Guide to Ruff Surf Codes

Matthew Morriss
NSF Fellow Utah Geological Survey
PhD Candidate - University of Oregon

February 26, 2020

Abstract

This manual was written to document the code and functions compiled and put together by Matthew Morriss during his 6 month internship at the Utah Geological Survey. These codes are stored at the UGS GitHub page: SurfRuffFinder

1 Introduction

The majority of the code on this GitHub page is written to support two aims:

- Measure surface roughness, using **4** different routines
- Use a Receiver Operating Characteristic to predict areas that could be landslides

Outline The remainder of this document will cover the details related to each major Python script and function. Each function purpose will be outlined as well as inputs and outputs. The ideal workflow will also be described for someone approaching these tools as a complete novice.

All dependencies and additional python packages needed for this code to run properly is highlighted in the DocString of each function. I will not be listing them all here.

1. Example Scripts

- `Snow_basin_Roughness_Maps.py`
- `Snow_basin_Landslide_Maps.py`

2. Functions

   - `conv2_mexh.py`
   - `conv2_mexh_var.py`
   - `conv2_mexh2.py`
   - `STDS.py`
   - `RMSH.py`
   - `DCE_preprocess.py`
   - `DC_eig_par.py`
   - `ROC_plot.py`

2 Surface Roughness Methods

I used four different methods for measuring the surface roughness across the landscape. These methods are documented more fully in Berti et al. (2013) and references cited therein. Moreover, when it is published through the UGS, I will include in the GitHub repository a technical open-file-report that goes into more details on the methods.

Methods are listed below

1. Continuous wavelet transform (CWT; Booth et al., 2009)
2. Standard Deviation of Slope (STDS; Frankel and Dolan, 2007)
3. Root Mean Squared Height (RMSH; Shepard et al., 2001)
4. Directional Cosine Eigenvector (DCE; McKean and Roering, 2004)

3 Example Scripts

The GitHub repository has two good entry points, written as sepaate scripts, which call all of the necessary functions of measuring surface roughness and making predictive landslide maps: `Snow_basin_Roughness_Maps.py` and `Snow_basin_Landslide_Maps.py`. These functions are described in brief detail below. Much of this documentation is also included with the code in the form of a long DocString. As there is much overlap between these two scripts, I will only detail the more complex, the landslide mapping script in great detail.

3.1 `Snow_basin_Landslide_Maps.py`

This script was written to accomplish two tasks:

1. Measure surface roughness using 4 different methods
2. Loop through different moving window sizes for each method
3. Apply the Receiver Operating Characteristic to make a predictive map of landslides

INPUTS

- `sb_less_steep.tif` A 2 m dem, projected in UTM covering area near Snowbasin Ski area.
- `no_ls_patch4.tif` A patch of terrain not containing a landslide for testing with the CWT method
- `ls_patch_4.tif` A patch of terrain that covers a portion of the hummocky surface of a landslide
- `ls_boolean_boolean_map_match.tif` A map that covers the same area as the DEM above. This is essentially a digitized geologic map, wherein 0s are areas that are not mapped *a priori* as landslides and 1s are areas that *are* landslides.

OUTPUTS

For each method, there will be multiple output files of surface roughness, measured with different window sizes. These maps will cover the same area as the entire input DEM. There are also predictive maps of where landslides *should* be based on thresholding of roughness.

- Surface Roughness maps, labeled by method and window size

- Landslide maps, labeled by method and window size
- ROC curve for each window size, and a compiled plot of all ROC curves for all window sizes, for each method
- Excel spreadsheet for the area under the curve for each window size and the critical threshold (R_c) for roughness

3.2 `Snow_basin_Roughness_Maps.py`

This script was written to accomplish one task:

1. Measure surface roughness using 4 different methods, using the optimal window size for detecting landslides, as determined by ROC plot previously.

INPUTS

- `sb_less_steep.tif` A 2 m dem, projected in UTM covering area near Snowbasin Ski area.
- `no_ls_patch4.tif` A patch of terrain not containing a landslide for testing with the CWT method
- `ls_patch_4.tif` A patch of terrain that covers a portion of the hummocky surface of a landslide

OUTPUTS

For each method, there will be multiple output files of surface roughness, measured with a single window size. These maps will cover the same area as the entire input DEM.

