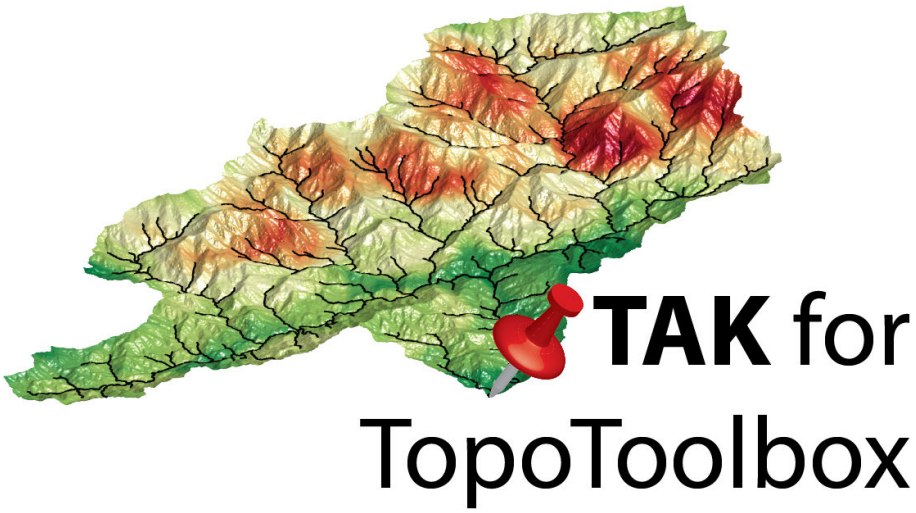
Topographic Analysis Kit (TAK) Manual

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

TAK Release Version 1.1.0



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*, 7:87–95, 2019. doi: 10.5194/esurf-7-87-2019

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.

The easiest way to use TAK is to just download a copy of the GitHub repository to your computer and work from that. The disadvantage is that to update you need to delete the old version and replace with a complete download of the new version (even if just a single function was updated). To avoid this, a suggested strategy for more advanced users 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 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). **This means that after you have installed the compiled TAK functions, double clicking on the TAK.exe file or a shortcut to it will result in an error indicating that it is missing required arguments. You must run it from the command line as described below!** The procedure for running the compiled TAK functions 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 and Section A of the Appendix.

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 standard Matlab 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 exist in the regular Matlab version. 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 Error Reporting

We have tried to ensure that all options work properly within both the Matlab and compiled versions of TAK, but unexpected errors may occur. The preferred method of error reporting is to use the built in 'Issues' function on the GitHub page for TAK 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. Before submitting an issue or contacting Adam, please check both open and closed issues to see if the bug has already been reported or fixed in an updated version. Whether submitting an issue or emailing Adam, please provide as much information as possible on the error (e.g. cut and paste the error message you received) and what you were doing when it occurred.

4 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) with square pixels (i.e. dx = dy) 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 you 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*).

TAK relies on either TopoToolbox or Matlab functions for importing geographic data and thus is limited to the data types supported by these. Specifically, for raster data (e.g. digital elevation models, other gridded data), TAK can interpret and/or export data that are GeoTiffs or ESRI ArcGIS ASCII files (both of which are readable by a variety of GIS programs including ArcGIS, QGIS, and GDAL). For importing or exporting vector data, TAK uses shapefiles (which are again readable and writable by a variety of GIS programs including ArcGIS, QGIS, or OGR).

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.

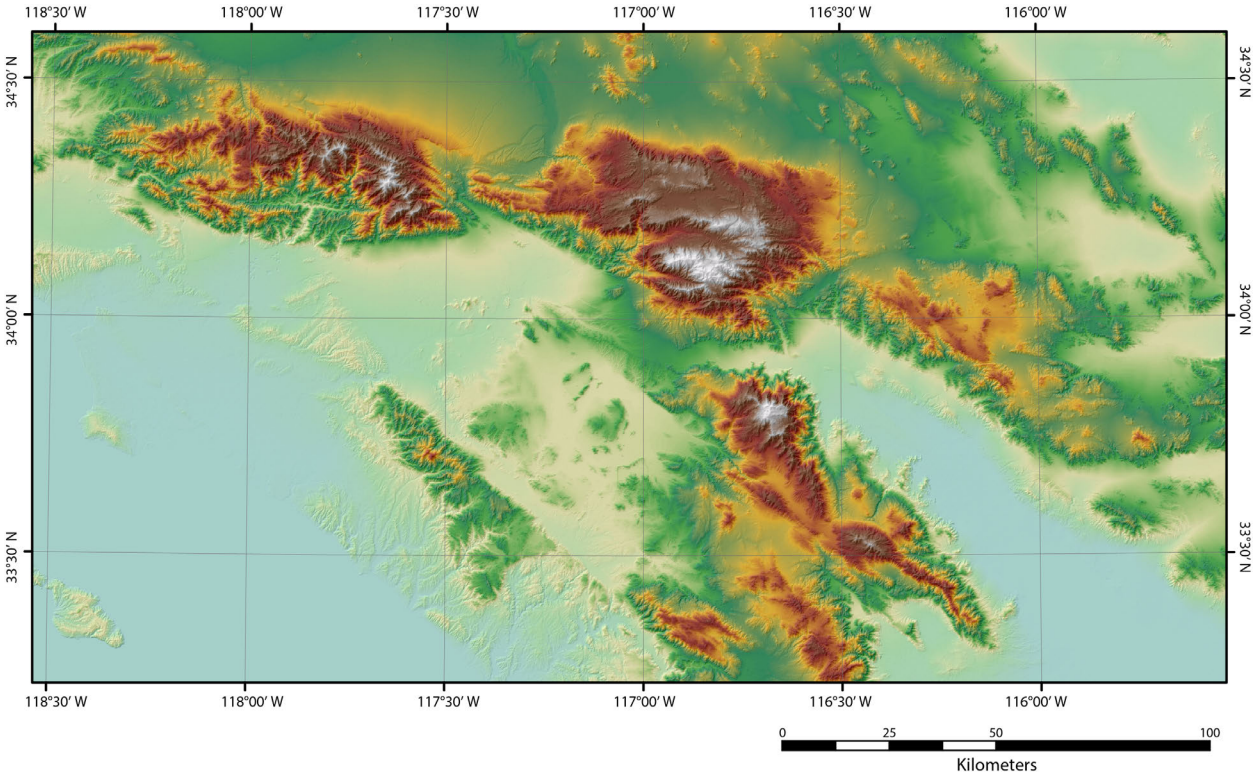


Figure 1: Example dataset area used in this manual.

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*

Matlab, and thus TopoToolbox and TAK, generally require that an entire dataset be loaded into active memory before it can be used. This means that the size of DEM you can successfully process and use in TAK will be dependent on the available memory (i.e. RAM) of the computer on which you're running TAK along with the operating system (Mac OS X, Windows, and Linux versions of Matlab deal with memory in slightly different ways). It's also not as straightforward as looking at the size of the input DEM as you need to account for all the datasets that need to be in active memory for many TAK functions, e.g. a GRIDobj, a FLOWobj, and a STREAMobj. If you are encountering memory issues, there aren't too many simple solutions, but the best bets are to either spit your data into smaller segments (though you must be careful to not introduce errors by truncating portions of drainage basins of interest) or use a dataset with a larger cellsize.

5 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 6. 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.

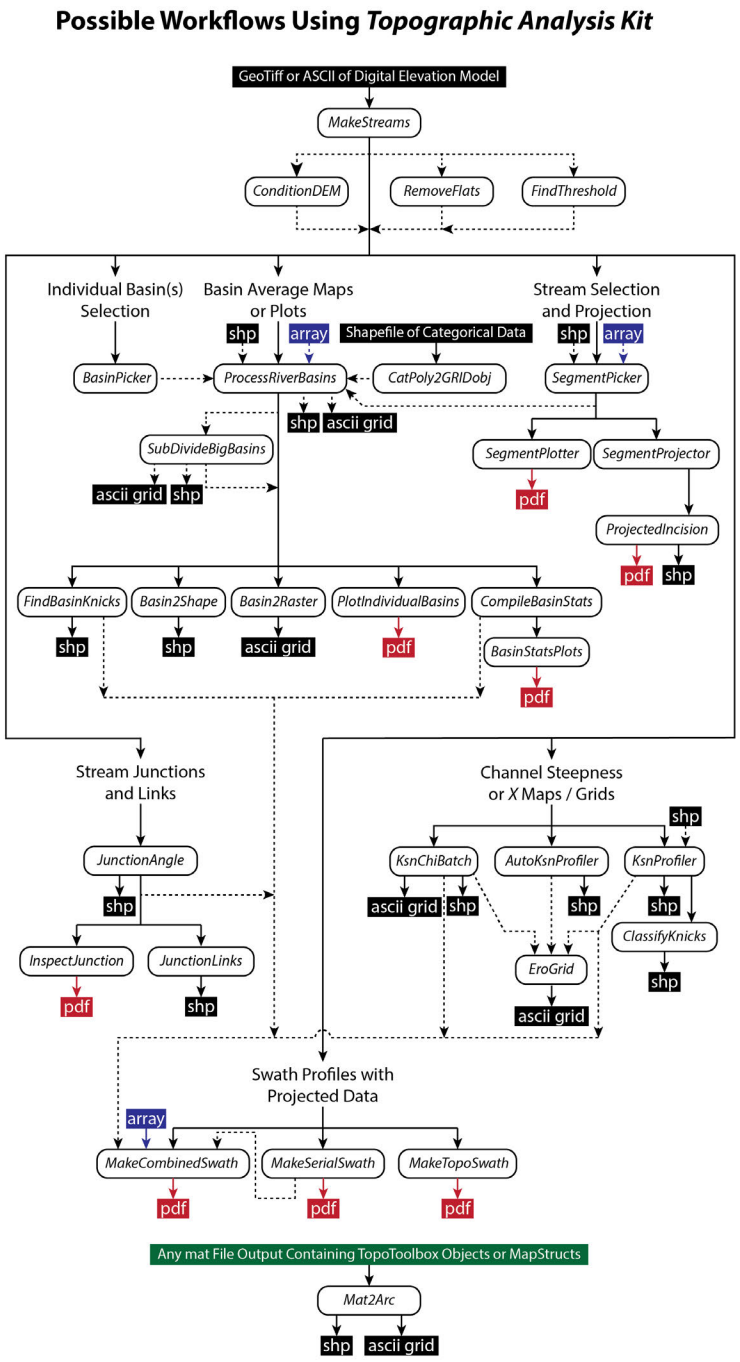


Figure 2: Suggested workflows through TAK functions depending on desired outcome and purpose of analysis. Also highlighted are the nature of the outputs produced by different functions. Definitions of inputs and outputs: shp - a shapefile containing vector data, the geometry of which, e.g. points, lines, polygons, depends on the tool in question; array - a Matlab array, i.e. a matrix of numbers; ascii grid - an ESRI Ascii text file that is interpretable as gridded raster data with projection information by many GIS programs; pdf - a figure output as a PDF.

6 Matlab and TopoToolbox Crash Course

The following sections are written for people who have never used Matlab (Sections 6.1, 6.2, 6.3) or TopoToolbox (Section 6.4). 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.

6.1 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 6.4), arrays, cell arrays, tables, and structures/geographic data structures.

