Quick guide to ‘discontinuities_2D’

Jeff Crompton
email: jcrompto@sfu.ca

October 28, 2016

Contents

1 Acquiring the program and setting up the file system

All scripts and functions have been stored on github.com, so you will need to get a **git**[1] account and install **git bash**[2] on your computer. To run the code you will also need Matlab, which can be downloaded from the Mathworks [3] website using an Activation Key provided by SFU IT services [4]. Once downloaded, you will need to create a folder to store the code. As an example, I will work with the following parent directory: `D:\code\digitizeTraces`.To link this folder with git, and to download the code, type the following commands:

- Open the **git bash** shell and type '**cd** `D:\code\digitizeTraces`'
- Now that you are in the parent directory, initialize a git repository by typing **git init**
- Link to the repository by typing
  git remote add origin `https://github.com/jwheelsc/discontinuities_2D.git`
- Pull the code with the following two lines
  git fetch --all
  git pull origin master

All of the files should now be in your folder. In the way that the code was designed for sharing, the output data is written to the **output** folder, while all figures are written to the **figures** folder. As such, the figures and output files will be overwritten when running certain parts of the code. Once you have imported the image of choice into your working directory, here are three of many possible ways to work on a project:

- Use the same working directory for every project, and once finished with a project, manually export the figures and output files to a desired location.
- Create a new directory for every project, and copy the code into each of these project directories.
- Create a separate folder that will store output and figures for each project, and change the code so that the the figures and output are written directly to those locations. This is the recommended option, but requires some programming knowledge.

2 Tools for tracing

In all scripts and functions, there is text written in green font, which means that it is commented out by the `%` sign, and not being executed when the file is run. This text often contains useful information to explain the code, and can tell you how to alter the code to suit your needs. If something is not working in the code, you might get an error message in red font that will be output to the Matlab command line. If you read the error message, you may be able to debug the code yourself.

[1]https://github.com/
[2]https://git-scm.com/downloads
[3]https://www.mathworks.com/
[4] https://www.sfu.ca/itservices/technical/software/matlab/NamedUserLicense/NU1.html

2.1 Initializing a project

1. **Running the statup file:** Open and run the `statup.m` script in Matlab. A script can be run using Matlab's text editor GUI system by clicking the green **Run** button, pressing **F5**, or from the Matlab command line by typing `run startup.m`.

2. **Setting initial parameters:** Before you can start tracing, you will need to specify a few parameters such as window size, location, sub-window size, etc. Open `msfFunc.m`, and start by typing the name of your image into the single quotation marks for the variable labeled `image_name`. If you know the scale of the image in pixels per meter, then enter it into the variable labeled `pxpm`. If you do not yet know your scale, but know the distance of an object on the outcrop, see item 3. When tracing the discontinuities, you may want to work within a sub-window that is smaller than your entire window. To define the sub-window size in meters, enter a number for the variable labeled `ws`. If tracing within the sub-windows, then you will want to move from one sub-window to the next such that the ends of your traces can be seen be seen in neighbouring sub-windows. You can set the fraction of sub-window overlap by changing the fraction within the variable labeled `ol`.

3. **Acquiring a scale:** To acquire a scale in pixels per meter, open the script labeled `getScale.m`, and run the script. The image will open in Matlabs GUI image viewer. You will be prompted to zoom in to the portion of the image that contains your physical scale bar. To do so, click on the magnifying glass icon, zoom in, and press `F5`. You can then click on both ends of your object. A dialog box will appear, and you can enter the distance in meters, upon which a scale in pixels per meter will be output to the screen. Enter this scale into `msfcFunc.m` as described in item 2. This is a basic example of using one scale bar, but in reality there should be multiple scales throughout the window. I have also written a Matlab program to transfer a 3D scale from a `.ply` file to a 2D image using n points with `n!` scale bars, and this program can be distributed separately upon request.

4. **Setting the tracing window:** To select your tracing window, open and run `draw_and_plot_sets.m`. The first time you run this script, it will create an output file called `setsFile.mat`, which will store the cell array `allSets`. This array contains all of the traces that you digitize. If you want to restart your tracing, go into the `output` folder and delete the file. If you want to use the full image as your tracing window, then you do not need to input any window limits. If this is the case, open the `msfcFunc.m` file, and make sure that the square brackets are left empty for the `xlms` and `ylms` variables. To select a window within the image, use the magnifying glass tool and the hand-pan tool to zoom into your desired tracing location. Once you are satisfied with your window location, you can either select the coordinates manually using Matlab's coordinate selction tool, or if you are satisfied that the current viewing window also be your tracing window, then type `get(gca,'xlim')` and `get(gca,'ylim')` into the command line. Once you have obtained the coordinates, enter the lower and upper x and y coordinates into the square brackets in the `xlms` and `ylms` variables within the `msfcFunc.m` function (e.g. `xlms = [1050 2200]`, which are the coordinates in pixels).

2.2 Tracing the discontinuities

1. **Push Buttons:** Now you are ready to trace the discontinuities. There are push buttons and a drop down menu to select your drawing tools, but these can only be viewed by maximizing the image viewer. You can change the location of your buttons by playing with the multiplier for the `xButLoc` variable within the `draw_and_plot_sets.m` script.

