient water is present in the source material. Lahars can be hundreds of thousands to hundreds of millions of cubic meters in volume. However, deposits left by past lahars demonstrate that high consequence, low probability events can reach billions of cubic meters in volume, such as the Osceola Mudflow at Mount Rainier, Washington, that filled valleys to 200 m and flowed 120 km downstream (Scott and others, 1995; Vallance, 2000) or the Chillos Valley Lahar at Cotopaxi Volcano, Ecuador (Mothes and others, 1998), that filled valleys to 180 m and flowed more than 300 km downstream.

Lahars are natural processes that become hazards when they affect people and property. These flows are a hazard at many of the world's volcanoes (fig. 1), and have claimed thousands of lives at Armero, Colombia, from eruption of Nevado del Ruiz in 1985, and the towns of El Povenir and Rolando Rodriguez from flank collapse at Casita Volcano, Nicaragua, in 1998. As lahars move downstream, the largest particles within the flow tend to migrate to the front and margins where direct impact can abrade or destroy anything in their path. As a lahar comes to rest, buildings and land may be buried by layers of rock debris and mud.

Figure 1. Photograph showing deposits left by a lahar at Mount St. Helens, Washington, 1982. The lahar (the dark deposit on the snow) flowed from the crater. Part of the lahar entered Spirit Lake (lower left corner), but most of it flowed west down the Toutle River, eventually reaching the Cowlitz River, 80 kilometers downstream. (Photograph taken by Tom Casadevall, U.S. Geological Survey, March 21, 1982).

Volcano hazard-zonation maps (fig. 2) depict estimates of potentially hazardous areas from lahars and other volcano processes that may help mitigate impact on society and infrastructure if used for development planning or event response. For volcanoes, the hazard-zonation map often includes a proximal zone encompassing most or all of the volcano edifice. This proximal zone is susceptible to a variety of processes such as pyroclastic flows, debris flows, lava flows, and ballistic projectiles. Distal hazard zones extend away from the proximal zone surrounding the volcano edifice, most often along stream channels, typically indicating areas of potential inundation and runout by future lahars.

To aid the creation of hazard-zonation maps, Iverson and others (1998) developed an empirical and statistically based forecasting method to predict inundation and runout of lahars that is rapid, reproducible, and objective. Their central tenets of the method are the same as those used by geologists using knowledge of past lahar deposits to constrain hazard zones:

> "(1) inundation by past lahars provides a basis for predicting inundation by future lahars; (2) distal lahar hazards are confined to valleys that head on volcano flanks; (3) lahar volume largely controls the extent of inundation downstream; (4) voluminous lahars occur less often than small lahars; and (5) no one can foretell the size of the next lahar to descend a given drainage" (Iverson and others, 1998).

Iverson and others (1998) assumed that far-reaching, distal lahars will originate at proximal sources and focused on sudden-onset lahars that typically evolve from rock and ice avalanches, pyroclastic flows, or lake-breakout floods that originate high on volcano flanks or at the summit. Although lahars evolve in size and composition as they move downstream, a key assumption of Iverson and others (1998) was that the maximum lahar discharge produces the maximum inundation of valley cross-sectional area, a quantity of primary interest for delineating hazard zones rather than the lahar deposit (fig. 3) that often has a much smaller cross-sectional area.

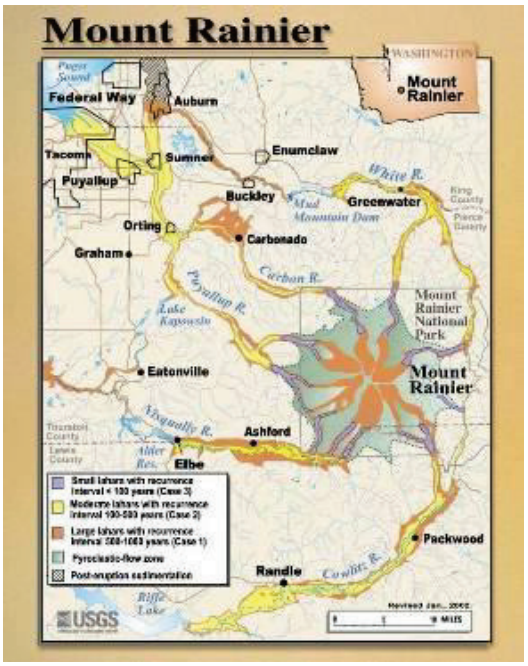

Figure 2. Volcano hazard map of Mount Rainier, Washington (Scott and others, 1995). Edifice is encircled by proximal hazard zone (green) with distal hazard zones along major drainages (yellow and red).

Figure 3. Photograph showing difference between maximum lahar discharge and lahar deposit. U.S. Geological Survey hydrologist (about 2 meters tall) stands on deposit and examines mudline left behind from the May 18, 1980 eruption of Mount St. Helens. (Photograph taken by Lyn Topinka, U.S. Geological Survey, 1980).

Iverson and others (1998) used scaling analysis to predict the mathematical form of equations that relate lahar volume (V) to a cross-sectional inundation area (A) (fig. 4a) and a planimetric inundation area (B) (fig. 4b).

$$A = 0.05\ V^{2/3} \tag{1}$$

$$B = 200\ V^{2/3} \tag{2}$$

where

A	cross section area;
B	planimetric area; and
V	volume.

The equations were derived statistically by using data from mapping of 27 lahar paths at nine volcanoes (Iverson and others, 1998).

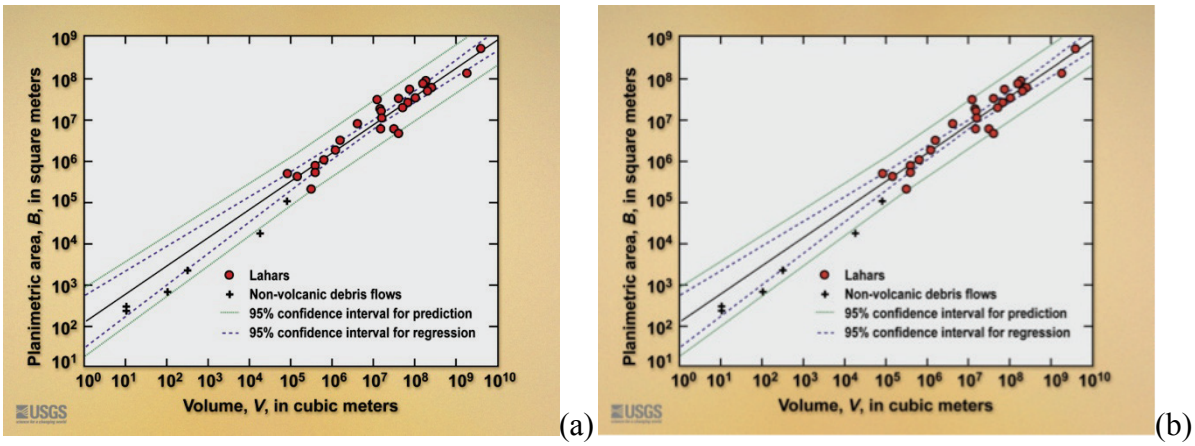

Figure 4. Scatter plots showing (a) inundated valley cross-sectional area, A, as a function of lahar volume V. The best-fit log-log regression line (solid) and 95-percent confidence intervals for regression (dashed lines) and prediction (green lines) also are shown. (b) inundated planimetric area B as a function of lahar volume V. The best-fit log-log regression line (solid) and 95-percent confidence intervals for regression (dashed lines) and prediction (green lines) also are shown.

Laharz Overview

Laharz (Schilling, 1998) is software designed to calculate a proximal hazard zone and automate equations (1) and (2) to run within a Geographic Information System (GIS) over three dimensional topography in order to estimate distal hazard zones. Laharz is written in the ArcInfo Macro Language (AML) that runs within the GRID portion of ArcInfo Workstation, and was designed to delimit areas of potential lahar inundation from one to four user-specified lahar volumes; producing one estimated lahar-inundation hazard zone for each volume over one or more stream drainages. Typically, the planimetric area of a lahar inundation hazard zone increases in width and length as lahar volume increases. Plotting hazard zones of smaller area over hazard zones having larger area (referred to as nesting of hazard zones) shows progressively larger areas of inundation from progressively larger volumes. These hazard zones can be displayed with other types of volcano hazard information in a GIS, such as a proximal hazard zone (fig. 5), infrastructure, hydrology, population, and contours or shaded relief map to produce volcano hazard-zonation maps. Such maps show proximity to and intersection of potential hazard zones with people and infrastructure.

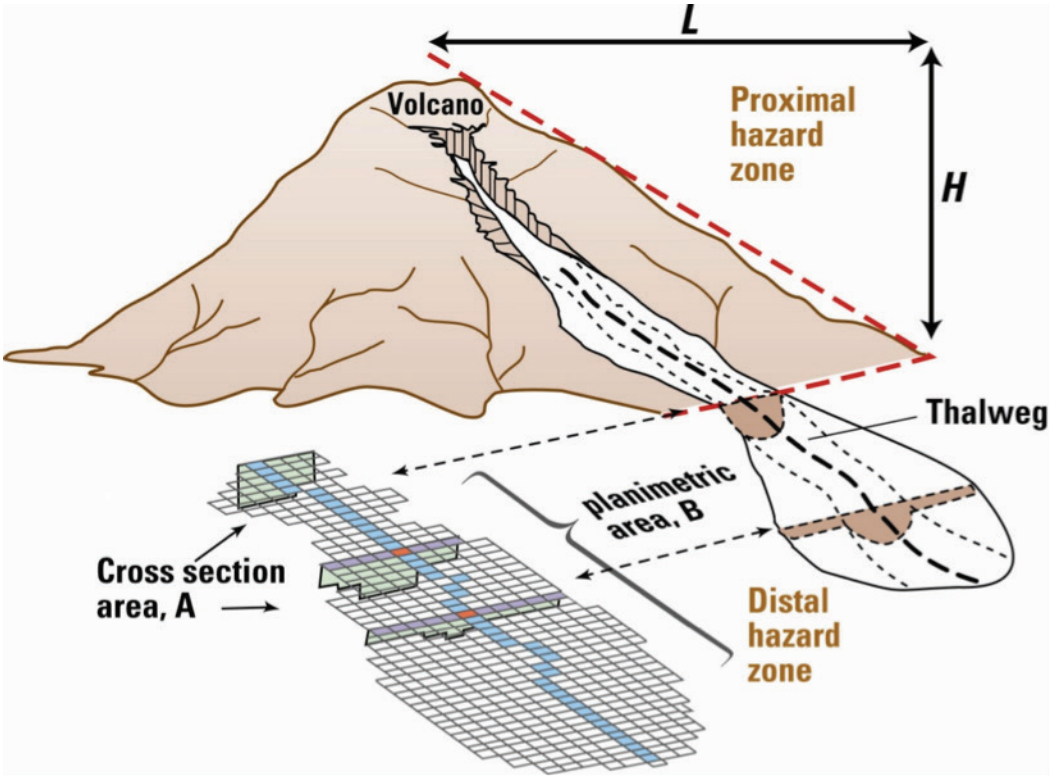

Figure 5. Diagram showing association between dimensions of an idealized lahar and cross-sectional (*A*) and planimetric (*B*) areas calculated by Laharz_py for a hypothetical volcano. The ratio of vertical drop (*H*) to horizontal runout distance (*L*) describes the extent of proximal volcano hazards. Laharz_py identifies cells where raster streams and the proximal-hazard zone boundary intersect as potential locations to begin calculations of distal lahar-inundation hazard zones. (Modifed from Iverson and others, 1998.)

Digital Topography

The most important dataset required to run Laharz is an accurate digital elevation model (DEM), a regular array of cells where each cell stores an elevation value and when combined compose three dimensional topography for a portion of the Earth. DEMs are available from many sources and vary in accuracy, resolution (size of cell), projection, and extent. Regardless of source, Laharz requires a DEM that has X-, Y-, and Z-coordinate values in units of meters. In addition, the extent of the DEM used to create lahar-inundation hazard zones should include the volcano edifice as well as the downstream extent of major stream drainages. Finally, even though DEMs vary in resolution, it is important to have a cell size that is appropriate for calculations. DEMs having cells with kilometer resolution will not be useful if a drainage of interest is 500 m wide. Generally, resolutions ranging between 1 and 30 m yield reasonable results for volcano topography. There is a tradeoff when considering resolution of a DEM; calculations for potential inundation along a river channel using larger sized cells accomplish the task quicker than when using smaller sized cells, whereas DEMs having smaller cells usually represent topography with greater precision.

Laharz documentation (Schilling, 1998) describes how the software can correct potential elevation errors (where flow direction is undefined) by use of a function to iterate across the DEM automatically, filling depressions that are shallower than a specified threshold value (Environmental Systems Research Institute, 1994). Often depressions in DEMs are erroneous elevations, referred to as "sinks," are often created during the generation of a DEM and inhibit surface-flow routing. However, some sinks may represent real surface depressions, such as quarries or natural erosional potholes (Jenson and Domingue, 1988). Thus, the user must be familiar with land features represented by the DEM to determine whether sinks in their DEM are errors. The resultant depressionless grid is then used to generate supplementary surface hydrology (flow routing) raster datasets.

These flow routing functions derive three raster datasets (fig. 6) from the DEM of flow direction, flow accumulation, and stream delineation that form the means to traverse a channel downstream and calculate areas of potential lahar inundation. Laharz begins calculations of lahar-inundation hazard zones at user specified X,Y coordinates, typically where a raster stream intersects the proximal-hazard zone boundary.

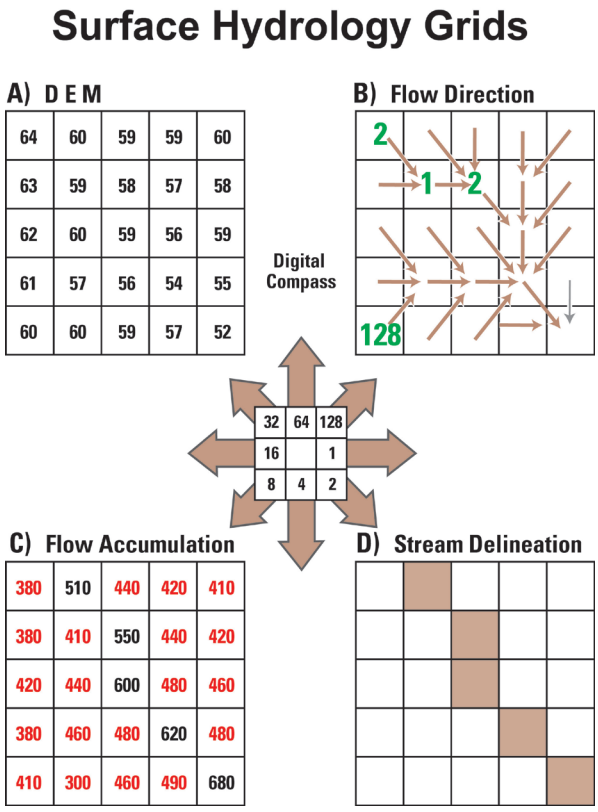

Figure 6. Diagram of part of a digital elevation model (DEM) and corresponding supplementary surface hydrology grids. (*a*) DEM where cells store values of elevation, in meters. (*b*) Flow direction grid derived from the DEM. Arrows represent the direction of flow; tails of arrows identify each individual cell; and arrowheads lie within the adjacent lowermost downstream cell. (*c*) The flow accumulation function uses the flow direction raster to calculate, for each cell, the number of upstream cells that flow into it. (*d*) Cells in the flow accumulation dataset with values (black text) greater than or equal to a user-specified stream delineation threshold (in this example a value of 500 cells) to identify stream cells (brown) in a stream raster.

Proximal Hazard Zone Boundary

Height/length runout (H/L) cones, sometimes referred to as energy-line or mobility cones (Sheridan, 1979), have an apex that usually coincides with a volcano summit and a slope determined by a ratio of vertical drop (H) to horizontal runout distance (L) characteristic for different volcano processes. The intersection of the cone with topography describes the predicted distance traveled (runout) by debris avalanches that originate on volcano flanks (Siebert, 1984) or summits (Hsu, 1975) and by pyroclastic flows (Sheridan, 1979; Beget and Limke, 1988). Values of H/L ratios typically range from about 0.1 to 0.3 yielding a cone that mathematically intersects topography. This line of intersection defines a boundary around a volcano edifice often used as a proximal hazard zone, depending on the size and type of the proximal event (Hayashi and Self, 1992). Laharz assumes that lahar inundation (primarily deposition) begins at the boundary of the proximal hazard zone and continues downstream, away from, or distal to, the volcano.

Laharz Algorithms

Laharz calculates a cross-sectional area and a planimetric area from user specified volumes according to equations (1) and (2). Although the cross-sectional area remains constant throughout a channel length, emulating the maximum discharge of a lahar, the shape of each cross section will vary downstream. Laharz uses the elevation values of the DEM to determine the shape of each cross section, beginning each section typically at the lowest cell elevation in the profile, a stream cell that represents the stream thalweg. The software (Schilling, 1998) builds each cross section upward using values of the DEM and tracks cell elevations progressively outwards from the thalweg stream cell. Laharz stores the X and Y (planimetric) locations of cells it encounters as it constructs a cross section. Laharz stops calculation of the cross section when the area of the cross section is greater than or equal to the calculated cross-sectional inundation area (A). A key concept of using a DEM is that each cell used to calculate a cross section occupies an finite area and thereby a fraction of the predicted planimetric area (B). As stream channels meander, cell locations from one cross section may coincide with cell locations stored from a previous cross section. In these cases, even though a single cell may contribute to many cross sections, a cell location contributes to planimetric area calculations only once. When cross sections for a stream cell are complete, the locations of cells occupied during construction form, in total, the planimetric inundation area. Laharz stops constructing the current inundation zone when the area of the current inundation zone is greater than or equal to the planimetric area (B).

Laharz_py

Laharz_py is written in the Python™ programming language (Python Software Foundation, 2013) and runs within ArcGIS (version 10 or later; Environmental Systems Research Institute,, 2012) as a set of scripts (hereinafter referred to as tools) that are grouped within an ArcMAP toolbox. These seven tools are:

1. *Create Surface Hydrology Rasters* (creates supplementary raster datasets for calculations),
2. *Generate New Stream Network* (creates a new raster stream network using a user defined stream threshold),
3. *Hazard Zone Proximal* (creates a proximal hazard zone),
4. *Lahar Distal Zones* (creates areas of potential inundation from user supplied volumes),
5. *Lahar Distal Zones with Conf Levels* (creates areas of potential inundation from a user supplied volume and two additional inundation areas showing range of possible results based on user-selected level of confidence),
6. *Merge Rasters by Volume* (combines areas along separate drainages derived from one volume), and
7. *Raster to Shapefile* (converts raster to vector dataset).

Three of the tools prepare initial datasets, two of the tools use statistically based empirical equations to forecast areas of potential inundation by future lahars, and two of the tools convert datasets to different forms. Laharz_py uses redesigned algorithms for the potential inundation tools. The software tracks cells encountered for multiple cross sections from one or more input volumes simultaneously, rather than the time-consuming process of iterating individual volumes in the earlier version. As a result, multi-volume scenarios that have taken minutes to hours in the previous version now take seconds. Although the use of each tool will be demonstrated in the following section as an exercise at Mount Rainier, Washington, new approaches for the two lahar inundation tools that calculate distal hazard zone areas warrants some description.

The *Lahar Distal Zones* tool operates similarly to the previous version of Laharz. However, now users can input up to seven volumes and identify multiple starting locations using a textfile. In addition, Laharz_py can calculate areas of potential inundation for several volcano processes. Griswold and Iverson (2008) used statistics to derive equations that relate inundation areas to flow volumes for debris flows, equations (3) and (4), and for rock avalanches, equations (5) and (6); where (A) is cross sectional inundation area, (B) is planimetric inundation area, and (V) is volume. With this tool, users choose whether to calculate potential inundation areas from lahars debris flows or rock avalanches.

$$A = 0.01\ V^{2/3} \tag{3}$$

$$B = 20\ V^{2/3} \tag{4}$$

$$A = 0.02\ V^{2/3} \tag{5}$$

$$\mathrm{B} = 20\ V^{2/3} \tag{6}$$

where

A	cross-sectional area;
B	planimetric area; and
V	volume.

The previous version of the Laharz software and the new *Lahar Distal Zones* tool, take a user-specified volume (V), and uses the regression to predict the mean expected (A) and (B) (fig. 7a). The 95-percent confidence intervals for prediction (green lines in fig. 7a) show the probable range of (A) and (B) values for a given volume.

The *Lahar Distal Zones with Conf Levels* tool allows the user to input a single volume and to select the confidence level appropriate for their problem. The purpose of calculating and displaying the range of lahar inundation possible for a given volume is to convey to the user of the data some of the uncertainties that are inherent to this empirical method of hazard estimation. As the confidence level changes, so does the range of corresponding (A) and (B) values, such that the range of statistically valid values grows if greater confidence is specified, or is reduced if lesser confidence is selected. After a level of confidence is selected, the *Lahar Distal Zones with Conf Levels* tool calculates the upper and lower confidence limits for the input volume (V) and from these values, the tool calculates two additional sets of (A) and (B) values (fig. 7b).

