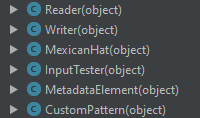
**Software description document for the article: *What is the shape of a tree? Detecting vegetation patterns by assessing similarities in high resolution elevation models***

1. **Instrument development**
2. **General description**

The accompanying software tool (available from the corresponding author) was developed in two variants, one in Esri's ArcGIS Platform and the other using only Opensource libraries. We used Python as the scripting language. The instrument’s back-end is common for both versions and consists of the following main processing tasks: raster to matrix conversion; Mexican Hat wavelet generation; 2D convolution; vector point data extraction from matrices; tangent calculation on matrices; output functions to shapefiles, rasters and tables. All these functions were written in arcpy (most of the ArcGIS version), gdal, pandas and scipy (the entire OpenSource version and various tasks in ArcGIS version). The front-end part was developed entirely different. For ArcGIS, we used the ArcGIS Desktop/ ArcGIS Pro framework only to configure the back-end script as a geoprocessing tool, and the instrument runs in that software, while for the OpenSource version, we used the Python library tkinter to build the front-end look from scratch. The ArcGIS version works on both Python 2 and 3, without the need to perform any script adjustments. The OpenSource version only works for Python 3 and can be easily deployed using the Conda environment created for this purpose. The output of the software differs according to the processing approach chosen but consists of CSV tables or raster files. A more detailed explanation in the developing of the instrument can be found in the additional document.

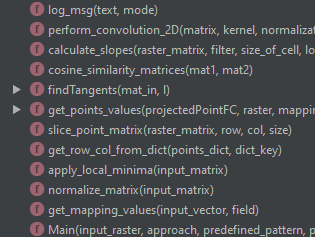
1. **Common core**

The common core of the tool has the following common Python classes and associated functions (as shown in the image below). The actual code can be seen and downloaded from the GitHub address.



* *Reader*: is tasked with reading the input rasters in many formats (.tif, .png, ESRI Geodatabase Raster, etc.) and converting them to a numpy array for faster processing. The ArcGIS tool uses *arcpy* for conversion, and the OpenSource one uses *gdal.*
* *Writer*: writes the numpy array on which certain filters were applied, back to a raster file. The ArcGIS tool uses *arcpy* for conversion, and the OpenSource one uses *gdal.*
* *MexicanHat*: is the class which generates Mexican Hat wavelet filters based on certain input parameters (sigma, d value, step site… see article).
* *InputTester*: class tasked with checking the input data for errors before the actual processing. This class checks the following:
  + The same spatial reference of input raster and point data;
  + Invalid values for d (matrix size);
  + Invalid values for Mexican Hat wavelet filter;
  + Single band raster for the elevation raster.
* *MetadataElement*: class which gathers metadata at runtime (input raster, processing parameters, output data), and sends it to the logging class;
* *CustomPattern*: class which reads input *csv* or *txt* files from an input folder, which contain square matrices of custom filters which can be applied on the input elevation raster;

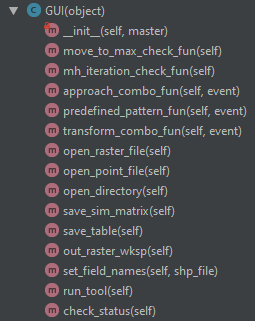
The common core also includes the following funcitons:



* *log\_msg*: logs a message to the Python standard output;
* *perform\_convolution\_2d*: performs a standard 2D convolution on the input matrices;
* *calculate\_slopes*: calculates the slope value of each cell in an input matrix;
* *cosine\_similarity\_matrices*: calculates the cosine similarity value on two input matrices;
* *findTangents*: function which is called when Transformations 3 or 4 are chosen on the input processing parameters;
* *get\_points\_values*: function which transforms the input point vector data in row/column data based on the raster extent;
* *slice\_point\_matrix*: slices a *dxd* array from the input raster, based on the *d* value, which defines the spatial scale;
* *get\_row\_col\_value\_from\_dict*: function to query a dictionary of data created from the input point vectors;
* *apply\_local\_minima*: function which applies a local minima transformation on the input matrix, according to Transformations 2 and 4;
* *normalize\_matrix*: normalizes an input matrix (divides each value to the norm of the matrix);
* *get\_mapping\_values*: function which maps an input vector data to a Python dictionary, for a faster access;
* *Main*: the main function which starts the tool;

