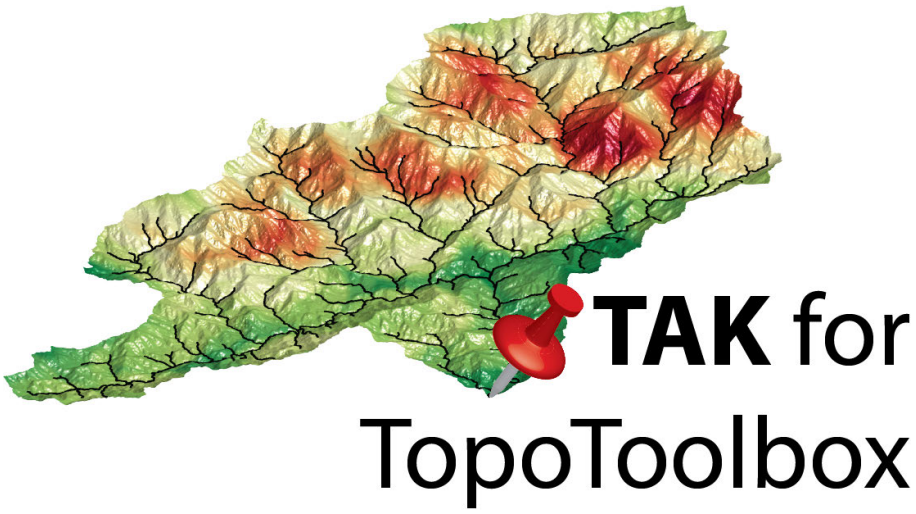
Topographic Analysis Kit (TAK) Manual v1.0

Adam M. Forte[1] and Kelin X. Whipple[2]

[1]Department of Geology and Geophysics, Louisiana State University, Baton Rouge, LA
[2]School of Earth and Space Exploration, Arizona State University, Tempe, AZ

July 9, 2018



Contents

1 Attribution

If you use or modify TAK functions for use in a publication, please cite the main TAK publication:

- Adam M. Forte and Kelin X. Whipple. Short Communication: The Topographic Analysis Kit (TAK) for TopoToolbox. *Earth Surface Dynamics*, in review. doi: 10.5194/esurf-2018-57

Also, please cite the original TopoToolbox publications as TAK could not function without TopoToolbox:

- Wolfgang Schwanghart and Nikolaus J. Kuhn. TopoToolbox: A set of Matlab functions for topographic analysis. *Environmental Modelling and Software*, 25(6):770–781, 2010. ISSN 13648152. doi: 10.1016/j.envsoft.2009.12.002
- Wolfgang Schwanghart and Dirk Scherler. Short Communication: TopoToolbox 2 - MATLAB based software for topographic analysis and modeling in Earth surface sciences. *Earth Surface Dynamics*, 2:1–7, 2014. doi: 10.5194/esurf-2-1-2014

2 Download and Install

2.1 Matlab Functions

The TAK functions were written (and periodically updated) to work with the most up to date version of TopoToolbox, this can be downloaded from Wolfgang Schwanghart's GitHub page. The TAK functions are available from Adam Forte's GitHub page. Both the TopoToolbox and TAK functions (and all of their subfolders) must be on your matlab path for the functions to work properly.

A suggested strategy for both the TopoToolbox and TAK functions is to 'fork' copies of these repositories and use these forked versions on your local machine. Periodically, you can sync your fork with the master of the original repository to merge in any changes that have been made to either TopoToolbox or TAK. If you are uncomfortable with the command line, this can also be done on the web version of GitHub by 'comparing' branches (for updating TopoToolbox, make sure the 'head fork' is wschwanghart/topotoolbox and the 'base fork' is your forked version of TopoToolbox, follow a similar procedure for TAK), click on the 'Create pull request' button, give the pull request a name in the 'Title' box (the new 'Create pull request' button will be grayed out until you provide a name) click 'Create pull request', click 'Merge pull request', and then finally click 'Confirm merge'. Your forked version will now be up to date with the master of TopoToolbox or TAK.

2.2 Compiled Functions

In attempt to make these functions accessible to a wider range of users, we have also produced compiled versions of these functions that do not require Matlab to run. Within the main repository, there is a folder called 'Compiled_Versions' that contains three folders associated with the compiled versions, 'cmpMfiles', 'Windows', and 'MacOSx'. The 'cmpMfiles' folder contains modified versions of all the matlab functions that were compiled, for reference (you will likely need to look at these to understand the inputs to the compiled functions). The compiled versions all have a 'cmp' prefix on them to differentiate them from the main functions (these are all functional in Matlab on their own, though it's not generally recommended that you use these in Matlab over their main counterparts), but as described below, when calling a function from the command line of the compiled TAK, you use the name of the desired function without the 'cmp' prefix. The master function is 'TAK.m' (for which there is not have a comparable function in the non-compiled versions). This master function will not successfully run in Matlab as it is designed to translate

inputs from the command line to the other functions in a way that works only when used in a deployed (i.e. compiled) mode. The 'Windows' and 'MacOSX' folders each contain an executable named *InstallTAKandMatlabRuntime* (with either an .exe or .app extension depending on the OS). Double clicking this executable will install the Matlab runtime environment that is necessary to run compiled matlab code and the executable for TAK. To keep the size of this install file manageable, this executable needs to access the internet to download the Matlab Runtime. Additionally, you will likely need to have administrator privileges on your local machine in order to successfully install Matlab Runtime. If you have previously installed the Matlab runtime environment, versions of just the TAK executable can be found in the 'Files_Only' folder.

Compiled TAK functions are designed to be run from the command line (i.e. cmd prompt on Windows, terminal on Mac OS X). The procedure for this differs between Windows and Mac OS X and is described in the readme that appears along with the executables. After installing Matlab Runtime, on Windows, open a command prompt and navigate to the location of the TAK executable and then simply call TAK:

```
TAK arguments
```

The procedure is a little more complicated on Mac OS X, here there is a shell script to run the application but it requires that you specify the location of the Matlab Runtime environment. For example, after installing Matlab Runtime and navigating to the location of the TAK executable and this shell script (with a .sh file extension) in a terminal window:

```
./run_TAK.sh /location/of/runtime arguments
```

As described in the readme associated with the Mac version, you can avoid having to give the location of the Matlab Runtime environment every time you run TAK by permanently setting the DYLD_LIBRARY_PATH location in your path.

Compiled TAK expects that the first argument (after whatever is required to call the main TAK function depending on the operating system you are using) will be the name of the function you wish to use and the second argument will be the **full path** (it really does need to be the full path) to your working directory. TAK expects that all input data that you reference are stored in this working directory and it will store all output data in this working directory (or folders created within this working directory). Generally, the compiled versions of the functions take the same input as the matlab functions described in detail in later sections, but in some cases, because of the limitations of the style of inputs allowed, these inputs are slightly different (e.g. functions require a text file input instead of a cell array input, etc). To understand the inputs to the compiled functions, please refer to the headers of the relevant codes stored in the cmpMfiles folder.

It is important to note that the form of inputs are also different at the command line. In short, no input should be included in quotes regardless of the input type and each input should be separated by a single space. For example to call the compiled version of *MakeStreams* on Mac OS X:

```
./run_TAK.sh /location/of/runtime MakeStreams /location/of/working/directory
    dem.tif 1e6 topo_files no_data_exp auto
```

where 'MakeStreams' is the name of the function you want to use, '/location/of/working/directory' is the working directory where input data will be read from and output data will be stored, 'dem.tif' is the first required input of MakeStreams and is the name of the dem file to import, '1e6' is the second required input of MakeStreams and is the threshold accumulation area to use, 'topo_files' is the third required input of MakeStreams and is the name for output files (note that this is NOT a required input for the non-compiled version of MakeStreams), 'no_data_exp' is the name of an optional parameter for MakeStreams, and 'auto' is the value being passed to that optional parameter. Some standards for the inputs; 1) when the function requires that you provide a name of an input file, it expects that

the file includes the file extension (e.g. .txt, .mat, .shp, etc) because it will use this file extension to make sure you're providing the correct kind of file, 2) any text file input is expected to have a .txt extension, 3) theoretically comma or tab delimited data are both fine for text file inputs, but in practice, comma delimited files are preferred, and 4) when the function requires (or allows as an optional input) the name of an output file, DO NOT include a file extension as this name will likely be used for several different files with different extensions and the function will handle appending the proper file extensions. The header of 'cmp*.m' files for each function contain example inputs to the command line to run that function.

The compiled versions are designed to mostly behave exactly the same as the non-compiled functions running Matlab with a few minor exceptions (e.g. the compiled versions of *KsnProfiler* and *BasinPicker* do not allow for you to zoom into a location on the DEM before you start picking). All function names described in later sections are valid functions to invoke in the compiled version of TAK except *ksncolor*, *CatPoly2GRIDobj*, or *ProjectOntoSwath*. Also note that two functions exist in compiled TAK which do not in the regular versions. These functions, called as *PrepareAddGrids* and *PrepareCatAddGrids* at the command lines, are designed to prepare inputs for additional grids and additional categorical grids, respectively, for use in the compiled version of *ProcessRiverBasins*. Refer to the 'cmp' files for these two functions ('cmpPrepareAddGrids.m' and 'cmpPrepareCatAddGrids.m') for instructions for use.

3 Preparing Datasets for TAK

TopoToolbox and TAK functions are designed assuming that data is supplied to them are in a projected coordinate system with meters as linear units (e.g. UTM) and will produce unexpected results (or may error out) if you do not reproject your data into a suitable projection and coordinate system. Similarly, for functions that take multiple datasets as inputs (e.g. *ProcessRiverBasins*) it is expected that all of the datasets your provide are in the same projection system. We would generally recommend doing the reprojections in a GIS environment (e.g. ArcGIS, QGIS, GDAL, etc), but TopoToolbox does include some functions to reproject data (e.g. *reproject2utm*, *projectGRIDobj*).

It's important to note that projected data (e.g. geotiffs, ascii grids, shapefiles, etc) exported from Matlab will not include georeferencing information. The data is still projected (in the same projection as the original input data), but you will need to define the projection in the GIS program you are using.

In this this manual, we use an example dataset from Southern California to demonstrate outputs of some of the functions included as part of TAK (Figure 1). The example data is available within the released version of the TAK code on GitHub and is contained within a tarball called *'ExampleData.tar.gz'*. Within this tarball, you'll find:

- *SoCal_UTM_DEM.tif* - GeoTiff of the example data area shown in Figure 1 from Open Topography and reprojected into UTM 11N
- *prism_precip.tif* - GeoTiff of 30 year normals of precipitation from PRISM, data was cropped and reprojected into UTM 11N
- *geo_polygons.shp* - Polygon shapefile of Geology of California from California Department of Conservation, data was reprojected into UTM 11N
- *river_mouths.txt* - text file containing the river mouths used to generate basins in the example using *ProcessRiverBasins*

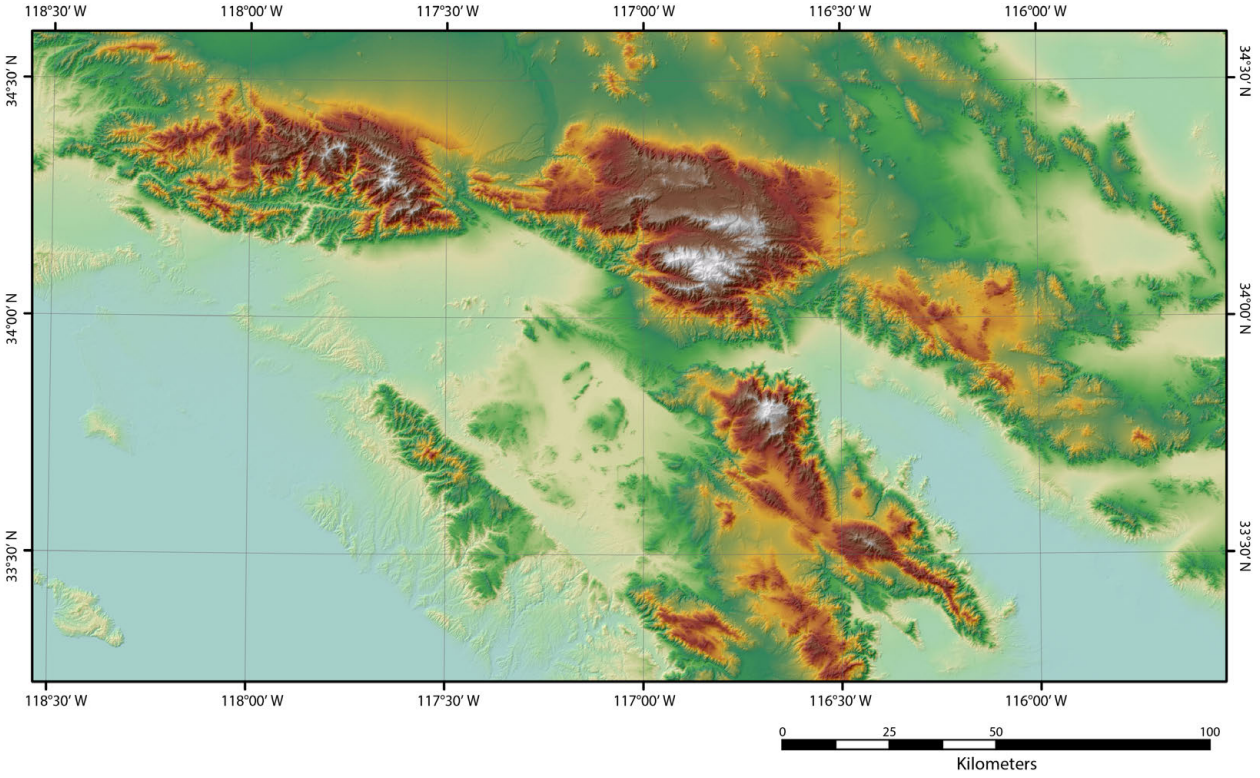


Figure 1: Example dataset area used in this manual.

4 Workflow

Possible workflows through the functions provided as a part of TAK are outlined in Figure 2. In the sections that follow, descriptions of each function are provided and in some cases, possible outputs are included as figures using the example dataset. In later sections we assume that you have a basic familiarity with Matlab data types (e.g. arrays, cell arrays, structures, tables, etc) and the four main TopoToolbox classes (i.e. *GRIDobj*, *FLOWobj*, *STREAMobj*, and *SWATHobj*), but we provide a brief primer in Section 5. This manual does not include all options for all functions; we refer users to the headers of each relevant function for a complete list of required and optional parameters. Instead, this manual is designed to highlight the basic utility of the functions along with some underlying rationale / methodology employed within some of the functions and to provide examples of some recommended use cases.

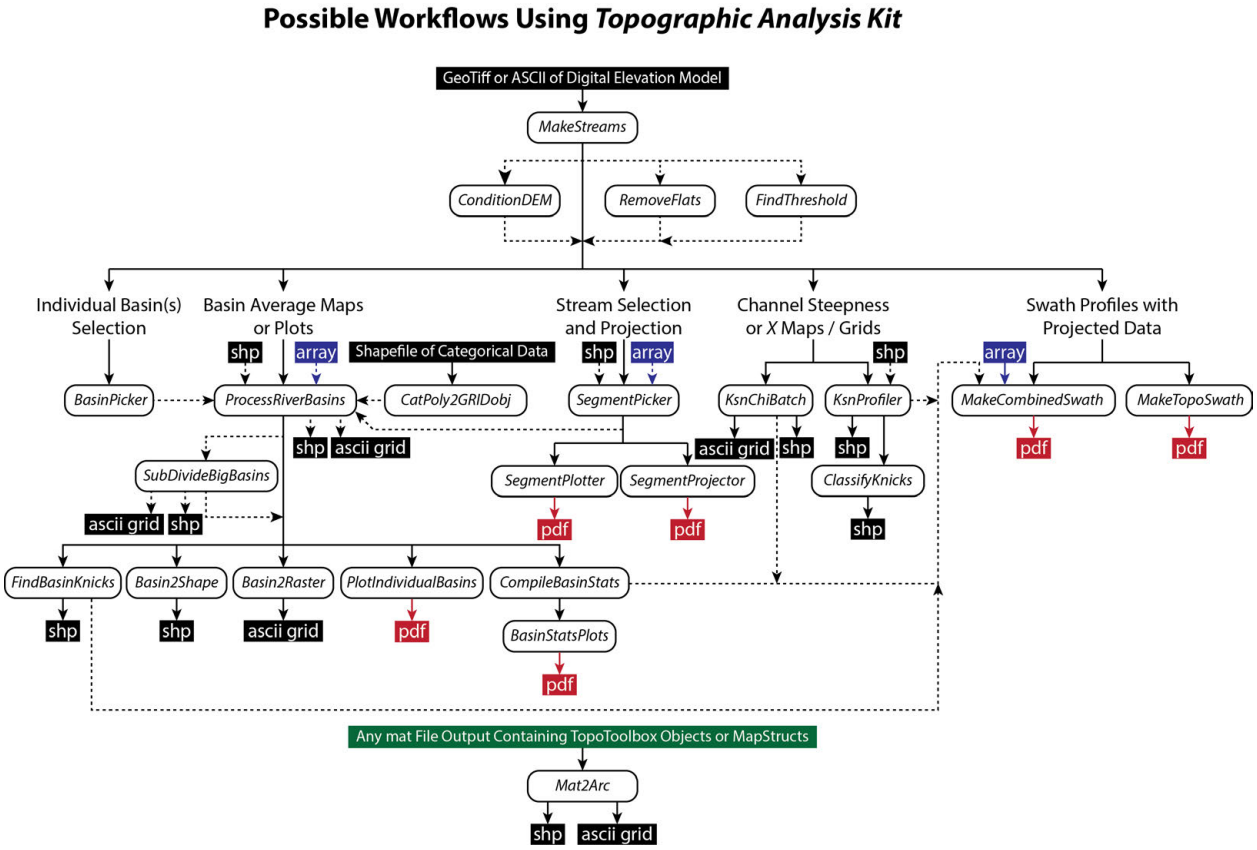


Figure 2: Possible work flows using the functions included in TAK.

5 Matlab and TopoToolbox Crash Course

The following sections are written for people who have never used Matlab (Sections 5.1, 5.2, 5.3) or TopoToolbox (Section 5.4). If you are familiar with basic manipulations of Matlab datatypes, general procedure for using Matlab functions, how to load and save variables in matfiles, and the four primary TopoToolbox classes, feel free to skip these sections and get right into Initial Data Processing with TAK.

5.1 Matlab Data Types

There are four Matlab data types that appear commonly in TAK functions as either inputs or ouptuts (in addition to TopoToolbox specific classes, described in section 5.4), arrays, cell arrays, tables, and structures/geographic data structures.

5.1.1 Arrays

Arrays are the most basic data type in Matlab and are essentially a matrix of numeric values. They have some specific rules associated with them, most importantly that they can only contain numbers and that there can be no

empty elements in an array:

```
% To make an array, values separated by semicolons indicate rows and values
    separated by commas indicate columns, and encasing them in brackets
    indicates that it is an array, so
a=[1;2;3];
% Will make an array with 3 rows and 1 column, whereas
a=[1,2,3];
% Will make an array with 1 row and 3 columns, and
a=[1,2,3;4,5,6];
% Will make an array with 2 rows and 3 columns
% The semicolon at the end of the previous commands suppresses any output to
    the command prompt
% Positions within a matlab array can be specified by their row and column
    position (note that the first index in Matlab is 1, not 0 like in C, Java,
    Python, etc), so calling
a(1,2)
% Will print 2 at the command prompt as this is the number in the first row
    and second column.
% Positions can also be specified by their 'linear index', which is a single
    number that starts counting from position (1,1) and proceeds down columns
    and then across rows, so
a(5)
% Will print 5 at the command prompt, this is equivalent to calling a(1,3)
```

5.1.2 Cell Arrays

Cell arrays are versatile data types that can be thought of as an array of containers. Cell arrays are primarily used in TAK in *ProcessRiverBasins*. Each 'cell' in a cell array can contain pretty much any other data type, e.g. a single number, and entire array, another cell array, etc:

```
% Making a cell array is similar to making an array, except you use curly
    brackets instead of square brackets
c={1,2,3};
% Will produce a 1 x 3 cell array
% Cell arrays can contain different types of data with different dimensions in
     each cell
a=[1,2,3;4,5,6];
c={a,5,'cells can contain strings too'};
% Will also produce a 1 x 3 cell array but this time with very different data
    stored in each cell
% To reference a particular cell, you need to use curly brackets again
c{1,2}
% Would print 5 to the comman prompt, where as
c{1}
% Would print the entire array stored in a to the command prompt
% You can also use parantheses to index a cell array, but in this case the
    result is a new cell array just containing the cells you specified
```

```
new_c=c(2);
% Would produce a 1 x 1 cell array containing the number 5 in the cell, whereas
new_c=c(1,2:3);
% Would produce a 1 x 2 cell array with the number 5 in the first cell, and the string 'cells can contain strings too' in the second cell
```

There are some useful functions to be aware of for converting between arrays and cell arrays, specifically *mat2cell*, *cell2mat*, and *num2cell*. These may be useful in preparing inputs for TAK functions, so we would suggest looking at the help files on these functions if you are having problems generating some of the required inputs for particular TAK functions.