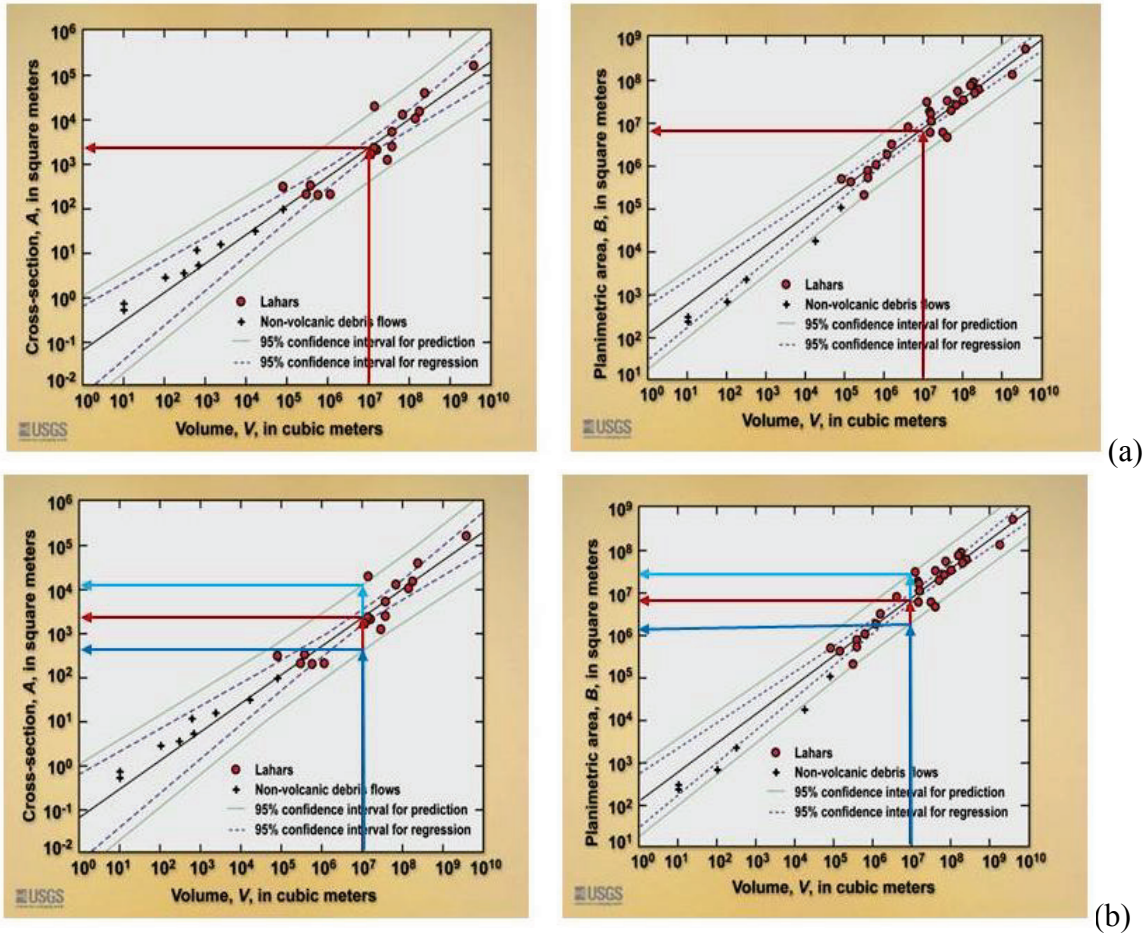

Figure 7. Scatter plots showing (a) for a selected V (here 10 million cubic meters), the regression predicts the mean (red line) expected at cross-sectional area, A, and planimetric area, B; and (b) the 95-percent confidence intervals for prediction show the probable dispersion (blue lines) of future A and B values. This dispersion increases if greater confidence is specified.

Three areas of potential inundation are calculated from a single volume, one showing the mean value of the regression, the other two showing the upper and lower confidence levels (fig. 8). During this process, the software opens and reads three text files (py_xxplanb.txt, py_xxsecta.txt, and py_xxttabl.txt) to calculate some basic statistical quantities described in detail in appendix A. Plotting all three areas of potential inundation forms a visual representation of the range of probable (A) and (B) values for the selected level of confidence, a useful visualization for decision making.

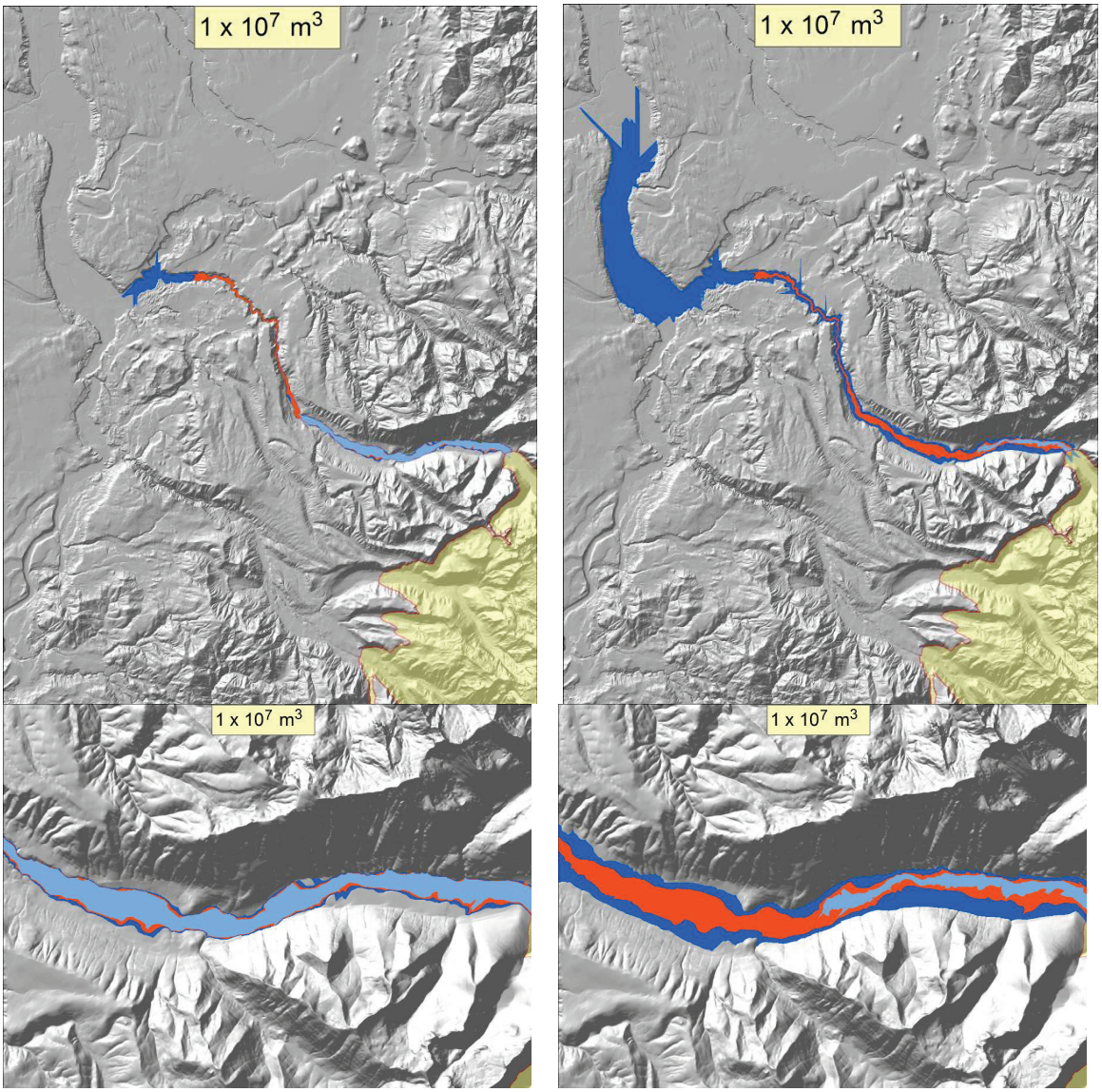

Figure 8. Light Detection and Ranging (LiDAR) images showing (in upper two images) the probable range of planimetric area (*B*) values with changing confidence limits (red is the mean value from the regression line, dark blue the upper confidence limit, light blue the lower confidence limit) from a single simulation of V (10 million cubic meters). The level of confidence shown is 50 percent (left upper image) and 99 percent (right upper image). The lower two images show the probable range of cross-sectional area (*A*) values with changing confidence limits (red is the mean value from the regression line, dark blue the upper confidence limit, light blue the lower confidence limit) from a single simulation of V (10 million cubic meters). The level of confidence shown is 50 percent (left lower image) and 99 percent (right lower image). This example is on the Carbon River, northwest flank of Mount Rainier, Washington.

Example Using Laharz_py

Figure 9 shows the files included in the distribution of Laharz_py_example that contains four subdirectories: laharz_py (contains Python™ scripts, hereinafter referred to as tools, for use with ArcGIS, version 10 or later), laharz_shapefiles (is empty), laharz_textfiles (contains text files used to run the software; note that the three text files with names beginning py_xx should be left unchanged as they are used by the confidence level version of Laharz_py to run properly), and mtrain_example (includes files and data to run this example for Mount Rainier, Washington) and Laharz_py.tbx a file that is an ArcGIS toolbox. The mtrain_example directory contains two raster datasets (mtrainier, the DEM, and mtrainiersh, a shaded relief dataset derived from the DEM). It also contains two subdirectories, laharz_textfiles and laharz_shapefiles that have been copied into this example directory. These two subdirectories must be copied into any Workspace directory in order for Laharz_py to function properly.

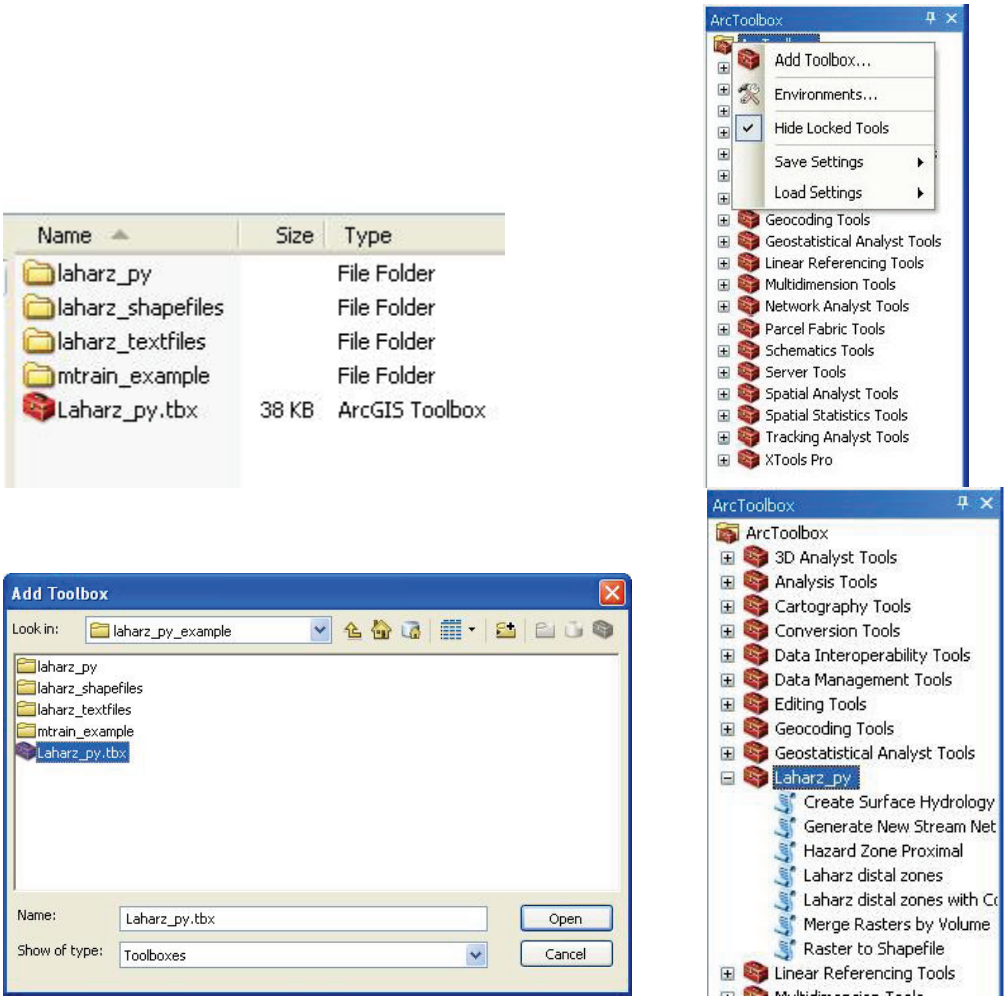

Figure 9. Screen captures showing a listing of files included with Lahar_py_example (upper left); right-click menu to add Laharz_py toolbox (upper right); navigation to the Laharz_py toolbox (lower left); and seven tools in the Laharz_py toolbox (lower right).

To begin using the Laharz_py software, start ArcMap and ensure the **ArcToolbox** window is open. In this example, it is affixed to the right part of the ArcMap interface. Right-click the root entry in the ArcToolbox tree, within the ArcToolbox window, at the top and select "Add toolbox…" from the popup menu (fig. 9). In the small window that appears, navigate to the Laharz_py_example directory and select Laharz_py.tbx toolbox (fig. 9). Within the ArcToolbox window, click on the small plus sign to the left of the Laharz_py toolbox to display the seven tools in the Laharz_py toolbox (fig. 9).

Each tool uses a similar graphical user interface (**GUI**) to enter or select appropriate inputs. The Laharz_py toolbox displays seven tools: *Create Surface Hydrology Rasters*, *Generate New Stream Network*, *Hazard Zone Proximal*, *Lahar Distal Zones*, *Lahar Distal Zones with Conf Levels*, *Merge Rasters by Volume*, and *Raster to Shapefile*. The first three tools typically are run once for a project to generate necessary datasets used for project simulations. The *Merge Rasters by Volume* and *Raster to Shapefile* tools are usually run after all simulations are complete. The *Lahar Distal Zones* tools are run as often as needed, typically once for each stream drainage.

For this example, click the **Add Data** button in ArcMap and navigate to the mtrain_example directory. Add the raster datasets **mtrainier** (DEM) and **mtrainiersh** (hillshade) to the project. The hillshade should be visible in the **ArcMap** window (fig. 10).

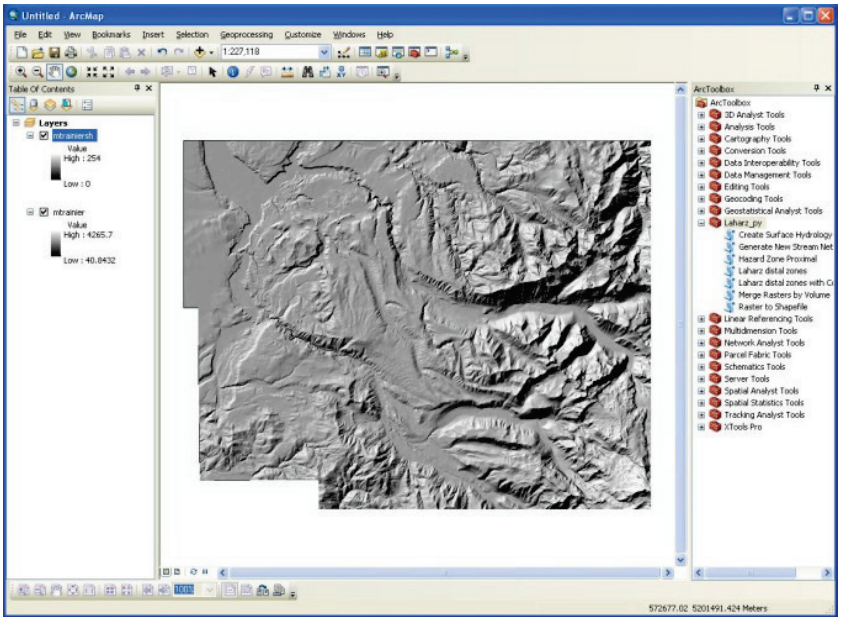

Figure 10. Screen capture showing raster datasets, mtrainfill and mtrainfillsh, added to project.

Creating Surface Hydrology Rasters

The first tool is labeled, *Create Surface Hydrology Rasters*. Double-click the tool to open a window to input field values (fig. 11). Enter or navigate to an ArcGIS Workspace, a directory that holds datasets for the project, then select a DEM from the current Workspace and enter a short prefix for names of supplementary raster datasets the tool will generate. Specify a stream threshold value (cells have greater than or equal to the threshold number of upstream cells flowing into it). Click on the **OK** button to start processing.

The tool creates a stream raster identifying cell locations where the flow accumulation raster is greater than or equal to the stream threshold value. As the tool runs, it will fill sinks in the DEM ensuring flow over the surface (fig. 6), and generate a flow direction, flow accumulation, and stream raster. The suffix "fill" is appended to the prefix to name the filled DEM, "dir" to name the flow direction raster, "flac" to name the flow accumulation raster, and "str" to name the stream delineation raster. For example, if the name of the input DEM is mtrainier and the user enters "mtr" as a prefix string, Laharz_py will name the four new rasters, mtrfill, mtrdir, mtrflac, and mtrstr; and store the rasters in the current workspace. The user must remember to use the filled, or depressionless, DEM for the rest of the program, rather than the original DEM.

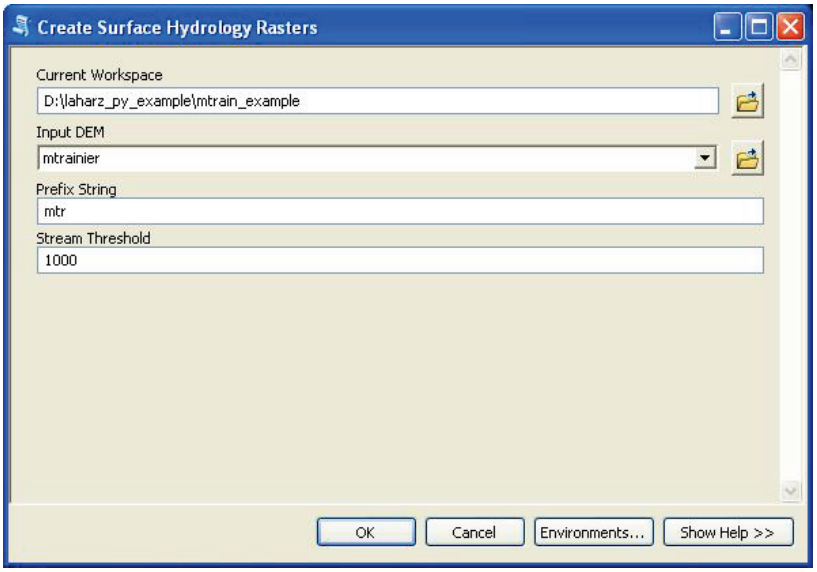

Figure 11. Screen capture showing window for creating supplementary surface hydrology rasters. Navigate to a Workspace (directory) that stores existing and new data sets for a project, then locates a DEM for the project and enters a prefix for supplementary raster data sets (keeping in mind that ArcInfo Grid names are restricted to 13 characters). The user also must specify a stream threshold value. Laharz_py will store locations of cells in a stream network where corresponding cells in the flow accumulation grid contain a value at or above the stream threshold. The software will run the FILL function to fill sinks (cell having undefined flow directions) in the DEM. The FILL function will fill sinks having a depth less than or equal to a threshold of 100 meters.

While each tool is running, a window displays progress (fig. 12) and the processing time when the tool is finished. When the tool is finished, the user can add these to the ArcMap session for viewing.

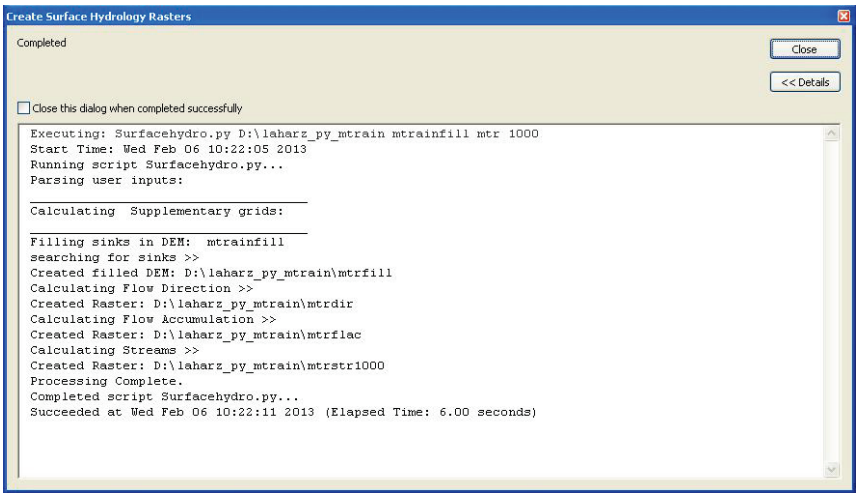

Figure 12. Screen capture showing progress and completion-time window when the Create Surface Hydrology Rasters is complete.