1. **OpenSource GUI**

The Graphical User Interface (GUI) for the OpenSource version was built using the *tkinter* Python library. The code structure is shown in the image below:



The GUI class has functions that bind the common core classes and functions to it’s internal tkinter GUI methods. An auxiliary external function, *task\_check\_status* checks the GUI upon starting for errors in input, by binding the *InputTester* common class to the GUI class. Also, the OpenSource version has a *Logging Handler* class which handles the logging of information during runtime, and is tasked to stop the processing if input data errors or runtime errors are encountered. All standard output is written in a logfile when the tool is finished running. The class inherits the standard Python Logging class.

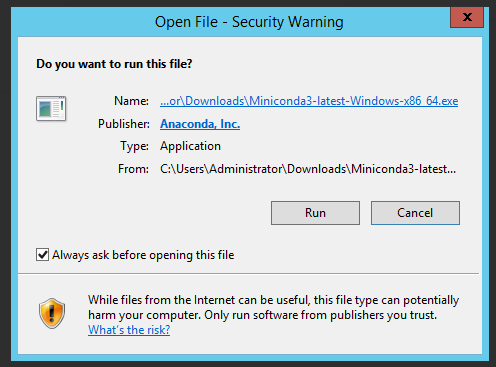
1. **ArcGIS GUI**

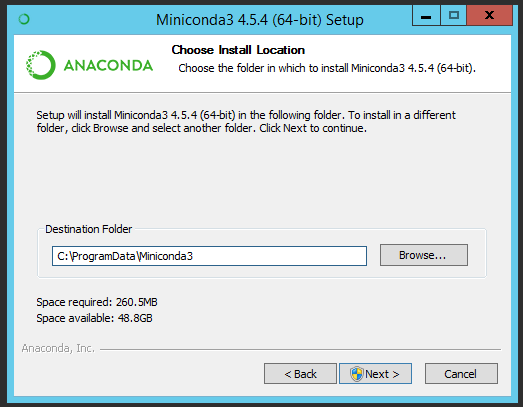
The ArcGIS GUI is built using the official Esri instructions found at the address: <https://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/basics/create-a-python-script-tool.htm>. The GUI is handled by ArcGIS Pro and ArcGIS Desktop. A separate Python script was developed to serve as a tool validation script, under the instructions at <https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/customizing-script-tool-behavior.htm>. The script is the equivalent of the OpenSource version function *task\_check\_status*.

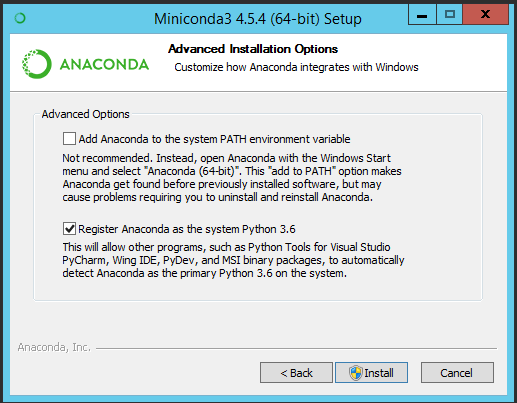
1. **Steps required to install the tool**
2. **Open-source version**
3. Download Miniconda for Python 3 x64 from the official website at <https://conda.io/miniconda.html>