6.1.1 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)
```

6.1.2 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.

6.1.3 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};
```

6.1.4 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');
```

6.2 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.If you open a '.m' file in a text editor or the Matlab editor, the first line, which defines how the function is called, will list the required inputs, e.g.:

```
% A function definition with one required input
function TestFunction(my_array)
```

If there optional inputs, there will simply be an apparent input named 'varargin', meaning that the function is capable of accepting variable arguments:

```
% A function definition with one required input and an arbitrary number of
    optional inputs
function TestFunction(my_array,varargin)
```

In many cases, values for 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);
```

6.3 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');
```

6.4 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
```

6.4.1 *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);
```

6.4.2 *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.

6.4.3 *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);
```

6.4.4 *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);
```

7 Initial Data Processing

7.1 *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.

7.2 *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*. It should be noted that while, as described in more detail in Schwanghart and Scherler [2014], TopoToolbox supports flow routing using either D8 or D∞ algorithms, TAK functions use the simpler D8 flow routing scheme. The minimum inputs to *MakeStreams* 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).

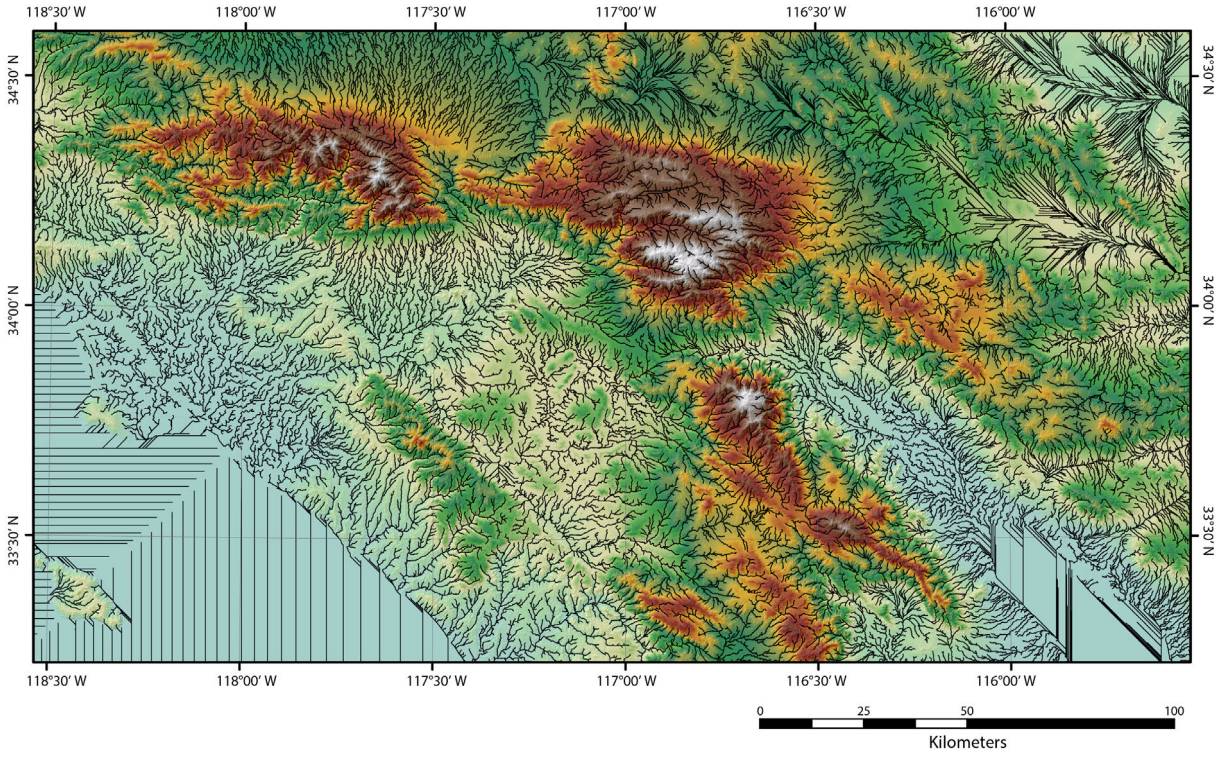


Figure 3: Result of running *MakeStreams* without any control for flat areas.

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');
```

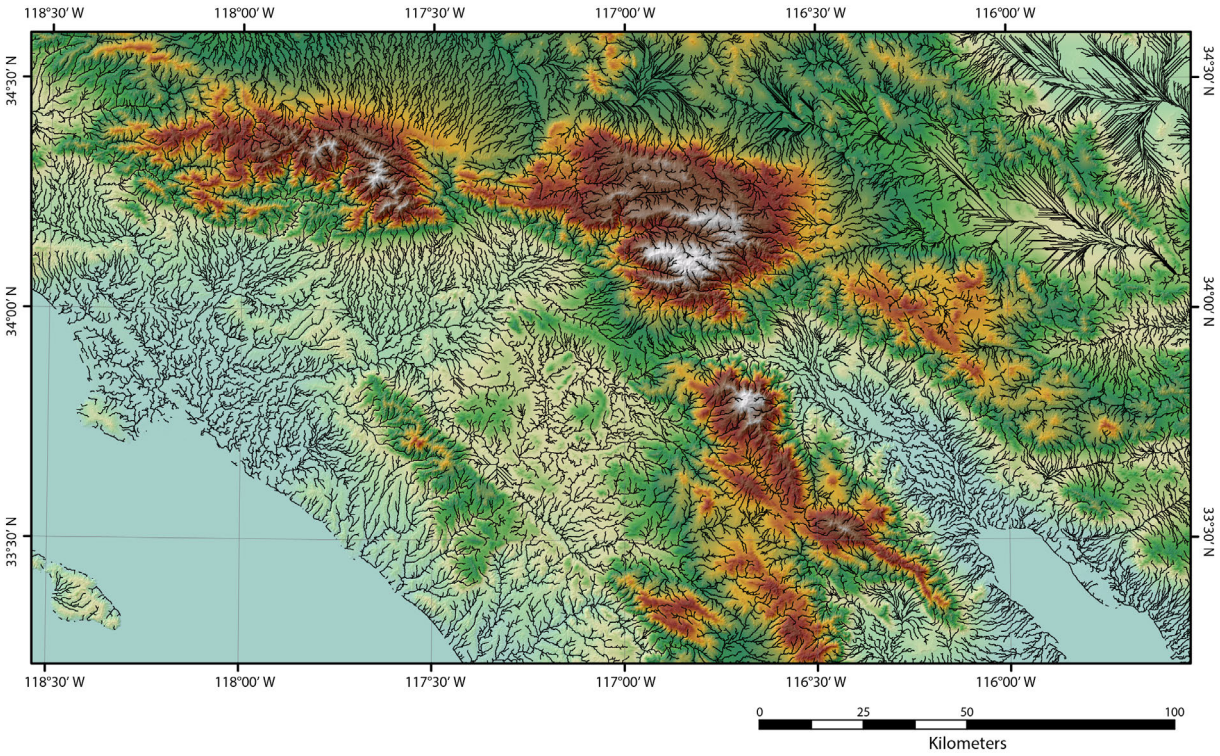
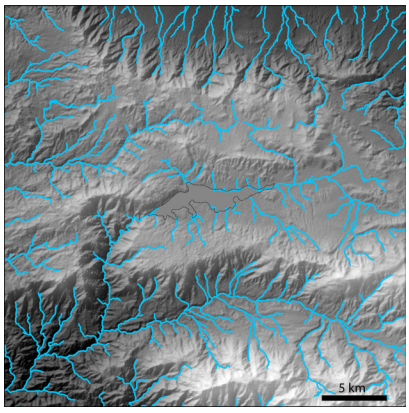
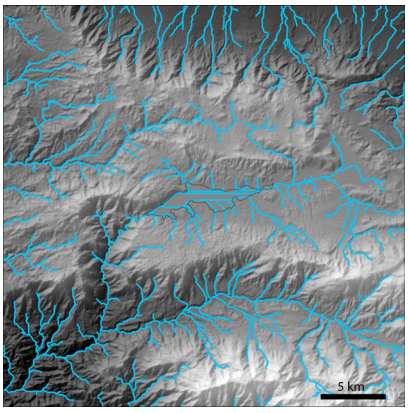


Figure 4: Result of running *MakeStreams* with auto removal of flat areas and *min_flat_area* set to 1e8.

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 considers an area 'flat' if the log of the gradient is undefined and then looks for connected pixels that are identified as 'flat.' The number of connected pixels which will subsequently be treated as a flat area, and set to 'NaN' in the resulting DEM, is controlled by setting a minimum area with the *'min_flat_area'* in m^2. 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).