If necessary, run the *Generate New Stream Network* tool (fig. 13) to generate another stream network using a different threshold than was produced with the *Create Surface Hydrology Rasters* tool.

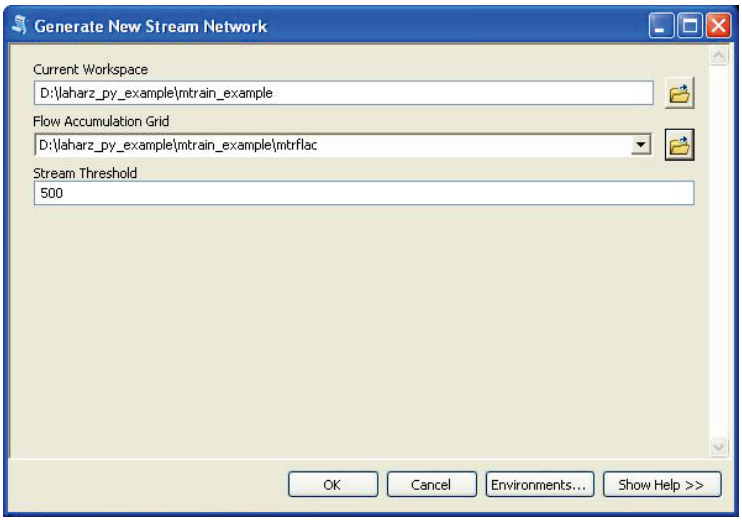

Figure 13. Screen capture showing window to generate an new stream network. The user enters or navigates to the Workspace, selects or navigates to a flow accumulation raster, and enters a stream threshold. The software labels the completed file with the same prefix as the flow accumulation raster, the letters “str” and the threshold value (for this example mtrstr500).

Creating a Proximal Hazard Zone Boundary

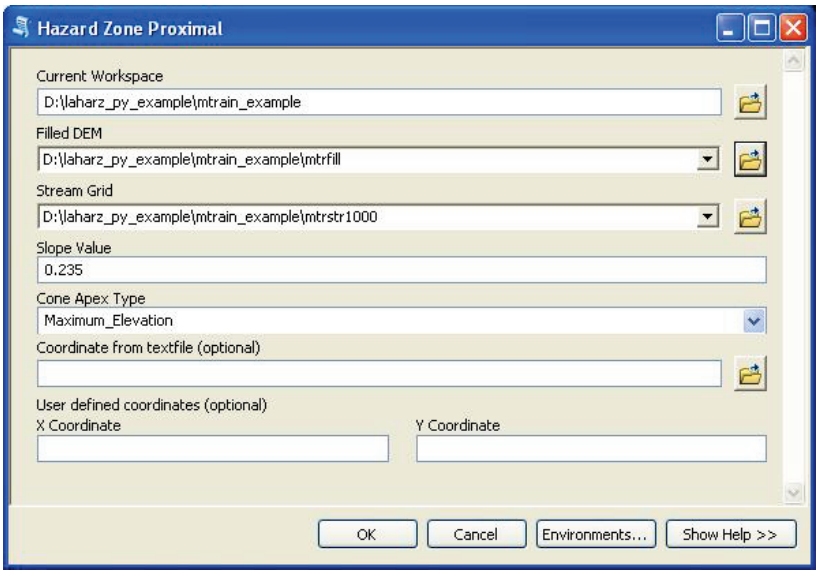

Figure 14. Screen capture showing window to create a proximal hazard zone boundary. Navigate to a Workspace, selects or enters a filled digital elevation model, selects or enters a stream raster and specifies a ratio of H to L (slope of H/L cone). A pull-down menu offers three alternatives to identify the location of the energy cone apex. The user can select the maximum elevation automatically (Maximum_Elevation), identify a textfile that contains X, Y coordinates (Textfile), or can type the coordinates manually (XY coordinates) as the location of the energy cone apex.

The *Proximal Hazard Zone Boundary* tool displays a window (fig. 14) to create a proximal hazard zone boundary. When the line of intersection between the energy cone and topography is completed, Laharz_py identifies all X,Y coordinates of cells where the cone boundary and streams intersect. Specify a Workspace, a filled DEM, a Stream raster, and a ratio of *H* to *L* (slope of the *H/L* cone). A pull-down menu offers three alternatives to identify the location of the cone apex. This choice determines whether the tool selects the maximum elevation automatically (Maximum_Elevation), identifies it from a textfile that contains X,Y coordinates (Textfile) of the apex, or accepts user-typed coordinates from the menu (XY coordinates) of the H/L cone apex. The simplest choice for the user is the Maximum_Elevation selection, provided that the highest elevation corresponds to the presumed volcanic vent location.

After the user identifies the apex location and the slope of the cone (specified using the Slope Value field on the Proximal Hazard Zone Boundary menu). The *Proximal Hazard Zone Boundary* tool calculates elevations of each cell in the cone raster according to the equation:

$$H_{Ap} - h_{cell} / L_{X,Yap} - l_{X,Ycell} = S \qquad (7)$$

where

H_{Ap}	height of the user-identified cone apex;
h_{cell}	cone height of the cell;
$L_{X,Yap}$	location of the user-identified cone apex;
$l_{X,Ycell}$	location of the cell; and
S	user-identified slope.

The tool uses the height and location of the cone apex and the slope that have been provided by the user. The tool calculates the denominator of equation (7) using a Euclidean distance algorithm (Environmental Systems Research Institute, 1994) and solves for h_{cell} at each cell in the raster.

The tool compares the values of the H/L raster with the elevations of the DEM at each cell location. In a temporary raster, the tool assigns the cells a unique arbitrary value where the H/L value is greater than the DEM elevation and a second unique arbitrary value where cell values are less than the DEM elevation. The tool uses these unique arbitrary values to create a polygon that marks the intersection between the H/L cone and the DEM, and is referred to as the Proximal Hazard Zone Boundary. The polygon boundary is converted back into a raster dataset, with the naming convention of hlcone<slope>_g, where <slope> is a user-specified slope value (for example, hlcone235_g is a cone having slope 0.235). As a final step, the Laharz_py software checks every cell to identify those cells where the proximal hazard zone boundary and the stream cells intersect (fig. 15). The X and Y coordinates of each of these cells is stored in a raster dataset, a shapefile, and a textfile.

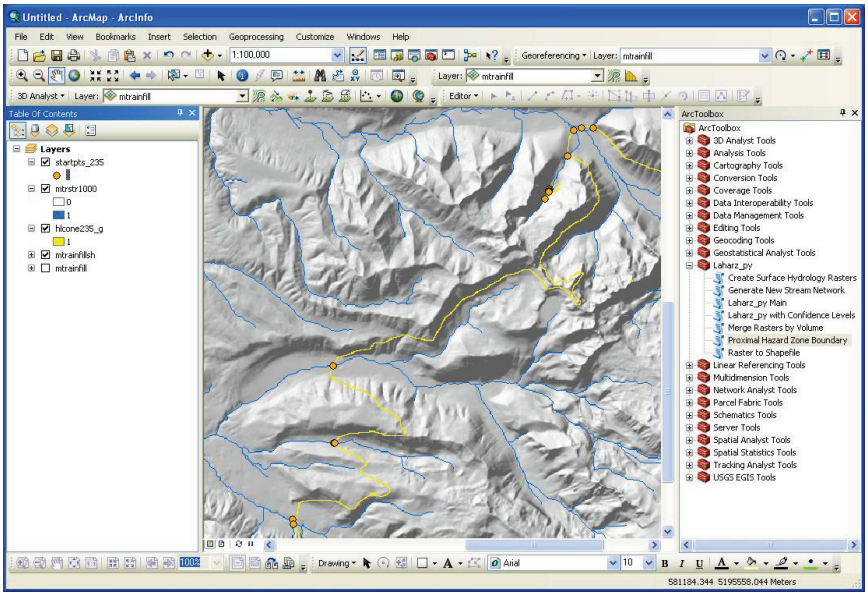

Figure 15. Screen capture showing proximal hazard zone boundary as the rasterized line, one cell in width, that marks the intersection of an H/L or energy cone with topography (yellow). The Laharz_py software calculates the boundary and identifies and stores X,Y coordinates where the boundary and streams (blue) intersect.

Create Lahar Inundation Zones

The *Lahar Distal Zone* tool will calculate areas of potential inundation for each drainage from user-specified X,Y coordinates, for each user-supplied debris volume. The tool creates a raster dataset for each drainage that can be plotted with other datasets such as a shaded relief map. Each raster dataset is named using the convention <drainage prefix identifier><number>. The user supplies the drainage prefix identifier, the number begins at 1 for the first coordinates supplied, 2 for the second coordinates, and so on. After the previous tools generate their datasets, double-click on the *Lahar Distal Zones* tool to display the tool menu (fig. 16). For this example, specify a textfile (volumesbig.txt, containing volumes of 10 and 30 million cubic meters) in the **Volume File** field. The file must contain between one and seven lahar volumes, specified in a comma-separated list, in cubic meters, on a single line. Users determine appropriate volumes on the basis of experience, judgment, fieldwork (if possible), and knowledge of the particular volcano (Iverson and others, 1998). In addition, the window requires fields of a Workspace, a filled DEM, prefix for the drainage (puycarb_ in this example), and a textfile containing X,Y coordinates of initiation sites for the simulations (threepnts_235.txt for this example). The **Type of Flow** pull-down menu offers three flow types: Lahar, Debris_Flow, and Rock_Avalanche. For this example, select **Lahar** (fig. 16) from the pull-down menu. The tool uses the equations (1) and (2) to calculate the cross-sectional area (A) and total planimetric area (B) (Iverson and others, 1998) for each of the user-specified lahar volumes. The tool will use equations (3) and (4) if the user selects **Debris_Flow** from the pull down menu, and equations (5) and (6) if the user chooses **Rock_Avalanche**.

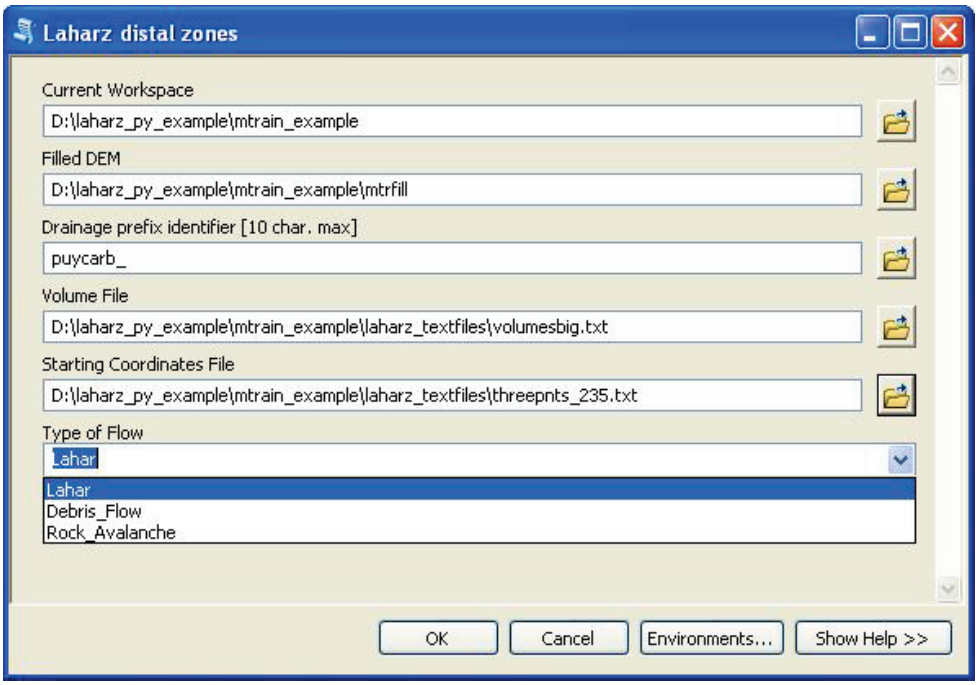

Figure 16. Screen capture showing interface to start *Lahar Distal Zones*. The user is required to fill in all fields. The user selects, navigates to, or enters a Workspace, a filled DEM, a filename prefix for the drainage (Puyallup-Carbon River drainage, for the Mount Rainier example), a textfile that contains the volumes (volumesbig.txt contains volumes of 10 and 30 million cubic meters) and a textfile containing X,Y coordinates of initiation sites for the simulations (threepnts_235.txt contains X and Y coordinates). A pull-down menu offers three alternative flow types: Lahar, Debris_Flow, and Rock_Avalanche. For this example, the user selects the flow type: Lahar.

When processing of inundation zones is complete, the tool creates a textfile, with the ".pts" file extension, and a raster of the inundated area for each drainage, for each lahar volume. The text files, named <drainage prefix identifier><run number>.pts, contains a list of the user-specified volumes, the cross-sectional and planimetric area values calculated by Laharz_py from user-specified volumes, and a list of decreasing planimetric areas, one row of values per stream cell.

The textfile (puycarb_1.pts) for the first drainage area and volume setting is shown in figure 17. Each raster is named using the drainage prefix identifier appended to a suffix of a sequential number (for example, 1,2,3, and so on).

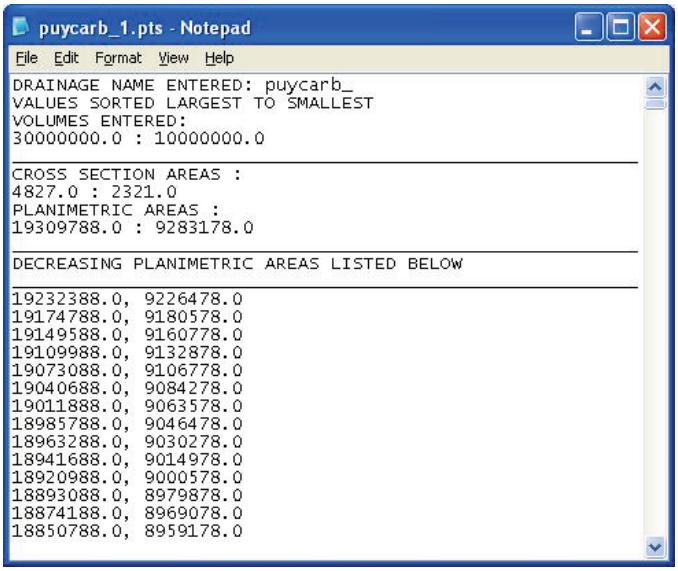

Figure 17. Screen capture showing a partial textfile listing for one of the .pts files created from the simulations. When a simulation is complete the decreasing planimetric area for a simulation is less than zero.

Use the **Add Data** button to view the potential lahar-inundation rasters over a shaded relief version of the DEM in ArcMap (fig. 18). This example results with a raster and a textfile for each drainage.

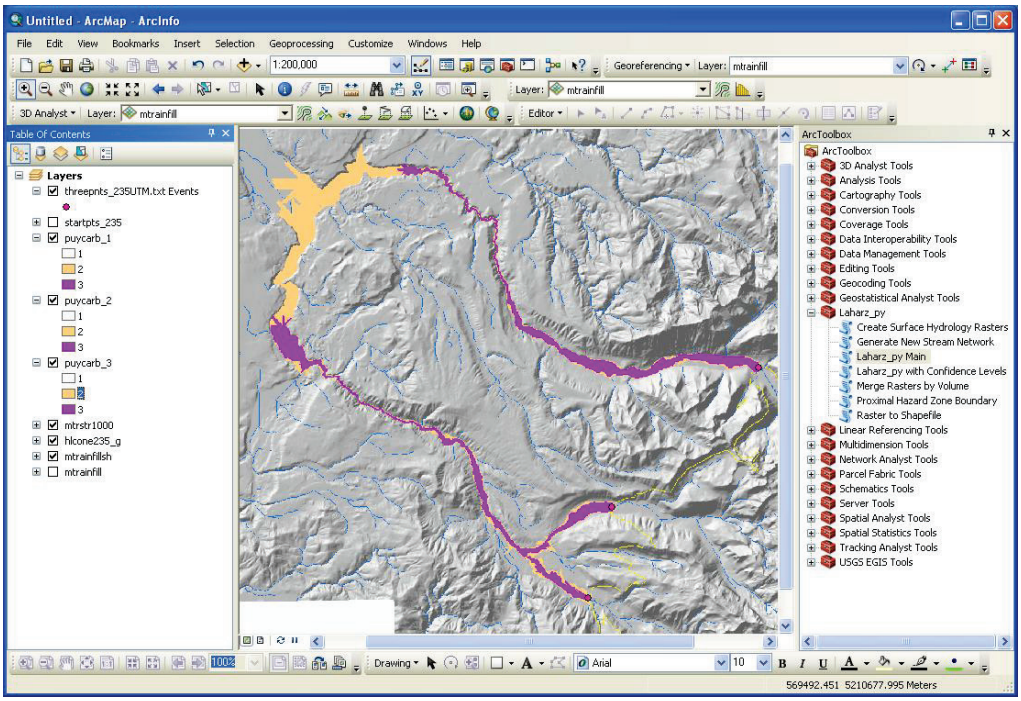

Figure 18. Screen capture showing three completed simulations. Smaller volumes are shown in purple, larger volumes are shown in orange.

Merge Rasters by Volume

After *Lahar Distal Zones* tool is complete, there are three rasters (puycarb_1, puycarb_2, and puycarb_3), one for each of the three drainages. Each raster contains the results for two volumes. Users may wish to combine the 10 million cubic meter results for the three drainages, and the 30 million cubic meter results for the three drainages. Double-clicking the *Merge Rasters by Volume* tool will display a menu (fig. 19). Specify a Workspace, a filled DEM, a filename of the textfile containing the list of rasters to merge (for this example the textfile is laharzruns.txt, the list of rasters to merge includes puycarb_1, puycarb_2, and puycarb_3), and a textfile that contained the input flow volumes named volumesbig.txt. Users can add the resulting two rasters, merge_2 and merge_3 (one for each volume) to the ArcMap project for viewing (fig. 20). These generic names can be changed to something more meaningful using ArcCatalog.

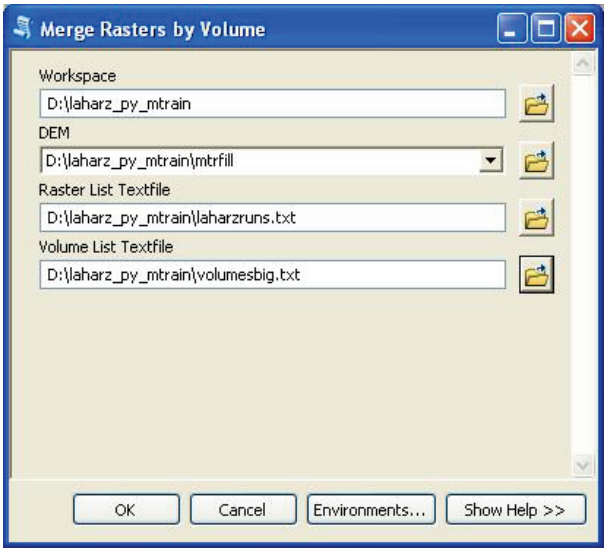

Figure 19. Screen capture showing menu for the *Merge Rasters by Volume* tool. A user specifies a Workspace, a filled digital elevation model , a textfile that lists the rasters to merge (for this example the textfile is laharzruns.txt, the rasters to merge are puycarb_1, puycarb_2, and puycarb_3), and a textfile that contains the volumes (textfile named volumesbig.txt used for the previous tool).

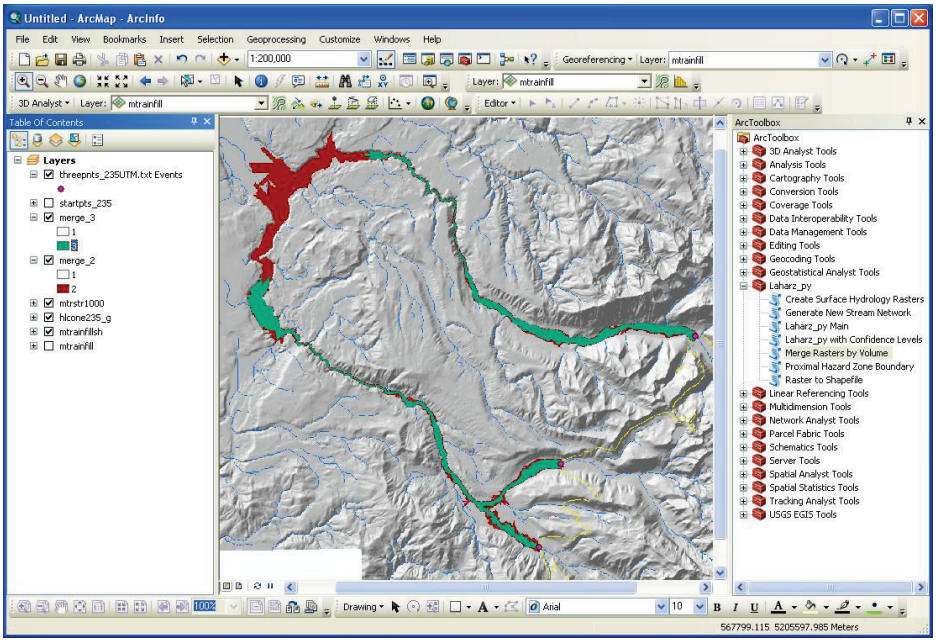

Figure 20. Screen capture showing merged raster datasets. Here the dataset merge_3 shows the 10 million cubic meter data from all three drainages (green) and Merge_2 shows the 30 million cubic meter data from all three drainages (dark red).