2. **Creating traces:** To create a trace, click the `create trace` button. This will allow you to create traces indefinitely, so to stop the program from waiting for a trace to be drawn, click the `stop tracing` button, and make one last trace. If you forget to click the `stop tracing` button, a loop will be forever running within the program, waiting for a trace to be made. This could possibly lead to further complications within the program. You can also break the loop by clicking the cursor on the command line, and pressing `Ctrl+c`. One you have selected the `create trace` button, simply hold down the left mouse click and drag the cursor, or trace the line with a stylus pen on a touchscreen computer.

3. **Trace properties:** A trace will be stored automatically after it is drawn, but to get rid of the excess line connecting both trace ends, you can click the `draw set` button, and traces will reappear in the color of choice with each trace enumerated by its line number. To choose a trace color, text color, and text font, search for the traceColor, textColor, and textFont variables in the `draw_and_plot_sets.m` script, and enter your desired values. Examples of colors are white (`‘w’`), black (`‘k’`), yellow (`‘y’`), magenta (`‘m’`), green (`‘g’`), blue (`‘b’`), and red (`‘r’`). If you do not want to see the line numbers, then comment out the line below the comment `%%% LINE NUMBERS`. Currently, line numbers are displayed at the top and bottom of the traces.

4. **Connecting and deleting traces:** To connect lines across sub-windows, click the `append line` button, and enter in the number of the lines you wish to append. Lines can be removed in the same way by clicking the `delete line` button. Note that, while lines are deleted and appended automatically, you will not be able to view the change until you click the `draw set` button, which will show you the new line numbers and will not plot deleted lines.

5. **The viewing window:** The viewing window can be changed using the drop-down menu, or by using push buttons such as `move right` or `move down`. Once you click the `draw set` button, you will be returned to the tracing window, but if you want to return to the sub-window in which you were last working, click the `return window` button.

6. **Plotting fewer traces:** There may be point at which there are so many traces that it is taking an undesirably long time to plot the traces when the window is refreshed. You can choose to plot only the last n traces by uncommenting the line below the comment that reads `%%% TRACE ONLY A PORTION?` in `draw_and_plot_sets.m`. If so, you need to comment out the subsequent `for` statement.

7. **Plotting traces of a given length:** If you want to view traces that span a defined fraction of the sub-window length, set the `longTraces` variable equal to 1, and change the threshold trace length using the `XTR` variable (both in `draw_and_plot_sets.m`). Make sure the traces appear as a different color than all other traces. Be sure to use this part of the code if only plotting a fraction of the traces.

3 Cleaning and backing up the data

Before analyzing the data, you will need to remove portions of traces that are outside of your window, traces that are completely outside of your window, and traces with less than two points. To do so, run the `clean_set.m` script. There will be numrous dialog boxes verifying that the data cleaning did not delete all of your traces. At this point, it is important to note that if your `setsFile.mat` is altered or deleted by accident, then all of your work is gone. It is therefore recommended to commit regular backups to the file. **Git** is an excellent tool for doing so, but any form of backup is better than none.

4 Analyzing the data

To compute the statistics of the data, open and run `run_analysis.m`. The code is running through each of the scripts outlined below. Each script can be run on it's own, but the first time that `run_analysis.m` is executed, `findWindowSize` needs to be run first, and `analyze_sets.m` needs to be run before `computeFrequency.m`. Results are printed out to a `results.mat` file in the output folder. Most of these results are printed to a dialog box at the end of the analysis, but it is possible to view additional data by loading the data from `results.mat`. Details of how the analysis is carried out will be released in future documentation.

1. `findWindowSize:` Given the scale, this function simply computes the window size in meters based on a bounding box that is determined by the limits of the traces.

2. `windowEdges:` This script finds the traces that are within a threshold distance of the window edge for all edges. Change the variable labeled `critical_d` to change the threshold distance in meters. An output figure labeled `windowEdges` will be printed to the `figures` folder.

3. `sets_stats.m`: This script plots a histogram of the trace lengths as a probability density function, and saves a figure labeled `length_distribution` to the `figure` folder. The script also computes length distribution statistics as printed to the figure, and shows the corresponding log normal and exponential fits based on the computed statistics.

4. `analyze_set_intersections.m`: This script finds the locations of the intersection points, and the intersecting angle at each location. See document () section () to understand the limitations of this technique.

5. `analyze_sets.m`: This script finds the locations at which a scanline intersect a trace. The intersection points are output to `.mat` files, and later analyzed by `computeFrequency.m`. The scanlines rotate through one to 176 degrees in increments of 5 degrees. The increment can be changed through the variable `intaAng`, while the lower and upper limits can be changed through the `lowerA` and `upperA` variables, respectively. For visual purposes, the scan lines are plotted at each angle of rotation, but you can choose to suppress the figure output in the code. Before the code is run, a dialog box will ask you how many scanlines you want per window area. You will have to play around with this number depending on the characteristics of your data.

6. `computeFrequency.m`:. This script will compute the spacing between intersection points along a given scanline, and plot the frequency as a function of scanline angle.

The figure will be labeled `scanline_angle` and printed to the `figures` folder. If the frequency versus scanline curve is really coarse, consider increasing the number of scanlines per window area. You may also want to decrease the increment angle between scanlines.

5 Concluding remarks

There is an additional bit of code that computes the true set spacing of underlying sets, but the program assumes that the total spacing is a function of only two underlying sets separated by some fixed angle (β). The analysis has many limitations (document () section ()), and can be distributed upon request. Lastly, I am happy to share this, but given the copyright policies set at SFU, it is not to be used outside of academic purposes without further consent.