5.1.3 Tables

Tables are similar to cell arrays in that elements within a table can contain a variety of different types of data, but they differ primarily in that they also allow you to specify column and row names and thus the way you access data within tables is different. Tables are primarily used in TAK in the *CompileBasinStats*, *CatPoly2GRIDobj*, and *BasinStatsPlots* functions. As an example, consider extracting information from the table output from *MakeCombinedSwath*:

```
% You can querry the size of a table with size
size(T)
% The entire contents of a particular column can be stored as a new variable by calling the name of the table, e.g. to extract the contents of the 'mean_ksn' column which contains basin averaged normalized channel steepness data in the table output from the CompileBasinStats
all_ksn=T.mean_ksn;
% You can also extract the contents of a single element within a column, e.g., to extract the 10th element of the 'mean_ksn' column
ksn10=T.mean_ksn(10,1);
% Individual elements in a table can contain data of variable sizes and types, e.g. in the table output from CompileBasinStats, each element of the 'mean_ksn' column will have one numeric value, where as each element of the 'hyps' column will have a n x 2 array containing the hypsometry information for a particular basin
```

Tables can also be a useful way to load in data for use in other functions, though generally you will need to convert to other Matlab data types to be valid input for TAK functions:

```
% To read in a file containing mixed data, e.g. a text file with columns named 'sample_names', 'sample_lat', and 'sample_lon' and containing the names of samples as characters, the latitude of samples, and the longitude of samples respectively
T=readtable('samples.txt');
% Particular columns can be easily extracted to arrays or cell arrays
lat=T.sample_lat; % Will produce an array named lat
lon=T.sample_lon; % Will produce an array named lon
```

```
samples=T.sample_names; % Will produce a cell array named samples because the
    sample_names column contains characters
% You can also reference particular rows of particular columns, for columns
    containing numbers you reference these like arrays, for columns containing
    characters you reference these like cell arrays
% Grabbing the 10th row of each column
lat10=T.sample_lat(10,1);
lon10=T.sample_lon(10,1);
name10=T.sample_names{10,1};
```

5.1.4 Structures

Structures allow you to group similar data and store them in containers referred to as fields. Structures can also have multiple dimensions like an array. In TAK, structures are used within the context of geographic data structures, which are a special subset of structures that contain specific fields and can be written out as shapefiles using the *shapewrite* command. Geographic data structures must have fields named, Geometry, X and Y (or Lon and Lat), and BoundingBox (unless the Geometry is Point). We refer interested readers to the Matlab Help documents for the specific requirements of the data stored in these fields if you wish to generate a valid geographic data structure that can be output as a shapefile on your own. Geographic information structures can also have additional fields which will be interpreted as fields in the output shapefile. Several functions produce geographic data structures, e.g. *ProcessRiverBasins* or *KsnChiBatch*. Consider the example of interacting with the *'MSNc'* geographic information structure stored within the outputs of *ProcessRiverBasins* that contains information related to k_{sn}:

```
% You can querry the dimensions of a structure with size
size(MSNc)
% Which in the case of 'MSNc' will be n x 1 depending on the size of the
    stream network
% You can also querry which fields are stored in 'MSNc' with fieldnames
fieldnames(MSNc)
% You can extract the contents of a given field from a particular dimension,
    for example to extract what's stored in the 'ksn' field in the 10th element
ksn10=MSNc(10,1).ksn;
% Examining these fields, you can see that they have variable sizes, for
    example the 'Geometry' field has a single entry per element that is 'Line',
     indicating that the shapefile Geometry type is Line, where as the X and Y
    fields will have n x 1 arrays specifying the X and Y coordinates of line
    segments
% You may wish to extract all the values from a specific field, e.g. all the
    normalized steepness values stored in the ksn field, but unlike with a
    table if you simply call a field without specifying a dimension, you will
    get the first element, not the entire list of elements
MSNc.ksn % Will print whatever ksn value is in the MSNc(1,1).ksn
% To extract all the ksn values stored in the ksn field, you must concatenate
    the field values
all_ksn=[MSNc.ksn] % will produce a row vector, 1 x m of ksn values
all_ksn=horzcat(MSNc.ksn) % will produce a row vector, 1 x m of ksn values
all_ksn=vertcat(MSNc.ksn) % will produce a column vector n x 1 of ksn values
```

```
% Valid geographic information structures can be output as shapefiles
shapewrite(MSNc,'ksn.shp');
% You can also import shapefiles into Matlab as geographic information
    structures
MS=shaperead('shape_name.shp');
```

5.2 Using Matlab Functions

All of the files included in TAK are written as Matlab functions. A Matlab function is stored in a '.m' file and is called by the name of that file:

```
% For a function file named TestFunction.m, you would call it like this at the
    command prompt in Matlab
TestFunction(my_array);
% In this example, the TestFunction has one input that is an array
```

All of the TAK functions have extensive 'headers', i.e. commented text that appears at the top of the .m file that contains a description of the use of the function along with lists of inputs and outputs. You can always open an .m file in Matlab or a text editor to view these, but you can also access them directly from the command line:

```
% Starting the call for a function like this
TestFunction(
% Will display a pop up showing the required inputs for the function and
    inclue a link labeled 'More Help...' that if clicked will open the header
    information for the function in a new window.
```

The majority of TAK functions have both required and optional inputs. As the names imply, required inputs are data or information the function must have to run where as optional inputs are inputs that can be omitted and the function will still run. In many cases, optional inputs are required for the function to run, but they have a default value that will be used if the user does not supply a value to the optional input (e.g. many TAK functions require a reference concavity, this is always specified as an optional parameter that will be set to 0.5 if you do not provide a different number). Required inputs for TAK functions will generally will take one of three forms:

1. The name of a variable stored in the workspace

```
% The header of TestFunction tells you that it has one required input that
    is a Matlab array named INPUT, this means you can provide any Matlab
    array (with any name) to TestFunction
TestFunction(my_array);
% Alternatively, if TestFunction says it requires two inputs, INPUT1 and
    INPUT2, i.e. the help pop up looks like TestFunction(INPUT1,INPUT2),
    and the first input is supposed to be a Matlab array and the second is
    supposed to be a Matlab cell array, then a valid input would be
TestFunction(my_array,my_cell_array);
% Always consult the header for specific requirements, some arrays have
    restrictions on their dimensions, e.g. they must be an n x 2 array.
    Inputs to TAK functions will be 'parsed' so if they do not meet the
    requirements, you will be informed of this and the function will error
    out
```

2. A character string defining an option or giving the name of a file or folder

```
% The header of TestFunction tells you it requires one input Method that defines a method and that the valid inputs to method are 'split' or 'join', so
TestFunction('split');
% Would be a valid call to the function, where as
TestFunction(split)
% Would not be a valid call
% Similarly, if a required input is the name of a file, you would give these in single quotes
TestFunction('my_file.txt');
```

3. A logical value

```
% The header of TestFunction tells you that it has one required input, Do_X and that this expects a logical value, then you could call it like this
TestFunction(true);
% Or like this
TestFunction(1);
% As 0 and 1 are equivalent to false and true, respectively
```

Optional inputs follow many of the same rules, but they importantly differ in that they require that they are proceeded by the name of the optional parameter:

```
% From the header of TestFunction, you learn that it has three required inputs, 1) data, which expects a Matlab array, 2) method, which specifies a method to use on the data that is either 'split' or 'join', and 3) save_output, which expects a logical value. TestFunction also has two optional parameters, 1) file_name, which expects the name of the file to be output and 2) extra_data, which expects another Matlab array. All of the following are valid calls to TestFunction
TestFunction(my_array,'split',true); % Running function with no optional inputs
TestFunction(my_array,'join',true,'file_name','my_file.txt');
TestFunction(my_array,'split',false,'extra_data',my_other_array);
TestFunction(my_array,'join',true,'extra_data',my_other_array,'file_name','my_file.txt');
% Note that the order in which you specify optional parameters doesn't matter, but the argument passed to an optional parameter must always immediately follow the name of the appropriate optional parameter
```

Many of the TAK functions also have outputs that will be stored as variables in the Matlab workspace after a successful run of the function. Outputs are specified like so:

```
% The header indicates that TestFunction from the previous example has two outputs, split_data and joined_data, you can specify the name of the variables for these outputs
```

```
[my_splits,my_joins]=TestFunction(my_array,'split',true);
% Now variables my_splits and my_joins will appear in your Matlab workspace
% If you don't actually care about one of the outputs, e.g. you only want
    my_joins, you can supply a ~ to any output you don't want to be output to
    the workspace
[~,my_joins]=TestFunction(my_array,'split',true);
```

5.3 Loading and Outputting Data

The outputs of many TAK functions are automatically saved as 'matfiles', which are versatile matlab files that can contain multiple variables and will have a '.mat' suffix. Some basic operations with matfiles:

```
% You can querry the contents of a matfile
whos('-file','Basin_1_Data.mat')
% Which will print out the list of variable names, their sizes, and the type
    of data stored in that variable to the workspace
% Alternatively, highlighting a matfile in the 'Current Folder' window will
    display the contents in the bottom left of the Matlab screen
% If you want to load in a particular variable, you can use load
load('Basin_1_Data.mat','DEMcc'); % Will load the DEMcc variable into the
    workspace
% You can load all variables contained in a matfile by not specifying any
    variables with the load command
load('Basin_1_Data.mat');
% The syntax for saving data into a matfile is similar
save('MyMat.mat','DEMcc');
% If the specified mat file name already exists, the previous action will
    overwrite the matfile. If you instead want to add the variable to the
    variables already stored in the matfile, you can use '-append'
save('MyMat.mat','DEMcc','-append');
```

5.4 TopoToolbox Classes

There are four primary TopoToolbox classes, *GRIDobj*, *FLOWobj*, *STREAMobj*, and *SWATHobj*. We refer interested users to the TopoToolbox documentation or Wolfgang Schwanghart's excellent blog for detailed discussions of these data classes, but below we provide a very brief description of these different classes. A general point to be aware of when using TopoToolbox classes is that many functions require several of these different classes and, unless otherwise stated, it is assumed that these are datasets that were generated together and from each other (e.g. the *STREAMobj* generated from a particular *FLOWobj* which in turn was generated from a particular *GRIDobj*). You generally don't need to worry about this if you are using the TAK functions exclusively, but if you ever get an error regarding datasets not aligning, check to make sure you are not mixing different TopoToolbox datasets that did not derive from the same DEM.

The TopoToolbox dataclasses are unique, but in terms of other Matlab data types, they are the most similar to a 1 dimensional structure, i.e. they contain a series of 'fields' that contain a variety of different data types:

```
% To extract the data array within a GRIDobj named DEM
```

```
elevations=DEM.Z; % Will be a n x m array of numeric values
% To extract the cellsize of a GRIDobj named DEM
cellsize=DEM.cellsize; % Will be a single value
% The contents of a TopoToolbox object can be querried with the 'fieldnames'
    function, just like a structure
fieldnames(DEM)
% Will output a cell array with the names of the fields contained within DEM
```

5.4.1 *GRIDobj*

*GRIDobj*s are for storing raster data. In TAK, they are how DEMs, flow accumulation rasters, and other additional gridded data is stored. It is very simple to generate a *GRIDobj*:

```
% GRIDobjs can be created from ascii grids or geotiffs
DEM=GRIDobj('/path/to/gridded__elevation_data.txt');
GRID=GRIDobj('/path/to/other_gridded_data.tif');
```

It is important to note that it is recommended that you project data into a projected coordinate system (e.g. UTM) before turning it into a GRIDobj, failure to do so will result in errors in various TAK functions.
If you want to plot a *GRIDobj*

```
% To plot a single GRIDobj
imagesc(GRID);
% To plot a hillshade colored by another GRIDobj (can provide the DEM as the
    second input to color by elevation)
imageschs(DEM,GRID);
```

5.4.2 *FLOWobj*

*FLOWobj*s are special data classes for storing flow routing information. Unlike a flow direction raster in ArcGIS, this is not something that can be easily visualized, but it is a crucial dataset used for almost all TAK functions.

5.4.3 *STREAMobj*

*STREAMobj*s are data classes for storing stream networks. To plot a *STREAMobj*,

```
%To plot a map of a STREAMobj
plot(S);
% To plot a longitudinal profile of the streams in a STREAMobj
plotdz(S,DEM);
```

5.4.4 *SWATHobj*

*SWATHobj*s are data classes for storing swath data extracted from *GRIDobj*. To create a basic plot of a *SWATHobj*,

```
%To plot swath profile of a SWATHobj
plotdz(SW);
```

6 Initial Data Processing

6.1 *CheckTAKDependencies*

TopoToolbox and TAK require several different Matlab toolboxes. *CheckTAKDependencies* is a simple function that checks to see if you have licensed versions of all the required toolboxes.

```
% CheckTAKDependencies takes no inputs and has no formal outputs
% If running
CheckTAKDependencies
% Produces no warnings, then all of the TAK functions should work (or at least
    , they shouldn't fail because of missing dependencies!)
% Alternatively you may see text like:
'Warning: Fatal error: You do not have a license for the Mapping Toolbox,
    TopoToolbox will not function properly'
% If you are missing a crucial TopoToolbox, or this:
'Warning: You do not have a license for the Statistics and Machine Learning
    Toolbox, some functions will not work properly'
% If you are missing a Toolbox that are only used by some TAK functions
```

TAK requires licenses for the Image Processing Toolbox, Mapping Toolbox, Optimization Toolbox, and Statistics and Machine Learning Toolbox. If you do not have all licenses for all of these, you may need to use the Compiled Functions.

6.2 *MakeStreams*

MakeStreams is a simple wrapper around creating the basic TopoToolbox objects needed for the majority of other TAK functions, specifically a digital elevation model (DEM) as a *GRIDobj*, a flow direction dataset as a *FLOWobj*, a flow accumulation grid as a *GRIDobj*, and a stream network as a *STREAMobj*. The minimum inputs are the location of a valid DEM as either a geotiff or ascii grid and a minimum threshold drainage area (in square map units) for beginning stream network definition:

```
[DEM,FD,A,S]=MakeStreams('/Users/aforte/GISdata/SoCal_UTM_DEM.txt',1e6);
```

The basic usage of *MakeStreams* will produce stream networks, that depending on the nature of your DEM, may include areas that are not of interest or should not be included in stream definition (Figure 3).

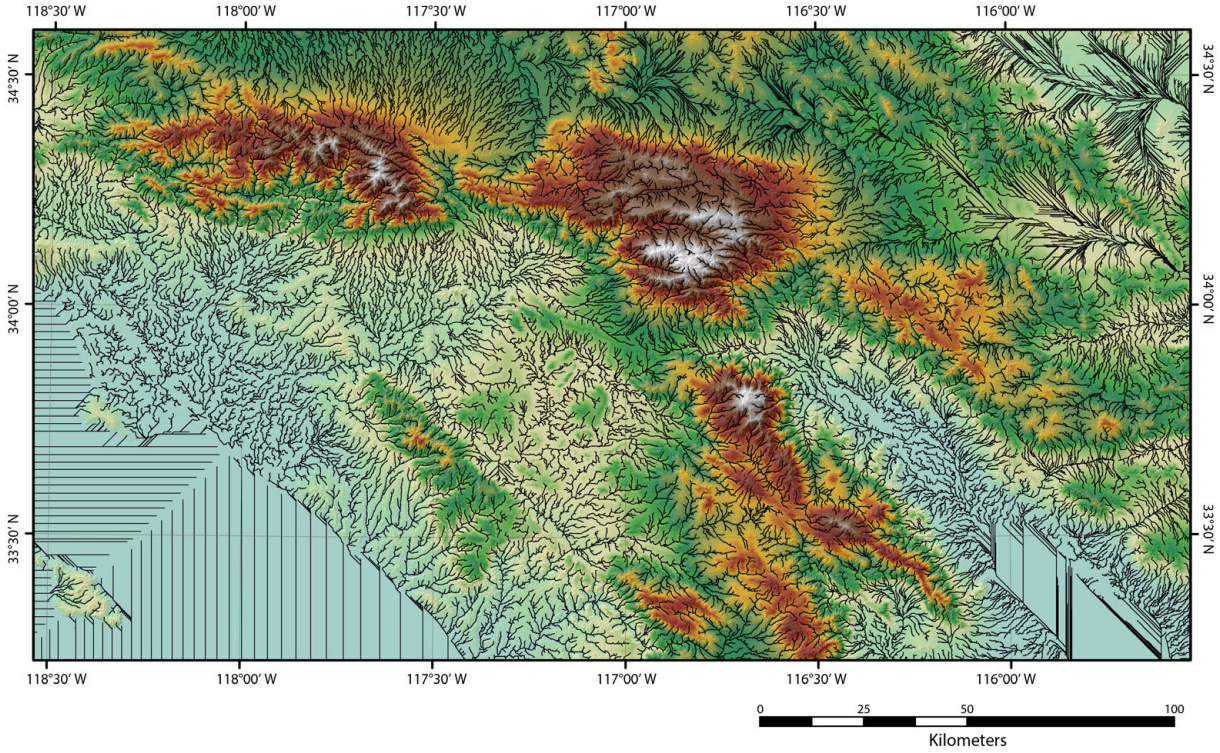


Figure 3: Result of running *MakeStreams* without any control for flat areas.

MakeStreams includes various simple options to filter the input DEM. This can be done through a logical expression, for example, if you wanted to set any portions of the DEM at or below 0 m elevation to no data (and thus suppress stream definition), you could use the following:

```
[DEM,FD,A,S]=MakeStreams('/Users/aforte/GISdata/SoCal_UTM_DEM.txt',1e6,'
    no_data_exp','DEM<=0');
```

There is also a built in auto filter that will identify true flats (i.e. areas of constant elevation) and set these to no data:

```
[DEM,FD,A,S]=MakeStreams('/Users/aforte/GISdata/SoCal_UTM_DEM.txt',1e6,'
    no_data_exp','auto');
```

Using this auto filter produces a more reasonable stream network and removes the Pacific Ocean and Salton Sea from the areas where streams are defined. (Figure 4). The auto filter can be controlled by setting a minimum area to identify as a flat to avoid finding small flat areas of DEMs that you do not wish to remove. This is set to a default minimum area of 1e8 m^2. Setting this minimum flat area to small values may result in a discontinuous stream network (Figure 5).

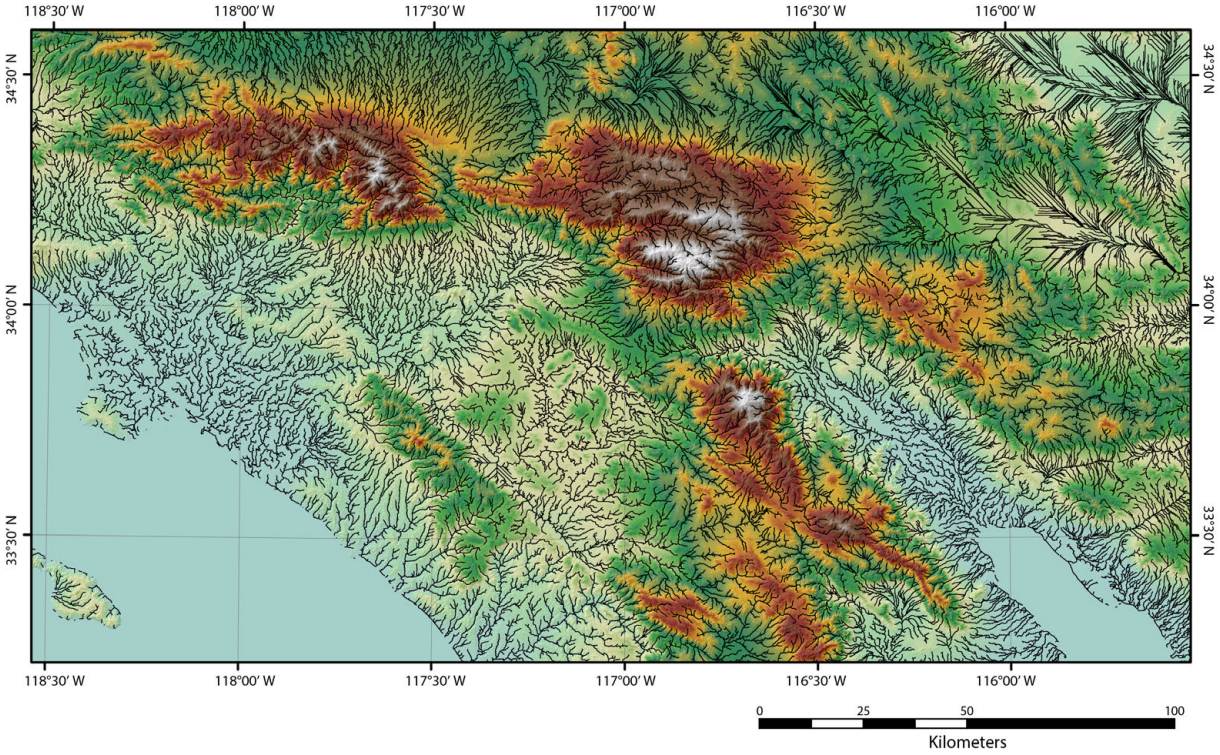
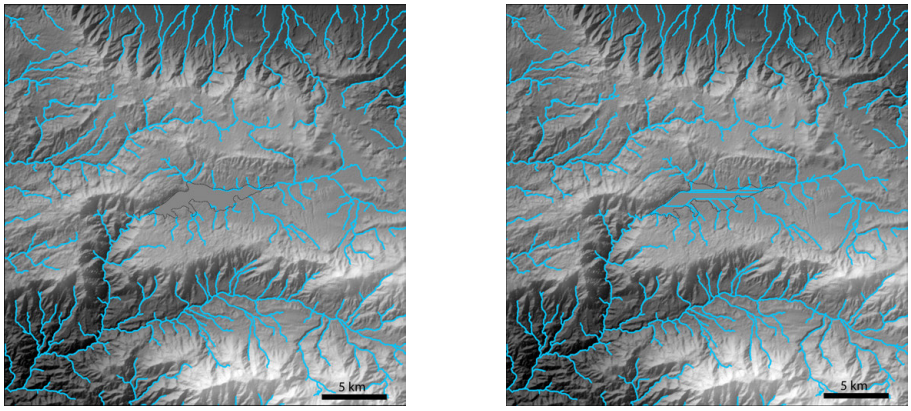


Figure 4: Result of running *MakeStreams* with auto removal of flat areas and *min_flat_area* set to 1e8.