Raster to Shapefile

Users may want to convert the raster dataset results to a vector shapefile, a smaller sized dataset and convenient to distribute. Double-clicking the tool *Raster to Shapefile* will display a menu (fig. 21). Select or navigate to a Workspace, enter a name for the new shapefile (in this example, mtr10mcm), and select or navigate to a raster data set to convert (merge_3). Users can add the resulting shapefile (mtr10mcm) to the ArcMap project for viewing (fig. 22).

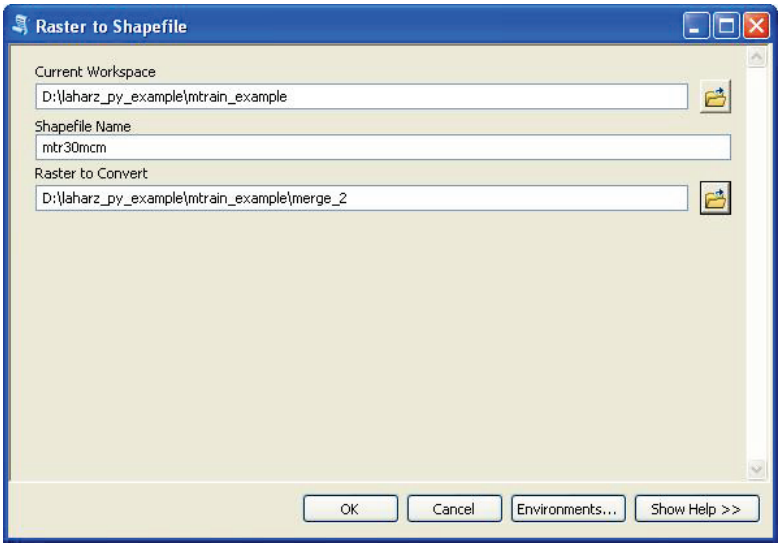

Figure 21. Screen capture showing window to convert a raster dataset to a vector shapefile using the *Raster to Shapefile* tool. Specify a Workspace, a name for the new shapfile (in this example, mtr30mcm), and a raster dataset to convert (merge_2).

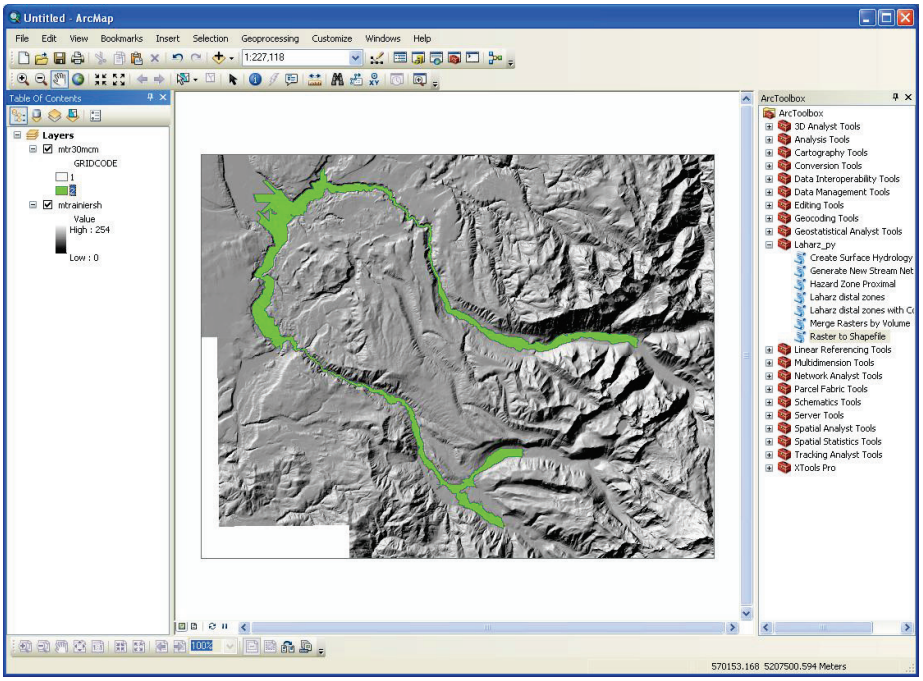

Figure 22. Screen capture showing mtr30mcm shapefile. The shapefile is a polygon vector dataset colored according to the attribute GRIDCODE in the attribute table.

Creating Lahar Inundation Zones with Confidence Levels

The *Lahar Distal Zones with Conf Levels* tool calculates the mean value for cross-sectional (*A*) and planimetric (*B*) areas as well as the probable range of possible cross–sectional (*A*) and planimetric areas (*B*) values with user-selected confidence limits. The user selects the *Lahar Distal Zones with Conf Levels* tool from the Laharz_py toolbox. From the tool menu, specify a Workspace, a filled DEM, a filename prefix for the drainage, a textfile that contains a single volume (numbers only, no commas or periods), a textfile containing X,Y coordinates to begin the simulation, and a level of confidence from the pull-down menu in appropriate fields (fig. 23). Users can add the resulting raster (carbon_1) to the ArcMap project (fig. 24) that shows the results from the entered volume as well as results showing the range of possible (*A*) and (*B*) areas for the selected level of confidence.

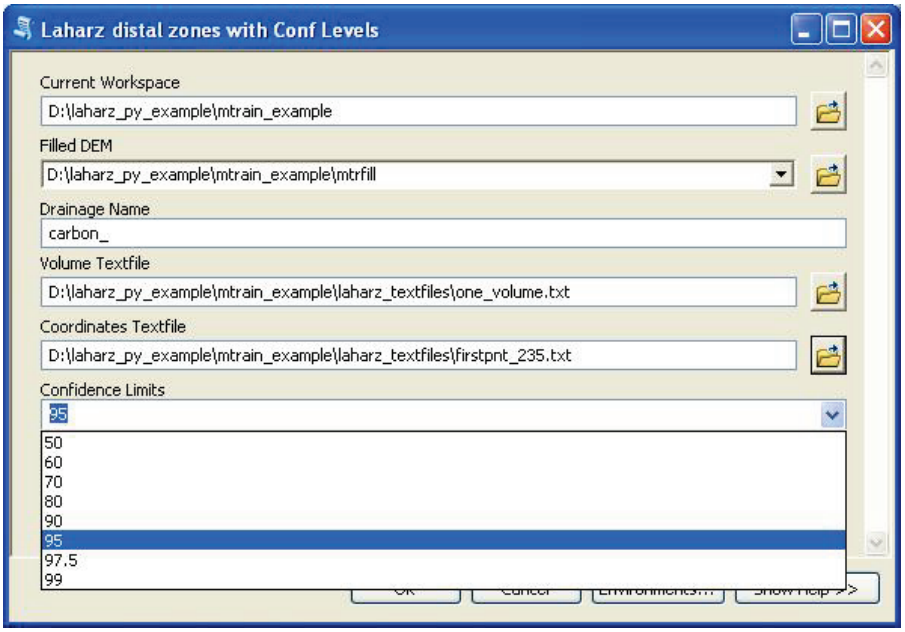

Figure 23. Screen capture showing interface to begin Laharz_py calculations with user-selected level of confidence. All fields must be filled. A user specifies a Workspace, a filled DEM, a filename prefix for the drainage (for this example, carbon_), a textfile that contains a single volume for a drainage (only numbers, no commas or periods, for this example a 1 million cubic meter volume) and a textfile containing X,Y coordinates of initiation sites for the simulations (in this example, one coordinate in a file named firstpnt_235.txt). A pull-down menu offers, discrete levels of confidence ranging from 50- to 99-percent confidence (for this example, 95 should be selected).

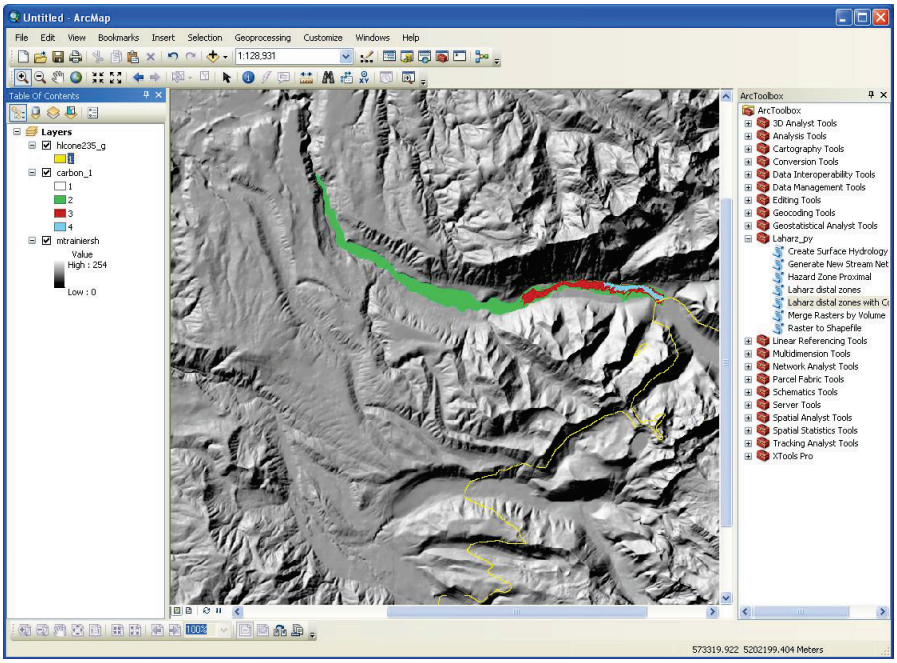

Figure 24. Screen capture showing carbon_1 raster results. The raster shows the volume entered (1 million cubic meters shown in red) as well as the results from volumes that correspond to the 95-percent confidence levels (green and light blue).

Acknowledgments

The author thanks Richard Iverson and James Vallance for their work, patience, and helpful discussions developing the original methodology. Using the levels of confidence, decision making arose from the observations in the field by Andrew Lockhart and led to a new method (appendix 1) being developed by Richard Iverson for incorporation into the software. Julie Griswold and Richard Iverson conducted the statistical work with non-volcanic debris flows and rock avalanches. This report was improved considerably by the careful and thoughtful reviews by Roland Viger and Rob Wardwell. Teaching a number of courses of Laharz with Julie Griswold, who offered thoughtful comments and good humor, as well as suggestions from many excellent students, have led to improvements of the software.

References Cited

Beget, J.E., and Limke, A.J., 1988, Two-dimensional kinematic and rheological modeling of the 1912 pyroclastic flow, Katmai, Alaska: Bulletin of Volcanology, 50, p. 148–160.

Environmental Systems Research Institute, 1994, Cell-based modeling with GRID: Environmental Systems Research Institute Inc., Redlands, California, p. 309–327.

Environmental Systems Research Institute, 2012, ArcGIS: Environmental Systems Research Institute Inc., Redlands, California.

Griswold, J.P., and Iverson, R., 2008, Mobility statistics and automated hazard mapping for debris flows and rock avalanches: U.S. Geological Survey Scientific Investigations Report 2007-5276, 59 p.

Hayashi, J.N., and Self, S., 1992, A comparison of pyroclastic flow and debris avalanche mobility: Journal of Geophysical Research, v. 97, no. B6, p. 9063–9071.

Hsu, K.J., 1975, Catastrophic debris streams, sturzstroms, generated by rockfalls: Geological Society of America Bulletin, v. 86, p. 129–140.

Hann, C.T., 2002, Statistical methods in Hydrology, 2d edition: Iowa State University Press, Ames, Iowa, 378 p.

Iverson, R.M., Schilling, S.P., and Vallance, J.W., 1998, Objective delineation of areas at risk from inundation by lahars: Geological Society of America Bulletin, v. 110, no. 8, p. 972–984.

Jenson, S.K., and Domingue, J.O., 1988, Extracting topographic structure from digital elevation data for geographic information system analysis: Photogrammetric Engineering and Remote Sensing, v. 54, p. 1593–1600.

Mothes, P.A., Hall, M.L., and Janda, R.J., 1998, The enormous Chillos Valley Lahar—An ash-flow-generated debris flow from Cotopaxi Volcano, Ecuador: Bulletin of Volcanology, v. 59, p. 233–244.

Newhall, C.G., Hendley II, J.W., and Stauffer, P.H., 1997, Lahars of Mount Pinatubo, Philippines: U.S. Geological Survey Fact Sheet 114-97, http://pubs.usgs.gov/fs/1997/fs114-97/.

Python Software Foundation, 2013, Python programming language–Official website: Python Software Foundation, accessed February 4, 2014, at http://python.org/

Schilling, S.P., 1998, LAHARZ—GIS Programs for automated mapping of lahar-inundation hazard zones: U.S. Geological Survey Open-File Report 98-638, 80 p.

Scott, K.M., Vallance, J.W., and Pringle, P.T., 1995, Sedimentology, behavior, and hazards of debris flows at Mount Rainier, Washington: U.S. Geological Survey Professional Paper 1547, 56 p.

Sheridan, M.F., 1979, Emplacement of pyroclastic flows—A review, *in* Chapin, C.E., and Elston, W.E., eds., Ash-flow tuffs: Geological Society of America Special Paper 180, p. 125–136.

Siebert, L., 1984, Large volcanic debris avalanches—Characteristics of source areas, deposits, and associated eruptions: Journal of Volcanology and Geothermal Research, 22, p. 163–197.

Vallance, J.W., 2000, Lahars, *in* Sigurdsson, H., Houghton, B.F., McNutt, S.R., Rhymer, H. and Stix, J., eds., Encyclopedia of Volcanoes: Academic Press, p. 601–616.

This page left intentionally blank

Appendix A. Strategy for Computing and Portraying Confidence Limits in Inundation Predictions Using Laharz (R.M. Iverson, written commun., October 2007)

The statistical procedures outlined here are amalgamated from several sources, primarily from Haan (2002). Steps 1–3 provide baseline statistical information necessary to calculate confidence limits for predictions. Steps 4–6 depend on user-selected flow volumes and confidence intervals.

1. If new data are added to the set reported by Iverson and others (1998), redo the linear regression and analysis of variance to obtain prediction equations of the form:

$$\log A = \log C + \frac{2}{3}\log V \tag{1}$$

The implication here is that data for A and V are log-transformed prior to doing the regression. To simplify ensuing equations, it is useful to rewrite (1) in the form:

$$y = c + \frac{2}{3}x \tag{2}$$

where $y = \log A$, $c = \log C$ and $x = \log V$.

2. The next step is to find some basic statistical quantities. The number of data pairs used in the regression is n. The degrees of freedom in the regression model, which has only one adjustable parameter, c, is therefore n-1. The residual sum of squares associated with the regression, calculated as described in the appendix of Iverson and others (1998), describes the sum of deviations of observed y values from those predicted by the regression line (2). Here the residual sum of squares is denoted by Σe^2. The standard error of the regression model is then defined as:

$$s = \sqrt{\frac{\Sigma e^2}{n-1}} \tag{3}$$

This is identical to the definition in the appendix of Iverson and others (1998).

3. To calculate prediction error, the first new quantity that needs to be defined is the standard error of the predicted mean response (y) for any fixed value of x. (This quantity is called the standard error of the mean, SE_m.) Any arbitrary fixed value of x is denoted by x^*, and our collection of n observed values of x is denoted by x_n. Mathematically, the standard error of the mean is defined as:

$$SE_m = s \times \sqrt{\frac{1}{n} + \frac{(x^* - \bar{x})^2}{\Sigma(x_n - \bar{x})^2}} \tag{4}$$

Importantly, note that this standard error varies as a function of x^*, which reflects the fact that there is more confidence in the regression line near the mean value of x (that is, $\bar{x}$) than for larger or smaller values of x. (This variability is what gives rise to the curvature of the confidence limits for regression lines.) The key thing about SE_m, from an operational standpoint, is that it is a function of x^*. Therefore, SE_m must be calculated for the full range of x values that might be used in Laharz. This calculation is probably best done by systematically incrementing x by some small fraction (for example, 0.001) of its full range and calculating SE_m at each point.

However, it is necessary to calculate or estimate (through interpolation) SE_m for all values of x (that is, log V) that are to be used in making Laharz predictions. Thus, it is probably best to fit a curve to the function $SE_m(x)$ so that interpolation can be very precise.

4. To calculate the prediction error, the next new quantity needed is the standard error of prediction, SE_p. This quantity is defined mathematically as:

$$SE_p = \sqrt{SE_m{}^2 + s^2} \tag{5}$$

This prediction error includes the effects of uncertainty represented by the standard error of the mean y (for any x^*) given by the calibrated regression equation as well as the error in estimation y for a new (user-selected) value of x, which is given by s. Note that the standard error of prediction always exceeds the standard error of regression.

5. The final new quantity needed in order to calculate prediction error is the *user-selected level of confidence*. The confidence interval about the regression line is then evaluated using the statistical t distribution, which assumes that true y values at any x are normally distributed about the mean y value estimated by regression (with standard error s). For example, if we want a 90 percent confidence interval, we need the value of $t_{0.1,n-1}$, where the subscript 0.1 is the value of $\alpha = 1 - 0.9$ for a two-tailed t distribution, and the subscript n-1 is the degrees of freedom of the regression model. (Note that statistics books can be a bit confusing in this respect, because they commonly use one-tailed rather than two-tailed t distributions. If we were using a one-tailed t distribution, we would use $\alpha = 1 - 0.95$ to calculate the 90 percent confidence limits.) Computation of t distributions from first principles is complicated, as it involves evaluation of gamma functions. So instead it is more efficient to use a look-up table to find relevant values of t. Such a look-up table is generated quite easily using the statistical functions in Excel®, for example.

6. Once the confidence level is chosen and the appropriate value of t is selected from the look-up table, the confidence intervals for prediction (denoted by $y\pm$) are given by:

$$y\pm = t_{\alpha,n-1} \times SE_p$$

This equation provides the value of y to be added or subtracted from the value predicted by regression in order to represent confidence limits for the $1-\alpha$ level. Finally, this new y value (that is A or B value inLaharz) is used to map inundation zones that portray the $1-\alpha$ confidence limits.