(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);
```

Finally, for large DEMs, the flow routing can take a long time. To speed this up, you can call upon the 'mex' files associated with the flow routing routines by setting the *'mex'* parameter to true. This will only work if you have precompiled the appropriate mex files from TopoToolbox on your machine, see the TopoToolbox function *'compilemexfiles.m'* for more information.

7.3 *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. We refer readers to the TopoToolbox documentation for detailed discussions of these smoothing routines, including Schwanghart and Scherler [2017] for consideration of the 'crs' algorithm. 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.

7.4 *RemoveFlats*

For topographic analysis, standard stream routing algorithms will usually route streams through lakes and playas, which in some circumstances is inappropriate and could produce misleading results, so 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. The function first fills all sinks in the DEM and then performs a morhpological erosion on the filled DEM with a 3x3 neighborhood. This means that any pixel will be set to the lowest elevation in a 3x3 neighborhood centered on that pixel. The eroded DEM is then morphologically dilated, with a neighborhood of between 3x3 to 9x9 depending on the users selection for the *'strength'* parameter. This dilation sets pixels within the neighborhood to the maximum value. When the user then selects a location on the DEM, all pixels which are connected (using the dilated DEM) are identified as a flat and set to NaN.

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.

7.5 *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.

8 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.

8.1 *SegmentPicker*

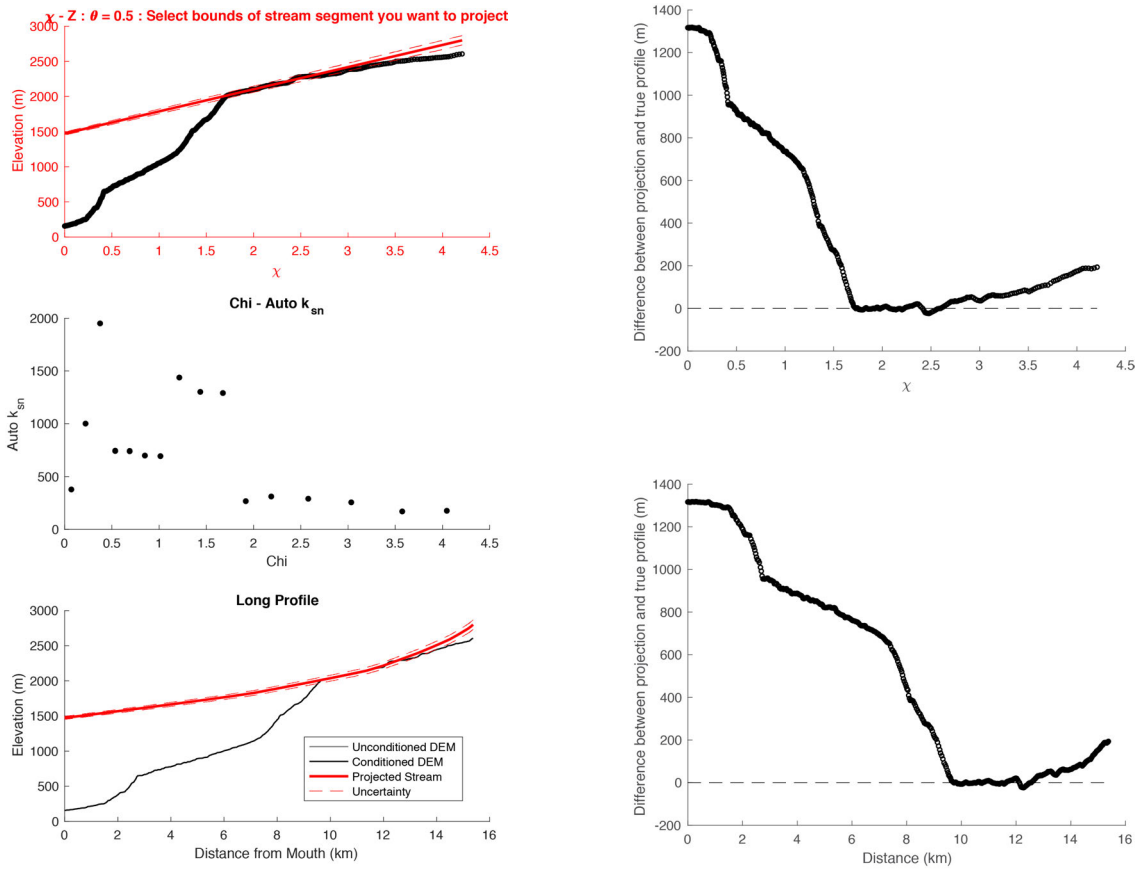
SegmentPicker is a function designed to select portions of larger stream networks. In terms of stream selection, the function has two primary modes, either 'down', i.e. you select individual streams based on a channel head location, or 'up', i.e. you select portions of networks above a given pour point. If selecting on the basis of channel head location, the function will interpret your selection to be the channel head that is the minimum euclidean distance between your selection and the actual channel heads. 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*.

8.2 *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. With the logical *'separate'* flag, you can produce individual figures for each segment. Alternatively, you can provide a list of identifying numbers (i.e. the ID number in the third column of the *SegmentPicker* output) to the *'subset'* parameter to only plot specific segments. You can also initiate and control labeling of stream segments with the *'label'* and *'names'* parameters.

8.3 *SegmentProjector*

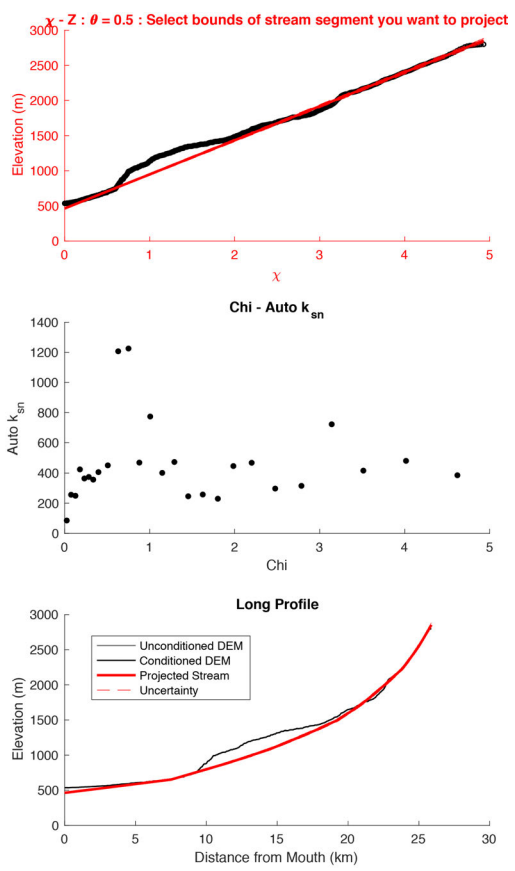
SegmentProjector allows you to interactively select a portion of a stream to project along the length of the entire stream (i.e. from the mouth to the channel head of the provided stream). This can be useful for a variety of questions, e.g. estimating the amount of uplift of a low relief portion of a landscape (Figure 6) or identifying portions of a stream profile that may be tectonically deformed or otherwise disturbed (Figure 7). In detail, the function clips out the portion of the stream you select and performs a least squares linear fit on the χ-elevation relationship along with a 95% confidence interval on this fit.



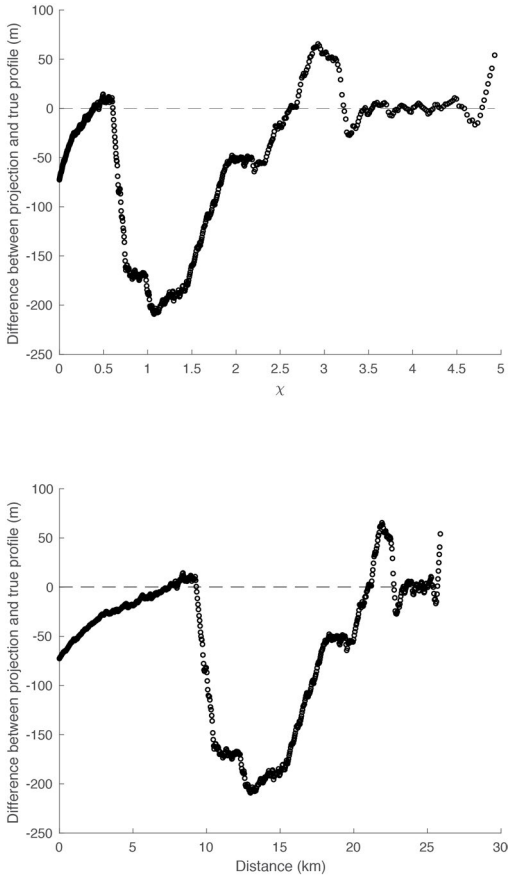
(a) Projected stream

(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.

8.4 *ProjectedIncision*

The *ProjectedIncision* function uses the output of *SegmentProjector* to estimate incision across an entire network. To do this, the function must assume that the topology of the drainage network in question has not changed over the time period of interest. The *ProjectedIncision* function takes the series of projected profiles that are calculated in *SegmentProjector* and assumes that the projected portions of the network represent relict patches of equilibrium river profiles, i.e. it assumes that at some point in the past, the entire landscape had the same k_{sn} as the given relict patch and uses the χ values of the modern drainage to estimate what the elevation of all portions of that relict stream net-

work would be and uses the difference between those elevations and the modern elevations (across the entire provided network) to calculate incision. If more than one section of stream was projected when using *SegmentProjector*, then the *ProjectedIncision* function will calculate projected elevations and implied incision amounts for each individual projected segment and then report the mean, minimum, maximum, and standard deviations of the projected elevations and incision. The function will also optionally display this estimated incision in a figure (e.g. Figure 8). Positive incision amounts indicate the projected elevations of the profile lie above the modern stream elevations whereas negative incision amounts indicate that the projected elevations of the profile lies below the modern stream elevations.

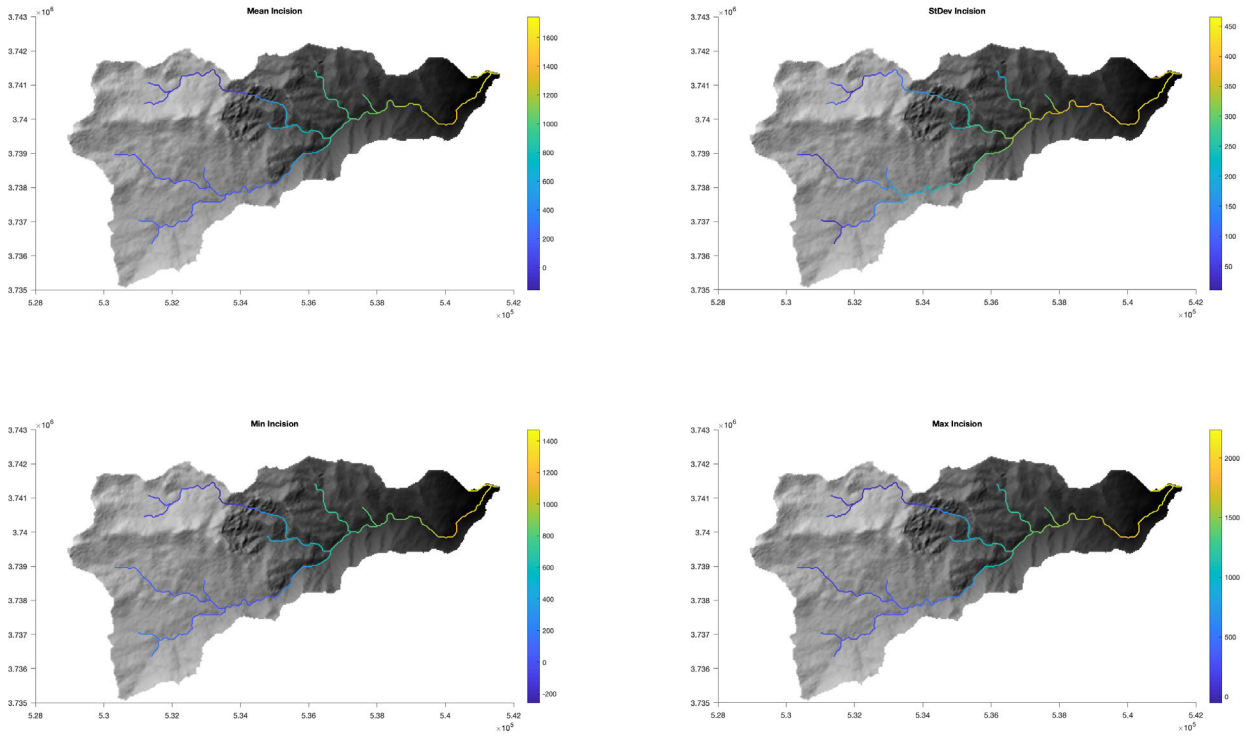


Figure 8: Example of projected incision amounts for Basin 402 (northeastern San Jacinto Mountains) using data from 5 projected streams.

9 Channel Steepness and χ Maps

9.1 *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 three 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,

$$S = k_{sn} * A^{-\theta} \tag{1}$$

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, [e.g., Wobus et al., 2006]. This *'smooth_distance'* is a user controlled parameter in *KsnChiBatch*.

Because of the way the averaging is done over a given smoothing distance by the underlying TopoToolbox *STREAMobj2mapstruct* function, sometimes portions of trunk streams can end up with anomalously high k_{sn} values as their values are influenced by smaller, steeper side tributaries. To partially control for this, we include an additional method of k_{sn} calculation where averaged k_{sn} values for trunk streams are calculated separately. This option can be used by setting

'ksn_method' to *'trunk'*. The determination of whether a stream is a trunk stream or not is based on stream order, which is controllable with the optional *'min_order'* parameter. By default, the *'min_order'* is set to 4, meaning that any stream of order 4 or greater is considered a trunk stream.

Both the default *'quick'* and *'trunk'* methods described above are rapid 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 9).

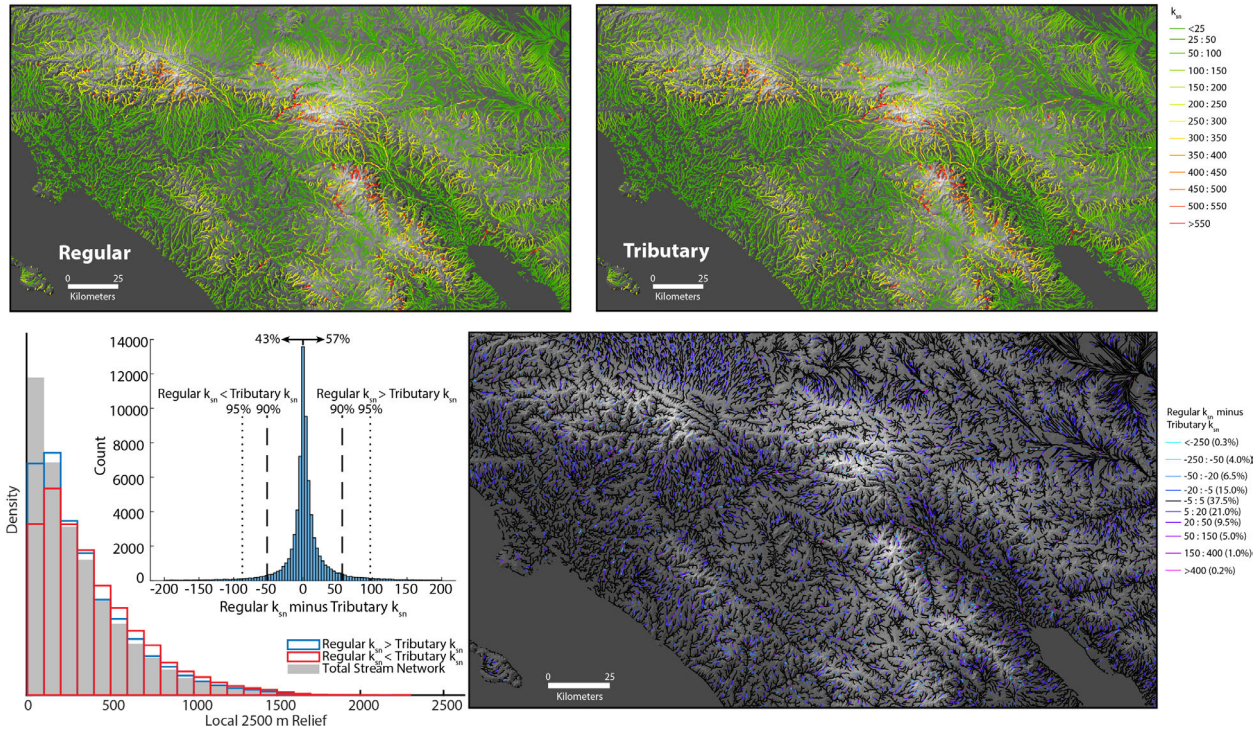


Figure 9: Comparison of the regular 'quick' method vs the 'tributary' method of calculating k_{sn} using *KsnChiBatch*.

While we have not conducted a full study of the differences of the basic *'quick'* vs *'trib'* 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 ± 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 9).

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.

If the *'product'* is set to to *'ksngrid'*, the continuous map of k_{sn} is made via spatial averaging within a moving circular window of the individual k_{sn} values along streams. The radius of this window by default is set to 5000 meters, but can be controlled via the *'radius'* function. Note that the production of this continuous map of k_{sn} can be time consuming. This output is suitable for use with the *EroGrid* function.

9.2 *AutoKsnProfiler*

The *AutoKsnProfiler* function can be viewed as a compromise between the *KsnChiBatch* and *KsnProfiler* functions in terms of automation and the degree of lumping segments of streams into similar steepness domains. In essence, what *AutoKsnProfiler* attempts to do is identify breakpoints in individual stream profiles and then fit a single k_{sn} value to segments between these breakpoints. This is identical to what *KsnProfiler* does, the difference being that with *KsnProfiler* you manually identify these breakpoints on individual stream portions. For detailed stream analysis where the individual steepness domains are important, it is still recommended that you use *KsnProfiler*, but *AutoKsnProfiler* can sometimes produces smoother (i.e. less noisy) and more realistic maps of k_{sn} when compared to *KsnChiBatch*.

In detail, *AutoKsnProfiler* calculates k_{sn} along the stream network as in *KsnChiBatch* and then calculates averaged k_{sn} over a user specified length scale, controllable with the *'segment_length'* parameter. The function then looks for changes along individual stream profiles in these binned k_{sn} values. If it identifies a significant change in these binned values, it identifies this as a break point and will then find the average k_{sn} for the segments of the stream between individual breakpoints. This is equivalent to what a user does manually using *KsnProfiler* (e.g. Figure 13).

The sensitivity of the function to detecting changes is primarily controlled by the *'thresh_ratio'* parameter which is a value between 0 and 1, where lower values imply that the threshold for detecting a change is lower (i.e. there will be more breakpoints). In detail, the sensitivity of the function is also influenced by the segment length and the reference concavity. To aid users in choosing values for these three parameters that are the most appropriate to their data and their desired outcomes, the *AutoKsnProfiler* function has an optional mode where you can visualize the outcomes for a single stream and iteratively change values for the threshold ratio, the segment length, or the reference concavity (e.g. Figure 10).

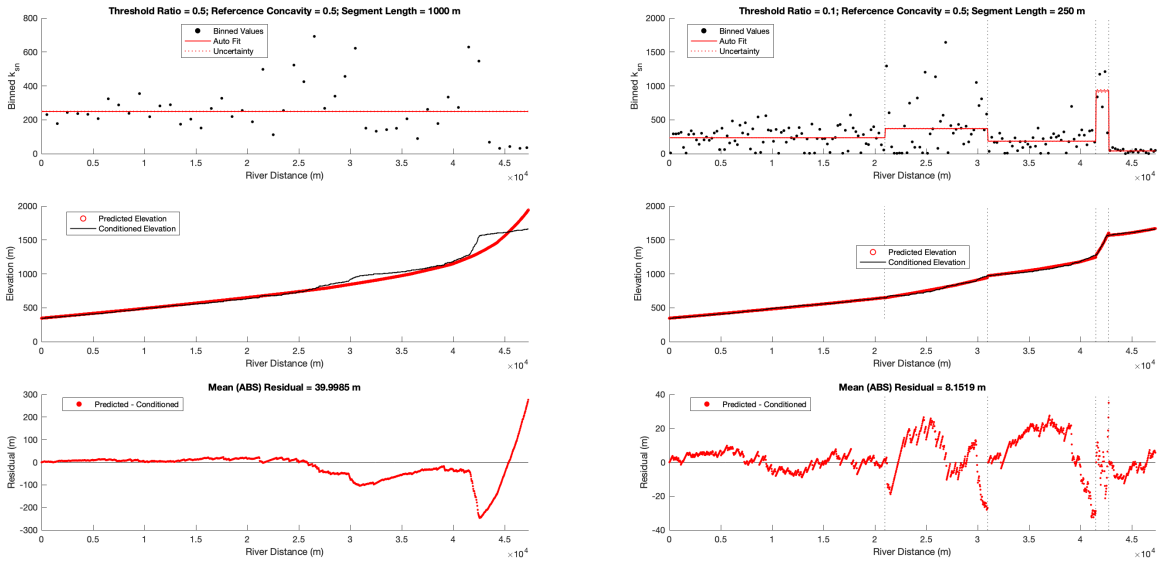


Figure 10: Example of exploring influences of parameter choices on the behavior of *AutoKsnProfiler* for a trial stream. Left panel shows results using the default settings and a threshold ratio of 0.5. Right panel shows results after decreasing threshold ratio to 0.1 and segment length to 250 meters.

By default, the selected stream will be the longest trunk stream in the provided network, but you can override this choice by providing the x-y coordinates of a particular channel head to the optional *'channeloi'* parameter. The function outputs node attributed lists containing the estimated k_{sn} and the residuals between the projected elevations and the true elevations. It also outputs the location of the breakpoints used (which could be interpreted as knick points) a mapstructure (suitable for use with the *shapewrite* function) and a structure containing the parameters used, which can be used in conjunction with the optional *'prev_param'* option to force a run of the *AutoKsnProfiler* to use the same parameters as a previous run.

9.3 *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.

9.3.1 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');
```