(a) Minimum flat area of 1e6

(b) Minimum flat area of 1e7

Figure 5: Difference in output of *MakeStreams* depending on value used for *min_flat_area* when auto removing flat areas in the area around Big Bear lake (outline of lake is shown in thin dotted black line) in the San Bernadino Mountains. When *min_flat_area* is 1e6 (5a), the lake is identified as a flat and removed, but when *min_flat_area* is 1e7 (5b) the lake is not identified as a flat and thus streams are routed through the lake.

MakeStreams also has controls for resampling the input DEM. This can be useful (and is sometimes necessary), because after reprojecting georeferenced data in a GIS program, the cellsize of the DEM can end up as a number with a lot of decimal places. This can cause problems in some TopoToolbox and TAK functions because of different rounding behaviors and thus cause the codes to think that two datasets do not line up, even when they do. To avoid this, *MakeStreams* will warn you if the provided DEM does not have a whole number cellsize and suggest that you use the resample option to fix this for later processing.

MakeStreams also has an options to provide a precipitation dataset (with or without a comparable runoff ratio grid) to automatically produce a weighted flow accumulation raster:

```
PRECIP=GRIDobj('/Users/aforte/GISdata/prism_precip.tif');
[DEM,FD,A,S]=MakeStreams('/Users/aforte/GISdata/SoCal_UTM_DEM.txt',1e6,'
    precip_grid',PRECIP);
```

6.3 *ConditionDEM*

DEMs can be extremely noisy and thus can produce very jagged stream profiles. This is both visually unappealing, but more importantly is problematic for calculations that use stream gradient (e.g. normalized channel steepness). TopoToolbox includes a variety of different algorithms to condition or smooth DEMs, either as an entire grid or specifically along specified stream networks. The *ConditionDEM* function is a wrapper around all of these different routines. Some are very computationally intensive and require several parameters, so you are advised to spend some time 'playing' with the parameters and choices to see the outcome. *ConditionDEM* produces example stream profiles comparing the unconditioned and conditioned result along with a map showing where elevations were changed to aid you in understanding what the chosen algorithm has done.

It is not strictly required to use the *ConditionDEM* function. All TAK functions that need smoothed channel profiles have built in conditioning, using the *mincosthydrocon* algorithm with an interpolation value of 0.1. If you want to use a different conditioning algorithm, you can use *ConditionDEM* to produce a conditioned DEM that you can then supply as an optional input to TAK functions that need a conditioned DEM.

6.4 *RemoveFlats*

For topographic analysis, we are often not interested in flat areas and it can be helpful to remove these from DEMs or set them to no data to restrict stream definition. Some of this can be accomplished with the simple controls built into *MakeStreams*, but when these flat areas are not perfectly flat or occur at multiple elevations, they can be problematic to remove without manual clipping of DEMs. The *RemoveFlats* function is an attempt to partially automate this process. It allows you to graphically identify areas (with a single click for each connected flat area) that you consider flats and the function will attempt to find contiguous areas. This function can sometimes be overly aggressive and remove areas of interest, so it is always best to inspect the results of the function before using it for subsequent processing (e.g. providing the filtered DEM produced here to *MakeStreams* to regenerate a stream network) and compare it with the basic output of *MakeStreams* to determine whether the results of *RemoveFlats* are suitable for use.

6.5 *FindThreshold*

Stream definition is commonly controlled by a minimum threshold accumulation area, i.e. streams are defined as anywhere within a DEM where the upslope drainage area exceeds a threshold value. The *FindThreshold* function is designed to aid you in choosing an appropriate minimum threshold area or, alternatively, manually setting this threshold

area on a stream by stream basis. It requires that you run *MakeStreams* initially, though the threshold area you use when running *MakeStreams* is not critical (though this will dictate the initial drainage density and thus the number of channel heads with which to work).

FindThreshold can be run in one of two ways. In the first way, when you provide a numeric input to the *'num_streams'* parameter, the function displays this user specified number of streams extracted all the way to the drainage divide and allows you to choose, on either a χ-elevation or slope-area plot where you think the hillslope to channel transition might be. In this mode, the function will produce a new stream network based on the average of your minimum threshold area choices, and will also give you the population of minimum threshold areas and associated distances from the drainage divide to the channel head for the streams you choose. Alternatively, *FindThreshold* if *'num_streams'* is set to *'all'*, it runs in a mode where the function iterates through all channel heads in the provided stream network and uses the same picking protocol as above to manually set the minimum threshold area for each stream. In this mode each stream will have a variable minimum threshold area, but the total number of streams (i.e. channel heads) will still be dictated by the drainage density of the input stream network. Running *FindThreshold* with *'num_streams'* set to *'all'* on a large stream network may be very labor intensive, for example, the southern California dataset with an initial threshold area of 1e6 m^2 has $>$ 120,000 channel heads so if you were to in *'all'* mode, you would have to manually identify the hillslope-channel transition on ALL of those channel heads.

7 Stream Selection and Projection

The three functions for stream selection and projection have some overlap with similar functions in TopoToolbox, but are slightly different in implementation or style of outputs. Exploring both to see which set better fits your needs is advisable.

7.1 *SegmentPicker*

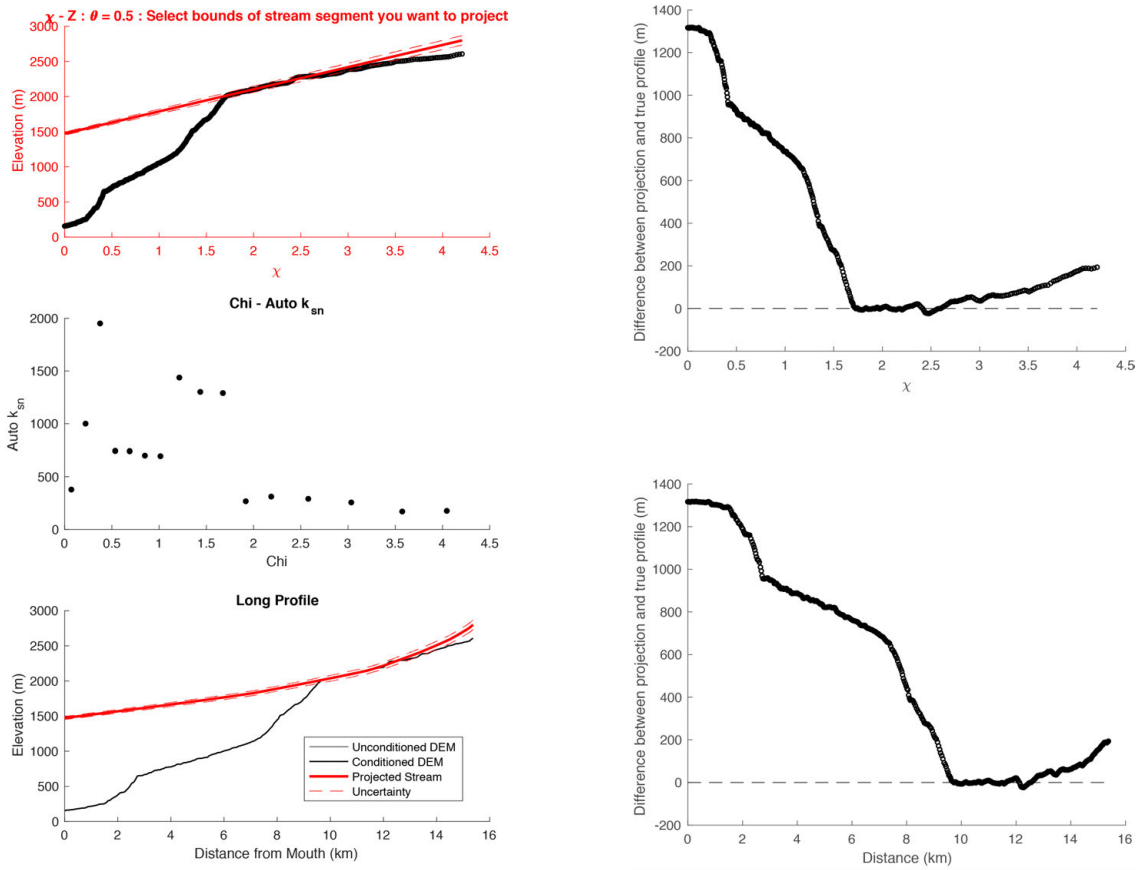
SegmentPicker is a function designed to select portions of larger stream networks. In terms of stream selection, the function has two primary modes, either 'down', i.e. you select individual streams based on a channel head location, or 'up', i.e. you select portions of networks above a given pour point. Channel selection can be done interactively within Matlab or by providing coordinates of channel heads or pour points as an array or a point shapefile. *SegmentPicker* shares some similarity to the TopoToolbox function *flowpathapp* or functionality available in *topoapp*.

7.2 *SegmentPlotter*

SegmentPlotter takes the output of *SegmentPicker* and plots each selected stream individually. The stream network is plotted as χ-elevation, longitudinal profile, and slope-area plots. There are various options for subsetting portions of picked streams, producing individual or single figures, and labeling.

7.3 *SegmentProjector*

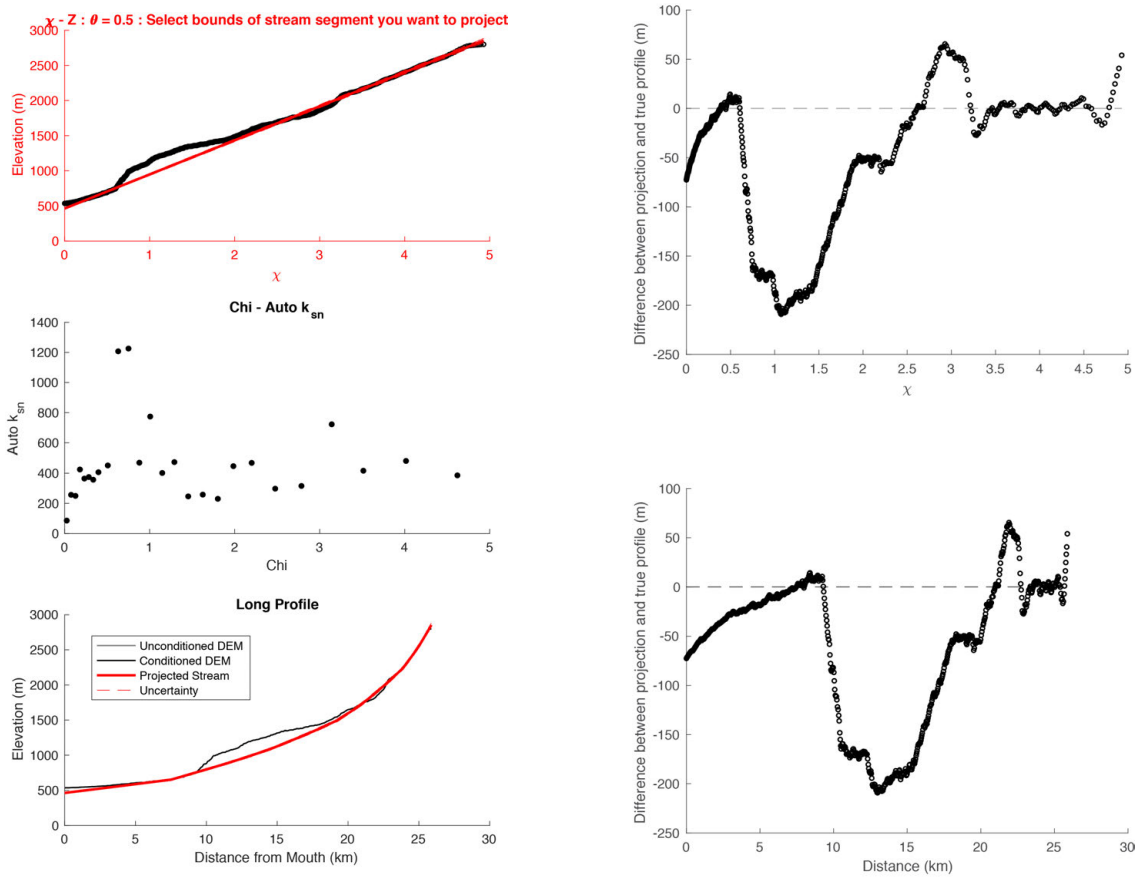
SegmentProjector allows you to select a portion of a stream to project along the length of the entire stream (i.e. from the mouth to the channel head of the provided stream). This can be useful for a variety of questions, e.g. estimating the amount of uplift of a low relief portion of a landscape (Figure 6) or identifying portions of a stream profile that may be tectonically deformed or otherwise disturbed (Figure 7).



(a) Projected stream

(b) Residual on projected stream

Figure 6: Example of *SegmentProjector* outputs for a stream within Basin 402 (northeastern San Jacinto Mountains) with a low relief upper portion.



(a) Projected stream

(b) Residual on projected stream

Figure 7: Example of *SegmentProjector* outputs for a stream within Basin 313 (northwestern San Jacinto Mountains) with possible localized deformation or landslide deposit within the profile.

SegmentProjector will iterate through all channel heads within a provided stream network, so it is suggested that you use *SegmentPicker* or some other means of selecting streams of interest before running *SegmentProjector*. Note that if you are using the output of *SegmentPicker*, you will need to load the *PickedSegments_*.mat* file and provide the *STREAMobj Sc* stored within this file as the required *STREAMobj* input to *SegmentProjector*. *SegmentProjector* is similar to the TopoToolbox *streamproj* function.

8 Channel Steepness and χ Maps

8.1 *KsnChiBatch*

KsnChiBatch is designed to be similar to the original 'batch' mode in Profiler51 for creating normalized channel steepness (k_{sn}) maps, but has many more options. *KsnChiBatch* can be used to produce stream networks with values of k_{sn} or χ or continuous grids of either k_{sn} or χ. There are options to produce outputs as either shapefiles or ascii grids for use in a GIS program or matlab outputs for display (e.g. using *PlotKsn*) or for analysis. There are also options

to remove incomplete portions of stream networks (which can be problematic for both χ and k_{sn} calculations) and control outlet elevation (which can be important for correctly interpreting χ anomalies). The methodology employed for dealing with χ in *KsnChiBatch* is identical to that as in the companion DivideTools GitHub repository, the details of which are described in Forte and Whipple [2018] [Link to Journal Site] .

The outputs of the *KsnChiBatch* function differ depending on the type of map being calculated. If the required *'product'* input is set to *'ksn'*, then the function will save a polyline shapefile, but if the *'product'* is *'chi'*, *'chigrid'*, or *'ksngrid'* the function will save an ascii grid. χ maps are output as grids to avoid averaging of chi values along a stream network during the production of a shapefile. If you wish to create a shapefile of a χ map, you can use a GIS program to create a shapefile from the ascii grid (e.g. in ArcGIS, convert the ascii to a raster and then use the raster to polyline function).

You can also specify that you want the function to produce outputs to the workspace with the optional *'output'* parameter. The number of outputs will again vary with the product:

```
% When 'ksn' is the 'product', two outputs will be produced, a GRIDobj with
    averaged ksn values stored in nodes along the stream network  (
    KSN_STREAM_GRID) and a geographic information structure of ksn values (
    ksn_ms)
[KSN_STREAM_GRID,ksn_ms]=KsnChiBatch(DEM,FD,A,S,'ksn');
% When 'ksngrid' is the product, a GRIDobj of interpolated ksn values will be
    produced
[KSN_GRID]=KsnChiBatch(DEM,FD,A,S,'ksngrid');
% When 'chimap' is the product, a GRIDobj with chi values stored in nodes
    along the stream network will be produced
[CHI_MAP]=KsnChiBatch(DEM,FD,A,S,'chimap');
% When 'chigrid' is the product, a GRIDobj with chi values in all nodes not
    excluded by outlet conditions will be produced
[CHI_GRID]=KsnChiBatch(DEM,FD,A,S,'chigrid');
% When 'chi' is the product, the chi map output will come first
[CHI_MAP,CHI_GRID]=KsnChiBatch(DEM,FD,A,S,'chi');
% When 'all' is the product, all of the previous outputs will be produced in
    the following order
[KSN_STREAM_GRID,ksn_ms,KSN_GRID,CHI_MAP,CHI_GRID]=KsnChiBatch(DEM,FD,A,S,'all
    ');
```

Values of k_{sn} can be calculated in two primary ways. The default *'ksn_method'* (and the same method used in the original Profiler51 or the TopoToolbox *ksn* function) is denoted *'quick'* and calculates k_{sn} across the entire stream network simultaneously by solving the equation $S = k_{sn} * A^{-\theta}$, where S is the gradient, A is the drainage area, and θ is a reference concavity. This result will typically be extremely noisy because of small variations in gradient, so k_{sn} values are usually averaged over some length, depending on the resolution of the data. This *'smooth_distance'* is a user controlled parameter in *KsnChiBatch*.

The default method described above is quick to calculate, but can sometimes smear out true abrupt changes in k_{sn} at confluences. For this reason, by setting *'ksn_method'* to *'trib'*, k_{sn} can also be calculated with *KsnChiBatch* where network segments (i.e. stream segments between confluences) are selected, divided into sub-segments determined by

the *'smooth_distance'*, and then the average k_{sn} of each sub-segment is calculated by finding the best fit slope on the χ-elevation relationship for this sub-segment (k_{sn} is equivalent to the slope of the χ-elevation relationship when the reference drainage area is set to 1). The *'quick'* and *'trib'* methods do produce different k_{sn} patterns, though while subtle, do show some systematic behavior (Figure 8).

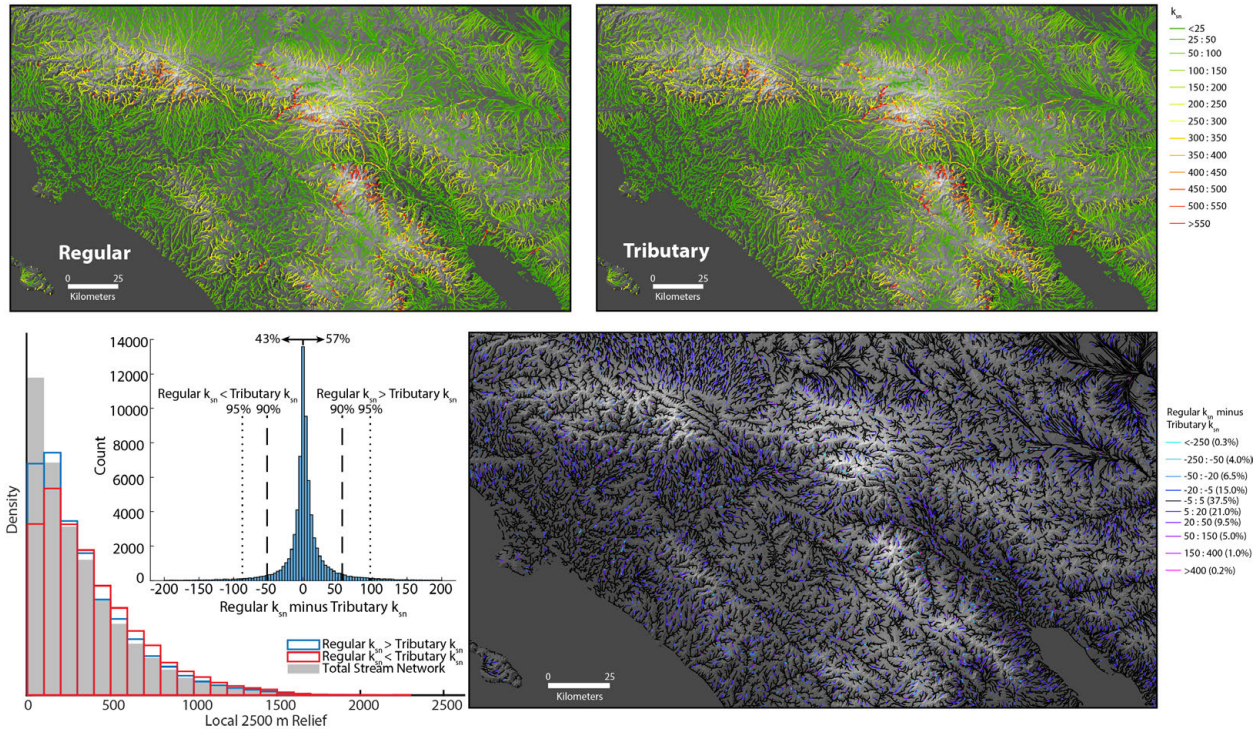


Figure 8: Comparison of the regular 'quick' method vs the 'tributary' method of calculating k_{sn} using *KsnChiBatch*.

While we have not conducted a full study of the differences of these two methods, the southern California example suggests that generally the regular *'quick'* method of k_{sn} calculation may be slightly biased towards higher k_{sn} values, though 90% of k_{sn} values of the two methods are within $\pm$50 of each other (this is with a reference concavity of 0.50, so absolute magnitudes of k_{sn} and thus magnitudes of deviations between these two methods will scale with choice of reference concavity). The k_{sn} values from the regular method are generally systematically larger than the tributary method values in low relief areas, where as generally k_{sn} values from the tributary method tend to be higher than regular values in higher relief portions of the landscape (Figure 8).

The geographic information structure (and shapefile) produced from both methods for each segment (controlled by the *'smooth_distance'*) will have fields containing values for k_{sn} (*'ksn'*), mean drainage area (*'uparea'*), mean gradient (*'gradient'*), and the difference between the conditioned stream elevation and non-conditioned stream elevation (*'cut_fill'*) to inform the user whether an anomalous k_{sn} value may because of an overly aggressive conditioning scheme. If the *'trib'* method is used, the geographic information structure and shapefile will include an extra field, *'chi_r2'*, which is the R^2 value on the χ-elevation fit and can be interpreted as a measure of the linearity of each sub-segment.