Appendix B. Code

surface_hydro.py

```
# ---------------------------------------------------------------------------
# surface_hydro.py
#
# Usage: surface_hydro.py is attached to Laharz_py.tbx (toolbox)
#  sys.argv[1] a workspace
#  sys.argv[2] a DEM, input surface raster
#  sys.argv[3] a prefix string for surface hydrology datasets
#  sys.argv[4] threshold value to demarcate a stream; default is 1000
#
#  This program creates surface hydrology datasets from an input raster (DEM)
# It fills sinks in the original DEM, then calculates flow direction, flow
accumulation,
# and delineates streams from flow accumulation according to the stream threshold
#
# ---------------------------------------------------------------------------

# Start Up - Import system modules
import sys, string, os, arcpy
from arcpy import env
from arcpy.sa import *

# Check out license
arcpy.CheckOutExtension("Spatial")


def main():
  try:
    #=========================================================================
    # Assign user inputs from menu to appropriate variables
    #=========================================================================
    arcpy.AddMessage("Parsing user inputs:")

    env.workspace = sys.argv[1]     # set the ArcGIS workspace
    env.extent = sys.argv[2]      # set extent to input DEM
    env.snapRaster = sys.argv[2]    # ensure new rasters align with input DEM
    env.cellSize = sys.argv[2]     # set cell size of new rasters same as input DEM
    env.scratchWorkspace = env.workspace # set the ArcGIS scratchworkspace

    Input_surface_raster = sys.argv[2] # name of DEM
    PreName = sys.argv[3]        # prefix for surface raster data sets
    Stream_Value = sys.argv[4]     # threshold of flow accumulation to demarcate
streams

    # local variables
    curdir = env.workspace       # current directory
    textdir = curdir + "\\laharz_textfiles\\"
    shapedir = curdir + "\\laharz_shapefiles\\"

    if Stream_Value == '#':
      Stream_Value = "1000" # provide a default value if unspecified
```

```
    #=============================================
    # Set filenames and directories
    #=============================================
     fillname = curdir + "\\" + PreName + "fill"
     dirname = curdir + "\\" + PreName + "dir"
     flacname = curdir + "\\" + PreName + "flac"
     strname = curdir + "\\" + PreName + "str" + str(Stream_Value)

     #=============================================================================
     arcpy.AddMessage( "_____________________________________")
     arcpy.AddMessage( "Calculating Supplementary grids:")
     arcpy.AddMessage( "_____________________________________")
     arcpy.AddMessage( "")
     arcpy.AddMessage( "Filling sinks in DEM: " + Input_surface_raster)

     # Fill
     arcpy.AddMessage( 'searching for sinks >>')
     tempa = Fill(Input_surface_raster)
     tempa.save(fillname)
     arcpy.AddMessage( 'Created filled DEM: ' + fillname)

     # Flow Direction
     arcpy.AddMessage( "Calculating Flow Direction >>")
     tempa2 = FlowDirection(fillname)
     tempa2.save(dirname)
     arcpy.AddMessage( 'Created Raster: ' + dirname)

     # Flow Accumulation
     arcpy.AddMessage( "Calculating Flow Accumulation >>")
     tempa3 = FlowAccumulation(dirname, "", "INTEGER")
     tempa3.save(flacname)
     arcpy.AddMessage( 'Created Raster: ' + flacname)

     # Applying stream threshold
     arcpy.AddMessage( "Calculating Streams >> ")
     tempa4 = GreaterThan(flacname, int(Stream_Value))
     tempa4.save(strname)
     arcpy.AddMessage( "Created Raster: " + strname)

     arcpy.AddMessage( "Processing Complete.")

     arcpy.AddMessage( "")
   except:
     print arcpy.GetMessages(2)

if __name__ == "__main__":
   main()
```

new_stream_network.py

```
# ---------------------------------------------------------------------------
# new_stream_network.py
#
# Usage: new_stream_network.py is attached to Laharz_py.tbx (toolbox)
#  sys.argv[1] a workspace
#  sys.argv[2] a DEM, input surface raster
#  sys.argv[3] threshold value to demarcate a stream; default is 1000
#
#
#  This program creates a single stream network raster from an input raster (DEM)
# It assumes their is an existing flow direction and flow accumulation rasters
# It uses the new threshold to calculate a new stream network
# ---------------------------------------------------------------------------

# Start Up - Import system modules
import sys, string, os, arcpy
from arcpy import env
from arcpy.sa import *

# Check out license
arcpy.CheckOutExtension("Spatial")

def main():
  try:

 #=============================================================================
 # Assign user inputs from menu to appropriate variables
 #=============================================================================
    arcpy.AddMessage("Parsing user inputs:")

    env.Workspace = sys.argv[1]     # set the ArcGIS workspace
    arcpy.extent = sys.argv[2]      # set extent to input DEM
    arcpy.SnapRaster = sys.argv[2]    # ensure new rasters align with input DEM
    env.scratchWorkspace = env.Workspace  # set the ArcGIS scratchworkspace

    # local variables
    curdir = env.Workspace          # current directory
    Flow_accum_raster = sys.argv[2]   # flow accumulation raster

    Stream_Value = sys.argv[3]      # stream threshold
    if Stream_Value == '#':
      Stream_Value = "1000" # provide a default value if unspecified


    arcpy.AddMessage( "_____________________________________")
    arcpy.AddMessage( "Calculating New Stream network:")
    arcpy.AddMessage( "_____________________________________")
    arcpy.AddMessage( "Flow Accumulation Raster: " + Flow_accum_raster)
    if Flow_accum_raster.endswith("flac"):
      atemp = Flow_accum_raster.rstrip("flac")
      atemp2 = os.path.basename(atemp)
    arcpy.AddMessage( "Prefix name is: " + atemp2)
    arcpy.AddMessage( "")
    arcpy.AddMessage( "")
    strname = curdir + "\\" + atemp2 +"str" + str(Stream_Value)
```

```
 # Applying threshold
    arcpy.AddMessage( "Calculating new Stream paths:")
    tempb = GreaterThan(Flow_accum_raster, int(Stream_Value))
    tempb.save(strname)

    arcpy.AddMessage( "Created raster: " + strname)
    arcpy.AddMessage( "")

    arcpy.AddMessage( "Processing Complete.")


  except:
    print arcpy.GetMessages(2)

if __name__ == "__main__":
  main()
```

proximal_zone.py

```
# ---------------------------------------------------------------------------
# proximal_zone.py
#
# Usage: hlcone_startpts.py is attached to Laharz_py.tbx (toolbox)
#  sys.argv[1] a workspace
#  sys.argv[2] a DEM, input surface raster
#  sys.argv[3] an input stream raster
#  sys.argv[4] a decimal slope value for the H/L cone; default is 0.3
#  sys.argv[5] an apex choice for the cone (max elev, XY point, or textfile coords
#  sys.argv[6] coordinates for apex of cone from text file
#  sys.argv[7] coordinates for apex of cone from typed coordinates
#
#

#  This program calculates a raster dataset that represents an H/L cone from
# the input raster (DEM), user defined slope, and user identified location of the
cone apex
# The resulting line where the cone intersects the DEM defines the boundary
# Point coordinates where the boundary intersects streams are stored in a text file
# In general, cells between the line and the apex are above the cone surface
whereas
# Outboard of the line, the cone surface is below the DEM.
# ---------------------------------------------------------------------------

# Start Up - Import system modules
import sys, string, os, arcpy
from arcpy import env
from arcpy.sa import *

# Check out license
arcpy.CheckOutExtension("Spatial")

#=============================================================================
# Local Functions
#=============================================================================

def ConvertTxtToList(atxtfilename,alist,dattype):
  # ====================================
  # Parameters:
  #  atxtfilename: name of a textfile
  #  alist: empty python list
  #  dattype: keyword 'volumes' or 'points' to determine how text file
  #       is manipulated
  #
  # Opens volume or starting coordinate textfile,
  # appends values read from textfile to the list,
  # volume list is sorted smallest to largest
  #
  # Returns: list
  # ====================================
```

```
 afile = open(atxtfilename, 'r')

  for aline in afile:

    if dattype == 'volumes':
      x = aline.rstrip('\n')
      y = round(float(x.lstrip(' ')))
      alist.append(y)

    else:
      if aline.find(',') <> -1: # if it does have a ','
        x = aline.rstrip('\n')
        y = x.split(',')
        for i in range(len(y)):
          y[i] = round(float(y[i].lstrip(' ')))
        if dattype == 'volumes':

          y.sort()
          alist = y
        else:
          alist.append(y)
  afile.close()

  return alist

def makepointlist(cellx,celly,oneptlist):
  # ====================================
  # Parameters:
  #  cellx: X coordinate of the current cell
  #  celly: Y coordinate of the current cell
  #  oneptlist: a list to store coordinates
  #
  # Appends the X and Y of current cell to a list
  #
  # ====================================

  onePoint = arcpy.Point(cellx, celly)
  oneptlist.append(onePoint)

  return oneptlist

def main():
  try:

    #=========================================================================
    # Assign user inputs from menu to appropriate variables
    #=========================================================================
    arcpy.AddMessage("Parsing user inputs:")
    env.workspace = sys.argv[1]    # set the ArcGIS workspace
    Input_surface_raster = sys.argv[2] # name of DEM
    Input_stream_raster = sys.argv[3] # name of stream raster
    slope_value = sys.argv[4]     # decimal slope entered by user
    apex_choice = sys.argv[5]     # max elev, XY point, or textfile coords
    coordstxt = sys.argv[6]      # name of textfile with apex coordinates
    coordspnt = sys.argv[7]      # coordinates entered at keyboard
```

```
#============================================
# Set the ArcGIS environment settings
#============================================

   env.scratchWorkspace = env.workspace
   env.extent = Input_surface_raster
   env.snapRaster = Input_surface_raster
   curdir = env.workspace
   textdir = curdir + "\\laharz_textfiles\\"
   shapedir = curdir + "\\laharz_shapefiles\\"

   #============================================
   # report dem selected back to user
   #============================================
   arcpy.AddMessage( "_________ Input Values _________")
   arcpy.AddMessage( 'Surface Raster Is: ' + Input_surface_raster)
   arcpy.AddMessage( 'Stream Raster Is: ' + Input_stream_raster)

   #============================================
   # Set filenames and directories
   #============================================
   BName = os.path.basename(Input_surface_raster)
   DName = env.workspace + "\\"

   # if the filename suffix is "fill", get prefix name
   if BName.endswith("fill"):
     PreName = BName.rstrip("fill")

   # assign complete names for supplementary files,
   # path and basename(prefix and suffix)
   fillname = DName+PreName+"fill"
   dirname = DName+PreName+"dir"
   flacname = DName+PreName+"flac"
   strname = DName+PreName+"str"

   pfillname = PreName+"fill"
   pdirname = PreName+"dir"
   pflacname = PreName+"flac"
   pstrname = PreName+"str"


   arcpy.AddMessage("full path fill :" + fillname)
   arcpy.AddMessage("full path dir  :" + dirname)
   arcpy.AddMessage("full path flac :" + flacname)
   arcpy.AddMessage("full path str  :" + strname)

   arcpy.AddMessage("partial path fill :" + pfillname)
   arcpy.AddMessage("partial path dir  :" + pdirname)
   arcpy.AddMessage("partial path flac :" + pflacname)
   arcpy.AddMessage("partial path str  :" + pstrname)

   # assign the flow direction and flow accumulation grids to variables
   Input_direction_raster = dirname
   Input_flowaccumulation_raster = flacname
```

```
 # use slope value as part of grid and shapefile name

    y = slope_value.split('.')
    slopename = str(y[1])

    hlconename = "hlcone"+slopename+"_g"
    hlshapename = shapedir + "startpts_"+slopename+".shp"
    hlgridname = "stpts_g"+slopename


    # ====================================
    # set file name for two textfiles storing X,Y coordinates
    # one storing an arbitrary single starting point
    # the other file storing all stream/hl cone intersections as starting points
    # ====================================
    txfileuno = textdir + "firstpnt_"+slopename+".txt"
    txfile = textdir + "startpnts_"+slopename+".txt"


    arcpy.AddMessage( "")
    arcpy.AddMessage( "_____________________________")
    arcpy.AddMessage( "Calculating H/L Cone: ")
    arcpy.AddMessage( "_____________________________")
    arcpy.AddMessage( "")
    arcpy.AddMessage( "Apex Choice is :" + apex_choice)
    arcpy.AddMessage( "")

    # variables for above and below the cone
    cone_gt_elev = "0"
    cone_lt_elev = "1"

# ====================================
# Apex_choice is either Maximum elevation,
# textfile elevation, or manually entered coordinate
# as apex of an H/L cone
# ====================================
# if maximum elevation, find it and store elevation, make const_g and cond_g
    if apex_choice == "Maximum_Elevation":

      arcpy.AddMessage("Searching for Maximum Elevation:")
      arcpy.AddMessage( "    ")
      currhipnt = arcpy.GetRasterProperties_management(Input_surface_raster,
"MAXIMUM")
      arcpy.AddMessage("Maximum Elevation Found: " + str(currhipnt))
      arcpy.AddMessage( "    ")

      arcpy.AddMessage("Creating grid with constant values of MAXIMUM Elevation:")
      arcpy.AddMessage( "    ")
      const_g = CreateConstantRaster(currhipnt, "FLOAT", Input_surface_raster,
Input_surface_raster)

      arcpy.AddMessage( "Creating grid with single data value at highest
elevation:")
      arcpy.AddMessage( "    ")
      cond_g = Con(Raster(Input_surface_raster) == const_g,const_g)
```

```
  if apex_choice == "XY_coordinate":
      coords = str(coordspnt)
      currhipnt = arcpy.GetCellValue_management(Input_surface_raster,coords,"")
      a = coords.split(' ')
      x = a[0]
      y = a[1]

      arcpy.AddMessage("Entered coordinates are: " + coords)
      arcpy.AddMessage("Elevation at that coordinate is: " + str(currhipnt))

      # make a point list
      onePointList = []
      onePointList = makepointlist(float(x),float(y),onePointList)

      arcpy.AddMessage("Creating grid with value of elevation at coordinate: ")
      arcpy.AddMessage( "    ")
      const_g = CreateConstantRaster(currhipnt, "FLOAT", Input_surface_raster,
Input_surface_raster)

      # Process: Extract By Points - cond_g
      cond_g = ExtractByPoints(Input_surface_raster,onePointList,"INSIDE")

    if apex_choice == "Maximum_Elevation" or apex_choice == "XY_coordinate":

 # ====================================
 # Create the cone
 # ====================================
      # Calculate Euclidean Distance
      arcpy.AddMessage( "Calculating Euclidean Distance of each cell from SELECTED
Location:")
      arcpy.AddMessage( "    ")
      eucdist_g = EucDistance(cond_g)

      # Multipy slope value by euclidean distance
      arcpy.AddMessage( "slope is: " + slope_value)
      arcpy.AddMessage( "Multiplying slope value times euclidean distance")
      arcpy.AddMessage( "    ")
      slopeeuc_g = Times(eucdist_g, float(slope_value))

      # Subtractions
      arcpy.AddMessage( "Processing values:")
      arcpy.AddMessage( "    ")
      hl_cone_g1 = Minus(const_g, slopeeuc_g)
      c_minus_dem = Minus(hl_cone_g1,Input_surface_raster)

      # LessThan and SetNull

      hl_cone_g2 = Con(c_minus_dem > 0, 1)
      hl_cone_g2.save(curdir + "\\" + "xhltemp")
```

```
 if apex_choice == "Textfile":
      interlist = []
      gridlist = []
      hipnts = []
      onePointList = []
      xstartpoint = []
      xstartpoints = []                # array to hold point
      dattype = "apex"
      xstartpoints = ConvertTxtToList(coordstxt,xstartpoints,dattype)

      arcpy.AddMessage("xstartpoints array is : " + str(xstartpoints))
      arcpy.AddMessage( "    ")
      numxstartpnts = len(xstartpoints)
      arcpy.AddMessage("number of points in array is : " + str(numxstartpnts))
      arcpy.AddMessage( "    ")

      for i in range(len(xstartpoints)):
        xstartpoint = xstartpoints[i]

        currx = float(xstartpoint[0])            # float X coord
        curry = float(xstartpoint[1])            # float Y coord
        coords = str(currx)+" "+str(curry)

        # make a point list

        onePointList = makepointlist(currx,curry,onePointList)

        # Get Cell Value at X,Y
        Result = arcpy.GetCellValue_management(Input_surface_raster,coords)
        currhipnt = Result.getOutput(0)
        hipnts.append(currhipnt)

    # ====================================
    # if textfile or manually entered coordinate, make const_g and cond_g
    # ====================================
    if apex_choice == "Textfile":
      for i in range(len(onePointList)):
        arcpy.AddMessage("Creating grid with value of elevation at coordinate: ")
        arcpy.AddMessage( "    ")
        const_g = CreateConstantRaster(hipnts[i], "FLOAT", Input_surface_raster,
Input_surface_raster)

        # Process: Extract By Points - cond_g
        cond_g = ExtractByPoints(Input_surface_raster,onePointList[i],"INSIDE")

        # ====================================
        # Create the cone
        # ====================================
        # Calculate Euclidean Distance
        arcpy.AddMessage( "Calculating Euclidean Distance of each cell from
SELECTED Location:")
        arcpy.AddMessage( "    ")
        eucdist_g = EucDistance(cond_g)
```

```
  # Multipy slope value by euclidean distance
  arcpy.AddMessage( "slope is: " + slope_value)
  arcpy.AddMessage( "Multiplying slope value times euclidean distance")
  arcpy.AddMessage( "    ")
  slopeeuc_g = Times(eucdist_g, float(slope_value))

  # Subtractions
  arcpy.AddMessage( "Processing values:")
  arcpy.AddMessage( "    ")
  hl_cone_g1 = Minus(const_g, slopeeuc_g)

  c_minus_dem = Minus(hl_cone_g1,Input_surface_raster)

  # LessThan and SetNull

  hl_cone_g2 = Con(c_minus_dem > 0, 1,0)
  hl_cone_g2.save(curdir + "\\" + "hl_cone_g2" + str(i))

  gridlist.append("hl_cone_g2" + str(i))

interlist.extend(gridlist)


if len(gridlist) > 1:
  arcpy.AddMessage( "    ")

  arcpy.CopyRaster_management(gridlist[0],"grid1")

  del gridlist[0]

  for i in range(len(gridlist)):
    temp = Con(Raster("grid1") > 0, Raster("grid1"),Raster(gridlist[i]))
    temp.save(curdir + "\\" + "temp")
    if arcpy.Exists(curdir + "\\" + "grid1"):
      arcpy.Delete_management(curdir + "\\" + "grid1")
    arcpy.CopyRaster_management("temp","grid1")
    if arcpy.Exists(curdir + "\\" + "temp"):
      arcpy.Delete_management(curdir + "\\" + "temp")
  if arcpy.Exists(curdir + "\\" + "grid1"):
    arcpy.CopyRaster_management("grid1","xhltempx")
    arcpy.Delete_management(curdir + "\\" + "grid1")
  for i in range(len(interlist)):
    if arcpy.Exists(curdir + "\\" + interlist[i]):
      arcpy.AddMessage( "Deleting:" + interlist[i])
      arcpy.Delete_management(curdir + "\\" + interlist[i])
else:
  if arcpy.Exists(curdir + "\\" + "hl_cone_g20"):
    arcpy.CopyRaster_management("hl_cone_g20","xhltempx")
    arcpy.Delete_management(curdir + "\\" + "hl_cone_g20")
if arcpy.Exists(curdir + "\\" + "xhltempx"):
  temp = Con(Raster("xhltempx") > 0, Raster("xhltempx"))
  temp.save(curdir + "\\" + "xhltemp")
  arcpy.Delete_management(curdir + "\\" + "xhltempx")
```

```
 # ====================================
 # convert the gridline of intersection of the cone and DEM to a polygon
 # ====================================
 # Raster to Polygon
    arcpy.AddMessage( "Converting cone/elevation intersection Raster to Polygon:")
    arcpy.AddMessage( "    ")
    arcpy.RasterToPolygon_conversion(curdir + "\\" + "xhltemp", curdir + "\\" +
"hl_cone_1.shp", "SIMPLIFY", "VALUE")
    arcpy.RasterToPolygon_conversion(curdir + "\\" + "xhltemp", curdir +
"\\laharz_shapefiles\\" + "hl_cone" + slopename + ".shp", "SIMPLIFY", "VALUE")

    # convert the polygon to a polyline
    # Polygon Feature To Line
    arcpy.AddMessage( "Converting Cone Polygon to Line:")
    arcpy.AddMessage( "    ")
    arcpy.FeatureToLine_management(curdir + "\\" + "hl_cone_1.shp", curdir + "\\" +
"hl_cone_2.shp", "", "ATTRIBUTES")

    # ====================================
    # convert the polyline back to a raster
    # ====================================
    # Polyline to Raster
    arcpy.AddMessage( "Converting Cone line back to a Cone outline Raster:")
    arcpy.AddMessage( "    ")
    arcpy.PolylineToRaster_conversion(curdir + "\\" + "hl_cone_2.shp", "GRIDCODE",
hlconename, "MAXIMUM_LENGTH", "NONE", Input_surface_raster)

    arcpy.AddMessage( "")
    arcpy.AddMessage( "_____________________________")
    arcpy.AddMessage( "Finding Intersection locations of streams")
    arcpy.AddMessage( "and H/L Cone and store coordinates as start points:")
    arcpy.AddMessage( "_____________________________")
    arcpy.AddMessage( "")

    # intersect_g
    arcpy.AddMessage( "Finding intersections of streams and hlcone:")
    arcpy.AddMessage( "    ")
    intersect_g = Plus(hlconename, Input_stream_raster)

    # Extract by Attributes
    arcpy.AddMessage( "Extracting intersections from raster:")
    arcpy.AddMessage( "    ")
    temp11c = ExtractByAttributes(intersect_g, "VALUE = 2")
    temp11c.save(curdir + "\\" + hlgridname)

    # Raster to Point
    arcpy.AddMessage( "Converting intersection locations to point shape file:")
    arcpy.AddMessage( "    ")
    arcpy.RasterToPoint_conversion(hlgridname, hlshapename, "VALUE")

    # Add X and Y coordinates to shape file
    arcpy.AddMessage( "adding X and Y coordinates to attribute table:")
    arcpy.AddMessage( "    ")
    arcpy.AddXY_management(hlshapename)
```

```
arcpy.AddMessage( "")
arcpy.AddMessage( "_____________________________________")
arcpy.AddMessage( "Writing X and Y locations of ")
arcpy.AddMessage( "intersections to a textfile:")
arcpy.AddMessage( "_____________________________________")
arcpy.AddMessage( "")

# ====================================
# open files for write, write heading to file of list of coordinates
# ====================================
runo = open(txfileuno, 'w')
report = open(txfile, 'w')

# set up search cursor
rows = arcpy.SearchCursor(hlshapename,"","","POINT_X; POINT_Y","")

# Get the first feature in the searchcursor
row = rows.next()

# local variables
currentloc = ""
count = 1

# Iterate through the rows in the cursor
while row:
  if currentloc != row.POINT_X:
    currentloc = row.POINT_X
  # write first X, Y to file
  if count == 1:
    runo.write(" %d,%d" % (row.POINT_X, row.POINT_Y))
  report.write(" %d,%d" % (row.POINT_X, row.POINT_Y))
  report.write("\n")
  row = rows.next()
  count += 1

# close files
report.close()
runo.close()

# report writing to files complete
arcpy.AddMessage( "")
arcpy.AddMessage( "File write complete...")
```

```
 # ====================================
 # Cleaning up intermediate grids and shape files
 # ====================================

    arcpy.AddMessage( "_____________________________")
    arcpy.AddMessage( "Cleaning up intermediate files...")
    arcpy.AddMessage( "_____________________________")
    if arcpy.Exists(curdir + "\\" + "const_g"):
      arcpy.Delete_management(curdir + "\\" + "const_g")
    if arcpy.Exists(curdir + "\\" + "above_g"):
      arcpy.Delete_management(curdir + "\\" + "above_g")
    if arcpy.Exists(curdir + "\\" + "c_minus_dem"):
      arcpy.Delete_management(curdir + "\\" + "c_minus_dem")
    if arcpy.Exists(curdir + "\\" + "notequ_g"):
      arcpy.Delete_management(curdir + "\\" + "notequ_g")
    if arcpy.Exists(curdir + "\\" + "cond_g"):
      arcpy.Delete_management(curdir + "\\" + "cond_g")
    if arcpy.Exists(curdir + "\\" + "Eucdist_g"):
      arcpy.Delete_management(curdir + "\\" + "Eucdist_g")
    if arcpy.Exists(curdir + "\\" + "Slopeeuc_g"):
      arcpy.Delete_management(curdir + "\\" + "Slopeeuc_g")
    if arcpy.Exists(curdir + "\\" + "hl_cone_g1"):
      arcpy.Delete_management(curdir + "\\" + "hl_cone_g1")
    if arcpy.Exists(curdir + "\\" + "hl_cone_g2"):
      arcpy.Delete_management(curdir + "\\" + "hl_cone_g2")
    if arcpy.Exists(curdir + "\\" + hlgridname):
      arcpy.Delete_management(curdir + "\\" + hlgridname)
    if arcpy.Exists(curdir + "\\" + "hl_cone.shp"):
      arcpy.Delete_management(curdir + "\\" + "hl_cone.shp")
    if arcpy.Exists(curdir + "\\" + "hl_cone_1.shp"):
      arcpy.Delete_management(curdir + "\\" + "hl_cone_1.shp")
    if arcpy.Exists(curdir + "\\" + "hl_cone_2.shp"):
      arcpy.Delete_management(curdir + "\\" + "hl_cone_2.shp")
    if arcpy.Exists(curdir + "\\" + "intersect_g"):
      arcpy.Delete_management(curdir + "\\" + "intersect_g")
    if arcpy.Exists(curdir + "\\" + "xhltemp"):
      arcpy.Delete_management(curdir + "\\" + "xhltemp")

    arcpy.AddMessage( "Processing Complete.")
    arcpy.AddMessage( "")


  except:
    arcpy.GetMessages(2)

if __name__ == "__main__":
  main()
```