9.3.2 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 [Wobus et al., 2006] 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 11).

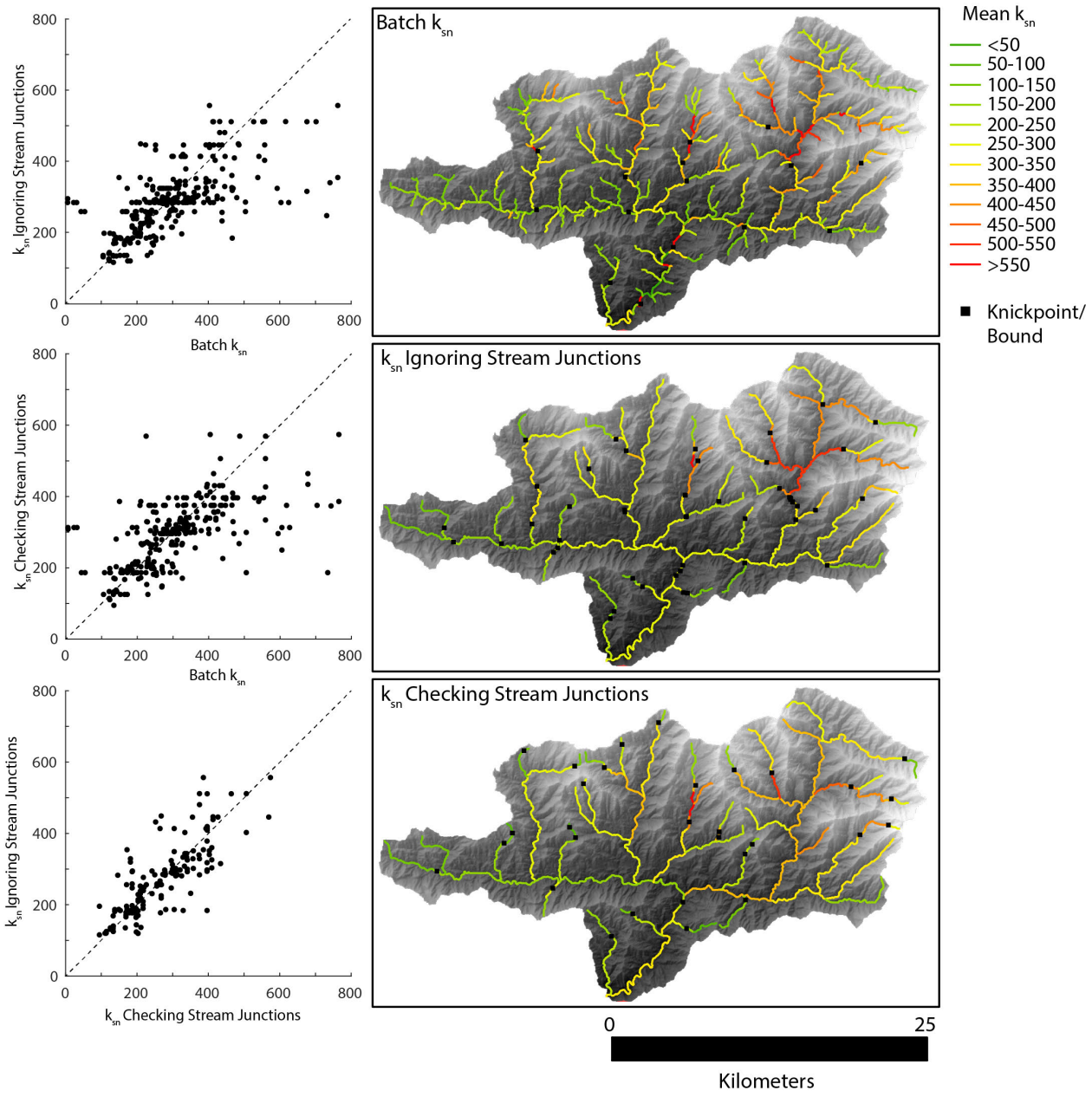


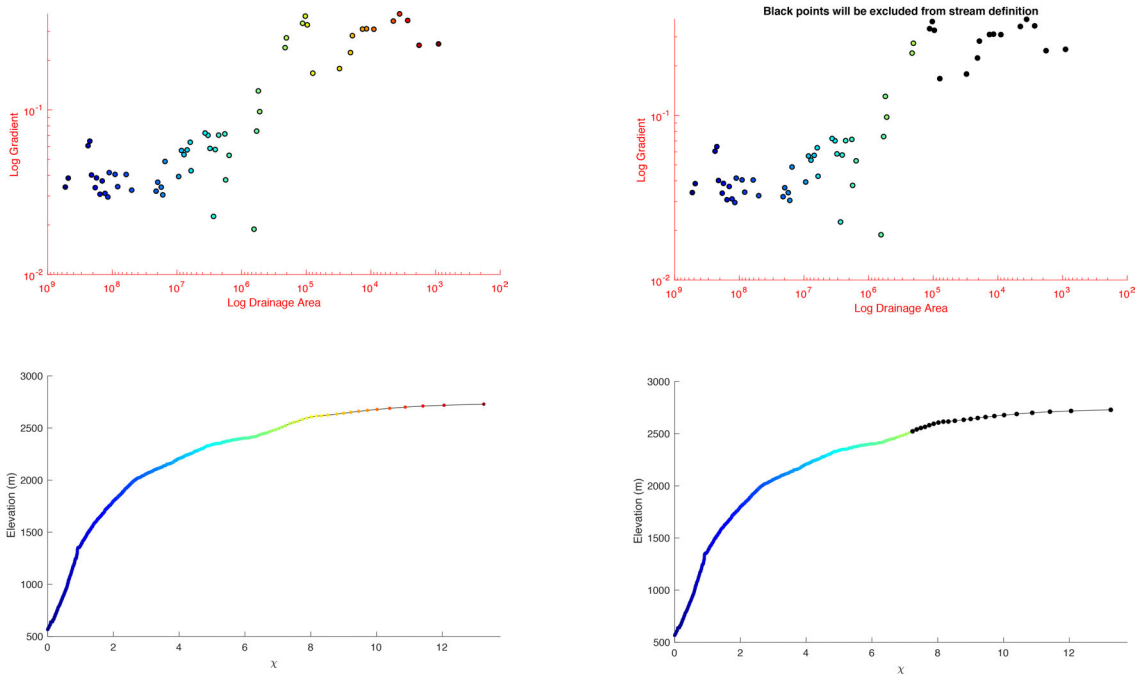
Figure 11: Comparison of k_{sn} values for Basin 56, calculated using the batch method (using *KsnChiBatch*) or *KsnProfiler* either ignoring or checking stream junctions.