8.2 *KsnProfiler*

For detailed analysis of streams, it is often necessary to manually select knickpoint bounded steam segments for which you wish to calculate average k_{sn} values. This was the primary purpose of the original Profiler51 code and we have produced the *KsnProfiler* function to replicate and improve upon the original Profiler51 methodology. In detail, *KsnProfiler* calculates k_{sn} of user selected segments by finding the best fit linear slope of the χ-elevation relationship for the segment in question. It should be noted that 1) calculation of k_{sn} via this method will produce identical results to finding the intercept of a fit in slope-area space (providing that the reference area in the χ calculation is set to 1, as it is in *KsnProfiler*) so there is no need to distinguish between k_{sn} values calculated by a χ-elevation slope or a slope-area intercept and 2) though the publication associated with the original Profiler51 codes discussed k_{sn} calculation in terms of slope intercepts (Wobus et al. [2006]), internally, the code used the slope of a linear fit on a χ-elevation relationship to calculate k_{sn}. *KsnProfiler* has a large number of optional inputs, so you should familiarize yourself with all of the ways in which it can be run by reading through the header of the function. We highlight some of the main options in the following sections.

8.2.1 Stream Selection

There are four different ways in which you can define which streams you would like to fit. The default is an interactive channel selection method where you choose streams (from all of the streams in the provided *STREAMobj*) manually by clicking near a channel head of interest. In this case, the call of *KsnProfiler* is relatively simple:

```
[~,~,~,~]=KsnProfiler(DEM,FD,A,S);
```

While this can be nice especially as it will update the map with k_{sn} values as you continue to pick and fit streams, it can be unstable or slow if you are trying to analyze a very large area (though you can use the *'plot_type'* parameter to switch to a map plot that is more optimized for larger datasets). The three other selection methods do not involve a map of streams, so is generally more stable for large datasets. The other options are called like:

```
% To iterate through all streams in the provided STREAMobj:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'input_method','all_streams');
% To iterate through all streams and fit any stream over 50 km in total
    length:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'input_method','stream_length','
   min_length_to_extract',50000);
% To fit streams based on a defined list of channel head locations provided
    as an array of channel head locations (chl in the example below):
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'input_method','channel_heads','
    channel_head_list',chl);
% To fit streams based on a defined list of channel head locations provided
    as an array of channel head locations from a point shapefile of channel
    heads:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'input_method','channel_heads','
   channel_head_list','channel_heads.shp');
```

8.2.2 Dealing with Stream Junctions

Similar to the methodology employed in *KsnChiBatch*, *KsnProfiler* gives you the option to fit across stream junctions or only fit portions of streams upstream of junctions. This is controlled with the optional parameter, *'junction_method'*, which by default is set to *'check'*.

```
% To run KsnProfiler in default mode where fits do not occur across stream
    junctions, no argument is required for 'junction_method':
[~,~,~,~]=KsnProfiler(DEM,FD,A,S);
% If you still want to specify it so that there is a record in the workspace:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'junction_method','check');
```

In this mode, the first time a particular stream segment within a given connected network is chosen, the entire stream profile will be displayed (i.e. from the channel head to the outlet) and used during the fitting process. For any subsequent chosen streams that share a portion of a previously fit stream, only the unique (i.e. the portion of the stream upstream of junctions) will be displayed and fit. This allows you to analyze portions of streams independently to avoid fitting across confluences and also avoids the stacking effect downstream of confluences where shared portions of picked streams are fit multiple times. By setting *'junction_method'* to *'ignore'*, you can operate *KsnProfiler* similar to the original Profiler51 where all stream chosen stream segments are displayed in their entirety and stream junctions are ignored.

```
% To run KsnProfiler similar to how Profiler51 operated and fit all selected
    streams in their entirety:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'junction_method','ignore');
```

There is a related optional parameter, *'stack_method'* that deals with the overlapping portions of streams allowing you to either generate multiple polylines (one for each fit) that will be stacked on top of each other in the resulting shapefile or to average values node by node in any overlapping segments.

```
 % Ignoring stream junctions and stacking polylines in output shapefile:
 [~,~,~,~]=KsnProfiler(DEM,FD,A,S,'junction_method','ignore','stack_method','
     stack');
 % Ignoring stream junctions and averaging values in overlapping segments
     before producing output shapefile:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'junction_method','ignore','stack_method','
    average');
```

The two different junction methods (i.e. *'check'* vs *'ignore'*) will produce slightly different results, but are generally more similar to each other than when compared to k_{sn} values calculated via batch processing using *KsnChiBatch* (Figure 9).

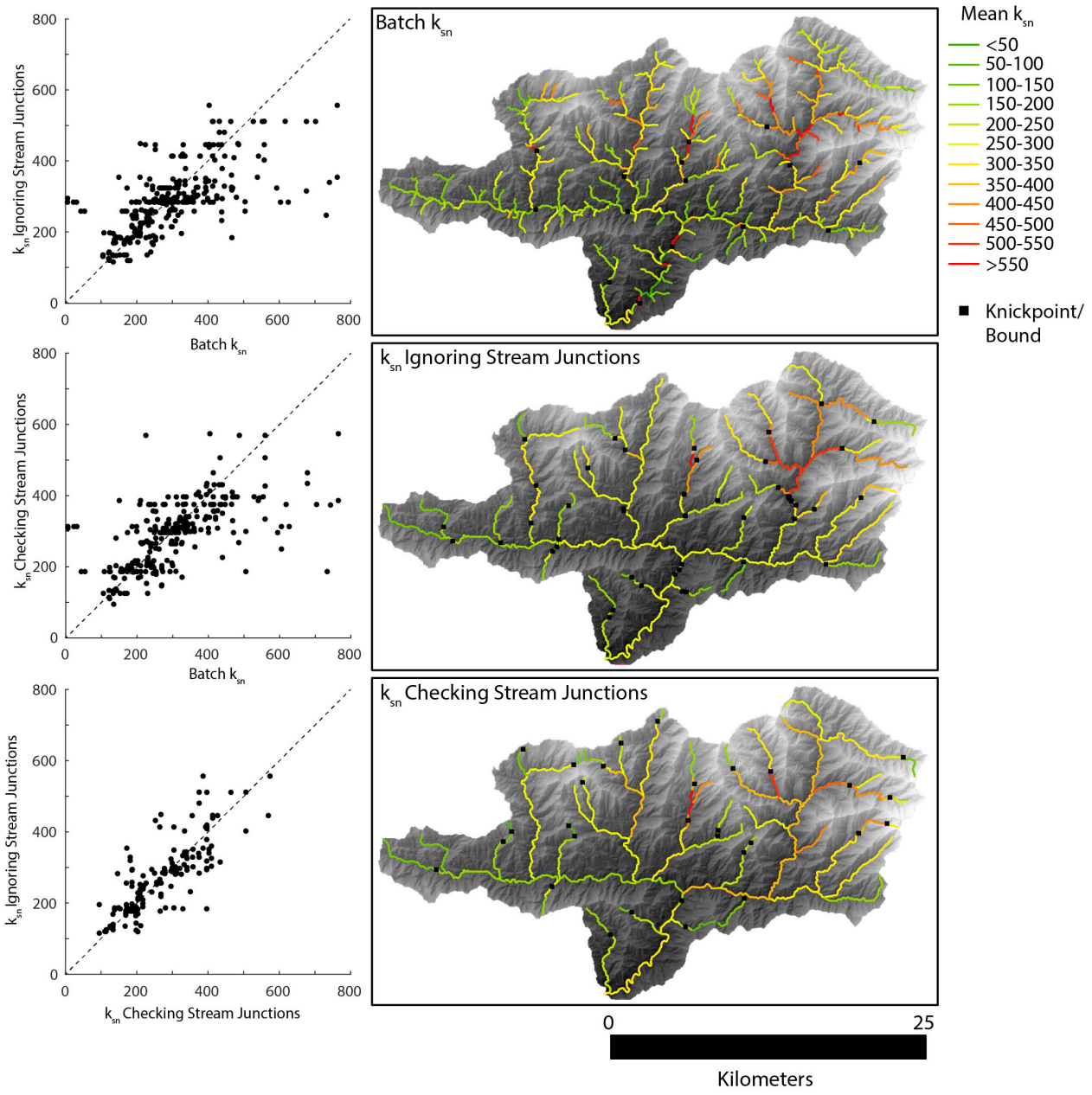


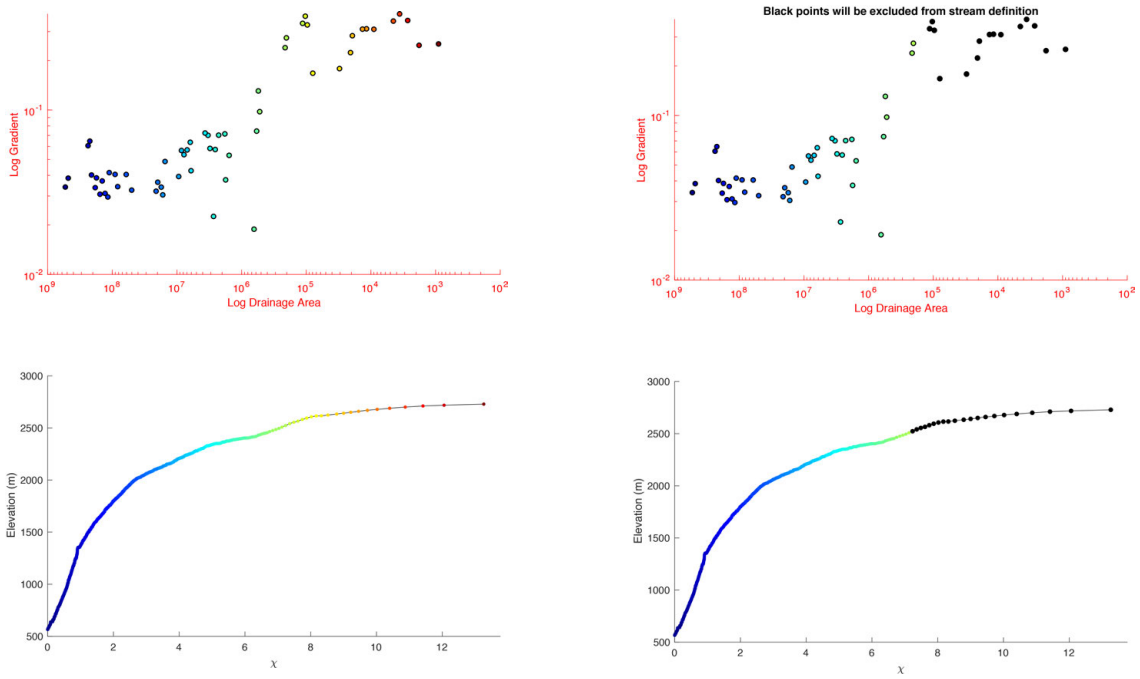
Figure 9: Comparison of k_{sn} values for Basin 56, calculated using the batch method (using *KsnChiBatch*) or *KsnProfiler* either ignoring or checking stream junctions.

8.2.3 Defining the Minimum Threshold Area

By default, the *KsnProfiler* function will use the stream network exactly as supplied, but there are several options for more precisely controlling the minimum threshold area for streams. The first option is to use *FindThreshold* to

regenerate the stream network and use this new *STREAMobj* as the input to *KsnProfiler*. Alternatively, you can use a built in function of *KsnProfiler* via the optional *'redefine_threshold'* parameter to choose the minimum threshold area on either a slope-area or χ-elevation plot (Figure 10).

```
% To redefine minimum threshold area for each stream chosen stream segment
    individually:
[~,~,~,~]=KsnProfiler(DEM,FD,A,S,'redefine_threshold',true);
```



(a) Before threshold area selection

(b) After threshold area selection

Figure 10: Example of defining minimum threshold area on a slope-area plot using *KsnProfiler*, black dots are portions of the stream network that will be excluded from channel definition.

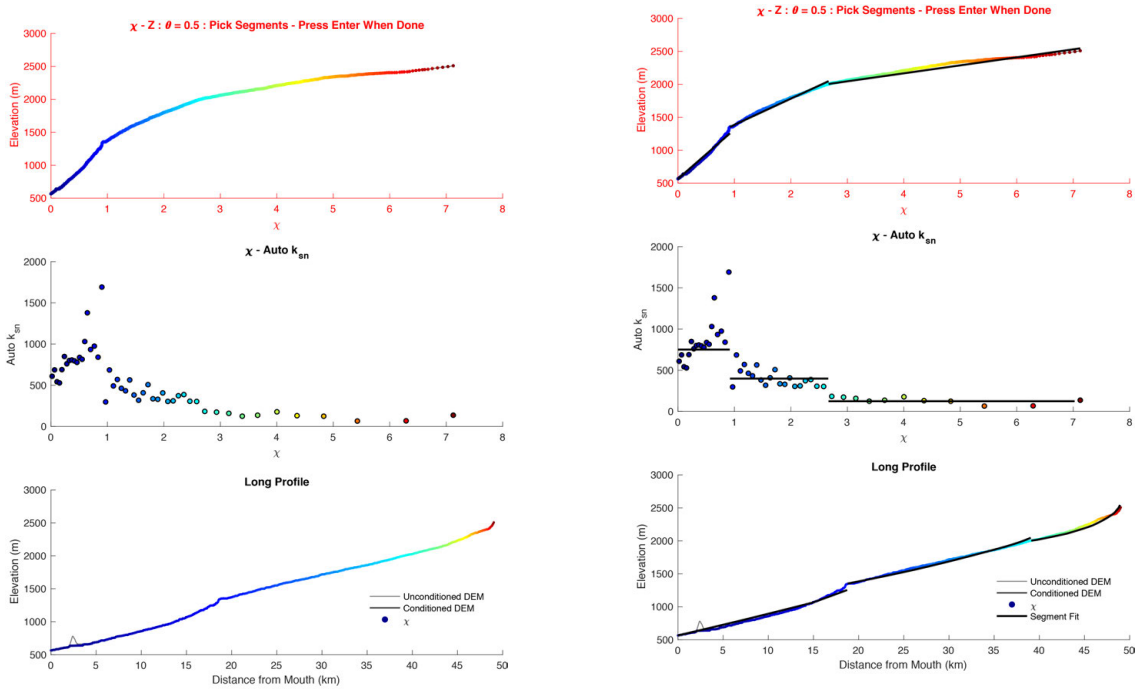
8.2.4 Restarting and Recovering from Errors

The *KsnProfiler* function is designed to be very robust in case of errors (both actual errors and user error). Throughout the normal operation of the function, the user is able to redo any step in the stream selection or fitting processes. While the function is running, a temporary mat file is also saved that stores all the values necessary to restart if the code fails for some reason. A failed run can be restart by using the *'restart'* optional parameter. The *'restart'* option can also be used to pick up where you left off from a user terminated session (i.e. you indicated to the function that you were done, but subsequently decided you wanted to keep fitting streams). Whether you are restarting a run because of failure or simply picking up where you left off, you do not need to recreate the exact call to the function as the function also saves a record of all parameter values and uses these saved values during a restart run (note though that you still need to provide the required datasets).

8.2.5 General Use

KsnProfiler can be run either with a user provided reference concavity or in a mode where the best fit concavity is found for each selected stream individually. Note that k_{sn} values can only be compared between streams if the same reference concavity is used, so it is generally recommended that unless you are explicitly only interested in the concavity of streams, that you leave the *concavity_method* set to the default *ref* mode. Regardless of your choice for *concavity_method*, a best fit concavity is calculated for each stream segment and is recorded in the outputs. During general use of *KsnProfiler*, for any stream that is selected, you will be prompted to select segment boundaries on a plot (Figure 11a).

The exact construction of the plot will depend on a variety of optional input parameters. You can choose segment boundaries on either a χ-elevation plot, a longitudinal profile plot, or a slope-area plot. In detail, regardless of the plot on which you choose to select segments, the calculation of best fit k_{sn} is performed on the χ-elevation relationship for that segment. To ensure that the fit is not biased by the spacing of χ values that vary as a function of drainage area, the fit is performed on a spline interpolated version of the χ-elevation relationship for the segment with equal point spacing in χ. The function also produces a residual plot to aid in your assessment of the goodness of fit (Figure 12).



(a) Main channel display before segment selection

(b) Main channel display after segment selection showing fit segments

Figure 11: Segment selection and fitting in *KsnProfiler*

.

For all options, a plot of the batch k_{sn} vs a relevant quantity (i.e. either χ, distance, or log area) will be displayed to aid you in picking out potential segments. The plot on which you need to make your selection will be highlighted with red axes, though for all options, the choice is recorded internally based on the position of the cursor in the x coordinate,

so in the example (Figure 11a) you could click on segment boundaries based on values of χ on either the χ-elevation or χ-Auto k_{sn} plots and produce an accurate choice. The order in which you select segment boundaries doesn't matter and the outlet and channel head are automatically considered segment boundaries (i.e. you don't need to define these as segment boundaries). If you do not click anywhere in the plot, the function will treat the selected stream as one segment and fit a single k_{sn} value to the entire stream segment. Once you are done selecting segment boundaries, the function will find the best fit k_{sn} for each segment and display these (Figure 11b).

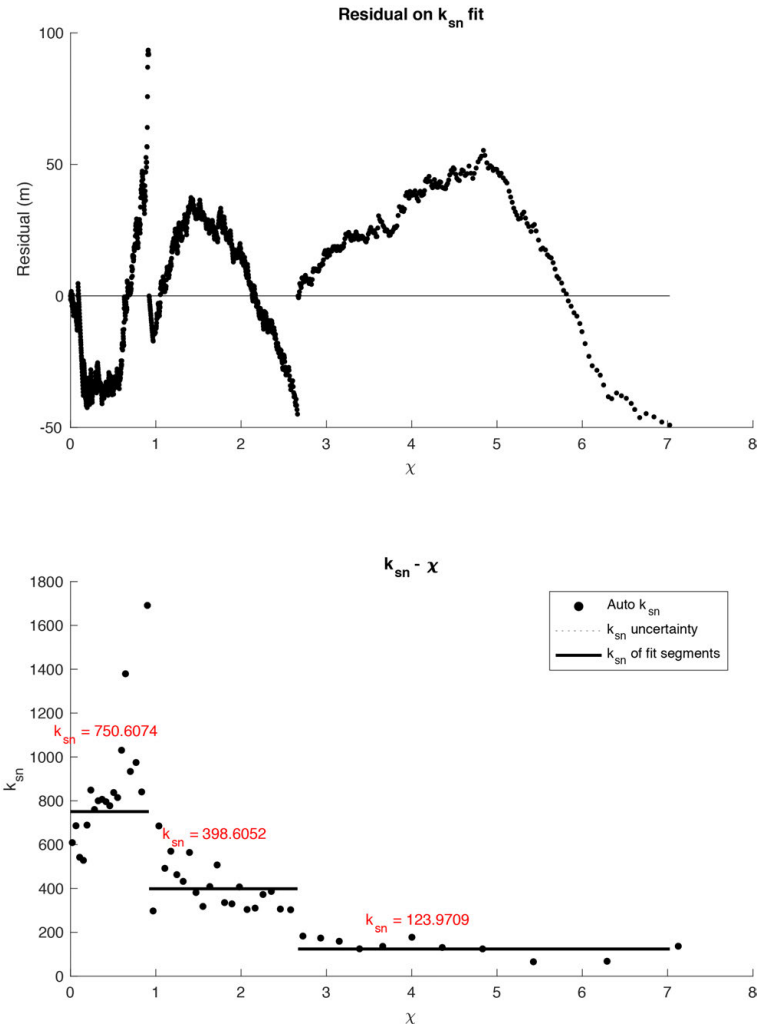


Figure 12: Residuals on k_{sn} fit. Note that uncertainty values on the k_{sn} fits are calculated and plotted, but in this example, the uncertainties are small enough that they are not distinguishable from the main k_{sn} lines.

8.2.6 Outputs

The *KsnProfiler* function produces four outputs to the workspace:

```
[knl,ksn_master,bnd_list,Sc]=KsnProfiler(DEM,FD,A,S);
```

where *knl* is an array of all stream locations with best fit k_{sn}, uncertainty values on this fit, concavity values, gradient, drainage area information, and reference ID numbers for the streams. This same output is also provided as a cell array, *ksn_master*, where individual streams are separated into cells. The segment boundaries are provided as an array of x, y, and z, locations in *bnd_list* and a *STREAMobj* of the selected streams is provided as *Sc*. Two shapefiles are produced, one as a polyline of the selected streams and containing all the information in the *knl* output and a point shapefile of segment boundaries (i.e. knickpoints) is also produced (assuming segment boundaries were selected for any stream). Depending on the setup of your *KsnProfiler* run, all of the plots generated during the fitting process may also be saved automatically.

8.3 *ClassifyKnicks*

The *ClassifyKnicks* is a companion function to *KsnProfiler* that allows you to iterate through all segment boundaries selected during a *KsnProfiler* run and provide a classification. This classification can either be numeric (e.g. 1, 2, 3, etc) or a character string (e.g., 'bound', 'slopebreak', 'knick', etc). If using character strings, these should be short (the shapefile format restricts entries in fields to 254 characters). The function will generate a new version of the knickpoint shapefile with this classification appended.

9 Basin Selection

9.1 *BasinPicker*

The *BasinPicker* function was originally designed to aid in the selection of sample locations for catchment averaged erosion rates (i.e. sand samples for ^{10}Be analysis), but can also be used as an interactive gateway to the *ProcessRiverBasins* function. *BasinPicker* takes the standard inputs:

```
[Outlets]=BasinPicker(DEM,FD,A,S);
```

and displays the DEM along with a map of local relief (radius of relief can be specified with optional *'rlf_radius'* parameter) and the provided stream network and prompts the user to choose a pour point / river mouth. The function will first confirm that you choose the correct portion of the stream network and then, if you answer in the affirmative, will display the plot of the χ-elevation and longitudinal profile for the streams within the selected basin and will also print out the mean local relief, mean channel steepness, and drainage area of the selected basin. The function will then ask if you wish to keep or discard this basin from the running list of outlets. The rationale being that if you are using this for sample site selection, you may not want to include basins that have major knickpoints, are below/above a particular drainage area, or do not meet some user defined morphometric criteria.