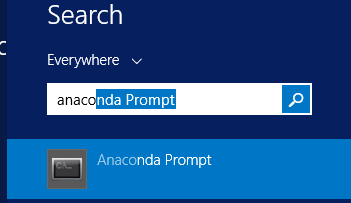
1. Install Miniconda on your machine



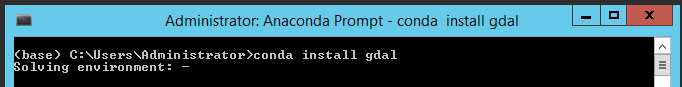


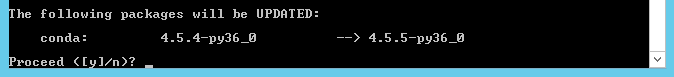


1. Open Anaconda Prompt

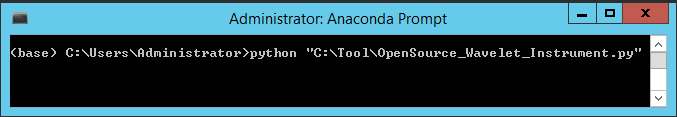


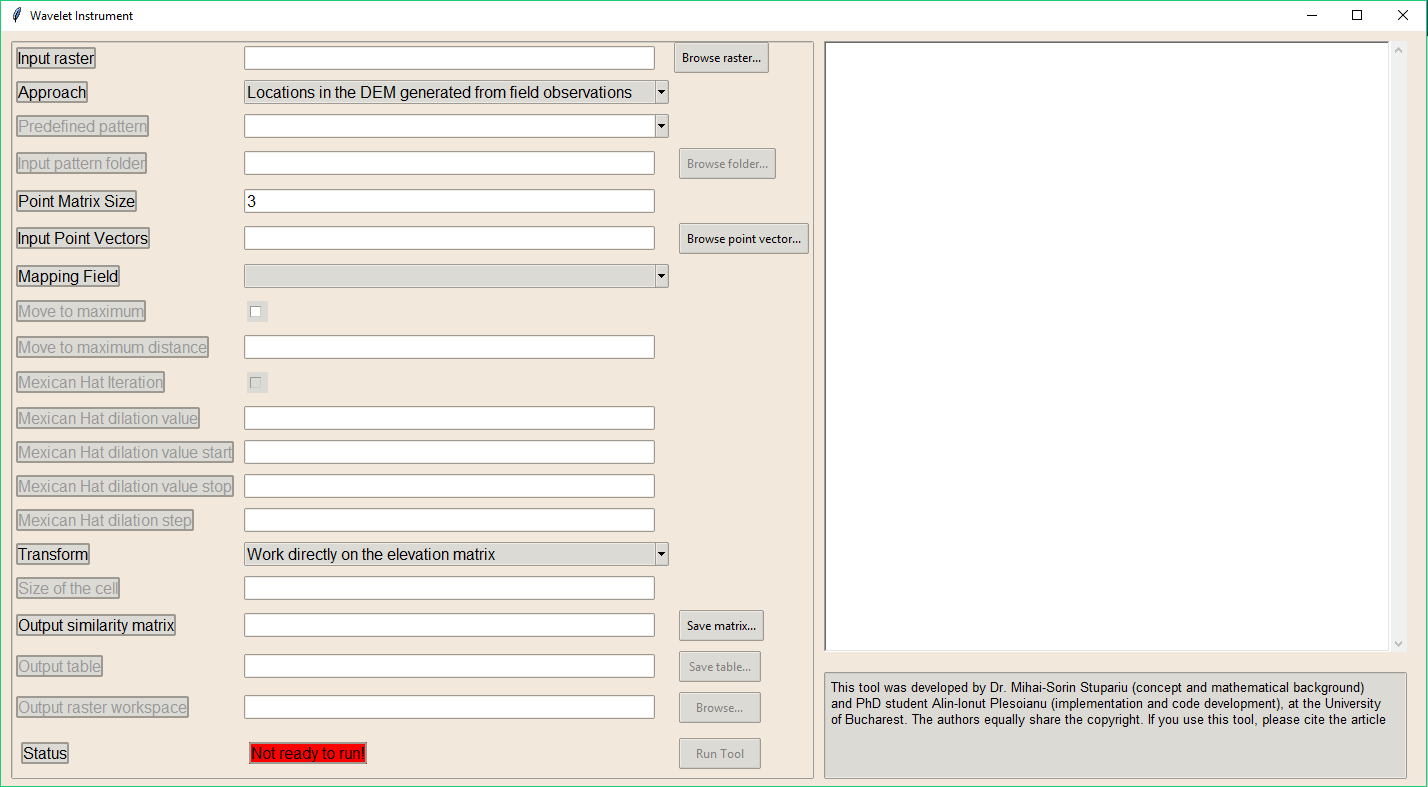
1. Use *conda* to install required packages (numpy, scipy, pandas, gdal)