distal_inundation.py

```
# ---------------------------------------------------------------------------
# distal_inundation.py
#
# Usage: laharz_py_main.py is attached to Laharz_py.tbx (toolbox)
#  sys.argv[1] a workspace
#  sys.argv[2] a DEM, input surface raster
#  sys.argv[3] name of the output .pts file (drainName)
#  sys.argv[4] text file storing the volumes
#  sys.argv[5] text file storing coordinates to start runs
#  sys.argv[6] flowType (lahar, debris_flow, rock_avalanche)
#
#
#  This program creates an estimate of area of potential inundation by a
hypothetical
# lahar, debris flow, or rock avalanche. Each planimetric areas stems from a single
input
# volume. The width and length of the planimetric area is governed by the
# cross sections calculated, centered at a stream cell. The calculations are
controlled
# by elevation values of cells from an input surface raster (DEM)
#
# ---------------------------------------------------------------------------

# Start Up - Import system modules
import sys, string, os, arcpy, math, time
from arcpy import env
from arcpy.sa import *
from math import *



# Check out license
arcpy.CheckOutExtension("Spatial")

#=============================================================================
# Local Functions
#=============================================================================

def ConvertTxtToList(atxtfilename,alist,dattype,conflim):
  # ====================================
  # Parameters:
  #  atxtfilename: name of a textfile
  #  alist: empty python list
  #  dattype: keyword 'volumes' or 'points' to determine how text file
  #       is manipulated
  #
  # Opens volume or starting coordinate textfile,
  # appends values read from textfile to the list,
  # volume list is sorted smallest to largest
  #
  # Returns: list
  # ====================================

  afile = open(atxtfilename, 'r')
```

```
for aline in afile:

    if conflim and dattype == 'volumes':
      x = aline.rstrip('\n')
      y = round(float(x.lstrip(' ')))
      alist.append(y)

    else:
      if aline.find(',') <> -1: # if it does have a ','
        x = aline.rstrip('\n')
        y = x.split(',')
        for i in range(len(y)):
          y[i] = round(float(y[i].lstrip(' ')))
        if dattype == 'volumes':

          y.sort()
          alist = y
        else:
          alist.append(y)
  afile.close()

  return alist

def CalcTime(tottime):
  # ====================================
  # Parameters:
  #  tottime: result of subtracting a start time from an
  #       end time
  #
  # Calculates the hours, minutes, seconds from a total
  # number of seconds.
  #
  # Returns: string of hours, minutes, seconds
  # ====================================

  timehr = 0
  timemin = round(tottime /60)
  timesec = tottime - (timemin * 60)
  if timemin > 60:
    timehr = round(timemin/60)
  if timehr > 0:
    timemin = timemin - (timehr * 60)
  if timehr > 0:
    return str(timehr) + " hrs, " + str(timemin) + " mins, " + str(timesec) + "
secs."
  else:
    return str(timemin) + " mins, " + str(timesec) + " secs."

def CalcArea(involllist,coeff,areaoutlist):
  # ====================================
  # Parameters:
  #  involllist: list of volumes
  #  coeff: coefficient based on lahar, debris flow, or rock avalanche
  #  areaoutlist: list of calculated planimetric or cross section areas
  #
  # Calculate cross section and planimetric areas
  #
  # Returns: list of calculated areas
  # ====================================
```

```
  for i in range(len(invollist)):
    areaoutlist.append(round((invollist[i] ** 0.666666666666666) * coeff))
  return areaoutlist

def CalcCellDimensions(dem):
  # ====================================
  # Parameters:
  #  dem: name of digital elevation model
  #
  # Extract the length of a cell in a DEM and
  # calculate the length of a cell diagonal
  #
  # Returns: value of cell width, value of cell diagonal
  # ====================================

  xxwide = arcpy.GetRasterProperties_management(dem,"CELLSIZEX")
  cwidth = float(xxwide.getOutput(0))
  tempdiag = math.sqrt((pow(cwidth,2) * 2))
  cdiag = round(tempdiag * 100) / 100

  return cwidth,cdiag

def StdErrModMean(ABpick,path,UserVol,confLim):
  # ====================================
  # Parameters:
  #  ABpick: 'A' cross section or 'B' planimetric areas
  #  path:  a path to workspace
  #  UserVol: user chosen volume
  #  confLim: level of confidence
  #
  #  calculates the confidence limits (upper and lower) depending on
  #  the user selected level of confidence. Calculates the volumes
  #  that correlate with those levels of confidence according to
  #  method outlined in accompanying text Appendix.
  #
  # Returns: two volumes, calculated from upper and lower confidence limits
  # ====================================

  #================================
  # Set intercepts according to A cross section or B planimetric area.
  # Set the appropriate textfile for statistics
  #================================
  if ABpick == 'A':
    anintercept = -1.301   # B + 2.301 # A - 1.301
    decintercept = 0.05    # B 200   #A 0.05
    txtfil = path + "laharz_textfiles\\" + "py_xxsecta.txt"
    arcpy.AddMessage("A - textfile:  " + str(txtfil))

  if ABpick == 'B':
    anintercept = 2.301   # B + 2.301 # A - 1.301
    decintercept = 200    # B 200   # A 0.05
    txtfil = path + "laharz_textfiles\\" + "py_xxplanb.txt"
    arcpy.AddMessage("B - textfile:  " + str(txtfil))

  afile = open(txtfil, 'r')


 #================================
```

```
# Initialize totlogv for use in
# standard error of mean
#==================================
rss = 0
totlogv = 0
count_n = 0 # number of data lines in file

for aline in afile:

  if aline.find(',') <> -1: # if it does have a ','
    x = aline.rstrip('\n')
    y = x.split(',')


  #==================================
  #  Loop to read file
  #  extract entries to calculate Mean of Log (V)
  #==================================
  aloc = y[0]
  avol = y[1]
  anarea = y[2]

  # convert the current volume read in file to a
  # log base 10 Volume (e.g. 10000000 => 7)
  logavol = log10(float(avol))

  # Add the log base 10 Volume to the sum totlogv
  # for standard error of the mean calculation
  # (i.e. Average Log V)used later
  totlogv = totlogv + logavol

  # Convert the current cross sectional area
  # (observation yi) to log base 10 Area
  logayi = log10(float(anarea))

  # Calculate the predicted cross sectional area as
  # shown in appendix and calculated for LaharZ
  logaypred = (logavol * 0.666666666667) + anintercept

  # Calculate the difference between measured (yi)
  # and predicted (ypred)
  # cross sectional areas then square the results
  sqdifyiypred = (logayi - logaypred) * (logayi - logaypred)

  # residual sum of squares => Sum e^2
  rss = rss + sqdifyiypred

  # update count_n => number of lines, i.e. number of observations
  count_n = count_n + 1

# Close file.
afile.close()
```

```
#==================================
# Calculate Residual Mean Square and then
# Calculate Standard Error of the Model
#==================================

 # variable count_n is the number of lines in the file, calculate n - 1
 nminusone = count_n - 1
 rms = rss / nminusone
 semodel = sqrt(rms)

 #==================================
 # Calculate Mean (Average) of Log (V)
 #==================================

 # calculate X bar used in Standard error of the mean
 # use the total of the log10 volumes and count of observations
 meanlogv = totlogv / count_n

 #==================================
 # open file for reading second time
 #==================================
 afile = open(txtfil, 'r')

 #==================================
 # variable meandiftotal is sum of
 # difference between each Log (V) and Mean Log (V)
 #==================================

 meandiftotal = 0.0
 oneovern = 1.0 / count_n
 nminusone = count_n - 1

 for aline in afile:
   if aline.find(',') <> -1: # if it does have a ','
     x = aline.rstrip('\n')
     y = x.split(',')

   #==================================
   # Calculate denominator of sum of square of
   # differences in standard error of the mean
   #==================================
   aloc = y[0]
   avol = y[1]
   anarea = y[2]

   # convert volume read in file to log base 10 Volume
   logavol = log10(float(avol))

   # difference between each converted Log (V) and
   # the Mean Log (V) calculated from first time
   # through the loop
   diflogvmean = logavol - meanlogv

   # Square the difference
   difsquared = diflogvmean * diflogvmean
```

```
  # add the current square of the difference between
  # observation and the mean of all observations to the
  # total to calculate the sum square of differences
  meandiftotal = meandiftotal + difsquared

# Close file.
afile.close

#==================================
# Get t-table file from user and open the file.
#==================================
txtfil = path + "laharz_textfiles\\" + "py_xxttabl.txt"

#==================================
# open t-table text file
#==================================
afile = open(txtfil, 'r')
count = 0

for aline in afile:
  if aline.find(',') <> -1: # if it does have a ','
    x = aline.rstrip('\n')
    y = x.split(',')
  linenum = y[0]
  cf50 = float(y[1])
  cf70 = float(y[2])
  cf80 = float(y[3])
  cf90 = float(y[4])
  cf95 = float(y[5])
  cf975 = float(y[6])
  cf99 = float(y[7])

  count = count + 1

  if count == nminusone:
    break

# Close file.
afile.close

#==================================
# Standard error of the mean
# SEm = s * SQRT( 1/n + (X* - Xmean)^2/sum(Xn - Xmean)^2)
# Value of s and sum(Xn - Xmean)^2 are completed
# Need to calculate X* - Xmean, difference between user
# selected X* and mean of X
#==================================

LogUserV = log10(float(UserVol))

UserApow = float(UserVol) ** 0.66666666666667

UserregressA = round(UserApow * decintercept)

# calculate difference of log UserV and log (V) mean
difmean = LogUserV - meanlogv
```

```
# square the result
difmeansq = difmean * difmean

#==================================
# Calculate the Standard Error of the Mean,
# SEm - Need to calculate for each user V
#==================================
sem = (semodel * sqrt(oneovern + (difmeansq / meandiftotal)))

sep = sqrt((semodel * semodel) + (sem * sem))

#==================================
# Calculate
# positive and negative confidence limits
#==================================
ypm50 = (cf50 * sep)
ypm70 = (cf70 * sep)
ypm80 = (cf80 * sep)
ypm90 = (cf90 * sep)
ypm95 = (cf95 * sep)
ypm975 = (cf975 * sep)
ypm99 = (cf99 * sep)

# positive values
pcfl50 = (ypm50 + log10(UserregressA))
pcfl70 = (ypm70 + log10(UserregressA))
pcfl80 = (ypm80 + log10(UserregressA))
pcfl90 = (ypm90 + log10(UserregressA))
pcfl95 = (ypm95 + log10(UserregressA))
pcfl975 = (ypm975 + log10(UserregressA))
pcfl99 = (ypm99 + log10(UserregressA))

# negative values
ncfl50 = (log10(UserregressA) - ypm50)
ncfl70 = (log10(UserregressA) - ypm70)
ncfl80 = (log10(UserregressA) - ypm80)
ncfl90 = (log10(UserregressA) - ypm90)
ncfl95 = (log10(UserregressA) - ypm95)
ncfl975 = (log10(UserregressA) - ypm975)
ncfl99 = (log10(UserregressA) - ypm99)


# list the positives in base 10
pos50a = 10 ** pcfl50
pos70a = 10 ** pcfl70
pos80a = 10 ** pcfl80
pos90a = 10 ** pcfl90
pos95a = 10 ** pcfl95
pos975a = 10 ** pcfl975
pos99a = 10 ** pcfl99
```

```
# list the negatives in base 10
  neg50a = 10 ** ncfl50
  neg70a = 10 ** ncfl70
  neg80a = 10 ** ncfl80
  neg90a = 10 ** ncfl90
  neg95a = 10 ** ncfl95
  neg975a = 10 ** ncfl975
  neg99a = 10 ** ncfl99
```