After each confirmed selection, *BasinPicker* will append this to an *'Outlets.mat'* file. If you run *BasinPicker* when an *'Outlets.mat'* file is in the active directory (or on your path), the code will attempt to populate the map with previous outlet selections. You can also provide an additional *GRIDobj* if you wish to consider an additional gridded dataset in selecting basin outlet locations:

```
PRECIP=GRIDobj('/Users/aforte/GISdata/precip.tif');
[Outlets]=BasinPicker(DEM,FD,A,S,'extra_grid',PRECIP);
```

The provided extra grid does not need to the same dimensions as the input DEM, etc, but it does need to be in the same projection. If you provide an extra grid, the mean values of this grid within the selected basin(s) will also be displayed in the plots.

10 Basin Average Maps and Plots

A major part of the *Topographic Analysis Kit* are tools designed for efficient selection and analysis of basin averaged data.

10.1 *ProcessRiverBasins*

10.1.1 Basic Operation

The workhorse function within the broader basin averaging set of tools is *ProcessRiverBasins*. *ProcessRiverBasins* was initially designed to efficiently clip out a series of watersheds from a larger DEM for use in ArcGIS, but has expanded much beyond that capability. The basic operation of *ProcessRiverBasins* requires the standard inputs along with a list of river mouths above which watersheds will be extracted. This list of river mouths can be provided as a Matlab array, e.g. the list of outlets output from *BasinPicker* is a valid input for *ProcessRiverBasins*:

```
% Using output of BasinPicker to run ProcessRiverBasins
load('Outlets.mat','Outlets');
ProcessRiverBasins(DEM,FD,A,S,Outlets,'basin_dir');
```

where *'basin_dir'* is the name of a folder (or full path of a folder) in which to store all of the datafiles that will be produced during a *ProcessRiverBasins* run. If the provided folder name does not exist, the function will create the folder. You do not have to use *BasinPicker* to generate the river mouth input, you only need to provide an array of x and y locations with ID numbers. Alternatively, you can provide a point shapefile where individual points are placed in locations where you wish there to be river mouth and each point has an identifying number (you can use the default ID number that ArcGIS will generate, but it is recommended that you make a separate field in your shapefile and manually provide ID numbers) :

```
% Using a point shapefile to run ProcessRiverBasins
ProcessRiverBasins(DEM,FD,A,S,'points_shape.shp','basin_dir');
```

As a caution, if you are going to use a GIS program (e.g. ArcGIS or QGIS) to select river mouth locations, it is strongly recommended that you use the stream shapefile output from *MakeStreams* as opposed to a stream network generated using flow routing in the GIS program of your choice. This is because flow routing algorithms vary slightly and thus absolute stream locations vary. Prior to the clipping process, *ProcessRiverBasins* will snap the provided river mouths to the provided stream network, so accidental selection of the wrong basin is possible if the selection of river mouths was done on an alternative stream network shapefile. Similarly, if you are using *ProcessRiverBasins* to clip out and calculate statistics on true sample locations (e.g. locations of detrital sediment samples as recorded by a GPS), it is again recommended that you ensure that these locations lie on the correct portion of the stream network generated by *MakeStreams* or incorrect basins may be clipped. **This may require moving true locations to lie on the correct flow routed stream!**

A final option for fully automated selection of river mouths allows you to provide an elevation as the river mouth parameter. This will place river mouths on the stream network at every location the stream network drops below this provided elevation:

```
% To create basins with outlets above 1000 meters elevation
ProcessRiverBasins(DEM,FD,A,S,1000,'basin_dir');
```

River mouths provided to *ProcessRiverBasins* can be non-nested or nested as each basin is processed independently of all other basins. If you're goal is to simply to generate a large dataset of arbitrary small basins within a landscape, we recommend placing river mouths at strategic locations (e.g. where streams exit a mountain range, Figure 13) and use *SubDivideBigBasins* to automatically subdivide basins as opposed to manually selecting large numbers of nested, sub-basins.

The function will use the river mouth locations to extract basins. Each basin will have its own basic TopoToolbox files along with various derived quantities, e.g. geographic data structure of k_{sn} using a reference concavity, geographic data structure of k_{sn} using a best fit concavity for that watershed (remember that k_{sn} values calculated with different concavities are not comparable when considering these outputs!), hypsometry and hypsometric integral for the basin, slope map, and statistics (mean, standard deviation or standard errors) on the majority of these quantities. Because it is time consuming, local relief is not calculated by default, but you can specify that you wish to calculate relief at a variety of radii using the *'calc_relief'* and *'relief_radii'* parameters:

```
% Calculate relief with a radius of 2500 m for all basins
ProcessRiverBasins(DEM,FD,A,S,Outlets,'basin_dir','calc_relief',true,'
    relief_radii',2500);
% Calculate relief with at 1000, 2500, and 5000 m radii for all basins
ProcessRiverBasins(DEM,FD,A,S,Outlets,'basin_dir','calc_relief',true,'
    relief_radii',[1000 2500 5000]);
```

If relief is calculated, statistics on these relief grids are also calculated. For each basin, a single .mat file is saved in the specified directory. The naming convention for these files is *'Basin_#_Data.mat'* where # is the ID number in the river mouth input for a given basin. It is important that you do not change the names of these files as this will break other functions that use these as inputs. By default, files suitable for use in a GIS program are not generated for each basin. You can have the function do this automatically for all basins by setting the optional *'write_arc_files'* to true or manually by using *Mat2Arc* for the desired basins.

10.1.2 Extra Grids

You can also provide an arbitrary number of extra grids to have *ProcessRiverBasins* clip and calculate basin averaged statistics for these grids. These extra grids do not need to be the same dimensions or cellsize as the input DEM, but they do need to be in the same coordinate system and projection. Also, if any provided extra grids are smaller than the DEM and any selected basin includes areas that are not covered by the extra grid, this will produce biased statistics. Extra grids are provided to the *ProcessRiverBasins* function as a 2 column Matlab cell array where the first column contains the *GRIDobj* and the second column includes a name for this grid in conjunction with the *'add_grids'* parameter:

```
% To run ProcessRiverBasins and include two extra grids
% Create the two GRIDobjs from data of interest
PRECIP=GRIDobj('/Users/aforte/GISdata/precip.tif');
NDVI=GRIDobj('/Users/aforte/GISdata/ndvi_grid.tif');
% Make an empty cell array
AG=cell(2,2);
% Populate cell array with necessary data
% Note that location in cell arrays are referenced with curly brackets
AG{1,1}=PRECIP;
AG{1,2}='precip';
AG{2,1}=NDVI;
```

```
AG{2,2}='ndvi_val';
% Run ProcessRiverBasins
ProcessRiverBasins(DEM,FD,A,S,Outlets,'basin_dir','add_grids',AG);
```

As a note, it is not recommended that you provide a local relief grid as an extra grid, but rather use the *'calc_relief'* option. The reason for this is twofold, 1) because local relief is calculated with a moving window, simply clipping out watersheds from a larger local relief raster will bias the statistics within the clipped watersheds as edge pixels will be influenced by neighboring basins and 2) subsequent codes explicitly look for the relief datasets in *ProcessRiverBasins* outputs to determine behaviors.

10.1.3 Categorical Grids

Some data that you may want to include in ProcessRiverBasins is not natively supported as a *GRIDobj*, specifically non-numeric data stored in polygonal shapefiles, e.g. units from a geologic map or vegetation types. You can provide this data as a usable input to ProcessRiverBasins via the optional *'add_cat_grids'* parameter and with the help of the *CatPoly2GRIDobj* function. The input to the *'add_cat_grids'* parameter is expected as a 3 column Matlab cell array, where the first column is the *GRIDobj*, the second column is the lookup table (both of these are generated by *CatPoly2GRIDobj*), and a name for the grid:

```
% To run ProcessRiverBasins an additional categorical grid
% Generate the categorical grid and lookup table with CatPoly2GRIDobj
[GEO,geo_table]=CatPoly2GRIDobj(DEM,'geo_polygons.shp','PTYPE');
% Make an empty cell array
ACG=cell(1,3);
% Populate cell array with necessary data
ACG{1,1}=GEO;
ACG{1,2}=geo_table;
ACG{1,3}='rock_type';
% Run ProcessRiverBasins
ProcessRiverBasins(DEM,FD,A,S,Outlets,'basin_dir','add_cat_grids',ACG);
% For simple inputs like above, you can do most of this in one line after
    generating GEO and geo_table with CatPoly2GRIDobj
ProcessRiverBasins(DEM,FD,A,S,Outlets,'basin_dir','add_cat_grids',{GEO
    geo_table 'geology'});
```

As with extra grids, more than one categorical grid can be provided, simply add additional rows to the input cell array. The remaining functions in this section all are either designed to operate on the products of *ProcessRiverBasins* or serve as helper functions for *ProcessRiverBasins*, e.g. *CatPoly2GRIDobj*. Because it is assumed that mean values of a category are not meaningful, the mode category value for each basin is found and used as the statistic. The function also calculates the number of pixels of each basin that belong to each category and saves this within the outputs.

10.1.4 Understanding Outputs

Each basin .mat file (or sub-basin file if considering the outputs of the related *SubDivideBigBasins*) contains a lot of outputs. For the most part, subsequent functions are designed to use and process these basin files, but in some cases you may wish to use products directly so we provide a list of all the outputs and a brief description of each below.

1. Default outputs regardless of optional parameters or datasets:
    - *RiverMouth* - array storing the x and y coordinate and ID number of the outlet of the basin

- *DEMcc* - clipped *GRIDobj* of the hydrologically conditioned DEM of the basin
- *DEMoc* - clipped *GRIDobj* of the original DEM of the basin
- *out_el* - scalar value of elevation of the outlet of the basin in map units
- *drainage_area* - scalar value of the drainage area of the basin in square kilometers
- *hyps* - two column array of the hypsometry of the basin, first column is relative frequencies, second column is elevation
- *FDc* - clipped *FLOWobj* of the basin
- *Ac* - clipped *GRIDobj* of the flow accumulation raster of the basin
- *Sc* - *STREAMobj* containing the full stream network of the basin
- *SLc* - *STREAMobj* containing the largest connected full stream network of the basin (should be identical to *Sc* in almost all cases).
- *Chic* - structure containing information regarding chi, direct output of the TopoToolbox function *chiplot*.
- *Goc* - clipped *GRIDobj* of the gradient of the basin
- *MSc* - geographic data structure of the stream network containing k_{sn} information using a best fit concavity
- *MSNc* - geographic data structure of the stream network containing k_{sn} information using the user supplied reference concavity
- *KSNc_stats* - 5 column array containing the mean k_{sn}, standard error on the mean k_{sn}, standard deviation on the mean k_{sn}, minimum k_{sn}, and maximum k_{sn} of the clipped basin.
- *Gc_stats* - 5 column array containing the mean gradient, standard error on the mean gradient, standard deviation on the mean gradient, minimum gradient, and maximum gradient of the clipped basin.
- *Zc_stats* - 5 column array containing the mean elevation, standard error on the mean elevation, standard deviation on the mean elevation, minimum elevation, and maximum elevation of the clipped basin.
- *Centroid* - two column array containing the weighted x and y coordinate of the center of the basin.
- *ChiOBJc* - clipped *GRIDobj* with χ values along the stream network as defined by *Sc*.
- *ksn_method* - character array indicating whether k_{sn} was calculated with the quick or tributary method
- *gradient_method* - character array indicating whether the gradient was calculated using *gradient8* or *arcslope*.
- *KsnOBJc* - clipped *GRIDobj* with interpolated k_{sn} values for the entire watershed, note construction of this can sometimes fail for small basins so this may not appear in all basin files, but the user will be warned that particular basins do not included this output if that is the case.
- *theta_ref* - scalar value recording the reference concavity used for calculating k_{sn} and χ.

2. Outputs included if relief is calculated using the optional *'calc_relief'* parameter:

   - *rlf* - 2 column cell array where the first column contains a *GRIDobj* of local relief calculated for the clipped basin and the second column contains a scalar recording the relief radius used. This cell array will have the same number of rows as the number or relief radii provided to *ProcessRiverBasins*.
   - *rlf_stats* - a 5 column array containing the mean local relief, standard error on the local relief, standard deviation on the local relief, minimum local relief, and maximum local relief of the clipped basin. This array will have the same number of rows as the number of relief radii provided to *ProcessRiverBasins* and will be in the same order as the *rlf* output.

3. Outputs included if extra grids are provided with the optional *'add_grids'* parameter:

   - *AGc* - 2 colum cell array where the first column contains a clipped *GRIDobj* of the additional grid(s) provided to *ProcessRiverBasins* and the second column is the character string that is the name of the additional grid provided by the user in argument for the *'add_grids'* parameter. The cell array will have the same number of rows as the numer of additional grids provided to *ProcessRiverBasins*.
   - *AGc_stats* - 5 column array containing the mean of the additional grid, the standard error on the mean of the additional grid, the standard deviation on the mean of the additional grid, the minimum of the additional grid, and the maximum of the additional grid of the clipped basin. This array will have the same number of rows as the number of additional grids provided to *ProcessRiverBasins* and will be in the same order as the *AGc* output.

4. Outputs inlcuded if categorical grids are provided with the optional *'add_cat_grids'* parameter:

   - *ACGc* - 3 column cell array where the first column contains a clipped *GRIDobj* of the additional categorical grid(s) provided to *ProcessRiverBasins*, the second column is the look up table for the relevant categorical grid provided to *ProcessRiverBasins*, and the third column is the name of the categorical grid provided by the user in argument for the *'add_cat_grids'* parameter. The look up table included in the *ACGc* output will have an extra column named 'Counts' which will include the number of pixels within the clipped categorical grid that below to each category. The cell array will have the same number of rows as the number of additional categorial grids provided to *ProcessRiverBasins*.
   - *ACGc_stats* - one column array containing the mode of the clipped categorical grid. This array will have the same number of rows as the number of additional categorical grids provided to *ProcessRiverBasins* and will be in the same order as the *ACGc* output.

10.2 *CatPoly2GRIDobj*

As discussed in section 10.1.3, sometimes it is helpful to have categorical data stored as a *GRIDobj*, but *GRIDobj*s do not natively support non numeric values for cells. *CatPoly2GRIDobj* is designed to get around that by generating a valid *GRIDobj* with numeric values replacing categorical values and bundling this with a lookup table that serves as a key for the numeric values in the *GRIDobj*. *CatPoly2GRIDobj* requires a DEM input along with the name of a valid polygonal shapefile that contains the categorical data of interest and the name of the field within that shapefile that contains the specific data.

```
% Generate a categorical grid and lookup table for a geologic map for the rock
    type field with the name 'PTYPE' in the input shapefile
[GEO,geo_table]=CatPoly2GRIDobj(DEM,'geo_polygons.shp','PTYPE');
```

The output *GRIDobj* will have the same dimensions as the input DEM. The provided shapefile does not need to have the same dimensions (it can be larger or smaller than the input DEM), but it does need to be in the same coordinate system and projection. *CatPoly2GRIDobj* can only handle one field within a shapefile, so if you have a shapefile that has multiple fields you wish to convert to *GRIDobj*s, you will need to run *CatPoly2GRIDobj* multiple times. Any node of the input DEM that does not have categorical values associated with it will be set to '0' in the *GRIDobj*, which will correspond to a value of 'undef' in the lookup table. As a note, this function relies heavily on the underlying TopoToolbox function *polygon2GRIDobj*. At the time of writing, *polygon2GRIDobj* sometimes has issues dealing with nested or overlapping polygons that will result in areas being set to '0' and 'undef'.

10.3 *SubDivideBigBasins*

The *SubDivideBigBasins* function is designed to automatically divide up basins output from *ProcessRiverBasins* greater than a user specified drainage area. There are a variety of different methods for subdividing basins, including on the basis of stream order (i.e. outlets of streams of a user specified order within a basin are used as new river mouths above which to extract new sub-basins), confluences or upstream confluences (i.e. the former using the confluence as a river mouth, the latter using points immediately upstream of a confluences as river mouths), filtered confluences (i.e. using confluences as river mouths if those confluence points are above a user defined drainage area or percentage of the input basin drainage area), confluence points with trunk stream (i.e. river mouths will be points immediately upstream of every confluence with the main trunk within each specified basin), or filtered confluences with the trunk stream. Some examples:

```
% Subdivide basins stored in the 'basin_dir' folder that are greater than 100
    square kilometers in drainage  on the basis of stream order with outlets of
     second order streams serving as river mouths
SubDivideBigBasins('basin_dir',100,'order','s_order',2);
% Subdivide basins on the basis of confluences where confluences are areas
    with draiange areas greater than 5 square kilometers in drainage area
SubDivideBigBasins('basin_dir',100,'filtered_confluences','min_basin_size',5);
% Subdivide basins on the basis of trunk confluences greater than 5 square
    kilometers for any main basin greater than 50 square kilometers
SubDivideBigBasins('basin_dir',50,'filtered_trunk','min_basin_size',5);
```

These different subdivision schemes will result in the selection of different sub-basins (Figure 13) which in turn will result in different populations of basin statistics (Figure 15). Identical statistics and operations are performed on each sub-basin as were for the main basins produced by *ProcessRiverBasins*.

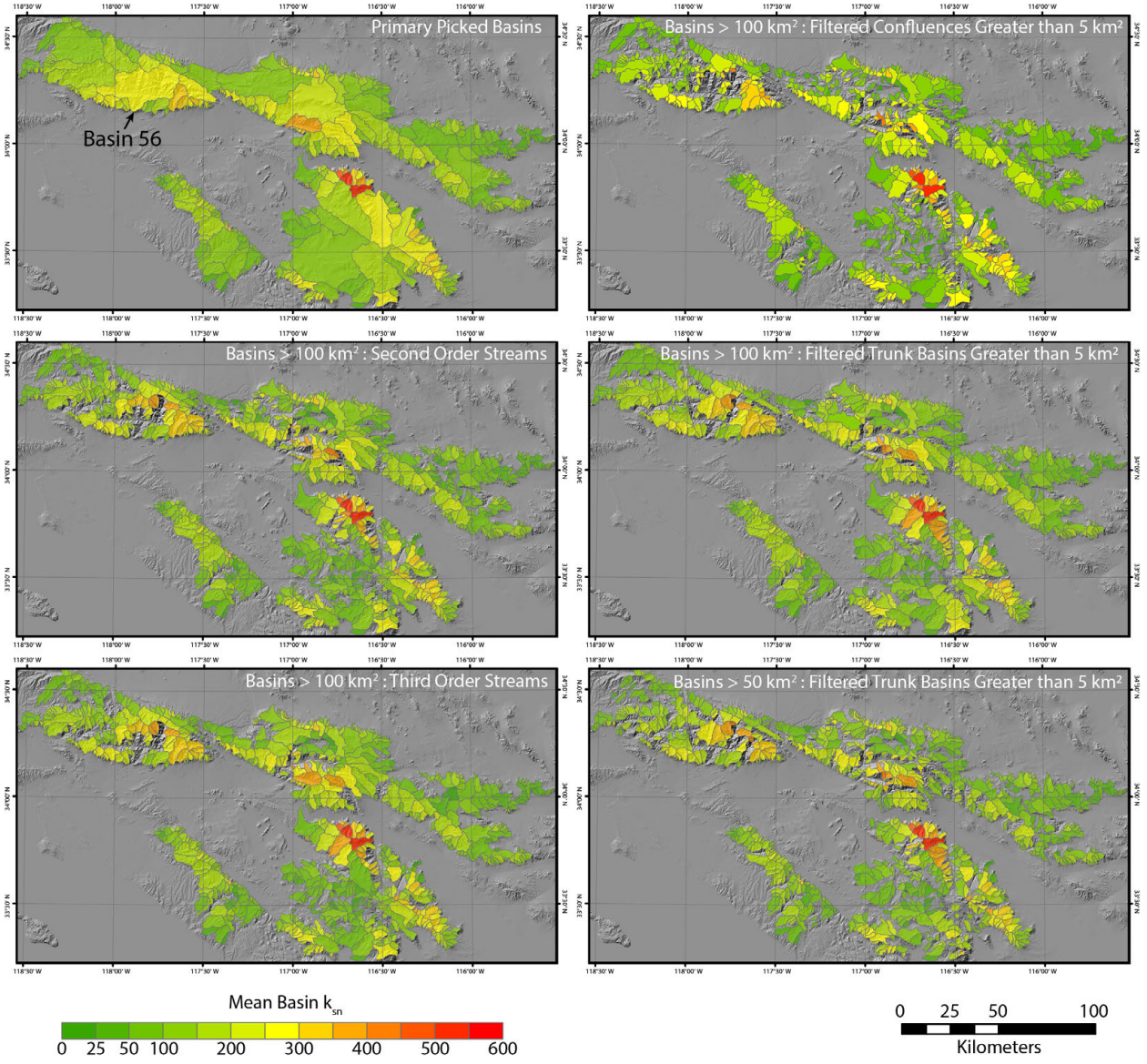


Figure 13: Comparison of mean k_{sn} values calculated for manually selected basins (upper left panel) from *ProcessRiverBasins* and for a variety of sub-basin division schemes using *SubDivideBigBasins*.