1. Run the tool

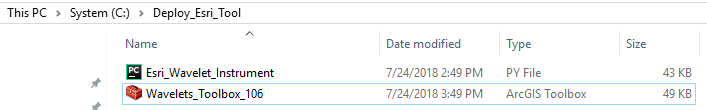




1. **Esri version**
2. Choose the correct tool version

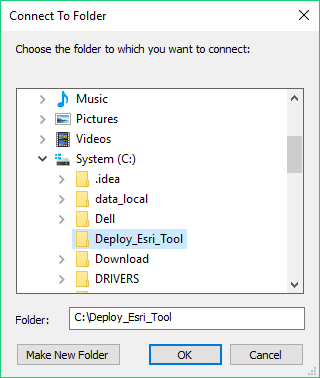
Depending on what version of ArcMap you are using, choose the corresponding tool version from the website.

1. Copy the toolbox *.zip* file to a local folder and unzip it

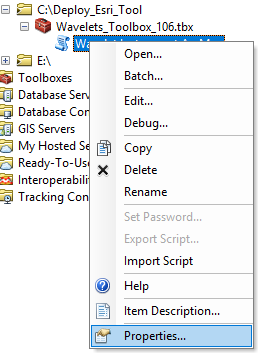


1. Connect to the folder using the Catalog window in ArcMap

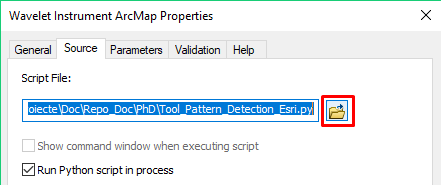


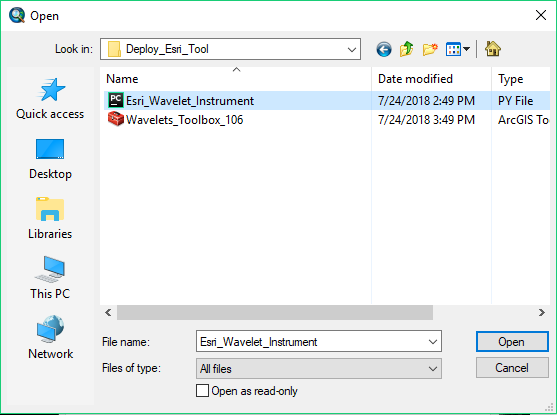


1. Open the properties of the *Wavelet Instrument ArcMap* tool



1. Change the path to the Python script

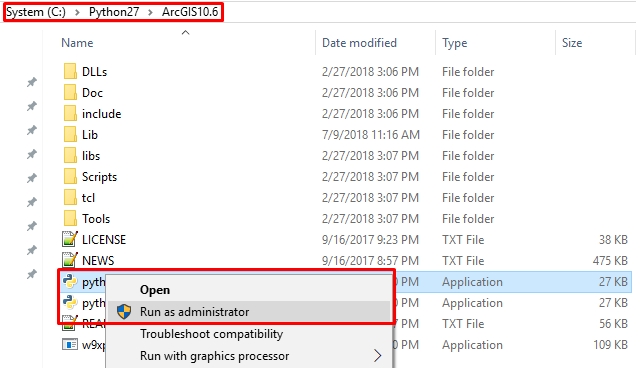




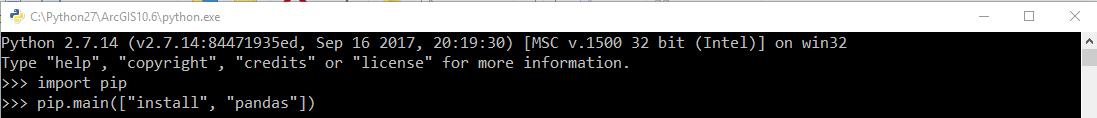
1. Install required Python libraries

6a. If you use ArcMap, the Python installation should be at the path *C:\Python27\ArcGIS10.X*, where X denotes the version (0 for 10.0, 1 for 10.1, etc…)

6b. Run *python.exe* as administrator



6c. Use *pip* to install the required Python libraries (numpy, scipy and pandas). First type *import pip*, then use *pip.main([“install”, “<package\_name>”])*, as in the example:



1. Run the script from the ArcMap Catalog window