if ABpick == 'A':

```
    arcpy.AddMessage("Cross Section Areas base 10")
    if confLim == '50':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos50a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg50a))
      areaAup = pos50a
      areaAdn = neg50a
    if confLim == '70':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos70a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg70a))
      areaAup = pos70a
      areaAdn = neg70a
    if confLim == '80':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos80a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg80a))
      areaAup = pos80a
      areaAdn = neg80a
    if confLim == '90':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos90a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg90a))
      areaAup = pos90a
      areaAdn = neg90a
    if confLim == '95':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos95a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg95a))
      areaAup = pos95a
      areaAdn = neg95a
    if confLim == '975':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos975a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg975a))
      areaAup = pos975a
      areaAdn = neg975a
    if confLim == '99':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos99a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg99a))
      areaAup = pos99a
      areaAdn = neg99a

  if ABpick == 'B':
    arcpy.AddMessage("Planimetric Areas base 10")
    if confLim == '50':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos50a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg50a))
      areaBup = pos50a
      areaBdn = neg50a
```

```
    if confLim == '70':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos70a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg70a))
      areaBup = pos70a
      areaBdn = neg70a
    if confLim == '80':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos80a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg80a))
      areaBup = pos80a
      areaBdn = neg80a
    if confLim == '90':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos90a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg90a))
      areaBup = pos90a
      areaBdn = neg90a
    if confLim == '95':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos95a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg95a))
      areaBup = pos95a
      areaBdn = neg95a
    if confLim == '975':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos975a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg975a))
      areaBup = pos975a
      areaBdn = neg975a
    if confLim == '99':
      arcpy.AddMessage("Upper Area " + confLim + " = " + str(pos99a))
      arcpy.AddMessage("Lower Area " + confLim + " = " + str(neg99a))
      areaBup = pos99a
      areaBdn = neg99a

  if ABpick == 'A':
    return areaAup,areaAdn
  if ABpick == 'B':
    return areaBup,areaBdn


def WriteHeader(headr):

  # ====================================
  # Parameters:
  #  drainName: name of the current run(s)
  #
  # writes header to drainName.pts file
  # documents the volumes entered and areas calculated
  #
  # ====================================
  # lists for string output text file
  outstrvolumeList = []
  outstrxsectAreaList = []
  outstrplanAreaList = []

  str_volumeList = []
  str_xsectAreaList = []
  str_planAreaList = []
```

```
  # Get headr dictionary values
  drainName=headr['drainName']
  ptsfilename=headr['ptsfilename']
  volumeList=headr['volumeList']
  masterXsectList=headr['masterXsectList']
  masterPlanList=headr['masterPlanList']


  outfile = file(ptsfilename, "a")
  outfile.write("DRAINAGE NAME ENTERED: " + str(drainName) + "\n")
  outfile.write("VALUES SORTED LARGEST TO SMALLEST"+ "\n")
  outfile.write("VOLUMES ENTERED:"+ "\n")
  for i in range(len(volumeList)):
    if i+1 == len(volumeList):
      str_volumeList.append(str(volumeList[i]) + "\n")
    else:
      str_volumeList.append(str(volumeList[i]) + " : ")
  outstrvolumeList = ''.join(str_volumeList)
  outfile.write(outstrvolumeList)
  outfile.write("_____________________________________________________________________"+ "\n")
  outfile.write("")
  outfile.write('CROSS SECTION AREAS :'+ "\n")
  for i in range(len(masterXsectList)):
    if i+1 == len(masterXsectList):
      str_xsectAreaList.append(str(masterXsectList[i]) + "\n")
    else:
      str_xsectAreaList.append(str(masterXsectList[i]) + " : ")
  outstrxsectAreaList = ''.join(str_xsectAreaList)
  outfile.write(outstrxsectAreaList)
  outfile.write('PLANIMETRIC AREAS :'+ "\n")
  for i in range(len(masterPlanList)):
    if i+1 == len(masterPlanList):
      str_planAreaList.append(str(masterPlanList[i]) + "\n")
    else:
      str_planAreaList.append(str(masterPlanList[i]) + " : ")
  outstrplanAreaList = ''.join(str_planAreaList)
  outfile.write(outstrplanAreaList)
  outfile.write("_____________________________________________________________________"+ "\n")
  outfile.write("DECREASING PLANIMETRIC AREAS LISTED BELOW"+ "\n")
  outfile.write("_____________________________________________________________________"+ "\n")
  endtimewh = time.clock()

def AppendCurrPointToPointArrays(cellx,celly,currxarea,planvals,B):
  # ====================================
  # Parameters:
  #  cellx: X coordinate of the current cell
  #  celly: Y coordinate of the current cell
  #  currxarea: copy of xsectAreaList and created in the CalcCrossSection function
  #       xsectAreaList is a list of the calculated cross section areas
  #  planvals: starts as a list of 0's; one for each planimetric area
  #
  # Track planametric cells in B array on the fly as cross sections are constructed
  #
  # Returns: planvals, B
  # ====================================
  currxareaCount = len(currxarea) + 1 # number of xsect areas + 1, changes over
time
  demArrayValue = B[cellx,celly]   # Array value at current X, Y
```

```
  if demArrayValue == 1:       # 1 is background value
    B[cellx,celly] = currxareaCount # set B value to current number of cross
section values
    planvals[currxareaCount - 2] = planvals[currxareaCount - 2] + 1 # increase
appropriate planvals count by 1
  else:
    if demArrayValue < currxareaCount:
      B[cellx,celly] = currxareaCount # set B value to current number of cross
section values
      planvals[demArrayValue - 2] = planvals[demArrayValue - 2] - 1  # remove from
one planvals column
      planvals[currxareaCount - 2] = planvals[currxareaCount - 2] + 1 # add to
another planvals column

  return planvals,B

def
CheckForWindowBoundaries(cellx,celly,cellbuffx,cellbuffy,cellelev,wXmax,wXmin,wYmax
,wYmin,A):
  # ====================================
  # Parameters:
  #  cellx: X coordinate of the current cell
  #  celly: Y coordinate of the current cell
  #  cellbuffx: X coordinate of previous cell
  #  cellbuffy: Y coordinate of previous cell
  #  cellelev: DEM elevation at current cell
  #  wXmax,wXmin,wYmax,wYmin: boundaries of DEM array
  #
  # Check if current(new) XY from function GetNextSectionCell is outside DEM
boundaries
  # if it is, set current elevation to 99999.0 and return to previous values of
row,col;
  # if is not, get next cell elevation
  #
  # Returns: X coordinate, Y coordinate, cell elevation
  # ====================================

  if cellx < wXmin or cellx > wXmax or celly < wYmin or celly > wYmax:
    cellx = cellbuffx
    celly = cellbuffy
    cellelev = 99999.0

  else: # get next row,col elevationzzz
    cellelev = A[cellx,celly]

  return cellx,celly,cellelev

def
GetNextSectionCell(cellx,celly,cellelev,cellposneg,currFlowDir,wXmax,wXmin,wYmax,wY
min,A):
  # ====================================
  # Parameters:
  #  cellx: X coordinate of the current cell
  #  celly: Y coordinate of the current cell
  #  cellelev: DEM elevation at current cell
  #  cellposneg: negative or positive 1 indicating left or right direction
  #  currFlowDir: current flow direction
  #  wXmax,wXmin,wYmax,wYmin: boundaries of DEM array
  #
```

```
  # uses current flow direction to set the operator to get next cross section cell
  # stores the current XY in buffer variables, calls CheckForWindowBoundaries
  # to check if out of bounds
  #
  # Returns: X coordinate, Y coordinate, cell elevation

 # ====================================

  if currFlowDir == 1:
    rowoper = -1
    coloper = 0
  elif currFlowDir == 2:
    rowoper = -1
    coloper = 1
  elif currFlowDir == 4:
    rowoper = 0
    coloper = 1
  elif currFlowDir == 8:
    rowoper = 1
    coloper = 1
  elif currFlowDir == 16:
    rowoper = 1
    coloper = 0
  elif currFlowDir == 32:
    rowoper = 1
    coloper = -1
  elif currFlowDir == 64:
    rowoper = 0
    coloper = -1
  elif currFlowDir == 128:
    rowoper = -1
    coloper = -1
  else:
    print "Bad flow direction ", currFlowDir

  cellbuffx = cellx
  cellbuffy = celly
  cellx = cellx + cellposneg * rowoper
  celly = celly + cellposneg * coloper
  cellx,celly,cellelev =
CheckForWindowBoundaries(cellx,celly,cellbuffx,cellbuffy,cellelev,wXmax,wXmin,wYmax
,wYmin,A)

  return cellx,celly,cellelev

def Check4Pop(currxarea):
  # ====================================
  # Parameters:
  #  currxarea: copy of xsectAreaList and created in the CalcCrossSection function
  #       xsectAreaList is a list of the calculated cross section areas
  #
  # If currxarea list is longer than 1,
  # check each of the section areas to see if it is a negative value.
  # If so, delete item from list by popping
  #
  # Returns: currxarea
  # ====================================
```

```
  negcount = 0
  currlength = len(currxarea)
  if len(currxarea) > 1:
    for i in range(len(currxarea)):
      if currxarea[i] < 0:
        negcount += 1
  while negcount > 0 and currlength > 1:
    currxarea.pop()
    negcount -= 1
    currlength -= 1
  return currxarea

def CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B):

  # ====================================
  # Parameters:
  #  currFlowDir: current flow direction
  #  currRow: row of the current cell
  #  currCol: column of the current cell
  #  wXmax,wXmin,wYmax,wYmin: boundaries of DEM array
  #  planvals: starts as a list of 0's; one for each planimetric area
  #
  # Calculates cross sections for a single stream cell
  # sets variables including whether direction is ordinal or diagonal,
  # gets initial XY row column of left and right cells, gets elevations for
  # comparison, and sets fill level.
  #
  # Main Loop identifies which scenario applies to left and right cell comparison
  #  compare elevations equal to fill level
  #  compare elevations less than fill level
  #  compare equal elevations
  #  compare unequal elevations
  # calculates the cross section and subtracts planimetric cells from total,
  # sets new fill level, calls AppendCurrPointToPointArrays and GetNextSectionCell
  # store location and move to next cell
  # updates cell count as appropriate, if elevations are equal, moves both left and
right
  # cells. If elevation is 99999 stops CalcCrossSection
  # If currxarea is a list longer than 1, check each of the section areas to see if
it
  # is a negative value. If so, delete (pop) item from list
  #
  # Returns: planvals, B
  # ====================================

  # Get sectn dictionary values
  wXmax=sectn['wXmax']
  wXmin=sectn['wXmin']
  wYmax=sectn['wYmax']
  wYmin=sectn['wYmin']
  cellDiagonal=sectn['cellDiagonal']
  cellWidth=sectn['cellWidth']
  A=sectn['A']


  currxarea = []
  count = 0
  currxarea.extend(xsectAreaList) # make a copy"
```

```
  #============================================
  # set cell dimension according to current
  # flow direction--use integer value currFlowDir
  #============================================
  if currFlowDir == 8 or currFlowDir == 128 or currFlowDir == 2 or currFlowDir ==
32:
    cellDimen = cellDiagonal
  else: # currFlowDir == 1 or currFlowDir == 16 or currFlowDir == 4 or currFlowDir
== 64:
    cellDimen = cellWidth

  #============================================
  # set X,Y coordinates of current stream cell
  # as right cell -- facing downstream (flow direstion)
  #============================================

  cellrightx = currRow
  cellrighty = currCol

  #============================================
  # calculate X,Y coordinates of left stream
  # cell -- facing downstream based upon current
  # flow direction
  #============================================

  if currFlowDir == 1:
    celllleftx = currRow - 1
    celllefty = currCol
  elif currFlowDir == 2:
    celllleftx = currRow - 1
    celllefty = currCol + 1
  elif currFlowDir == 4:
    celllleftx = currRow
    celllefty = currCol + 1
  elif currFlowDir == 8:
    celllleftx = currRow + 1
    celllefty = currCol + 1
  elif currFlowDir == 16:
    celllleftx = currRow + 1
    celllefty = currCol
  elif currFlowDir == 32:
    celllleftx = currRow + 1
    celllefty = currCol - 1
  elif currFlowDir == 64:
    celllleftx = currRow
    celllefty = currCol - 1
  elif currFlowDir == 128:
    celllleftx = currRow - 1
    celllefty = currCol - 1
  else:
    print "Bad flow direction ", currFlowDir


  #============================================
  # get elevations of left and right cells
  #============================================

  celllleftelev = A[celllleftx,celllefty]
```

```
  cellrightelev = A[cellrightx,cellrighty]

  #============================================
  # set filllevel equal to the lower of
  # the left or right elevations
  #============================================

  filllevel = cellrightelev
  cellcount = 0
  cellnorm = 1
  cellwneg = -1

  #============================================
  #       Main Loop
  #============================================

  while count < 1000000000:

    if currxarea[0] < 0:
      break
    #============================================
    # compare elevations equal to fill level
    #============================================

    if celllefteleV == filllevel or cellrightelev == filllevel:

      if celllefteleV == filllevel:

        planvals,B =
AppendCurrPointToPointArrays(celllefteX,celllefty,currxarea,planvals,B)
        celllefteX,celllefty,celllefteleV =
GetNextSectionCell(celllefteX,celllefty,celllefteleV,cellnorm,currFlowDir,wXmax,wXmi
n,wYmax,wYmin,A)
      else: #cellrightelev = filllevel

        planvals,B =
AppendCurrPointToPointArrays(cellrightx,cellrighty,currxarea,planvals,B)
        cellrightx,cellrighty,cellrightelev =
GetNextSectionCell(cellrightx,cellrighty,cellrightelev,cellwneg,currFlowDir,wXmax,w
Xmin,wYmax,wYmin,A)
      cellcount += 1

    #============================================
    # compare elevations less than fill level
    #============================================

    elif cellrightelev < filllevel or celllefteleV < filllevel:

      if cellrightelev < filllevel:

        for i in range(len(currxarea)):
          currxarea[i] = currxarea[i] - ((filllevel - cellrightelev) * cellDimen)

        currxarea = Check4Pop(currxarea)
        if currxarea[0] > 0:
          planvals,B =
AppendCurrPointToPointArrays(cellrightx,cellrighty,currxarea,planvals,B)
```

```
          cellrightx,cellrighty,cellrightelev =
GetNextSectionCell(cellrightx,cellrighty,cellrightelev,cellwneg,currFlowDir,wXmax,w
Xmin,wYmax,wYmin,A)

      else: # cellleftelev < filllevel
        for i in range(len(currxarea)):
          currxarea[i] = currxarea[i] - ((filllevel - cellleftelev) * cellDimen)

        currxarea = Check4Pop(currxarea)
        if currxarea[0] > 0:
          planvals,B =
AppendCurrPointToPointArrays(cellleftx,celllefty,currxarea,planvals,B)
          cellleftx,celllefty,cellleftelev =
GetNextSectionCell(cellleftx,celllefty,cellleftelev,cellnorm,currFlowDir,wXmax,wXmi
n,wYmax,wYmin,A)
      cellcount += 1

    #============================================
    # compare equal elevations
    #============================================

    elif cellrightelev == cellleftelev:
      for i in range(len(currxarea)):
        currxarea[i] = currxarea[i] - ((cellrightelev - filllevel) * (cellDimen *
cellcount))

      currxarea = Check4Pop(currxarea)
      if currxarea[0] > 0:
        filllevel = cellrightelev
        #============================================
        # move left and right
        #============================================
        planvals,B =
AppendCurrPointToPointArrays(cellleftx,celllefty,currxarea,planvals,B)
        cellleftx,celllefty,cellleftelev =
GetNextSectionCell(cellleftx,celllefty,cellleftelev,cellnorm,currFlowDir,wXmax,wXmi
n,wYmax,wYmin,A)
        planvals,B =
AppendCurrPointToPointArrays(cellrightx,cellrighty,currxarea,planvals,B)
        cellrightx,cellrighty,cellrightelev =
GetNextSectionCell(cellrightx,cellrighty,cellrightelev,cellwneg,currFlowDir,wXmax,w
Xmin,wYmax,wYmin,A)
        cellcount = cellcount + 2

    #============================================
    # compare unequal elevations
    #============================================

    elif cellrightelev > cellleftelev or cellrightelev < cellleftelev:
      if cellrightelev > cellleftelev:

        for i in range(len(currxarea)):
          currxarea[i] = currxarea[i] - ((cellleftelev - filllevel) * (cellDimen *
cellcount))

        currxarea = Check4Pop(currxarea)
        if currxarea[0] > 0:
          filllevel = cellleftelev
```

```
          planvals,B =
AppendCurrPointToPointArrays(cellleftx,celllefty,currxarea,planvals,B)
          cellleftx,celllefty,cellleftelev =
GetNextSectionCell(cellleftx,celllefty,cellleftelev,cellnorm,currFlowDir,wXmax,wXmi
n,wYmax,wYmin,A)
      else: # cellleftelev > cellrightelev
        for i in range(len(currxarea)):
          currxarea[i] = currxarea[i] - ((cellrightelev - filllevel) * (cellDimen *
cellcount))

        currxarea = Check4Pop(currxarea)

        if currxarea[0] > 0:
          filllevel = cellrightelev
          planvals,B =
AppendCurrPointToPointArrays(cellrightx,cellrighty,currxarea,planvals,B)
          cellrightx,cellrighty,cellrightelev =
GetNextSectionCell(cellrightx,cellrighty,cellrightelev,cellwneg,currFlowDir,wXmax,w
Xmin,wYmax,wYmin,A)
      cellcount += 1


    #============================================
    # hit an edge
    #============================================
    if cellleftelev == 99999.0 or cellrightelev == 99999.0:
      for i in range(len(currxarea)):
        currxarea[i] = -99999

    #============================================
    # update count of time through the MAIN LOOP
    # stops at 1000000000
    #============================================
    count += 1

  return planvals,B

#============================================
# End Local Functions
#============================================


def main(workspace, Input_surface_raster, drainName, volumeTextFile,
coordsTextFile, flowType):

  for i in [1]:
    #============================================================================
    # Assign user inputs from menu to appropriate variables
    #============================================================================
    starttimetot = time.clock() # calculate time for program run
    tottime = 0.0
    arcpy.AddMessage("Parsing user inputs:")

    arcpy.env.workspace=workspace

    if flowType == 'Lahar' or flowType == 'Debris_Flow' or flowType ==
'Rock_Avalanche':
      flowType = flowType     # lahar, debris flow, rock avalanche
      conflim = False
```

```
  arcpy.AddMessage("Running Laharz_py")
else:
  confLimitChoice = flowType    # selected confidence limit
  conflim = True
  arcpy.AddMessage("Running Laharz_py with confidence limits")

#=========================================
# report dem selected back to user
#=========================================
arcpy.AddMessage( "_________ Input Values _________")
arcpy.AddMessage( 'Input Surface Raster Is:' + Input_surface_raster)

#=========================================
# report inputs back to user
#=========================================
arcpy.AddMessage("Volume textfile  :" + volumeTextFile)
arcpy.AddMessage("Starting coordinates file :" + coordsTextFile)
arcpy.AddMessage("Drainage identifier :" + drainName)

# ===================================
# report flowType back to user
# ===================================
if conflim:
  # ===================================
  # fix flowType as Lahar
  # ===================================
  flowType = 'Lahar'
  arcpy.AddMessage("Flow Type is Lahar")
else:
  if flowType == 'Lahar':
    arcpy.AddMessage("Lahar Selected")
  if flowType == 'Debris_Flow':
    arcpy.AddMessage("Debris Flow Selected")
  if flowType == 'Rock_Avalanche':
    arcpy.AddMessage("Rock Avalanche Selected")
arcpy.AddMessage( "_________ Paths on Disk _________")

#=========================================
# Set the ArcGIS environment settings
#=========================================
env.scratchWorkspace = env.workspace
env.extent = Input_surface_raster
env.snapRaster = Input_surface_raster
currentPath = env.workspace

#=========================================
# Set filenames and directories
#=========================================
BaseName = os.path.basename(Input_surface_raster)
BaseNameNum= len(BaseName)
PathName = env.workspace + "\\"

# if the filename suffix is "fill", get prefix name
if BaseName.endswith("fill"):
  PrefixName = BaseName.rstrip("fill")

# assign complete names for supplementary files,
# path and basename(prefix and suffix)
fillname = PathName + PrefixName + "fill"
```

```
dirname = PathName + PrefixName + "dir"
flacname = PathName + PrefixName + "flac"
strname = PathName + PrefixName + "str"

# assign partial names, prefix and suffix without path
pfillname = PrefixName + "fill"
pdirname = PrefixName + "dir"
pflacname = PrefixName + "flac"
pstrname = PrefixName + "str"

# report full names including path
arcpy.AddMessage("full path fill :" + fillname)
arcpy.AddMessage("full path dir  :" + dirname)
arcpy.AddMessage("full path flac :" + flacname)
arcpy.AddMessage("full path str  :" + strname)

# assign the flow direction and flow accumulation grids to variables
Input_direction_raster = dirname
Input_flowaccumulation_raster = flacname

# ====================================
#  Set up List variables
# ====================================

volumeList = []    # list of input volumes
xstartpoints = []   # coordinates of starting cell
xsectAreaList = []  # list of cross section areas
planAreaList = []   # list of planimetric areas
checkPlanExtent = [] # copy of list of planimetric areas
masterXsectList = [] # copy of list of cross section areas
masterPlanList = []  # copy of list of planimetric areas
masterVolumeList = [] # copy of list of input volumes

# string lists for output text file
str_volumeList = []
str_xsectAreaList = []
str_planAreaList = []


# ====================================
# Convert DEM to NumPyArray and
# get row, column values for boundaries
# ====================================
arcpy.AddMessage("_________ Creating DEM Array _________")
A = arcpy.RasterToNumPyArray(fillname)

# ====================================
#  Get NumPyArray Dimensions
# ====================================
arcpy.AddMessage("_________ Get NumPyArray Dimensions _________")

arcpy.AddMessage('Shape is: ' + str(A.shape) + " (rows, colums)")
number_rows = A.shape[0]
number_cols = A.shape[1]
arcpy.AddMessage('Number of rows is: ' + str(number_rows))
arcpy.AddMessage('Number of columns is: ' + str(number_cols))

#=====================================================
# Setthe Xmin, Xmax, Ymin, Ymax values for DEM boundaries
```

```
    #=========================================================
    arcpy.AddMessage("_________ Set Window Boundaries _________")

    wXmin = 0
    wXmax = number_rows - 1
    wYmin = 0
    wYmax = number_cols - 1

    arcpy.AddMessage( "wXmin (TOP): " + str(wXmin))
    arcpy.AddMessage( "wXmax (BOTTOM): " + str(wXmax))
    arcpy.AddMessage( "wYmin (LEFT): " + str(wYmin))
    arcpy.AddMessage( "wYmax (RIGHT): " + str(wYmax))

    # ====================================
    # Call ConvertTxtToList function with volumes and
    # starting point locations
    # ====================================

   arcpy.AddMessage( "_________ Convert Textfiles to Arrays _________")

    volumeList = ConvertTxtToList(volumeTextFile, volumeList, 'volumes', conflim)
    numvolumes = len(volumeList)

    if conflim and numvolumes > 1:
      vList = []
      vList = append.volumeList[0]
      volumeList = []
      volumeList = append.vList[0]
      del vList
    arcpy.AddMessage("Volume List is: " + str(volumeList))

    xstartpoints = ConvertTxtToList(coordsTextFile, xstartpoints, 'points',
conflim)
    numstartpts = len(xstartpoints)
    arcpy.AddMessage("Points entered: " + str(xstartpoints))

    # ====================================
    # call CalcArea function with parameters of list of volumes,
    # appropriate coefficents, and an empty list to store
    # calculated cross section or planimetric area values
    # ====================================

    if flowType == 'Lahar':
      xsectAreaList = CalcArea(volumeList,0.05,xsectAreaList)
      planAreaList = CalcArea(volumeList,200,planAreaList)
    if flowType == 'Debris_Flow':
      xsectAreaList = CalcArea(volumeList,0.1,xsectAreaList)
      planAreaList = CalcArea(volumeList,20,planAreaList)
    if flowType == 'Rock_Avalanche':
      xsectAreaList = CalcArea(volumeList,0.2,xsectAreaList)
      planAreaList = CalcArea(volumeList,20,planAreaList)

    if conflim:
      # ====================================
      # Calculate the max and min values for selected confidence limits
      # then add the cross section and planimetric areas to respective list
      # ====================================
      oneVolume = volumeList[0]
      XSArea1, XSArea3 = StdErrModMean('A',PathName,oneVolume,confLimitChoice)
```

```
      PlanArea1, PlanArea3 = StdErrModMean('B',PathName,oneVolume,confLimitChoice)

      xsectAreaList.append(round(XSArea1))
      xsectAreaList.append(round(XSArea3))
      planAreaList.append(round(PlanArea1))
      planAreaList.append(round(PlanArea3))
      xsectAreaList.sort() # sort
      planAreaList.sort() # sort


    arcpy.AddMessage("Cross Section Area List is: " + str(xsectAreaList))
    arcpy.AddMessage("Planimetric Area List is: " + str(planAreaList))

    # ====================================
    # order volumes, cross section and planimetric areas (large to small)
    # make copy of the planimetric area called checkPlanExtent
    # to store calculated area
    # make master copies of planimetric and cross section areas and volumes
    # ====================================

    volumeList.reverse()         # order volumes large to small
    xsectAreaList.reverse()       # order xsection areas list large to small
    planAreaList.reverse()        # order planimetric areas large to small
    checkPlanExtent.extend(planAreaList) # make copy of planAreaList
    masterPlanList.extend(planAreaList) # master copy of planimetric areas
    masterXsectList.extend(xsectAreaList)# master copy of cross section areas
    masterVolumeList.extend(volumeList) # master copy of volumes



    # ====================================
    # Call CalcCellDimensions function to get
    # cell width and diagonal cell length of DEM cells
    # get the lower left corner location for array output
    # ====================================

    cellWidth, cellDiagonal = CalcCellDimensions(pfillname)
    xxllx = arcpy.GetRasterProperties_management(pfillname,"LEFT")
    lowLeftX = float(xxllx.getOutput(0))
    xxlly = arcpy.GetRasterProperties_management(pfillname,"BOTTOM")
    lowLeftY = float(xxlly.getOutput(0))

    # initialize count and stop flag (boolean)
    cellTraverseCount = 0
    allStop = False

    # ====================================
    # Create starting point coordinate array
    # Convert current coordinates to Arcpy Point
    # and append to startCoordsList
    # ====================================

    startCoordsList = []
    for b in range(len(xstartpoints)):
      apoint = xstartpoints[b]  # get coordinates as list of first point
      currx = float(apoint[0])  # assign first value of coord list (X) of first
point as float
      curry = float(apoint[1])  # assign second value of coord list (Y) of first
point as float
```

```
      xonePoint = arcpy.Point(currx, curry)
      startCoordsList.append(xonePoint) # append current point to a list

    arcpy.AddMessage("_________ Creating startpts_g _________")

    if arcpy.Exists(currentPath + "\\" + "startpts_g"):
      arcpy.Delete_management(currentPath + "\\" + "startpts_g") # delete existing
startpts_g

    # ====================================
    # Use cell locations of startCoordsList to create a
    # temporary grid with cells having a value
    # of 0 at starting points and all other cells
    # having values of NODATA; use isnull function
    # to create grid having 0's at start locations
    # and the other cells having values of 1
    # creates startpts_g grid that stores locations
    # will convert to array, search for 0's then
    # store row, column to start runs
    # ====================================

    tmpStartPoints = ExtractByPoints(Input_surface_raster,startCoordsList,"INSIDE")

    isnull_result = IsNull(tmpStartPoints)
    isnull_result.save(currentPath + "\\" + "startpts_g") # create startpts_g
having values of 0 and 1


    # ====================================
    #  Convert startpts_g grid to NumPyArray
    # ====================================

    arcpy.AddMessage("_________ Creating Starting Points Array _________")
    B = arcpy.RasterToNumPyArray(currentPath + "\\" + "startpts_g")

    # ====================================
    #  Convert flow direction grid to NumPyArray
    # ====================================

    arcpy.AddMessage("_________ Creating Flow Direction Array _________")
    C = arcpy.RasterToNumPyArray(Input_direction_raster)

    # ====================================
    #  Get row, column of all starting cells
    # ====================================

    arcpy.AddMessage('Total rows : ' + str(number_rows))
    arcpy.AddMessage('Total columns : ' + str(number_cols))
    # set counters to zero
    i = 0
    j = 0
    zerosCoordsList = []
    foundPt = []
    while j < number_rows:
      for i in range(number_cols):
        if B[j,i] == 0:
         arcpy.AddMessage('Found the zero : '+ str(A[j,i]))
         FoundX = j
         FoundY = i
```

```
      foundPt.append(j)
      foundPt.append(i)
      zerosCoordsList.append(foundPt)
      foundPt = []
  j = j + 1
arcpy.AddMessage('found points: ' + str(zerosCoordsList))
for r in range(len(zerosCoordsList)):
  aStartPoint = zerosCoordsList[r]
  currRow = aStartPoint[0] #startX
  currCol = aStartPoint[1] #startY
  B[currRow,currCol] = 1 # Remove the 0's, entire array completely 1's

mergeList = []
# ====================================
#  Begin loop for list of rows, columns
# ====================================

blcount = 0
for r in range(len(zerosCoordsList)):
  # ====================================
  #  Intialize variables and lists for new run
  # ====================================

  blcount = blcount + 1

  cellTraverseCount = 0
  allStop = False

  str_volumeList = []
  str_xsectAreaList = []
  str_planAreaList = []

  xsectAreaList = []
  xsectAreaList.extend(masterXsectList)

  planAreaList = []
  planAreaList.extend(masterPlanList)

  checkPlanExtent = []
  checkPlanExtent.extend(masterPlanList) # make copy of planAreaList

  volumeList = []
  volumeList.extend(masterVolumeList)

  planvals = []
  for m in range(len(checkPlanExtent)):
    planvals.append(0)

  # ====================================
  # Load a row, column
  # ====================================

  aStartPoint = zerosCoordsList[r]

  currRow = aStartPoint[0] #startX
  currCol = aStartPoint[1] #startY

  currFlowDir = C[currRow,currCol]
  arcpy.AddMessage("Current flow direction: " + str(currFlowDir))
```

```
      # ====================================
      # call WriteHeader function for drainName.pts file
      # ====================================

      ptsfilename = currentPath+"\\"+str(drainName)+ str(blcount)+".pts"
      arcpy.AddMessage("Current name: " + str(ptsfilename))
      if not os.path.exists(ptsfilename):
        outfile = file(ptsfilename, "w")
        arcpy.AddMessage( "Textfile Created: " + ptsfilename)
      else:
        outfile = file(ptsfilename, "a")
        arcpy.AddMessage( "Textfile Exists: " + ptsfilename)
      arcpy.AddMessage("Calling writeheader with: " + str(drainName))

     # ====================================
     #  Set up Dictionaries
     # ====================================

      headr={}
      headr['drainName']=drainName
      headr['ptsfilename']= ptsfilename
      headr['volumeList']= volumeList
      headr['masterXsectList']=masterXsectList
      headr['masterPlanList']=masterPlanList

      #WriteHeader(drainName,ptsfilename,volumeList,masterXsectList,masterPlanList)
      WriteHeader(headr)

      sectn={}
      sectn['wXmax']=wXmax
      sectn['wXmin']=wXmin
      sectn['wYmax']=wYmax
      sectn['wYmin']=wYmin
      sectn['cellDiagonal']=cellDiagonal
      sectn['cellWidth']=cellWidth
      sectn['A']=A



      while not allStop:
        # ====================================
        # just in case of problems
        # ====================================
        if cellTraverseCount > 90000000:
          break

        # ========================================
        # Create cross sections in directions other
        # than the direction of stream flow
        # ========================================

        #arcpy.AddMessage("First cross section")
        planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)

        # ========================================
        # Store current flow direction,
```

```
        # change flow direction to construct sections
        # in other two possible directions
        # ========================================

        savedir = currFlowDir # store current flow direction
        # Calculate two cross sections for each flow direction
        if currFlowDir == 32:
          currFlowDir = 16
          # 1 of 2 Cardinal flow directions
          #arcpy.AddMessage("Second cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 64
          # 2 of 2 Cardinal flow directions
          #arcpy.AddMessage("Third cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
        if currFlowDir == 128:
          currFlowDir = 64
          # 1 of 2 Cardinal flow directions
          #arcpy.AddMessage("Second cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 1
          # 2 of 2 Cardinal flow directions
          #arcpy.AddMessage("Third cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
        if currFlowDir == 2:
          currFlowDir = 1
          # 1 of 2 Cardinal flow directions
          #arcpy.AddMessage("Second cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 4
          # 2 of 2 Cardinal flow directions
          #arcpy.AddMessage("Third cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
        if currFlowDir == 8:
          currFlowDir = 4
          # 1 of 2 Cardinal flow directions
          #arcpy.AddMessage("Second cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 16
          # 2 of 2 Cardinal flow directions
          #arcpy.AddMessage("Third cross section - ordinal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)


        if currFlowDir == 1: # or currFlowDir == 4 or currFlowDir == 16 or
currFlowDir == 64:
          currFlowDir = 128
          # 1 of 2 Diagonal flow directions
          #arcpy.AddMessage("Second cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
```

```
          currFlowDir = 2
          # 2 of 2 Diagonal flow directions
          #arcpy.AddMessage("Third cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
        if currFlowDir == 4: # or currFlowDir == 4 or currFlowDir == 16 or
currFlowDir == 64:
          currFlowDir = 2
          # 1 of 2 Diagonal flow directions
          #arcpy.AddMessage("Second cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 8
          # 2 of 2 Diagonal flow directions
          #arcpy.AddMessage("Third cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
        if currFlowDir == 16: # or currFlowDir == 4 or currFlowDir == 16 or
currFlowDir == 64:
          currFlowDir = 8
          # 1 of 2 Diagonal flow directions
          #arcpy.AddMessage("Second cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 32
          # 2 of 2 Diagonal flow directions
          #arcpy.AddMessage("Third cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
        if currFlowDir == 64: # or currFlowDir == 4 or currFlowDir == 16 or
currFlowDir == 64:
          currFlowDir = 32
          # 1 of 2 Diagonal flow directions
          #arcpy.AddMessage("Second cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)
          currFlowDir = 128
          # 2 of 2 Diagonal flow directions
          #arcpy.AddMessage("Third cross section - diagonal")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)

        currFlowDir = savedir # restore the saved flow direction

        # ========================================
        # checkerboard on diagonal - move to new X,Y,
        # make section and restore to original X,Y
        # ========================================

        if currFlowDir == 2 or currFlowDir == 8 or currFlowDir == 32 or currFlowDir
== 128:
          # Checkerboard flow direction
          savex = currRow # store current X coordinate
          savey = currCol # store current Y coordinate
          if currFlowDir == 8:
            # east
            currRow = currRow + 1
          elif currFlowDir == 32:
            # southeast
```

```
            currCol = currCol - 1
          elif currFlowDir == 128:
            # south
            currRow = currRow - 1
          elif currFlowDir == 2:
            # southwest
            currCol = currCol + 1
          #arcpy.AddMessage("Fourth cross section ")
          planvals,B =
CalcCrossSection(sectn,currFlowDir,currRow,currCol,planvals,xsectAreaList,B)

          currRow = savex  # restore X coordinate
          currCol = savey  # restore Y coordinate

        # ====================================
        # Check planimetric area to see if run
        # should stop
        # ====================================

        planvals.reverse()
        numz = 0
        temp_plan = []

        for i in range(len(planvals)):
          numz = planvals[i] + numz

          temp_plan.append(numz * cellWidth * cellWidth)

        temp_plan.reverse()

        for i in range(len(checkPlanExtent)):
          checkPlanExtent[i] = planAreaList[i] - temp_plan[i]


        planvals.reverse()

        # ==========================================
        # write the remaining planimetric areas to file
        # remaining area - checkPlanExtent[0] - [6] or
        # 7 simultaneous runs
        # ==========================================
        outfile = file(ptsfilename, "a")
        iwrite = len(checkPlanExtent)
        if iwrite == 1:
          outfile.write(str(checkPlanExtent[0])+ "\n")
        elif iwrite == 2:
          outfile.write(str(checkPlanExtent[0])+", "+str(checkPlanExtent[1])+ "\n")
        elif iwrite == 3:
          outfile.write(str(checkPlanExtent[0])+", "+str(checkPlanExtent[1])+",
"+str(checkPlanExtent[2])+ "\n")
        elif iwrite == 4:
          outfile.write(str(checkPlanExtent[0])+", "+str(checkPlanExtent[1])+",
"+str(checkPlanExtent[2])+", "+str(checkPlanExtent[3])+ "\n")
        elif iwrite == 5:
          outfile.write(str(checkPlanExtent[0])+", "+str(checkPlanExtent[1])+",
"+str(checkPlanExtent[2])+", "+str(checkPlanExtent[3])+",
"+str(checkPlanExtent[4])+ "\n")
        elif iwrite == 6:
```

```
          outfile.write(str(checkPlanExtent[0])+", "+str(checkPlanExtent[1])+",
"+str(checkPlanExtent[2])+", "+str(checkPlanExtent[3])+",
"+str(checkPlanExtent[4])+", "+str(checkPlanExtent[5])+ "\n")
        elif iwrite == 7:
          outfile.write(str(checkPlanExtent[0])+", "+str(checkPlanExtent[1])+",
"+str(checkPlanExtent[2])+", "+str(checkPlanExtent[3])+",
"+str(checkPlanExtent[4])+", "+str(checkPlanExtent[5])+",
"+str(checkPlanExtent[6])+ "\n")

        # ========================================
        # check for negative planimetric values
        # if so, delete (pop) them
        # ========================================

        pnegcount = 0
        plandiflength = len(checkPlanExtent)
        if plandiflength > 1:
          for i in range(len(checkPlanExtent)):
            if checkPlanExtent[i] < 0:
              pnegcount += 1
        if pnegcount > 0 and plandiflength > 1:
          #arcpy.AddMessage("Popping...")
          planAreaList.pop()
          xsectAreaList.pop()
          checkPlanExtent.pop()
          pnegcount -= 1
          plandiflength -= 1

        # ====================================
        # Stop if done
        # ====================================

        if checkPlanExtent[0] < 0:
          endtimetot = time.clock()
          tottime = endtimetot - starttimetot


          stringtime = CalcTime(tottime)

          outfile.write("TOTAL TIME: " + str(tottime)+ " seconds" + "\n")
          outfile.write("TOTAL TIME: " + stringtime + "\n")
          outfile.write("TOTAL CELLS TRAVERSED: " + str(cellTraverseCount)+ "
cells" + "\n")
          outfile.close()

          allStop = True


        # ========================================
        # This function changes coordinates to move
        # downstream to appropriate stream cell
        # ========================================

        if cellTraverseCount < 9000000:

          if currFlowDir == 1:
            # east
            currCol = currCol + 1
          elif currFlowDir == 2:
```

```
            # southeast
            currRow = currRow + 1
            currCol = currCol + 1
          elif currFlowDir == 4:
            # south
            currRow = currRow + 1
          elif currFlowDir == 8:
            # southwest
            currRow = currRow +1
            currCol = currCol - 1
          elif currFlowDir == 16:
            # west
            currCol = currCol - 1
          elif currFlowDir == 32:
            # northwest
            currRow = currRow - 1
            currCol = currCol - 1
          elif currFlowDir == 64:
            # north
            currRow = currRow - 1
          elif currFlowDir == 128:
            # northeast
            currRow = currRow - 1
            currCol = currCol + 1
          else:
            #print "Bad flow direction ", currFlowDir
            arcpy.AddMessage("Bad flow direction")
        else:
          # ====================================
          #  Stop if infinite loop
          # ====================================
          endtimetot = time.clock()
          tottime = endtimetot - starttimetot

          stringtime = CalcTime(tottime)

          outfile.write("TOTAL TIME: " + str(tottime)+ " seconds" + "\n")
          outfile.write("TOTAL TIME: " + stringtime + "\n")
          outfile.write("TOTAL CELLS TRAVERSED: " + str(cellTraverseCount)+ "
cells" + "\n")
          outfile.close()

          allStop = True


        # ========================================
        # Get new flow direction
        # ========================================
        currFlowDir = C[currRow,currCol]
        arcpy.AddMessage("New Flow Direction is: " + str(currFlowDir))

        cellTraverseCount += 1

        arcpy.AddMessage("_____________________________________")
        arcpy.AddMessage(" NUMBER OF STREAM CELLS TRAVERSED: " +
str(cellTraverseCount))

        arcpy.AddMessage("")
```

```
      if allStop == True:
        arcpy.AddMessage("_____________________________________________")
        arcpy.AddMessage("_________ ALL STOP IS:" + str(allStop))

      arcpy.AddMessage("_________ Creating Grid " + str(drainName) + str(blcount) +
" from Array _________")
      if arcpy.Exists(currentPath + "\\" + str(drainName) + str(blcount)):
        arcpy.Delete_management(currentPath + "\\" + str(drainName) + str(blcount))
# delete existing test_sect
      myRaster = arcpy.NumPyArrayToRaster(B,arcpy.Point(lowLeftX,
lowLeftY),cellWidth,cellWidth)
      myRaster.save(env.workspace + "\\" + str(drainName) + str(blcount))

      mergeList.append(str(drainName)+str(blcount))

      # ====================================
      #  Restore B array to all 1's
      # ====================================

      i = 0
      j = 0
      while j < number_rows:
        for i in range(number_cols):
          if B[j,i] > 1:
            B[j,i] = 1
        j = j + 1

    arcpy.AddMessage("...Processing Complete...")
    arcpy.AddMessage("TOTAL TIME: " + str(tottime) + " seconds")

    arcpy.AddMessage("List of the files created: " + str(mergeList))
    arcpy.AddMessage("Volumes entered: " + str(volumeList))

    arcpy.AddMessage("Number of volumes entered: " + str(numvolumes))

    del A
    del B
    del C

if __name__ == "__main__":
  from sys import argv
  main(argv[1], argv[2], argv[3], argv[4], argv[5], argv[6])
```