We refer you to the function header for a full list of optional parameters for running *SubDivideBigBasins*, but it is important to highlight the *'SBFiles_Dir'* optional parameter. This parameter allows the user to specify the name of the folder that will contain the sub-basin mat files and is set to 'SubBasins' by default. This folder will be placed within the main basin directory (specified with required *'basin_dir'* parameter). If you want to generate different sets of sub-basins based on different subdivision schemes, it is important to provide unique folder names in which to place your different sub-basin datasets, otherwise overwriting and mixing of different subdivision schemes may occur. Sub-

basin mat files are saved as *'Basin_#_DataSubset_##.mat'* where # is the original basin number that is being subdivided and ## is a sequential number based on the number of sub-basins produced within that main basin. It is important that you do not change these file names as this will cause other functions which use these as inputs to fail or behave unexpectedly. River mouth IDs for sub-basins are assembled by appending the sequential number of the sub-basin as at least a 3 digit number (i.e. with the appropriate number of leading zeros) to the main basin number. For example, river mouth ID for the 7th sub-basin of basin number 50 will be 50007, where as the 14th sub-basin for basin number 156 will be 156014. If there are more than 999 sub-basins, the function will still work fine, e.g. the river ID number for the 1200 sub-basin of basin number 156 would be 1561200.

10.4 *FindBasinKnicks*

FindBasinKnicks allows you to find, mark, and optionally classify knickpoints within a basin file produced by *ProcessRiverBasins* or *SubDivideBigBasins*. Function will iterate through each stream in a given basin and allow the user to select knickpoint locations on a χ-elevation plot. If you set the optional *'classify_knicks'* parameter to true, you can also input a classification for each selected knickpoints. As with the *ClassifyKnicks* function for use with *KsnProfiler*, classification can be numeric or short character strings. The *FindBasinKnicks* function will output a Matlab table with location information for the selected knickpoints and optionally produce a shapefile by providing a valid character string to the optional *'shape_name'* parameter.

10.5 *PlotIndividualBasins*

PlotIndividualBasins will iterate through a directory containing basin files output from *ProcessRiverBasins* and produce a three panel plot displaying the χ-elevation, longitudinal profile, and slope-area relationships for each basin. Each plot will be titled with the basin number and the drainage area of the basin. If you wish to produce plots for sub-basins, you must provide the location of the sub-basins of interest with the optional *'location_of_subbasins'* parameter. Note that if a valid sub-basin directory is supplied, plots for a subdivided main basin will not be produced, i.e. if main basin 100 was subdivided, a plot for this basin will not be produced, but plots for all of its subbasins will be produced. If you want to produce plots for all main basins and sub-basins, you will need to run the code twice, one time providing a folder name for *'location_of_subbasins'* that is either empty or doesn't exist.

10.6 *Basin2Shape*

The *Basin2Shape* function will take the products of *ProcessRiverBasins* and *SubDivideBigBasins* and produce a single polygon shapefile containing all the basins and a variety of fields. The only required inputs are the original DEM provided to *ProcessRiverBasins* and the location of the outputs of *ProcessRiverBasins*. The function provides three options for which basins are included in the shapefile (depending on whether you ran *SubDivideBigBasins* or not) via the optional *'include'* parameter. Similar to other functions, if you wish to include subdivided basins, you must provide the name of the sub-basin folder. Assuming you ran both *ProcessRiverBasins* and *SubDivideBigBasins* the options are:

```
% To include only the original products of ProcessRiverBasins
[MS]=Basin2Shape(DEM,'basin_dir','include','bigonly');
% To include all basins, meaning that even the main basins that were
    subdivided will be inlcuded
[MS]=Basin2Shape(DEM,'basin_dir','include','all','location_of_subbasins','
    my_sub_basins');
 % To include non-subdivided basins and subdivided basin, meaning that main
     basins that were subdivided will  NOT be included
```

```
[MS]=Basin2Shape(DEM,'basin_dir','include','subdivided','
    location_of_subbasins','my_sub_basins');
```

The function will include information from the various statistics (e.g. mean elevation, mean k_{sn}, mean gradient) and will also include uncertainty values of the users choice (i.e. either standard deviations, standard errors, or both) on these statistics. The code detects whether statistics exist for relief or any extra grids and will include those as well.

If categorical grids were provided, the function will produce fields reporting the modal value for these but will save the modal values as the original categories (e.g. if rock types from a geologic map were provided and the most prevalent unit in a given basin is 'granite' then within the relevant field for that basin). The function will also produce a field to record what percentage of the basin is occupied by that modal category. There is also an optional parameter *'populate_categories'* that if set to true will produce a field for each category in the original categorical grid and for each basin will record the percentage of the basin occupied by that category.

With the optional *'extra_field_values'* and *'extra_field_names'* parameters, you can add additional values to the shapefile. The input to *'extra_field_values'* is expected as a cell array with a row for each basin (not an array within a cell array row) with the first column being the ID number of that basin (i.e. the third column in the river mouth array) and subsequent columns containing the values of interest you wish to add to the shapefile. Every basin must have a value for all extra fields or the code will produce an error. The cell array containing IDs and values of interest does not need to be in any particular order, the function will use the basin ID number to match the relevant data with the correct basin. The *'extra_field_names'* parameter expects a cell array where the columns contain character strings that will serve as the field names in the shapefile. These are expected to be in the same order as the columns in *'extra_field_values'* and should have one fewer column than the *'extra_field_values'* array (i.e. you do not include a field name for the ID number). For example, if for each watershed you had a text file containing the river mouth ID numbers, sample names, erosion rates, and uncertainty on erosion rates with the headers named 'ID', 'sample', 'E_rates', and 'E_unc' you could add these to the shapefile via the following:

```
% Load in your text file into a table specifying that it has a header
    containing the variable names, this is the easiest way to import mixed data
     containing both characters and numbers
T=readtable('SampleTable.txt','ReadVariableNames',true);
% The 'sample' column will already be a cell array because it contains
    characters, but the other columns will be numeric arrays so you will need
    to convert them to cell arrays and then horizontally concatenate them
    together
EFVal=horzcat(num2cell(T.ID),T.sample,num2cell(T.E_rates),num2cell(T.E_unc));
% This should produce a cell array with 4 columns and as many rows as there
    are basins
% Create the names for the extra fields
EFName{1,1}='sample';
EFName{1,2}='E_rate';
EFName{1,3}='E_unc';
% Run Basin2Shape
[MS]=Basin2Shape(DEM,'basin_dir','extra_field_names',EFName,'
    extra_field_values',EFVal);
```

Basin2Shape will output a geographic data structure that is suitable for production of a shapefile:

```
[MS]=Basin2Shape(DEM,'basin_dir');
shapewrite(MS,'shape_name.shp');
```

but the function will automatically produce a shapefile saved in the active directory unless you suppress this with the optional *'suppress_shape_write'* parameter.

10.7 *Basin2Raster*

The *Basin2Raster* is a simpler alternative to *Basin2Shape* that will output a *GRIDobj* and save out an ascii raster of the basin extents filled with values specified by the user. Possible values are mean k_{sn}, mean gradient, mean elevation, mean relief (you must specify which radius you want to use), R^2 value on the χ-elevation relationship (this can be a useful metric for assessing how well adjusted a basin is, R^2 values closer to 1 should have fewer significant deviations from a linear χ-elevation relationship, though this will also be influenced by the reference concavity used for calculating χ), drainage area in km^2, hypsometric intergral, basin ID number, best fit concavity, or the name of any additional grid or additional categorical grid originally provided to *ProcessRiverBasins*. If the name of an additional grid is provided, value will be the basin mean and if the name of a categorical grid is provided , value will be the basin mode.

If the basins provided to *Basin2Raster* were manually nested (i.e. you provided a list of river mouths to *ProcessRiverBasins* that included nested catchments and did not run *SubDivideBigBasins*), then you should set the optional *'method'* parameter to *'nested'* to ensure that all basins are visible in the output raster. If you did not manually nest any basins or ran *SubDivideBigBasins*, then you do not need to provide an argument to *'method'*.

10.8 *CompileBasinStats*

CompileBasinStats is desgined to aggregate information from a population of basins output from *ProcessRiverBasins* and *SubDivideBigBasins* either for export or for use with *BasinStatsPlots* or *MakeCombinedSwath*. The output of *CompileBasinStats* is a Matlab table, which can be exported easily as a textfile:

```
% Generate basin stats table
[T]=CompileBasinStats('basin_dir');
% Export table
writetable(T,'basin_stats.txt');
```

The inputs for and behavior of several optional parameters for *CompileBasinStats* are identical to those for *Basin2Shape* (e.g. including extra fields, controlling which basins are included in the output, specifying which type of uncertainty to include) and the Matlab table that *CompileBasinStats* outputs shares some similarities with the geographic data structure output by *Basin2Shape*, but *CompileBasinStats* has several additional options that *Basin2Shape* does not, mostly related to to categorical grids.

10.8.1 Recalculating Means Based on Categories

If categorical grids were provided to *ProcessRiverBasins*, then particular categories can be used to recalculate means within the basins based on subsets of these categories. Specifically, means can be recalculated by either applying an inclusive or an exclusive filter. For example, if you created a categorical grid for the rock type from a geologic map, you could use this functionality to recalculate means only for parts the landscape occupied by particular units (i.e. an inclusive filter) or recalculate means for the entire landscape except those covered by particular units (i.e. an exclusive filter):

```
% To calculate means only from portions of a basin defined by certain rock
    types
[T]=CompileBasinStats('basin_dir','filter_by_category',true,'filter_type','
    include','cat_grid','geology','cat_values',{'pCc','grMz','pC','gr-m','Pc','
    grPz','grpC','gr'});
% To calculate means from all portions of basins except where the rock types
    are certain types
[T]=CompileBasinStats('basin_dir','filter_by_category',true,'filter_type','
    exclude','cat_grid','geology','cat_values',{'Q','Qpc','Qg','Qls','Qs','Qv',
    'water','undef'});
```

where the argument provided to *'cat_grid'* is the name of the additional categorical grid of interest as originally supplied to *ProcessRiverBasins*, *'cat_values'* is a cell array containing the list of categories to use in the filter, and *'filter_type'* is either *'include'* or *'exclude'* depending on desired behavior.

Recalculated means will be produced for properties for which means were originally calculated (i.e. k_{sn}, elevation, gradient, relief (if it was calculated), and any additional grids (if provided)). Names for these recalculated means will appear in the table with '_f' appended to the column name, e.g. 'mean_ksn_f'. The original mean values calculated for the entirety of each basin will still appear in the table.

10.8.2 Populating Categories

Setting the optional *'populate_categories'* parameter to true will result in a column for each category in the provided categorical grids where the value in that column for a given basin is the percentage of pixels in that basin which belong to that category. For example, if the categorical grid was a geologic map with rock types, this would mean that there would be a column for each rock type and an entry for what percentage of each basin was covered by that rock type.

10.8.3 Means by Category

This functionality allows you to calculate means of particular metrics within each category. Using the rock type example, this would allow you to calculate things like mean k_{sn} within each rock type, within each basin. If a basin is not covered by a particular category, then the value for that category mean will be 'NaN'. To calculate means by category, you can use the *'means_by_category'* parameter:

```
% To calculate means using the geology additional grid for ksn and 2500 m
    local relief
[T]=CompileBasinStats('basin_dir','means_by_category',{'geology','ksn','
    rlf2500'});
```

where the first column in the cell array provided to *'means_by_category'* must be the name of the categorical grid provided to *ProcessRiverBasins* (in the example, *'geology'*) followed by valid names of particular quantities (see header of the function for full list).

10.9 *BasinStatsPlots*

Using the Matlab table output from *CompileBasinStats*, you can quickly plot a variety of basin averaged statistics, but we have written the *BasinStatsPlots* function to aid in you producing some potentially useful plots. These plot types are designed to allow you to explore populations of basins quickly and efficiently and are envisioned as primarily data exploration tools (i.e. you will probably want to design your own functions to analyze and plot data output from

'CompileBasinStats', with these plots as easy ways to guide further analysis). In the following sections, we describe the different plotting options available to you in *BasinStatsPlots* and provide some examples of outputs using the example dataset.

10.9.1 Basic Options

There are several options that span across multiple (or all) types of plots. If you wish to subset the basins by their geographic location (i.e. you only want to generate a plot for basins within a specific area) you can use the *'define_region'* parameter to either define a region based on input coordinates or, if you just set the *'define_region'* parameter to true, interactively select a bounding box. Note that the function uses the basin centroid as the location for determining whether a basin lies within or outside of the provided coordinates.

Many of the plots will attempt to plot uncertainty values as whiskers on points. You can control which estimation of uncertainty (standard error or standard deviation) is used or suppress drawing uncertainty whiskers with the *'uncertainty'* parameter. Some of the plots also allow you to color the points by a specified quantity using the *'color_by'* parameter (e.g. Figure 14) which expects the valid name of a column in the input table as a string:

```
% To color a plot of mean gradient vs mean ksn by mean precipitation, and
    assuming the mean precipitation is stored in a column named 'mean_precip'
    in the input table, T
BasinStatsPlots(T,'grd_ksn','color_by','mean_precip');
```

The *'color_by'* parameter is a valid input for Mean Gradient vs Mean k_{sn} - *'grd_ksn'*, Mean Gradient vs Mean Relief - *'grd_rlf'*, Mean Relief vs Mean k_{sn} - *'rlf_ksn'*, and Generic X-Y plot - *'xy'* plot types. You can control the colormap used for coloring points with the *'cmap'* parameter which expects the name of a valid colormap (e.g. 'jet'), the name of a function that produces a valid colormap (e.g. *ksncolor*) or a nx3 array of RGB values usable as a colormap.

Figures created by the function can be automatically saved using the *'save_figure'* parameter.

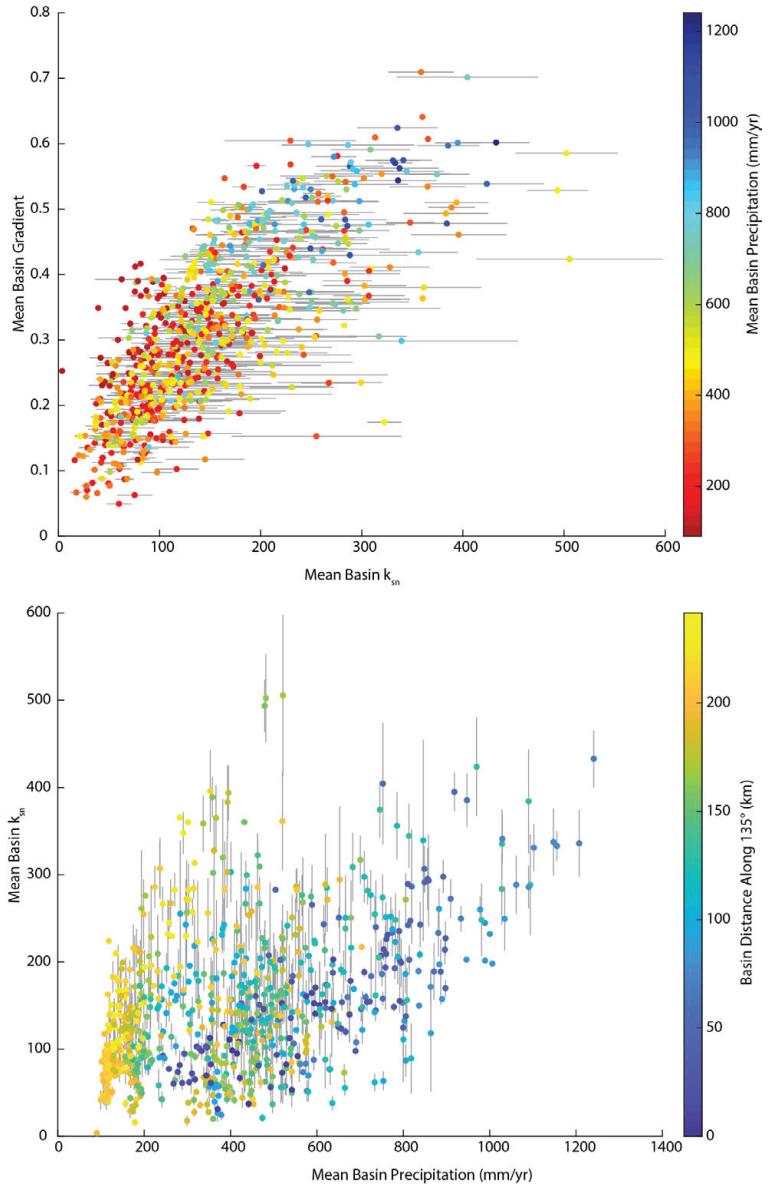


Figure 14: Selected outputs of *BasinStatsPlots* showing some options for plots using the *'color_by'* optional parameter.

10.9.2 Mean Gradient vs Mean k_{sn} - *'grd_ksn'*

Both mean hillslope gradient and mean k_{sn} have previously been shown to correlate to mean erosion rate and can be thought of as representing the hillslope and fluvial contributions, respectively, to the erosion rate. Thus, in the absence of quantitative erosion rate data for a portions of basins, there can be utility in examining the relationship between mean hillslope gradient and mean k_{sn}, see Forte et al., 2016 for a full discussion of the nuances of interpreting these relationships. The *'grd_ksn'* plot option will plot these two quantities against each other (Figure 15).

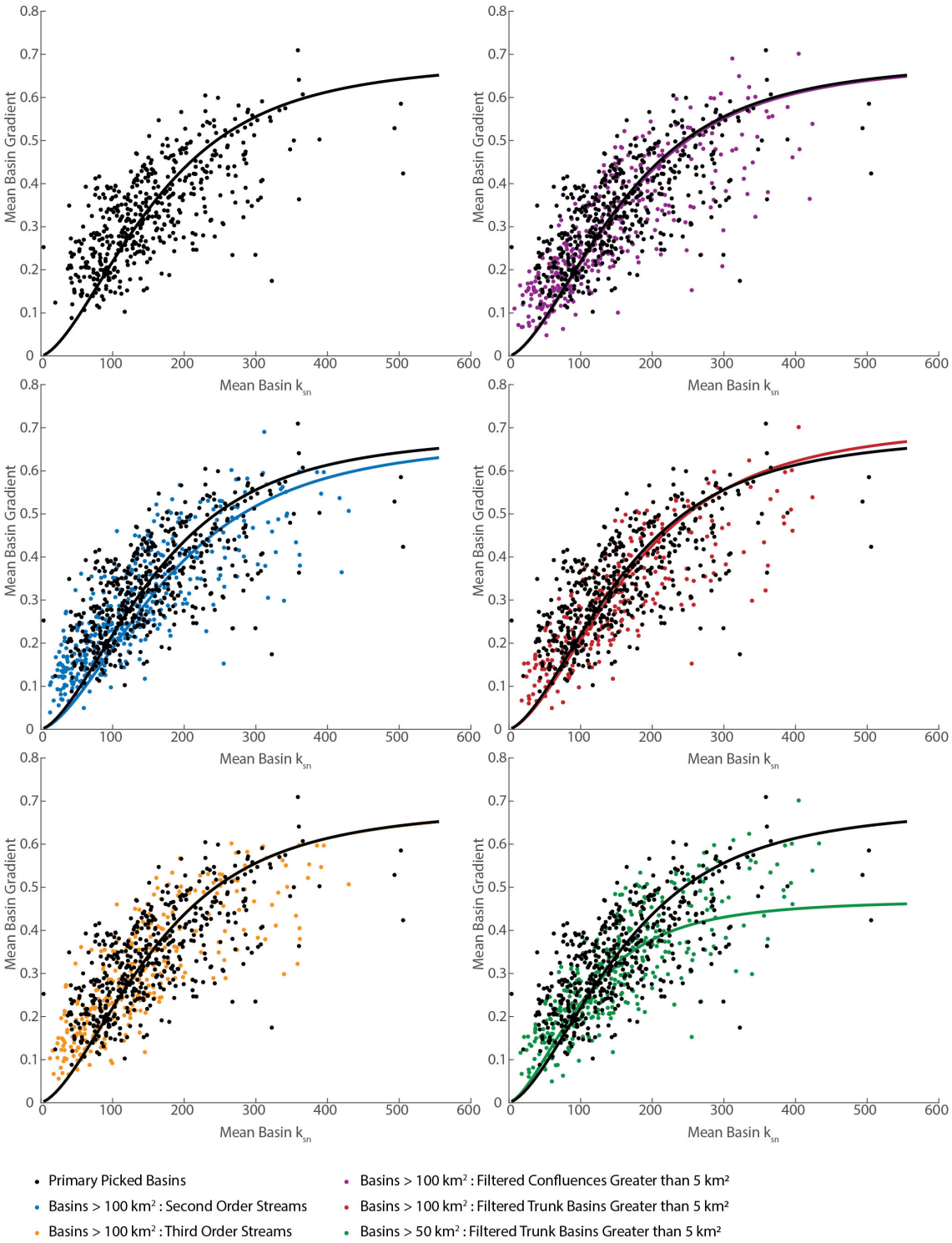


Figure 15: Comparisons of mean k_{sn} vs mean basin gradient and associated fits for a variety of sub-basin division schemes using *SubDivideBigBasins* using the *'grd_ksn'* plot from *BasinStatsPlots*. See Figure 13 for a map view of these different basin division schemes.

Via the optional *'fit_grd_ksn'* parameter, you can also use the relationship between to mean hillslope gradient and mean k_{sn} to estimate some morphometric parameters for the populations of basins, specifically the erosional efficiency, hillslope diffusivity, and limiting gradient. The fits are done via a constrained minimization of least absolute deviation on a relatively complicated equation which means that to successfully run the fits, you need access to the Optimization Toolbox in Matlab and that the fits are sensitive to the choice of initial parameters, which you can control with the optional *'start_diffusivity'*, *'start_erodibility'*, and *'start_threshold_gradient'* parameters. Also important is the optional *'n_val'* parameter (equivalent to the slope exponent in the stream power model of incision) which is not a free parameter in the fits, but will control the shape of the fluvial portion of the relationship. We refer readers who are more interested in the nuances of this fit and the (large) assumptions that go into it to Forte et al., 2016 and references therein. Exploring the shape of this relationship (and thus the estimated values for the parameters that control it) also highlights that choosing different methods of subdivision of basins using *SubDivideBigBasins* can produce statistically different populations of basins (e.g. Figure 15).