9.3.3 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 12).

```
% 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 12: 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.

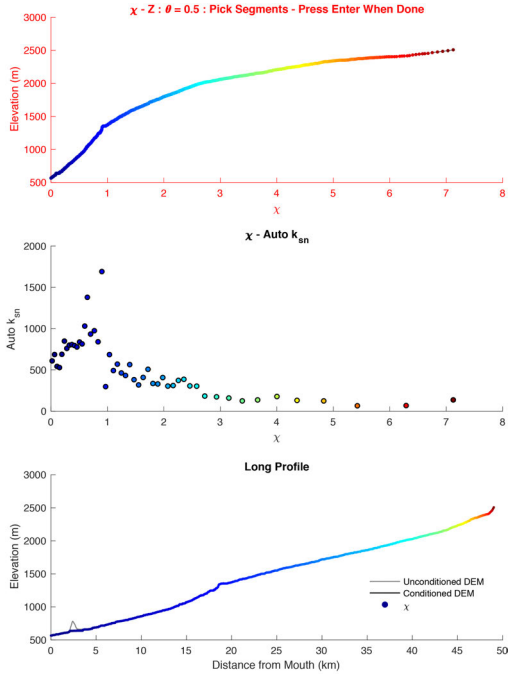
9.3.4 Restarting and Recovering from Errors

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).

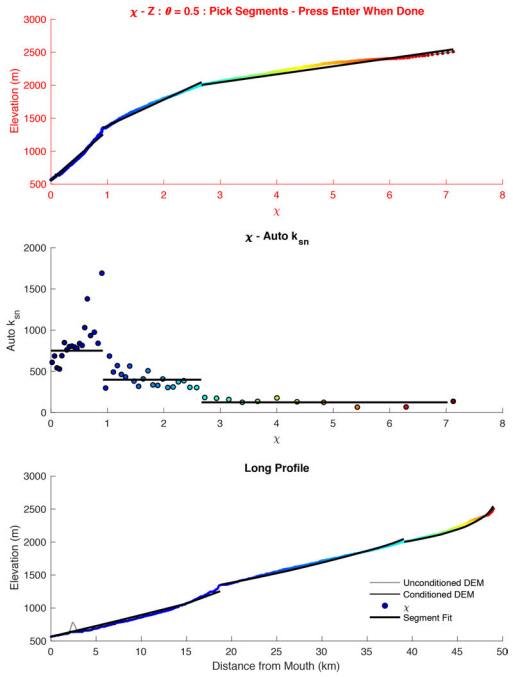
9.3.5 General Use

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. Finding the best-fit concavity relies on the TopoToolbox *chiplot* function, where this function searches for the concavity value that minimizes the difference between the true χ-elevation relationship of the stream segment and a linear χ-elevation relationship. During general use of *KsnProfiler*, for any stream that is selected, you will be prompted to select segment boundaries on a plot (Figure 13a). These manually selected segment boundaries will be output as both a matlab array and a point shapefile. It should be noted that because the selection of these segments is manual, individual selections are not strictly reproducible. However, because the code restricts segment boundaries to stream nodes, there are a limited number of options for segment boundaries so in practice, while tedious, results for a given stream can be mostly reproduced (i.e. if you were trying to reproduce previous segment location choices).

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 14).



(a) Main channel display before segment selection



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

Figure 13: 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 13a) 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 13b).

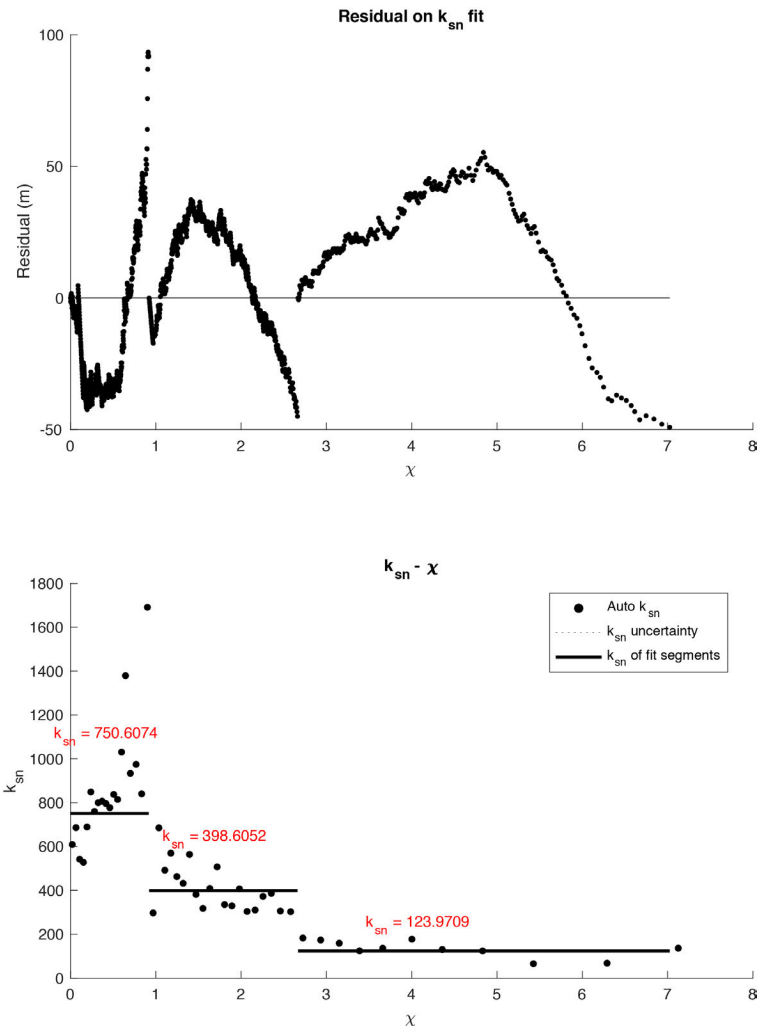


Figure 14: Residuals on k_{sn} fit. Note that uncertainty values on the k_{sn} fits are calculated and plotted, but in this example, the uncertainties are small enough that they are not distinguishable from the main k_{sn} lines.

9.3.6 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.

9.4 *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.

9.5 *EroGrid*

The *EroGrid* function will generate a *GRIDobj* of interpolated erosion rates based on an empirical relationship between erosion rate (e.g. as measured from catchment averaged ^{10}Be inventories) and k_{sn}. The *EroGrid* function assumes that you have estimated this relationship with a power law of the form:

$$\bar{k_{sn}} = C * \bar{E}^{\phi} \tag{2}$$

where $\bar{k_{sn}}$ is a basin averaged value of k_{sn}, $\bar{E}$ is basin averaged erosion rate, and C and ϕ are constants derived from fitting. It should be noted that later in this document we present this same relationship in a different form in Equation (12). The form in Equation (2) does not assume any particular erosion law, where as the form in Equation (12) is couched in terms of the well known 'stream power equation':

$$E = K * A^m * S^n \tag{3}$$

where E is incision rate, K is an erosional efficiency parameter, A is drainage area as a proxy for discharge, S is channel slope, and m and n are empirical constants often referred to as the 'area exponent' and 'slope exponent', respectively. The form of the relationship between k_{sn} and erosion rate presented in Equation (2) and the associated parameters can be recast in terms of the stream power equation and the parameters used in Equation (12) with:

$$C = K^{\frac{-1}{n}} \tag{4}$$

$$\phi = \frac{1}{n} \tag{5}$$

The function is agnostic with respect to the units of $\bar{E}$, e.g. this could be provided m/Myr or mm/yr, but it is assumed that the values for C and ϕ were determined with the empirical erosion rates ($\bar{E}$) in those units and the output *GRIDobj* will be in those same units.

The function expects one of the inputs to be a *GRIDobj* containing continuous values of k_{sn}, e.g. as output from *KsnChiBatch* when the *'method'* is set to *'ksn_grid'*. Alternatively, you can provide a mapstructure containing k_{sn} values, e.g. as output from *KsnChiBatch* when the *'method'* is set to *'ksn'*, and the *EroGrid* function will produce a continuous map of k_{sn} for use in the function.

Optionally, you can provide a standard deviation on the values of C and/or ϕ, which will be used to calculate uncertainty on the interpolated erosion rate:

```
% Producing a single erosion rate GRIDobj
[ERO]=EroGrid(DEM,KSN,100,0.5);
% Producing an erosion rate GRIDobj along with maximum and minimum erosion
    rate grids based on uncertainty in the C and phi values
[ERO,ERO_MAX,ERO_MIN]=EroGrid(DEM,KSN,100,0.5,'C_std',5,'phi_std',0.05);
```

Similarly, you can use incorporate uncertainty in the interpolated map of k_{sn} with the optional *'KSNstd'* input.

Finally, in some cases, the data may suggest that there are different relationships between erosion rate and k_{sn} based on the value of another grid. For example, if you had evidence that there were two different relationships between erosion rate and k_{sn} depending on the mean annual precipitation, you could provide a *GRIDobj* of mean annual precipitation and the 'bins' you wish to use to calculate erosion rates differently:

```
% Establish edges for bins you wish to apply to the GRIDobj of interest, in
    this case a grid of precipitation called PRECIP
% This example assumes that you believe there to be different relationship
    between erosion rate and ksn for areas that recieve 0-1 m/year, 1-3 m/yr,
    and 3-5 m/yr of precipitation per year
edges=[0 1 3 5];
% You must provide values for both C and phi for each bin, i.e. your entries
    for C and phi should have one fewer values than your entries to 'edges'
C=[100 150 200];
phi=[0.5 0.4 0.5];
% It is assumed that the entries to 'C' and 'phi' are in the same order as
    your bin edges, e.g. for any pixel with a precipitation value between 0 and
     1 m/yr, a C of 100 and phi of 0.5 will be used to determine the erosion
    rate based on the ksn value at that pixel
[ERO]=EroGrid(DEM,KSN,C,phi,'edges',edges,'VAL',PRECIP);
```

You can use this option to calculate erosion rates differently depending on the spatial variation in another grid along with uncertainties on k_{sn}, C, or ϕ, the only requirement is that there are the same number of entries to *'C_std'* and *'phi_std'* as there are to *'C'* and *'phi'*, respectively.

10 Stream Junctions

10.1 *JunctionAngle*

10.1.1 Measuring Junction Angles

The *JunctionAngle* function is designed to both measure the angle between streams at tributary junctions and provide simple predictions of what these junction angles should be under very simple assumptions of an idealized channel network. The function largely adopts nomenclatures and methods from Howard [1971]. Specifically, the function considers a stream junction to consist of two upstream links, referred to as 'tributary 1' and 'tributary 2', and a single downstream link. Tributary 1 is defined to be the larger of the two incoming links, with the determination of size based on the Shreve stream order of the two streams. In cases where the Shreve order of the two upstream links are the same, the determination of stream size is based on the drainage areas of the two upstream links. We adopt the approach in

Howard [1971] and consider the junction angle between the two upstream links, generally, as the product of two angles between each link and the upstream projection of the downstream link (Figure 15a).

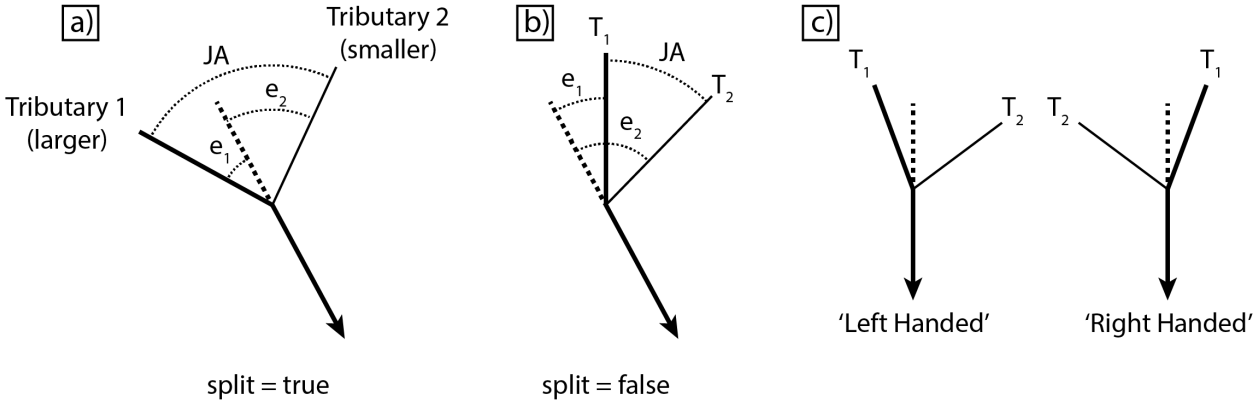


Figure 15: a) Diagram of angles measured by the *JunctionAngle* function and how these relate to the particular tributaries. b) Example of when the upstream projection of the downstream link does not bisect the two incoming links. If this were a real junction, the *'rotation'* for both tributaries would be clockwise (CW). c) Handedness as defined by James and Krumbein [1969]. Note that handedness for a junction is undefined if the Shreve order for the two tributaries are identical

These two angles, e_1 and e_2 referring to the angles between tributaries 1 and 2 and the upstream projection of the downstream link, respectively, are summed to find the junction angle (JA), such that:

$$JA = e_1 + e_2 \tag{6}$$

Not all junctions satisfy the assumption that both upstream links are bisected by the upstream projection of the downstream link, e.g. if both tributary 1 and tributary 2 lie on the same side of the upstream projection of the downstream link (Figure 15b). In these cases, the junction angle is instead determined by:

$$JA = \begin{cases} e_1 - e_2, & \text{if } e_1 \geq e_2 \\ e_2 - e_1, & \text{if } e_2 > e_1 \end{cases} \tag{7}$$

In these cases, the values reported for e_1 and e_2, i.e. the entries for *'e1_obs_angle'* and *'e2_obs_angle'* in the output table, will still be with reference to the upstream projection of the downstream link. For a particular junction, the value of *'split'* in the output table indicates whether the upstream projection of the downstream link bisects the tributaries (true) or does not (false). By definition, junction angles are undefined in cases where more than two upstream links meet at a single node within a *STREAMobj*.

To ensure that the angular relations between elements of the stream network are accurately represented and measured, the nodes of the provided *STREAMobj* are converted to geographic coordinates before any calculations are preformed. To calculate junction angles, the mean orientation of each link is determined by a linear fit (see Figure 16 for an example). The orientation of this mean line is sensitive to the distance over which the fit is determined. This distance can be controlled by the user with the *'fit_distance'* parameter and can be specified in map units as measured along the stream or by directly specifying the number of nodes along the stream you would like to use. Multiple values can be provided to *'fit_distance'* to calculate junction angles considering more than one distance:

```
% Finding junction angles with a single fit distance of 1000 meters
[junctions,IX]=JunctionAngle(S,A,DEM,1000);
% 'junctions' output in this case will be a table
% Finding junction angles considering multiple fit distances
[junctions,IX,mean_junctions]=JunctionAngle(S,A,DEM,[500 1000 2000 5000]);
% 'junctions' output in this case will be a cell array with 4 columns and
    containing tables for each different fit distance
% 'mean_junctions' output is an array that summarizes the variability in
    junction angles across the different fit distances provided
```

Depending on the details of the stream network and the value provided to *'fit_distance'*, not all links may be fit over the full distance, because the algorithm will only fit streams over portions of links that do not encounter another junction. For example, if a value of 1000 meters was provided to *'fit_distance'*, but for one of the links another tributary junction occurred 500 meters from the junction in question, then that particular link would only be fit over 500 meters. The extent to which the linear fit of the link is a good approximation of the true path of the link can be assessed with the R^2 value for the link in question, i.e. the *'e1_R2'*, *'e2_R2'*, or *'e3_R2'* values in the output table, or visually for individual junctions of interest with the *InspectJunction* function.

The *'IX'* output is a cell array containing the indices into the node attributed list for the provided *STREAMobj* for all junctions. Calculating this output is the most time consuming portion of the function, so there is an optional input to provide a previously calculated *'IX'* output when running *'JunctionAngle'*.

10.1.2 Junction Angle Prediction

Along with measuring the junction angles, the *'JunctionAngle'* function will predict what the junction angles of a given junction should be based on the geometric expectations laid out in Howard [1971]. This predicted junction angle can be calculated considering the drainage areas of the two upstream links with the equations:

$$e_{1predicted} = cos^{-1}\left(\left(\frac{A_1 + A_2}{A_1}\right)^{\theta_{ref}}\right) \tag{8}$$

$$e_{2predicted} = cos^{-1}\left(\left(\frac{A_1 + A_2}{A_2}\right)^{\theta_{ref}}\right) \tag{9}$$

Where A_1 and A_2 are the drainage areas directly upstream of the junction for tributaries 1 and 2 respectively and θ_{ref} is the reference concavity. Alternatively, the predicted junction angles can be calculated using the average channel slope where:

$$e_{1predicted} = cos^{-1}\left(\frac{S_3}{S_1}\right) \tag{10}$$

$$e_{2predicted} = cos^{-1}\left(\frac{S_3}{S_2}\right) \tag{11}$$

Where S_1, S_2, and S_3 are the mean slopes of tributaries 1 and 2 and the downstream link, respectively. In the slope based method, the predicted junction angles will be sensitive to the *'fit_distance'* as the mean slope of the links may vary depending on the portions of the stream considered. This is not the case for the drainage area method as this will not vary as a function of the fit distance, but it is instead sensitive to the choice of reference concavity. You can choose to calculate the predicted angles with the slope, area, or both methods depending on the input to the optional *'predict_angle_method'* argument.

10.1.3 Output Table

One of the primary outputs of the *JunctionAngle* function is a table with a variety of information for all junctions within the provided stream network. For any junction where there are more than two upstream tributaries meeting, all values except the junction number and x-y coordinates will be set to NaN as junction angles (and derivative values) are undefined in this case. The entries in the output table are as follows:

- *'junction_number'* - A unique ID number for the junction
- *'junction_x'* - the x coordinate of the junction
- *'junction_y'* - the y coordinate of the junction
- *'junction_angle'* - the angle (in degrees) between tributary 1 and tributary 2
- *'handedness'* - the handedness of the junction as defined by James and Krumbein [1969], which can be either 'Right', 'Left', or 'Undefined'
- *'split'* - logical indicating whether the upstream tributaries are bisected by the upstream projection of the downstream link (true) or not (false)
- *'e1_obs_angle'* - the measured angle between tributary 1 and the upstream projection of the downstream link
- *'e1_Apred_angle'* - the predicted angle between tributary 1 and the upstream projection of the downstream link if the *'predict_angle_method'* is set to *'area'* or *'both'*.
- *'e1_Spred_angle'* - the predicted angle between tributary 1 and the upstream projection of the downstream link if the *'predict_angle_method'* is set to *'slope'* or *'both'*.
- *'e1_rotation'* - orientation of tributary 1 with respect to the upstream projection of the downstream link, either counter-clockwise (CCW), clockwise (CW), or undefined
- *'e1_shreve'* - Shreve order of tributary 1
- *'e1_direction'* - flow azimuth of selected portion of tributary 1
- *'e1_distance'* - stream distance used to fit selected portion of tributary 1
- *'e1_num'* - number of nodes used to fit selected portion of tributary 1
- *'e1_R2'* - R^2 value on linear fit of selected portion of tributary 1
- *'e2_obs_angle'* - the measured angle between tributary 2 and the upstream projection of the downstream link
- *'e2_Apred_angle'* - the predicted angle between tributary 2 and the upstream projection of the downstream link if the *'predict_angle_method'* is set to *'area'* or *'both'*.
- *'e2_Spred_angle'* - the predicted angle between tributary 21 and the upstream projection of the downstream link if the *'predict_angle_method'* is set to *'slope'* or *'both'*.
- *'e2_rotation'* - orientation of tributary 2 with respect to the upstream projection of the downstream link, either counter-clockwise (CCW), clockwise (CW), or undefined
- *'e2_shreve'* - Shreve order of tributary 2

- *'e2_direction'* - flow azimuth of selected portion of tributary 2
- *'e2_distance'* - stream distance used to fit selected portion of tributary 2
- *'e2_num'* - number of nodes used to fit selected portion of tributary 2
- *'e2_R2'* - R^2 value on linear fit of selected portion of tributary 2
- *'e3_direction'* - flow azimuth of selected portion of downstream link
- *'e3_distance'* - stream distance used to fit selected portion of downstream link
- *'e3_num'* - number of nodes used to fit selected portion of downstream link
- *'e3_R2'* - R^2 value on linear fit of selected portion of downstream link

10.2 *InspectJunction*

The *InspectJunction* function is provided mainly as a way of evaluating outputs of *JunctionAngle*. *InspectJunction* expects the *STREAMobj* used in the original call to *JunctionAngle* along with the *'IX'* output from *JunctionAngle* and the number (i.e. the index of a particular junction as it appears in the output junction table) of a particular junction in question. Alternatively, you can provide an empty array to this number and manually select a junction to inspect on a map display of the stream network.

```
% To visually inspect the 50th junction in the output junction table
InspectJunction(S,IX,50);
% To manually select a juction to inspect
InspectJunction(S,IX,[]);
```

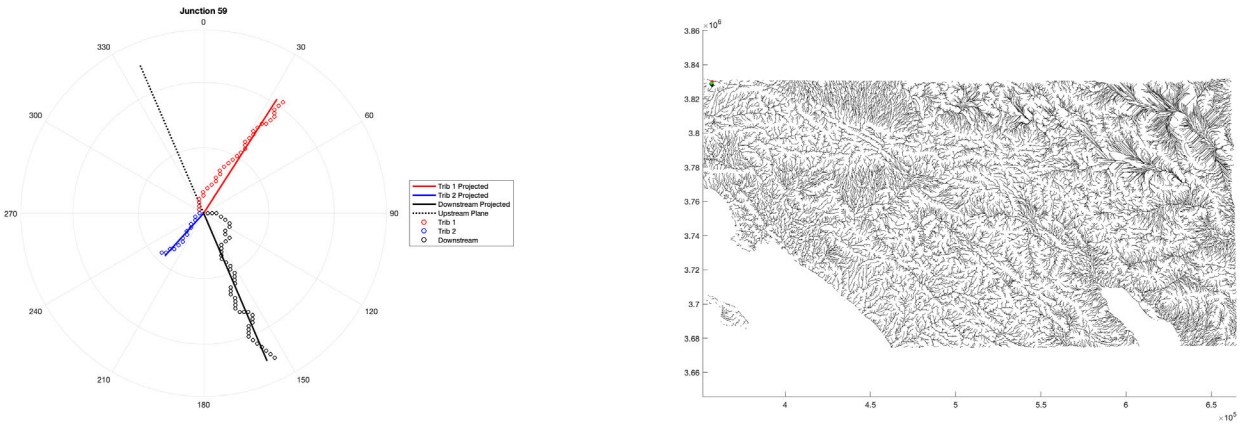


Figure 16: Details on the fit of a selected junction. Left plot is the selected portions of the stream showing the true stream path, as points, and the best fit lines used to calculate the junction angles, displayed in polar coordinates. The dashed line is the upstream projection of the downstream link, which is used to measure the pairs of junction angles reported in the junction table. Right plot is the map showing the location of the selected junction and the sections of streams in context.

With the optional inputs to *InspectJunction*, you can also manually change the length (either as stream length or number of nodes) over which the junction angels are fit to test the sensitivity of the junction angle to the fit distance. If no entries are provided to the optional *'fit_distance'* or *'num_nodes'*, the function will use the stream length selected during the original call to *JunctionAngle*, which will either be the single value supplied or the maximum value supplied if multiple fit distances were specified.

10.3 *JunctionLinks*

The *JunctionLinks* function uses the output of *JunctionAngle* and the classification of 'handedness' for individual junctions to classify individual links within the provided *STREAMobj* (see Figure 15 for a definition of handedness). The classification scheme is based on James and Krumbein [1969] and follows the basic rules depicted in Figure 17.

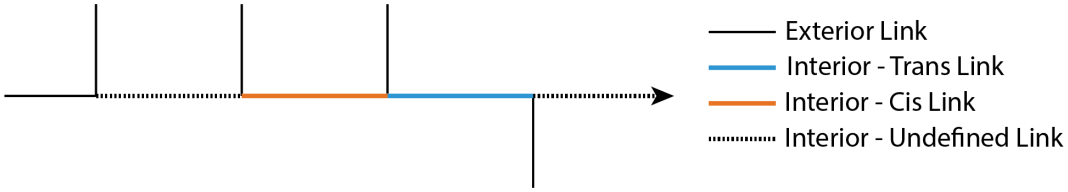


Figure 17: Distinction between exterior, interior - Cis, interior - Trans, and interior - Undefined links as laid out by James and Krumbein [1969] and implemented in the *JunctionLinks* function.

Specifically, the function will classify links as either 'Interior' or 'Exterior', with the distinction being whether a link is defined between two junctions (or an outlet) for the former or is defined by a channel head at its upstream end for the latter. The function also classifies links as either 'Cis', 'Trans', or 'Undefined'. Cis links have the same handedess for the junctions that defined either end and Trans links have different handedness at each end. A link will be Undefined if it is an exterior link or an interior link for which both junctions that define it do not have clear handedness. This may result if one of the ends of the link is an outlet, if one of the junctions has more than two streams meeting, or if the incoming streams at one of the junctions have the same Shreve stream order.

The *JunctionLinks* function outputs a table with information for each link (including which junctions as output in the junction table from *JunctionAngle* define the link). Optionally, the function can be used to produce a shapefile that contains this information for each link, but it should be noted that the production of this shapefile can be EXTREMELY time consuming. It is strongly recommended that if you have access to the Parallel Computing Toolbox for MATLAB that you leave the optional *'par'* parameter set to *true*. If you do not have access to this toolbox, the function will automatically detect this, so it is not necessary to set *'par'* to *false*.

11 Basin Selection

11.1 *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. If you have already calculated local relief for your area of interest, you can skip the recalculation by providing a *GRIDobj* of local relief to the optional *'rlf_grid'* parameter. You can also provide an additional *GRIDobj* if you wish to consider an additional gridded dataset in selecting basin outlet locations:

```
% Providing a precomputed relief grid
[Outlets]=BasinPicker(DEM,FD,A,S,'rlf_grid',RLF);
% Providing an extra grid
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.

In some instances, you may be selecting basins that are upstream of specific locations, e.g. if you want to calculate basin averaged statistics for watersheds above detrital sample sites for which you have GPS coordinates. Very often, the true location of streams, and samples from them, will be slightly different than the location of the stream that results from flow routing, such that just using the GPS location of a sample site will result in incorrect basin selection, requiring a laborious process of nudging true locations to lie on stream lines. This process can be sped by providing the list of coordinates to the optional *'refine_positions'* argument in *BasinPicker*. After providing these locations, the code will iterate through each position and show its location on the main map but also the location of the sample and the stream network in a separate zoomed in window, allowing you to precisely reposition the river mouth. The amount of zoom is controlled by the *'window_size'* parameter, where you can specify the size of the zoomed in window in kilometers. For example:

```
% Load a shapefile containing your GPS coordinates
ms=shaperead('my_gps_points.shp');
% Extract the X and Y coordinates and put them into a m x 2 array
gps_pnts=[[ms.X] [ms.Y]]
% Run BasinPicker with window size of the zoomed window set to 2 km
[Outlets]=BasinPicker(DEM,FD,A,S,'refine_positions',gps_pnts,'window_size',2);
```

12 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.

12.1 *ProcessRiverBasins*

12.1.1 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!** For an easy way to do this, see the *'refine_positions'* option in *BasinPicker*.

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 your 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 18) and use *ProcessRiverBasins* to process these large basins and then use *SubDivideBigBasins* to automatically subdivide basins as opposed to manually selecting large numbers of nested, sub-basins.

The *ProcessRiverBasins* function will use the river mouth locations to extract basins. For each basin there will be a Matlab .mat file which will contain 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. See the Understanding Outputs for a complete list of outputs stored within the .mat file. 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.

12.1.2 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.

Note for Compiled Versions: If you are running the compiled version of *ProcessRiverBasins*, the procedure for adding extra grids is slightly different. Instead of directly providing the extra grids to *ProcessRiverBasins*, you must first use the PrepareAddGrids function to prepare a .mat file containing these extra grids, the name of which is then provided to the compiled *ProcessRiverBasins* function.