- Surface Roughness maps, labeled by method and window size

4 Functions

This section details all of the supporting functions leveraged in the above scripts. More details and comments are available in the actual DocString within each function.

4.1 `conv2_mexh.py`

This function does the heavy lifting of:

1. Constructing a Mexican Hat Wavelet of a given size
2. Convolving that Mexican Hat Wavelet with a provided DEM

It is called from within the `conv2_mexh_var.py` function.

INPUTS

- patch - A full DEM or a piece of a DEM
- a - radius of a desire wavelet in terms of wavelet scale **S**, which is described in more detail in the main example scripts.
- dx - resolution of the DEM

OUTPUTS

- A DEM which has been convolved with a wavelet of a chosen scale. The new raster will give greater power to regions that emphasize the chosen wavelet scale.

4.2 `conv2_mexh_var.py`

This function calculates the wavelet spectra across a variety of chosen scales, provided by the user. This function is called in the earlier stages of spectral reconnaissance, when a user is trying to determine what wavelength has the greatest spectral difference between a landslide piece of terrain and non-landslide.

INPUTS

- patch - a piece of a DEM, landslide or not.
- scales - the wavelet scales over which the user wants to explore and determine how much of a difference in spectral power there is between the landslide patch and the non-landslide patch.
- dx - resolution of the DEM.

OUTPUTS

- Vcwt - an array of wavelet variance at different scales
- frq - array of frequencies over which variance is calculated
- wave - array of wavelengths (in m) over which variance is calculated.

4.3 `conv2_mexh2.py`

This is a stand-alone function used once the user has decided which wavelet scales they believe will do the best job of highlighting landslide terrain.

INPUTS

- patch - a whole DEM usually selected for processing
- a - wavelet scale chosen to filter DEM with
- dx = resolution of the DEM

OUTPUTS

- C - wavelet coefficients computed from the original DEM convolved with constructed wavelet.
- frq - frequency of wavelet at scale a
- wave - wavelength of wavelet (inverse of frequency)

4.4 `STDS.py`

This function is the first amplitude based measure of surface roughness examined herein. This function calculates the standard deviation of slope within a moving window across a provided DEM. The calculation is parallelized for the sake of computational efficiency.

INPUTS

- DEM - digital elevation model across which surface roughness will be calculated.
- w - window size
- cellsize - resolution of the DEM (in M)

OUTPUTS

- stdsGrid - surface roughness raster, covering same area as original DEM.

4.5 `RMSH.py`

This function measures surface roughness using the root mean squared height algorithm. This is accomplished in much the same way as the STDS algorithm, using a moving window in parallel.

INPUTS

- DEM - digital elevation model across whichi surface roughness will be calculated.
- w - window size over which the RMSH will be calculated

OUTPUTS

- RMSH grid - surface roughness raster, covering the same area as the original DEM.

4.6 `DCE_preprocess.py`

This function works in conjunction with the `DC_eig_par.py` function. In order to successfully parallelize the code in second function, some calculations had to be moved into a separate function call.

INPUTS

- DEM - digital elevation model across which surface roughness will be calculated.
- cellsize - resolution of the DEM (in M)
- w - size of moving window

OUTPUTS

- cx, cy, cz – directional cosines calculated in x, y, and z directions.

4.7 `DCE_eig_par.py`

This is the second part of the DCE method. This function calculates the eigen vector values within a moving window that scans the directional cosine grids.

INPUTS

- DEM - digital elevation model across which surface roughness will be calculated
- w - window size
- cx, cy, cz – directional cosines calculated in `DCE_preprocess.py`
- eps - measure of computational precision

OUTPUTS

- DCE grid - a grid of DCE calculated surface roughness.

4.8 `ROC_plot.py`

This function was written to calculate an ROC curve, comparing surface roughness to a boolean map of landslides (0 = not a landslide; 1 = mapped landslide).

INPUTS

- Roughness map (choose your method)
- lsRaster - boolean map of landslides over the exact same area as input roughness map and same grid size
- lowT - low threshold for roughness where algorithm starts (recommend checking values in a GIS)
- highT - high threshold for roughness where algorithm ends (recommend checking values from roughness map in a GIS)
- numDiv - number of divisions between high and low threshold

OUTPUTS The code outputs several figures and the necessary data to reconstruct an ROC plot

- FPr - false positive rate
- TPr - true positie rate
- lsMap - an array of potential landslides
- threshMax - critical threshold for identifying landslides.