10.9.3 Mean Gradient vs Mean Relief - *'grd_rlf'*

You can also plot mean gradient vs. mean local relief via the *'grd_rlf'* plot option. This should be similar to the result of Mean Gradient vs Mean k_{sn} - *'grd_ksn'* plot as mean local relief and mean k_{sn} are typically linearly related (depending on the relief radius). You can test the nature of the relationship between mean local relief and mean k_{sn} with the Mean Relief vs Mean k_{sn} - *'rlf_ksn'* plot option. The *'grd_rlf'* plot will use the relief radius specified with the optional *'rlf_radius'* parameter. If this specifies a relief radius that you did not calculate when running *ProcessRiverBasins*, this will result in an error.

10.9.4 Mean Relief vs Mean k_{sn} - *'rlf_ksn'*

Depending on the relief radius, the relationship between mean local relief and mean k_{sn} should be linear, but you can quickly test this with the *'rlf_ksn'* option which will plot these two quantities against each other. The *'rlf_ksn'* plot will use the relief radius specified with the optional *'rlf_radius'* parameter. If this specifies a relief radius that you did not calculate when running *ProcessRiverBasins*, this will result in an error.

10.9.5 Comparing Filtered and Non-Filtered Means - *'compare_filtered'*

If you calculated filtered means when running *CompileBasinStats*, the *'compare_filtered'* option will plot the filtered means against the unfiltered means and visually highlight basins that imply higher filtered values compared to non-filtered with red dots and basins that imply lower filtered values compared to non-filtered with blue dots (e.g. Figure 16). This plot option will produce an individual plot for any quantity for which a mean and a filtered mean was calculated (obviously, if filtered means were not calculated, this option will produce an error). In the example, we present examples of mean k_{sn} and mean gradient in the case of a filter excluding quaternary units from the means and an inclusive filter calculating means from granites, metamorphic rocks, and other 'basement' rocks (Figure 16). This allows you to assess relatively quickly that removing areas covered by quaternary units shifts means toward higher values, where as only calculating means based on 'hard rocks' also shifts many basins to higher means values, but not in all cases. This will of course depend on the nature of the filter and like the majority of the plots, this is designed as a data exploration tool to determine what aspects warrant deeper analysis.

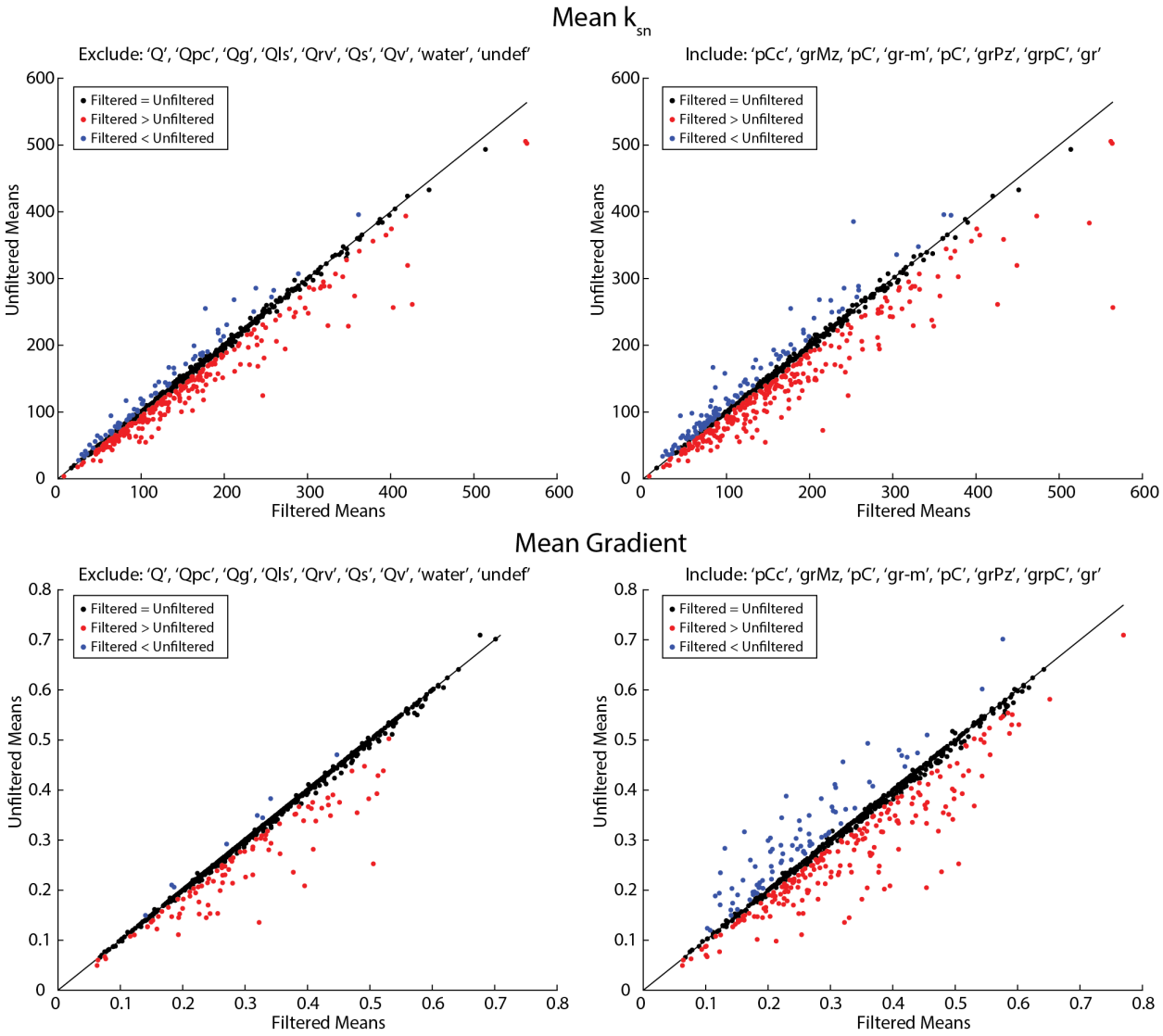


Figure 16: Selected outputs of *'compare_filtered'* plot from *BasinStatsPlots* comparing results of an excluding quaternary, water, and undefined areas vs including basement rocks.

10.9.6 Histograms of Category Means - *'category_mean_hist'*

If you calculate means by categories when running *CompileBasinStats*, you can use the *'category_mean_hist'* option to plot histograms of the means across all basins within given categories for a statistic of interest. Using this option requires providing an argument to the optional *'cat_mean1'* parameter to specify which statistic (e.g. mean k_{sn}) for which you wish to produce histograms.

10.9.7 Comparisons of Category Means - *'category_mean_compare'*

If you calculate means by categories when running *CompileBasinStats*, you can use the *'category_mean_compare'* option to compare two statistics of interest within the categories (e.g. plots of mean basin k_{sn} vs mean gradient as a function of rock type). Using this option requires providing arguments to the optional *'cat_mean1'* and *cat_mean2'* parameters to specify which statistics you want to compare.

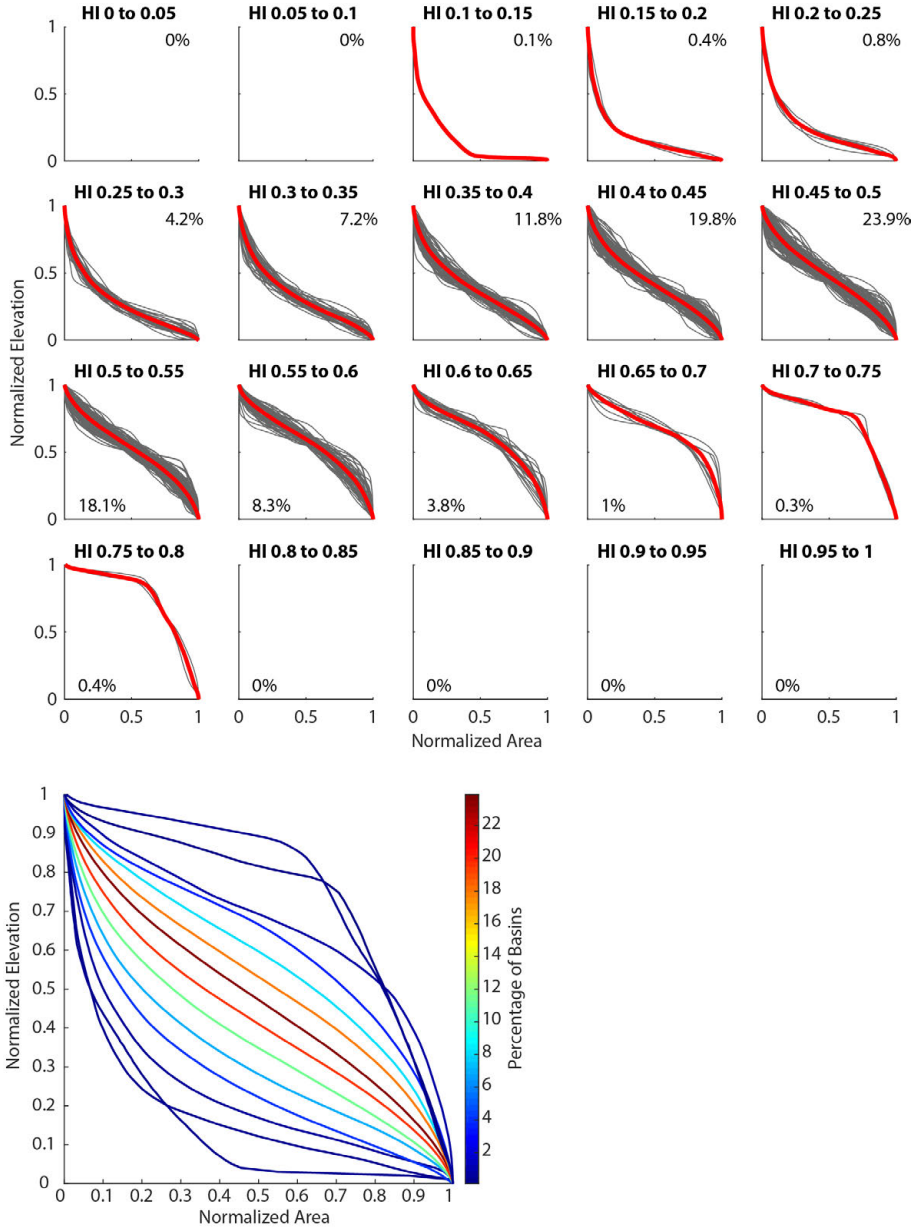


Figure 17: Selected output of *'stacked_hypsometry'* plots from *BasinStatsPlots*.

10.9.8 Basin Hypsometry - *'stacked_hypsometry'*

The hypsometry of a basin (e.g. the empirical cumulative distribution of elevations in that basin) and the hypsometric integral (e.g. the area underneath a normalized version of this curve) are classic morphometric parameters, which while simplistic, can still be useful in understanding first order characteristics of a landscape. The *'stacked_hypsometry'* plot option will produce a series of 5 plots, a subset of which are shown in Figure 17. This option will plot figures of stacked normalized hypsometries (not included in Figure 17), a version of the stacked normalized hypsometries displayed as heatmap (not included in Figure 17), stacked hypsometries with elevations vs percentage area (not included in Figure 17), a grid of normalized hypsometries with the mean hypsometry binned by the hypsometric integral with the percentages of the basins within that bin (top portion of Figure 17), and a plot of the mean hypsometries in each hypsometric integral bin and colored by the percentage of basins within that bin (bottom portion of Figure 17).

10.9.9 Comparing Distribution of Basin Means vs All Nodes -*'compare_mean_and_dist'*

The *'compare_mean_and_dist'* is designed to explore the distributions of particular statistics within the basins. This option can function in two ways. The first way is by comparing the distribution of a statistic of interest (e.g. mean gradient) across all nodes in all basins compared to the basin means across all basins (Figure 18). This requires that you specify a statistic to compare using the optional *'statistic_of_interest'* parameter.

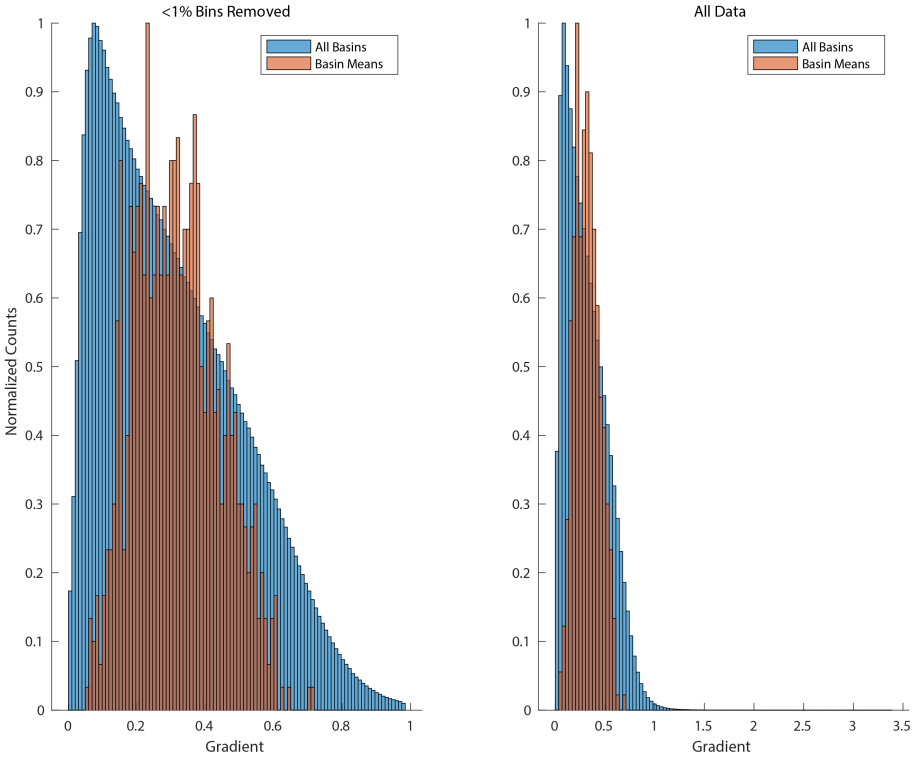


Figure 18: Sample output of *'compare_mean_and_dist'* plot from *BasinStatsPlots* comparing the distribution of gradient across all nodes in all basins (blue) to the distribution of mean gradients across all basins (orange).

The other way the *'compare_mean_and_dist'* plot option can be used is to explore the distribution of the statistic of interest of a particular basin compared to the mean value (Figure 19). This requires that you specify a statistic to compare using the optional *'statistic_of_interest'* parameter and specify a basin of interest by the basin ID number with the *'basin_num'* parameter.

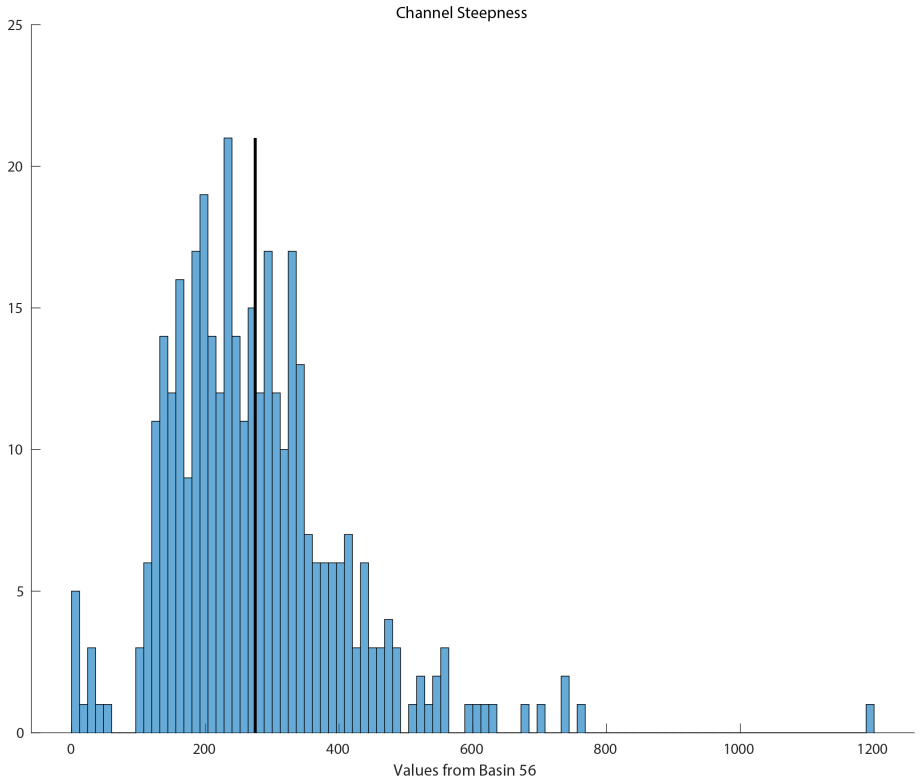


Figure 19: Sample output of *'compare_mean_and_dist'* plot from *BasinStatsPlots* comparing the distribution of k_{sn} within Basin 56 to the mean k_{sn} within that basin (the thick black line).

10.9.10 Grid of Bi-Plots of Means - *'scatterplot_matrix'*

The *'scatterplot_matrix'* plot is designed as a very quick data exploratory tool to look for potential relationships between basin statistics (Figure 20). The function will produce a grid of bi-plots comparing the mean values of all metrics for which means were calculated (though this will not include means by categories if calculated). The function also displays a simple second order polynomial fit on the relationship. In positions where values would be compared against each other, the function instead displays a histogram of the distribution of those mean values. For users familiar with R, this is designed to be largely similar to plots produced by the 'lattice' package.

As a note, if you calculated filtered means and leave the optional *'use_filtered'* to false (which is the default) this option will produce a very large matrix as it will include both the regular means and the filtered means. For an input table with filtered means, if you set *'use_filtered'* to true, the function will use the filtered means to populate the scatterplot matrix. If you want to display the regular means without the filtered means, provide an input table for which you did not calculate filtered means when running *CompileBasinStats*.

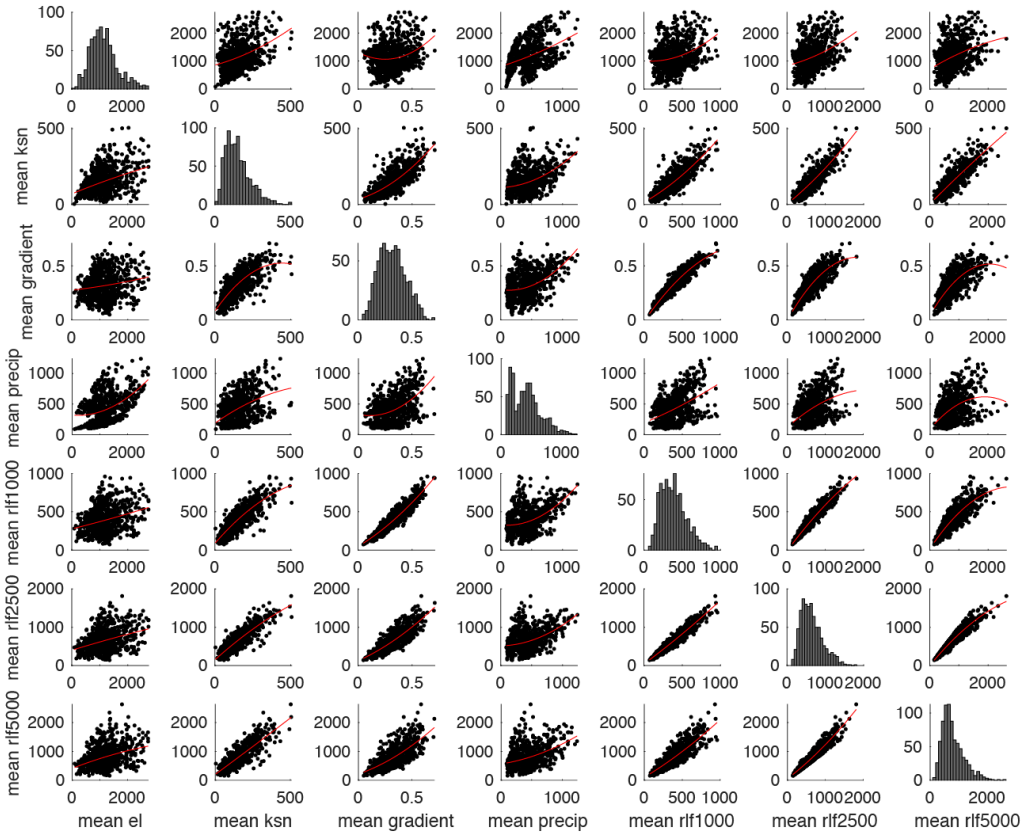


Figure 20: Sample output of *'scatterplot_matrix'* plot from *BasinStatsPlots*.

10.9.11 Generic X-Y plot - *'xy'*

The final plot option for *BasinStatsPlots* is a generic option to plot any value against any other value. To use this option you must specify the name of the columns to be used as the x and y values via the *'xval'* and *'yval'* parameters. Using conjunction with the *'color_by'* parameter can quickly produce plots to explore potential relationships within the basin data (e.g. Figure 14).

11 Swath Profiles with Projected Data

TopoToolbox includes a robust and simple function to produce swath topographic profiles. We provide two functions in TAK that build and expand upon this functionality and use some swath profiles through our example dataset to illustrate the use of these functions (Figure 21).