12.1.3 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.

Note for Compiled Versions: If you are running the compiled version of *ProcessRiverBasins*, the procedure for adding categorical grids is slightly different. Instead of running *CatPoly2GRIDobj* and providing the result directly to *ProcessRiverBasins*, you must instead use the PrepareCatAddGrids function to prepare a .mat file containing these categorical grids, the name of which is then provided to the compiled *ProcessRiverBasins* function.

12.1.4 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.

Note for Compiled Versions: The compiled version of *ProcessRiverBasins* (and the related *SubDivideBigBasins* function) will still produce the .mat file described above for use in other functions. However, if you wish to directly access the contents of each .mat file, there are several options available to you. For accessing the raster and vector data stored for each basin, you can run *ProcessRiverBasins* with the optional *'write_arc_files'* parameter set to true or alternatively use the compiled version of *Mat2Arc* after the completion of *ProcessRiverBasins* to extract and save versions of these files usable in a GIS program. For accessing the statistics, running the compiled version of *CompileBasinStats* will produce a table (that is output as a text file) that contains the majority of these statistics. As an alternative, you can also read the individual .mat files directly using the scipy.io module of python.

12.2 *CatPoly2GRIDobj*

As discussed in section 12.1.3, 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'.

Note for Compiled Versions: There is no compiled version of *CatPoly2GRIDobj*. The PrepareCatAddGrids function replaces it and prepares inputs directly for the compiled version of *ProcessRiverBasins*.

12.3 *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 18) which in turn will result in different populations of basin statistics (Figure 20). Identical statistics and operations are performed on each sub-basin as were for the main basins produced by *ProcessRiverBasins*.

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.

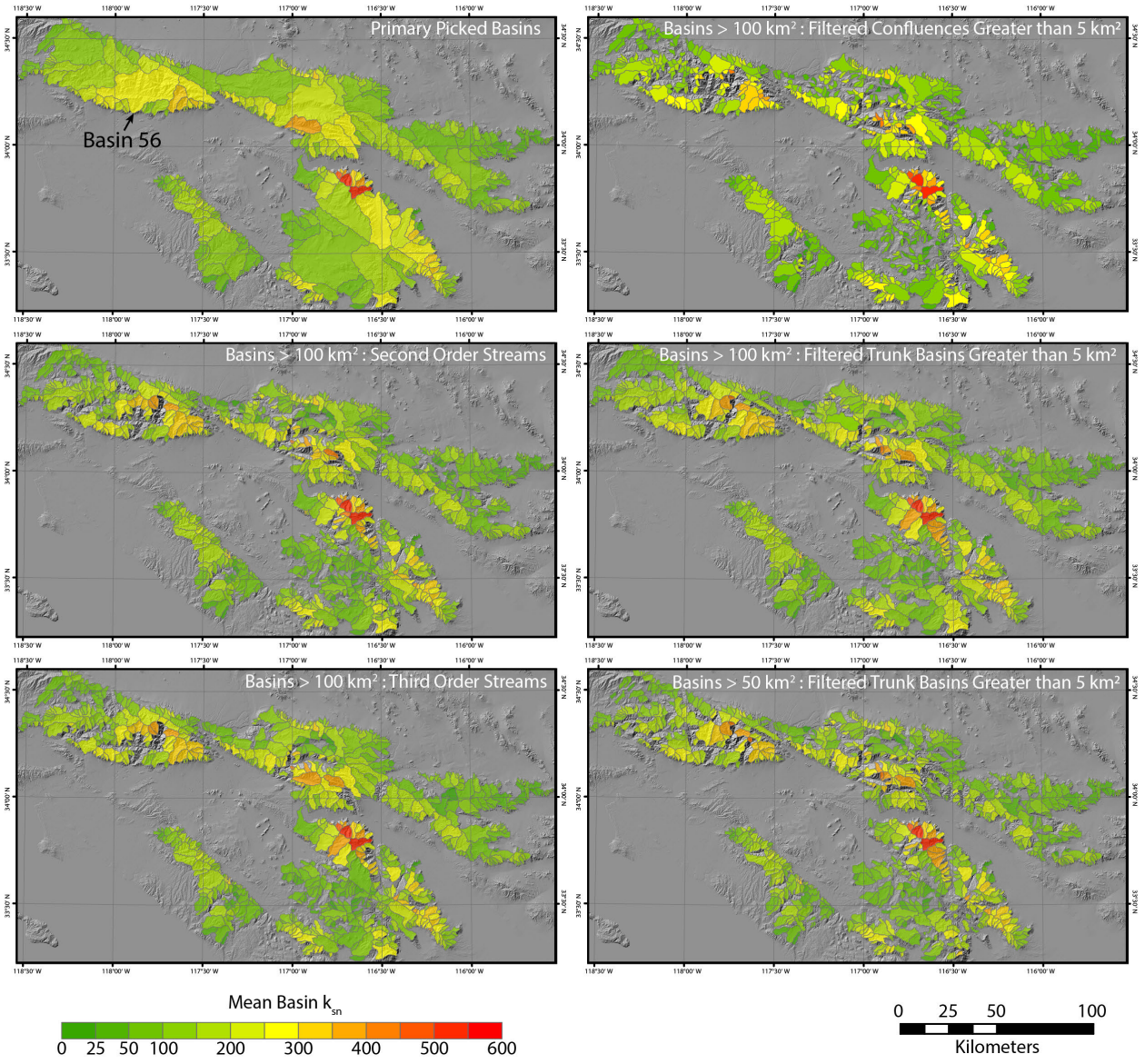


Figure 18: Comparison of mean k_{sn} values calculated for manually selected basins (upper left panel) from *ProcessRiverBasins* and for a variety of sub-basin division schemes using *SubDivideBigBasins*. Data is from the Southern California sample dataset.

12.4 *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 manually 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. Similar to other options in TAK where you manually define the location of a feature (on the basis of clicking on a plot), choices of basin knickpoints are not strictly reproducible, but because the code restricts the location of a knickpoint to being on a stream node, they are in practice closer to being reproducible.

12.5 *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.

12.6 *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.

You can also have the code do some limited extra calculations while producing the shapefile as well, specifically, you can recalculate mean and standard deviations and/or standard errors of k_{sn} using difference reference concavities by providing an argument to the *'new_concavity'* parameter:

```
% To recaculate mean normalized channel steepness with a reference concavity
    of 0.45
Basin2Shape(DEM,'basin_dir','new_concavity',0.45);
% To recalculate mean normalized channel steepness at reference concavities of
    0.45, 0.55, and 0.60
Basin2Shape(DEM,'basin_dir','new_concavity',[0.45 0.55 0.60]);
```

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.

Note for Compiled Versions: In the compiled version of Basin2Shape, extra field values are dealt with differently. Instead of providing separate inputs for the values and names for those values, a text file is provided to the *'extra_field_values'* parameter and the names of columns in that text file are used as the names of the resulting categories.

12.7 *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'*.

12.8 *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 recalculating mean k_{sn} values at new concavities) 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.

12.8.1 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.

12.8.2 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.

12.8.3 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).

Note for Compiled Versions: If using the compiled version CompileBasinStats and you wish to supply extra fields, the procedure is identical as described for the the compiled version of Basin2Shape.

12.9 *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.

12.9.1 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 19) 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.

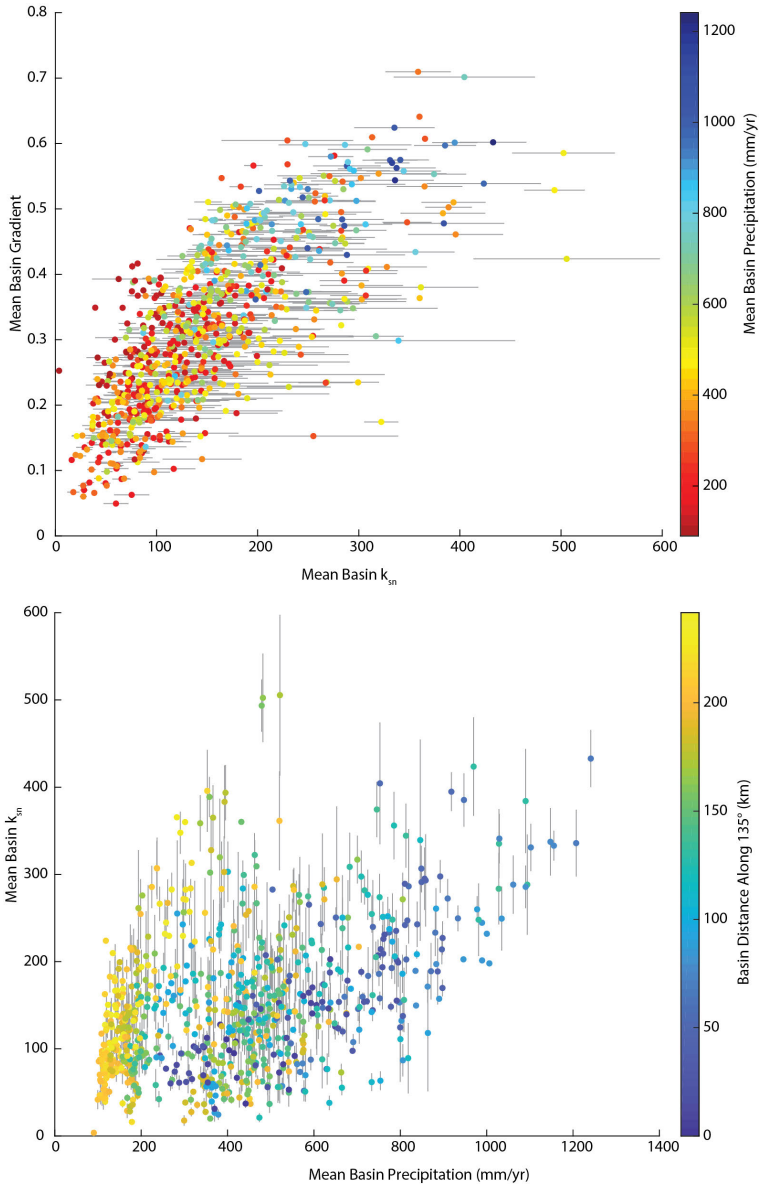


Figure 19: Selected outputs of *BasinStatsPlots* showing some options for plots using the *'color_by'* optional parameter. Data is from the Southern California sample dataset using the basins > 100 km^2 filtered by trunk confluences > 5 km^2.

12.9.2 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 reflect the morphologic response of hillslope and fluvial domains to the erosion rate, respectively. 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 20).

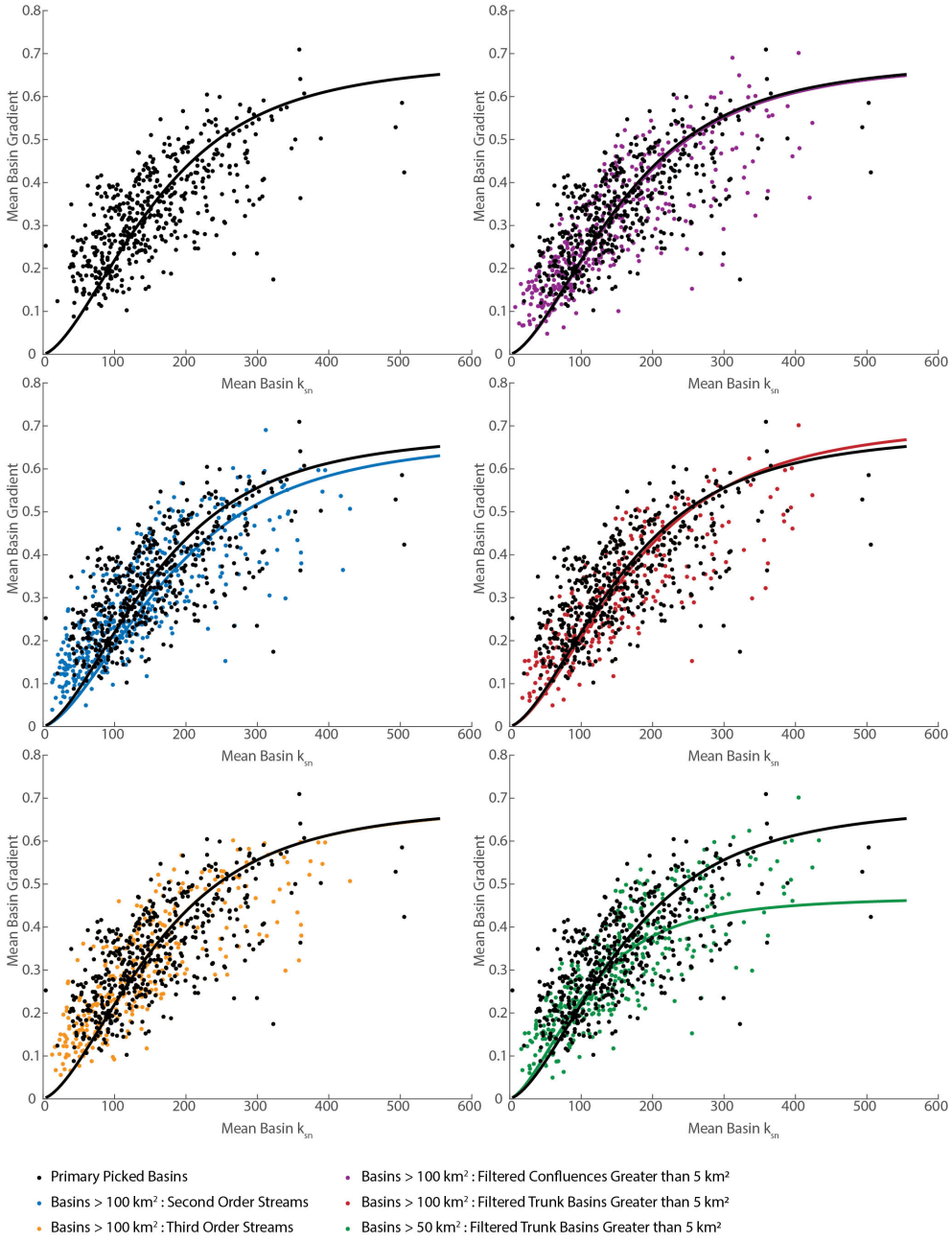


Figure 20: Comparisons of mean k_{sn} vs mean basin gradient and associated fits for a variety of sub-basin division schemes using *SubDivideBigBasins* using the *'grd_ksn'* plot from *BasinStatsPlots*. See Figure 18 for a map view of these different basin division schemes.