merge_runs.py

```
# ---------------------------------------------------------------------------
# merge_runs.py
#
# Usage: merge_runs.py is attached to Laharz_py.tbx (toolbox)
#  sys.argv[1] a workspace
#  sys.argv[2] a DEM, input surface raster
#  sys.argv[3] text file storing names of raster runs to merge
#  sys.argv[4] text file storing the volumes
#
#  This program will merge runs of the same volume from separate rasters.
#  The output is a raster containing cells for one volume from all runs at
#  a volcano.
# ---------------------------------------------------------------------------
```

```
# Start Up - Import system modules
import sys, string, os, arcpy, time
from arcpy import env
from arcpy.sa import *

starttimetot = time.clock() # calculate time for program run

# Check out license
arcpy.CheckOutExtension("Spatial")

#================================================================================
# Local Functions
#================================================================================

def ConvertTxtToList(atxtfilename,alist):
  # ====================================
  # Parameters:
  #  atxtfilename: name of a textfile
  #  alist: empty python list
  #
  # Opens volume or starting coordinate textfile,
  # appends values read from textfile to the list,
  # volume list is sorted smallest to largest
  #
  # Returns: list
  # ====================================

  afile = open(atxtfilename, 'r')

  for aline in afile:
    if aline.find(',') <> -1: # if it does have a ','
      x = aline.rstrip('\n')
      z = x.lstrip(' ')
      y = z.split(',')
      alist = y
  afile.close
  return alist
```

```
def main():
  try:
    #============================================================================
    # Assign user inputs from menu to appropriate variables
    #============================================================================

    arcpy.AddMessage("Parsing user inputs:")

    env.workspace = sys.argv[1]   # set the ArcGIS workspace
    Input_raster = sys.argv[2]   # name of DEM
    rasterTextFile = sys.argv[3]  # textfile of runs to merge
    volumesTextFile = sys.argv[4]  # textfile of volumes used to create runs


    #=============================================
    # Set the ArcGIS environment settings
    #=============================================
    env.scratchWorkspace = env.workspace # set ArcGIS scratchworkspace
    PathName = env.workspace + "\\"    # directory path
    rasterList = []            # empty list to store rasters
    volumeList = []            # empty list to store volumes



    # ====================================
    # Call ConvertTxtToList function with
    # rasters to merge and with volumns
    # ====================================

    arcpy.AddMessage( "________ Convert Textfile to List ________")

    rasterList = ConvertTxtToList(rasterTextFile, rasterList)
    numrasters = len(rasterList)
    rasterList.sort()

    volumeList = ConvertTxtToList(volumesTextFile, volumeList)
    numvolumes = len(volumeList)
    volumeList.sort()
    volumeList.reverse()

    # ====================================
    # Get the lower left cell coordinates
    # and cell size
    # ====================================

    xxcellsize = arcpy.GetRasterProperties_management(Input_raster,"CELLSIZEX")
    cellWidth = float(xxcellsize.getOutput(0))
    xxllx = arcpy.GetRasterProperties_management(Input_raster,"LEFT")
    lowLeftX = float(xxllx.getOutput(0))
    xxlly = arcpy.GetRasterProperties_management(Input_raster,"BOTTOM")
    lowLeftY = float(xxlly.getOutput(0))

    # ====================================
    # Convert DEM to NumPyArray and
    # get row, column values for boundaries
    # ====================================
    arcpy.AddMessage("_________ Convert DEM to Array _________")
    A = arcpy.RasterToNumPyArray(Input_raster)
```

```
# ====================================
#  Get NumPyArray Dimensions
# ====================================
arcpy.AddMessage("_________ Get Array Dimensions _________")

arcpy.AddMessage('Shape is: ' + str(A.shape) + " (rows, colums)")
number_rows = A.shape[0]
number_cols = A.shape[1]
arcpy.AddMessage('Number of rows is: ' + str(number_rows))
arcpy.AddMessage('Number of columns is: ' + str(number_cols))
del A

#=========================================
# For each volume in volume list
#=========================================
for x in range(len(volumeList)):
  z = x + 1
  w = x + 2
  s = 0
  t = 0
  A = arcpy.RasterToNumPyArray(Input_raster) # create numpyarray

  # ====================================
  #  Set array values to 1
  # ====================================
  while t < number_rows:
    for s in range(number_cols):
      A[t,s] = 1
    t = t + 1

  # ====================================
  # For each raster in the list of rasters
  # ====================================
  for r in range(len(rasterList)):
    B = arcpy.RasterToNumPyArray(rasterList[r]) # create numpyarray

    number_rowsi = B.shape[0]
    number_colsi = B.shape[1]

    i = 0
    j = 0

    while j < number_rowsi:
      for i in range(number_colsi):
        if B[j,i] > z:
          A[j,i] = w
      j = j + 1

    del B # delete numpyarray of rasters
    arcpy.AddMessage('Completed rasterlist number : ' + str(r+1))

  #================================
  # if raster already exists, delete it
  #================================
```

```
      currentname = PathName + "merge_" + str(w)
      if arcpy.Exists(currentname):
        arcpy.Delete_management(currentname)
      myRaster = arcpy.NumPyArrayToRaster(A,arcpy.Point(lowLeftX,
lowLeftY),cellWidth,cellWidth)
      # convert to integer raster
      outInt = Int(myRaster)
      outInt.save(currentname)
      # build vat for raster
      arcpy.BuildRasterAttributeTable_management(currentname)


      del A  # delete numpyarray of merged rasters

      arcpy.AddMessage('Completed merge of : ' + "merge_" + str(w))

    endtimetot = time.clock()
    tottime = endtimetot - starttimetot

    arcpy.AddMessage("...Processing Complete...")
    arcpy.AddMessage("TOTAL TIME: " + str(tottime) + " seconds")


  except:
    arcpy.GetMessages(2)

if __name__ == "__main__":
  main()
```

raster_to_shapefile.py

```
# ---------------------------------------------------------------------------
# raster_to_shapefile.py
#
# Usage: raster_to_shapefile.py is attached to Laharz_py.tbx (toolbox)
#  sys.argv[1] a workspace
#  sys.argv[2] name of new shapefile
#  sys.argv[3] name of existing raster data set
#
#  This program converts a raster data set of volumes into a polygon
# shapefile. The shapefile retains the coding from the raster, storing
# the information in an associated attribute table
# ---------------------------------------------------------------------------

# Start Up - Import system modules
import sys, string, os, arcpy, time
from arcpy import env
from arcpy.sa import *
#from math import *

starttimetot = time.clock() # calculate time for program run

# Check out license
arcpy.CheckOutExtension("Spatial")

def main():
  try:
    #=============================================================================
    # Assign user inputs from menu to appropriate variables
    #=============================================================================

    arcpy.AddMessage("Parsing user inputs:")

    env.workspace = sys.argv[1]   # set the ArcGIS workspace
    shapename = sys.argv[2]     # a name of new shapefile
    Raster_to_convert = sys.argv[3] # name of raster to convert

    #=============================================
    # Set the ArcGIS environment settings
    #=============================================
    env.scratchWorkspace = env.workspace # scratchworkspace
    curpath = env.workspace        # directory path
    PathName = env.workspace + "\\"    # set the path to files on disk

    #=========================
    # Set local variables
    #=========================
    inRaster = Raster_to_convert
    outPolygons = PathName + "laharz_shapefiles\\" + shapename + ".shp"
    field = "VALUE"

    #=========================
    # apply the conversion
    #=========================
    arcpy.RasterToPolygon_conversion(inRaster, outPolygons, "NO_SIMPLIFY", field)
    endtimetot = time.clock()
    tottime = endtimetot - starttimetot
```

```
    arcpy.AddMessage("...Processing Complete...")
    arcpy.AddMessage("TOTAL TIME: " + str(tottime) + " seconds")

  except:
    arcpy.GetMessages(2)

if __name__ == "__main__":
  main()
```