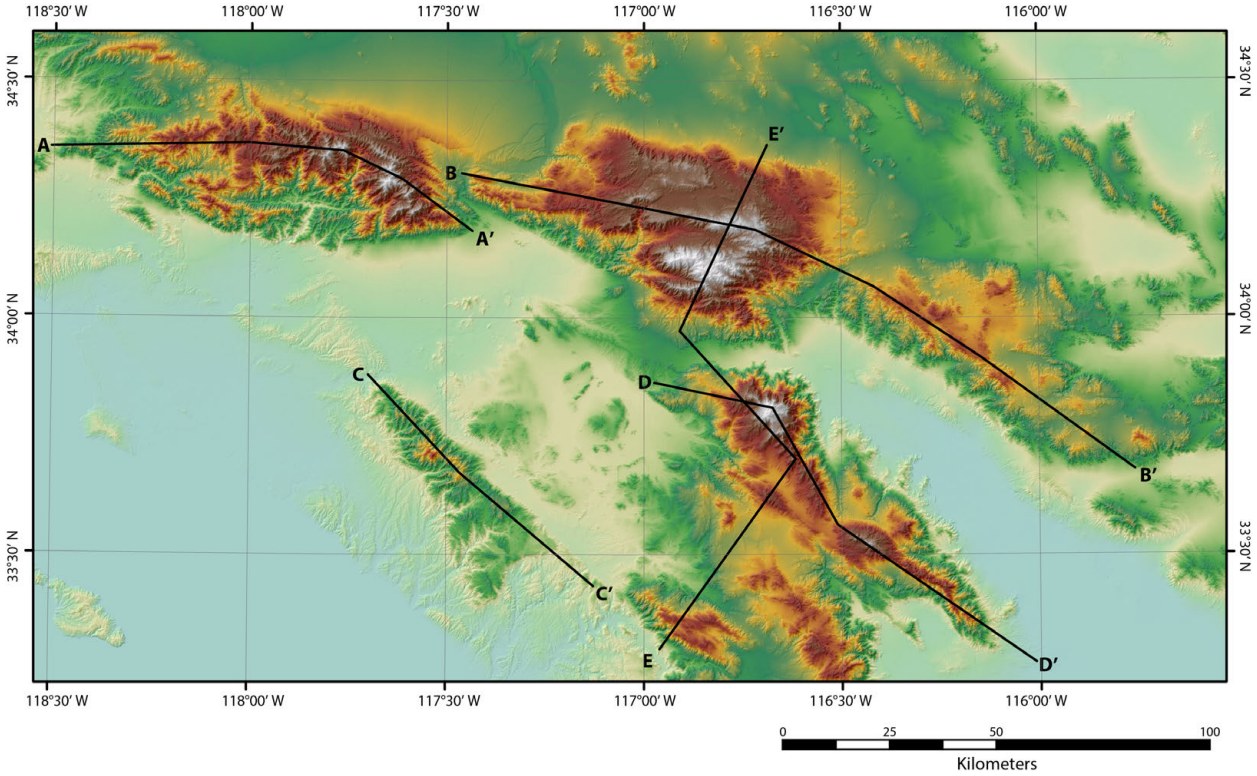


Figure 21: Location of swaths used to demonstrate uses of *MakeTopoSwath* and *MakeCombinedSwath*.

11.1 *MakeTopoSwath*

The *MakeTopoSwath* function is a simple wrapper around the basic functionality of the *SWATHobj* class included as a part of TopoToolbox. The main utility of *MakeTopoSwath* compared to simply using *SWATHobj* natively is more control on the plots produced including different styles of displaying the data, direct control of the vertical exaggeration via the optional *'vex'* parameter, and plotting the location of bends in a swath (Figure 22).

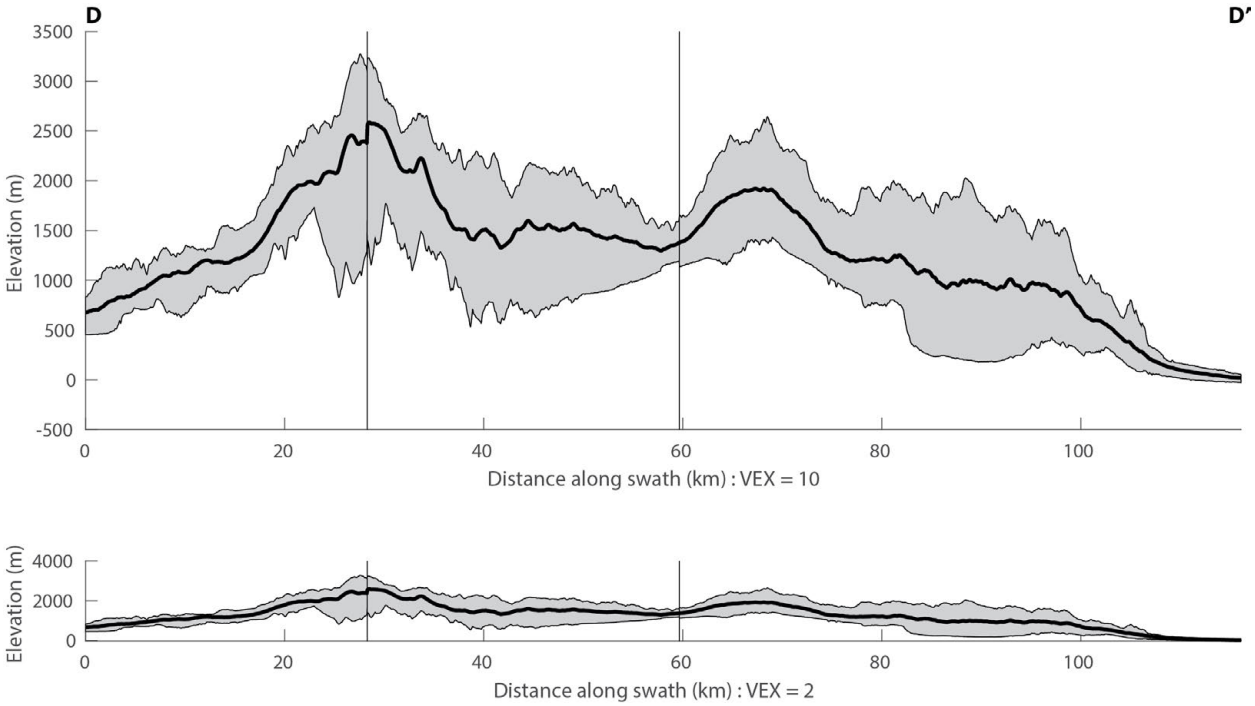


Figure 22: Basic swaths output from *MakeTopoSwath* highlighting the function of the *'vex'* parameter to control the vertical exaggeration.

In addition to the basic swath figure, with *MakeTopoSwath*, you can also display the output swath as an array of points (Figure 23) by setting *'plot_as_points'* to true or a heatmap (Figure 24) by setting *'plot_as_heatmap'* to true.

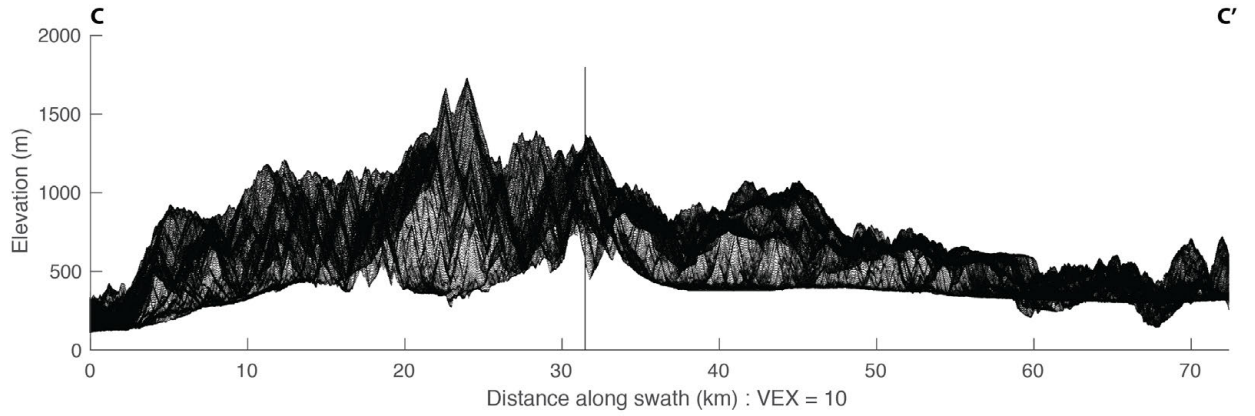


Figure 23: Swath output from *MakeTopoSwath* showing the result of plotting as points.

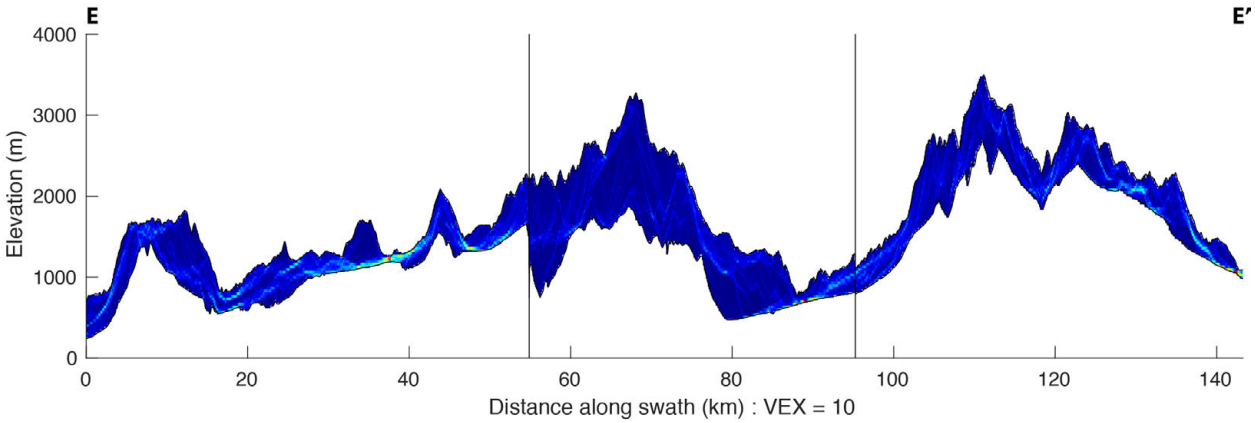


Figure 24: Swath output from *MakeTopoSwath* showing the result of plotting as a heatmap.

11.2 *MakeCombinedSwath*

It can be useful to visualize additional data projected onto a swath profile. The *MakeCombinedSwath* provides a flexible framework within which to produce such plots. There are a variety of types of data that can be plotted onto a swath profile:

- *'points3'* - generic point dataset with x, y, and z coordinates.
- *'points4'* - generic point dataset with x, y, and z coordinates with an additional value. The resulting plots will color the points by this additional value.
- *'points5'* - generic point dataset with x, y, and z coordinates with two additional values. The resulting plots will color the points by the first additional value and scale the points by the second value.
- *'eqs'* - plot optimized for earthquakes for data with x, y, depth, and magnitude. Points will be scaled by magnitude and colored by the distance from the center of the swath line (e.g. Figure 25).
- *'STREAMobj'* - will project portions of a provided STREAMobj onto the swath.
- *'ksn_chandata'* - will project the k_{sn} values from a 'chandata' mat file produced by the old Profiler51 code (some of us have A LOT of these sitting around).
- *'ksn_batch'* - will project the k_{sn} values contained in the geographic data structure output from the *KsnChiBatch* function.
- *'ksn_profiler'* - will project the k_{sn} values contained in the *'knl'* array output from *KsnProfiler*.
- *'basin_stats'* - will project basin data output from *CompileBasinStats*. You must specify a quantity to color basins by with the *'basin_value'* parameter and can optionally also scale point size by an additional quantity with the *'basin_scale'* parameter (e.g. Figure 26).
- *'basin_knicks'* - will project knickpoint locations as produced by *FindBasinKnicks*.

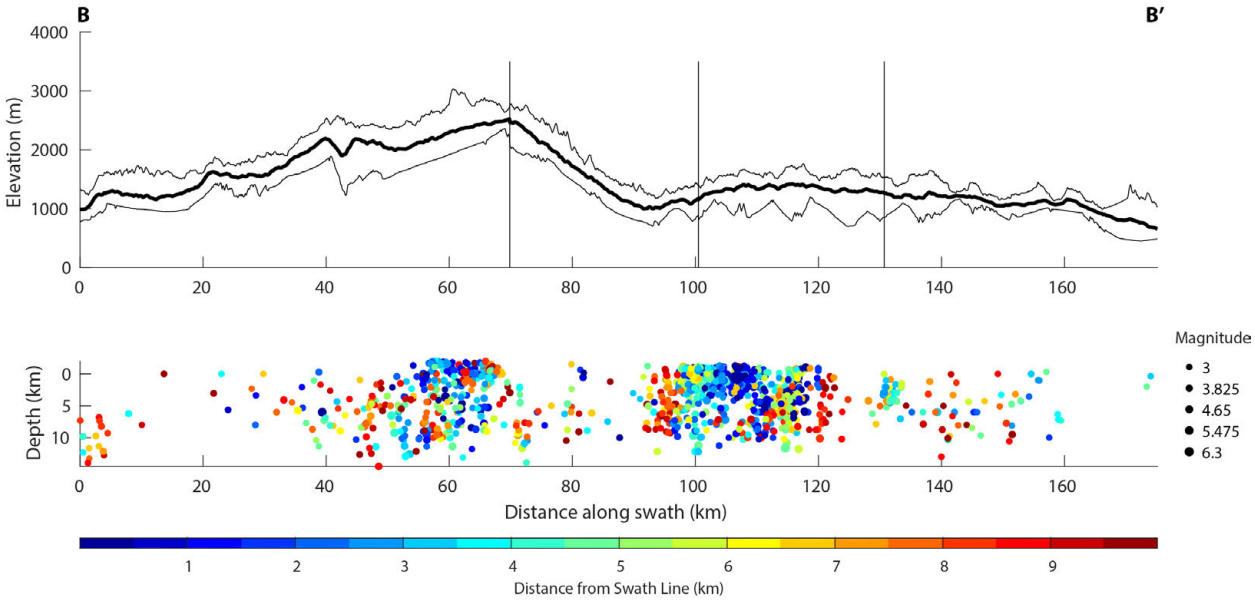


Figure 25: Swath output from *MakeCombinedSwath* projecting earthquake data

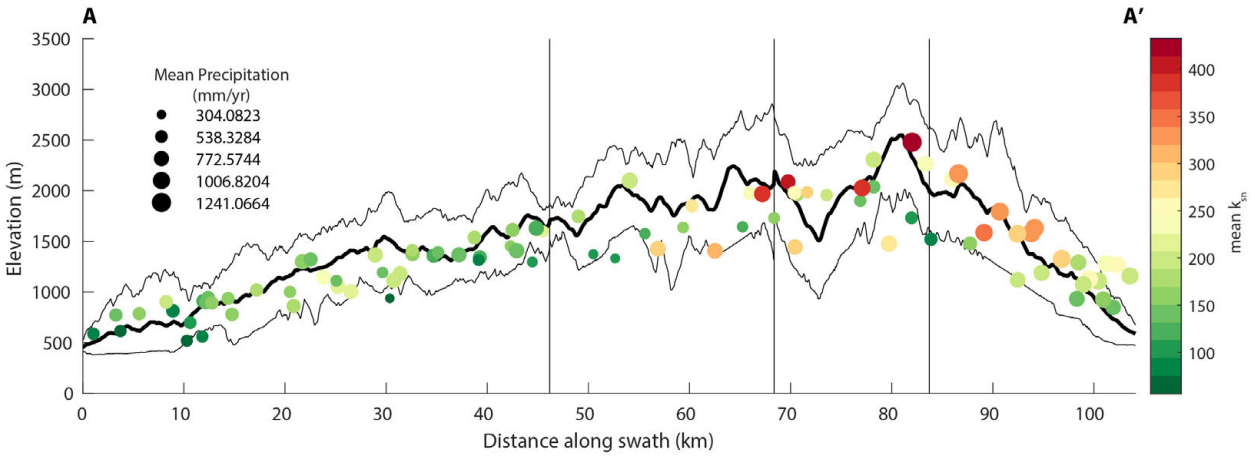


Figure 26: Swath output from *MakeCombinedSwath* projecting basin locations from *CompileBasinStats* and colored by mean k_{sn} and scaled by mean precipitation.

Importantly, the *MakeCombinedSwath* function is setup so that the width to sample the topography to produce the topographic swath is decoupled from the width from which to sample the provided data (you can of course make these equal if you want). By default the function will also plot a map displaying which data points are included in the projected data on the swath figure, you can suppress the drawing of this map with the optional *'plot_map'* parameter.

11.3 *ProjectOntoSwath*

The *ProjectOntoSwath* function is used in *MakeCombinedSwath* to do the data projection, but we include it as a separate function as it may be useful for users. *ProjectOntoSwath* finds the distance of a list of x and y points along and from the centerline of a provided *SWATHobj* based on a user defined width within which to sample the data. Distances of points that lie beyond the extent of the swath or lie outside of the data sampling width will be set to NaN. The function projects data onto each swath segment (i.e. if the swath line is bent) and finds the swath segment that data are closest to in tangential distance to avoid duplication of points.

12 Miscellaneous

12.1 *ksncolor*

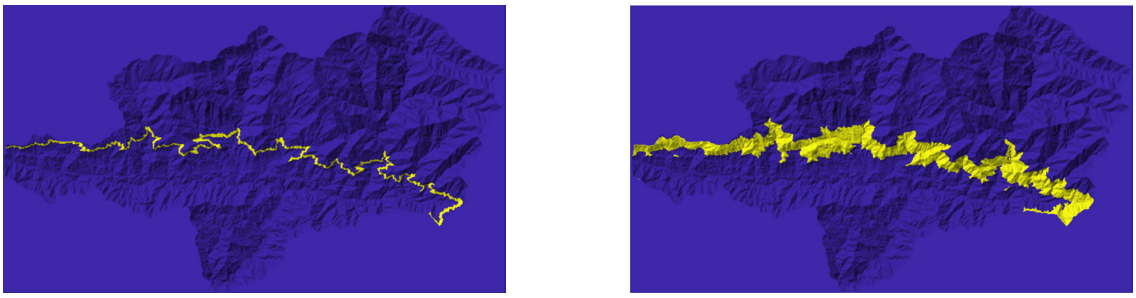
Function to generate a colormap that roughly approximates the stereotypical green-yellow-red color progression commonly used for k_{sn} data in publications, see Figure 26 for an example of the color map.

12.2 *PlotKsn*

The *PlotKsn* function will quickly plot a stream network colored by k_{sn} overlaying a hillshade colored by elevation. Uses the *ksncolor* colormap. You can provide a valid geographic data structure output from *KsnChiBatch* or *ProcessRiverBasins* or a shapefile output from *KsnChiBatch*, *KsnProfiler*, or *ProcessRiverBasins*. Optionally can also include a set of knickpoints output from *ProcessRiverBasins* or *FindBasinKnicks* to plot knickpoint locations.

12.3 *DippingBedFinder*

DippingBedFinder is a simple function to estimate where a planar dipping bed should occur within a landscape based on a known occurrence and some information about the bed (Figure 27). Specifically, you need to provide a location for the observation, a total thickness for the bed, an estimate of the height above the base of the bed at the point of observation, the strike of the bed (it's assumed you give this using the right hand rule), and the dip of the bed.



(a) Projection of a 100 meter thick bed, 50 meters up from the base.

(b) Projection of a 500 meter thick bed, 250 meters up from the base.

Figure 27: Example outputs from *DippingBedFinder* for a bed striking 280°and dipping 10°.

You can provide an empty array for the location of observation to choose a point on the provided DEM to use as the location. The location of the observation is marked on the figure with a white dot.

12.4 *Mat2Arc*

Mat2Arc is a helper function that takes any matfile as an input and will search its contents and produce ascii grids and shapefiles from any valid datasets. Specifically, will make ascii grids of any *GRIDobj*, will convert any *FLOWobj* to an Arc style flow raster an export as an ascii grid, and output any *STREAMobj* or recognizable geographic data structure as a shapefile.

13 Error Reporting

We have tried to ensure that all options work properly within TAK, but unexpected errors may occur. The preferred method of error reporting is to use the built in 'Issues' function on the GitHub website so that there is a record of user encountered errors. If you have forked a version of the code and think you can fix an error, issue a pull request with the change, collaboration is welcome! If all else fails, you can email Adam Forte as well. Whether submitting and Issue or emailing Adam, please provide as much information as possible on the error and what you were doing when it occurred.

References

Adam M Forte and Kelin X. Whipple. Criteria and tools for determining drainage divide stability. *Earth and Planetary Science Letters*, 493:102–117, 2018. doi: 10.1016/j.epsl.2018.04.026.

Adam M. Forte and Kelin X. Whipple. Short Communication: The Topographic Analysis Kit (TAK) for TopoToolbox. *Earth Surface Dynamics*, in review. doi: 10.5194/esurf-2018-57.

Wolfgang Schwanghart and Nikolaus J. Kuhn. TopoToolbox: A set of Matlab functions for topographic analysis. *Environmental Modelling and Software*, 25(6):770–781, 2010. ISSN 13648152. doi: 10.1016/j.envsoft.2009.12.002.

Wolfgang Schwanghart and Dirk Scherler. Short Communication: TopoToolbox 2 - MATLAB based software for topographic analysis and modeling in Earth surface sciences. *Earth Surface Dynamics*, 2:1–7, 2014. doi: 10.5194/esurf-2-1-2014.

Cameron W Wobus, Kelin X Whipple, Eric Kirby, Noah P Snyder, J Johnson, K Spyropolou, Benjamin T Crosby, and D Sheehan. Tectonics from topography: Procedures, promise, and pitfalls. In Sean D Willett, N Hovius, M T Brandon, and Donald Fisher, editors, *Tectonics, climate, and landscape evolution*, number 398, pages 55–74. The Geological Society of America, Boulder, CO, 2006.