Via the optional *'fit_grd_ksn'* parameter, you can also use the relationship between 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. In detail, the analysis relies on the idea that the morphology of both the hillslope and fluvial portions of a landscape vary as a function of erosion rate. We can use these relationships with erosion rates, even if the erosion rates are unknown, to estimate these parameters. For the fluvial portion of the landscape, we use:

$$E = K(k_{sn})^n \tag{12}$$

where E is mean erosion rate, K is fluvial erosional efficiency, k_{sn} is basin mean normalized channel steepness, and n is the slope exponent from the stream power law [e.g., Snyder et al., 2003, Ouimet et al., 2009]. For the hillslope portion of the basin, we use the combination of two equations from Roering et al. [2007], first an equation for variation of hillslope elevation z as a function of distance x:

$$z(x) = \frac{-{S_c}^2}{2\beta E}\left[\sqrt{D^2 + (2\beta E x/S_c)^2} - D\ln\left(\frac{\sqrt{D^2 + (2\beta E x/S_c)^2} + D}{2\beta E/S_c}\right)\right] \tag{13}$$

where S_c is a limiting hillslope gradient where sediment flux becomes infinite, D is a diffusivity coefficient, $\beta = (\rho_r/\rho_s)$ and ρ_r and ρ_s are the densities of rock and sediment and β is set to 2. This equation is in turn used to determine a mean hillslope gradient, S_{avg}, (as a function of erosion rate):

$$S_{avg} = \frac{z(0) - z(L_H)}{L_H} \tag{14}$$

where L_H is a characteristic hillslope length, which is found by numerically solving the following expression:

$$\left(\frac{E}{2K{L_H}^2}\right)^{\frac{1}{n}} = \frac{D{S_c}^2}{\beta E|L_H|}\left(\sqrt{1 + \left(\frac{\beta E L_H}{D S_c}\right)^2} - 1\right) \tag{15}$$

which is effectively finding the distance from the divide where the slope of the hillslope portion and fluvial portions of the landscape are equal and that the contributions from erosion of the hillslope and fluvial portions of the landscape are also equal [e.g., Howard, 1997, Perron et al., 2008]. The fits are done via a constrained minimization of least absolute deviation on the above set of equations, and to successfully run the fits, you need access to the Optimization Toolbox in Matlab. 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, which are starting values for D, K, and S_c respectively. Also important is the optional *'n_val'* parameter (i.e. n) which is not a free parameter (i.e. you must provide a value for *'n_val'* are use the default value of 2) 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 20). It should be emphasized that the values for the limiting gradient, fluvial erodibility, and hillslope diffusivity extracted from this fit should be considered with some level of skepticism. If quantitative erosion rates are available for a region of interest, using these erosion rates to estimate the referenced parameters will always yield a more reliable result. The purpose of using the relationships here is mostly for relative comparison between basins within a study region.

12.9.3 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.

12.9.4 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.

12.9.5 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 21). 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 21). 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.

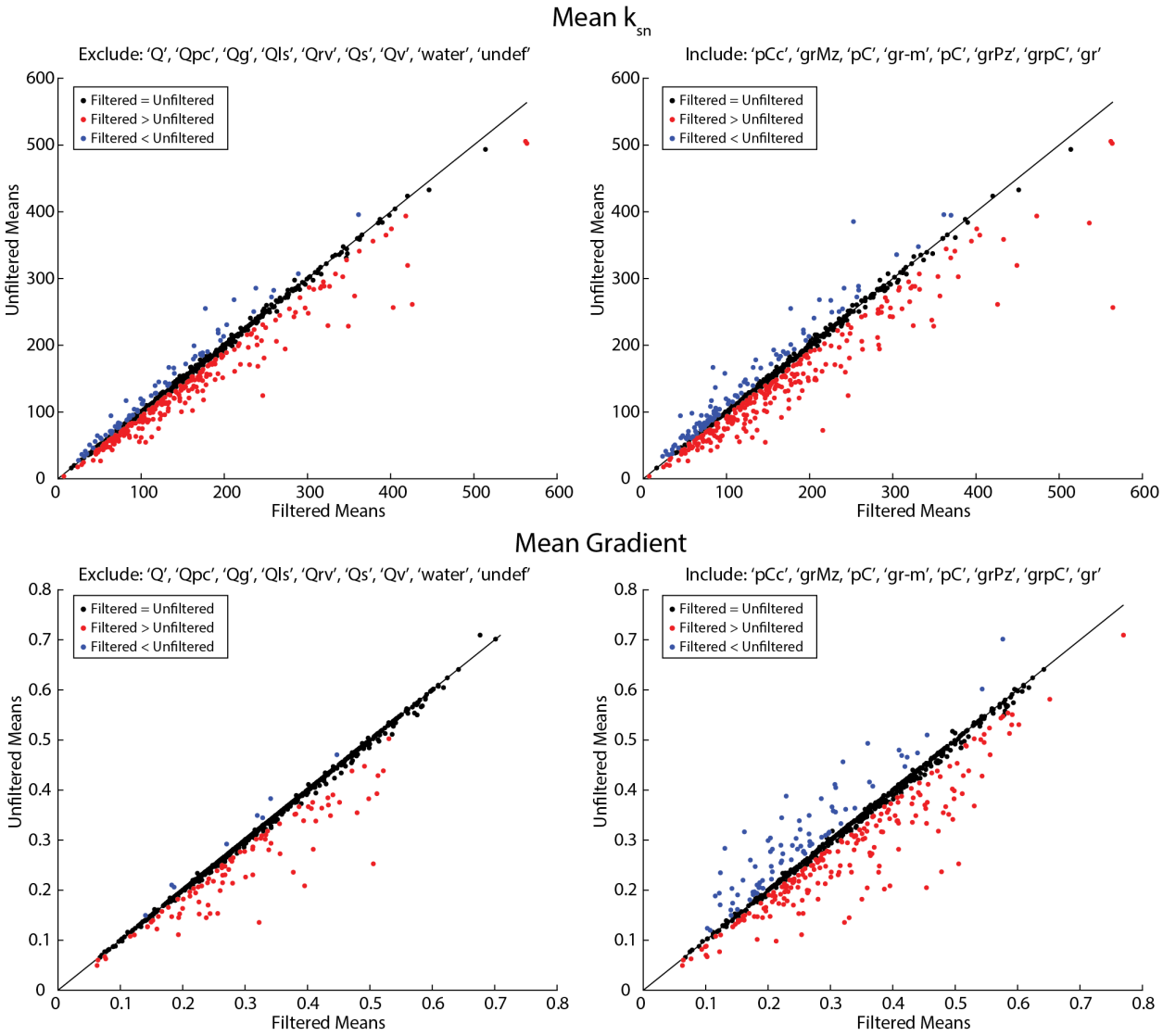


Figure 21: Selected outputs of *'compare_filtered'* plot from *BasinStatsPlots* comparing results of an excluding quaternary, water, and undefined areas vs including basement rocks. Data is from the Southern California sample dataset using the basins > 100 km^2 filtered by trunk confluences > 5 km^2.

12.9.6 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.

12.9.7 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.

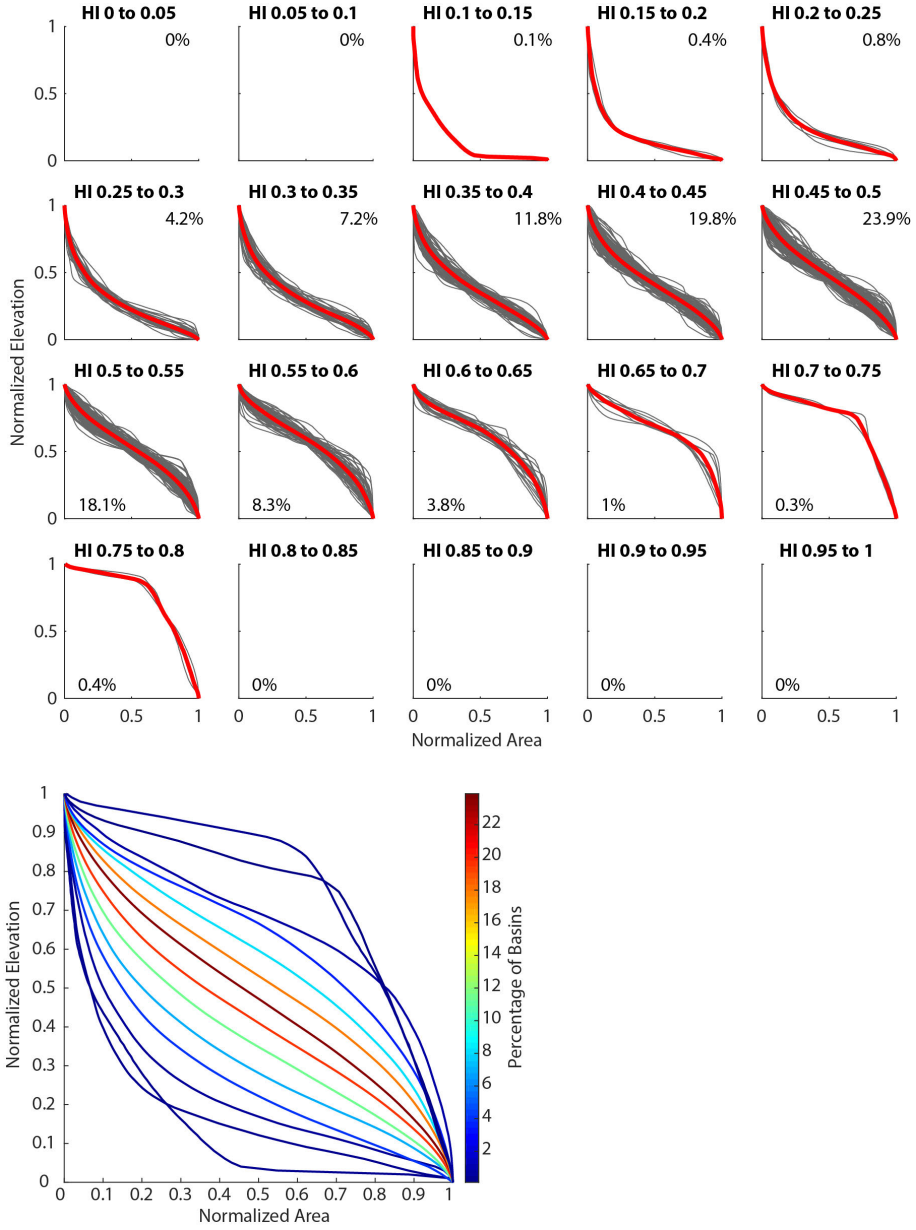


Figure 22: Selected output of *'stacked_hypsometry'* plots from *BasinStatsPlots*. Data is from the Southern California sample dataset using the basins > 100 km^2 filtered by trunk confluences > 5 km^2.

12.9.8 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 22. This option will plot figures of stacked normalized hypsometries (not included in Figure 22), a version of the stacked normalized hypsometries displayed as heatmap (not included in Figure 22), stacked hypsometries with elevations vs percentage area (not included in Figure 22), 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 22), 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 22).

12.9.9 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 23). This requires that you specify a statistic to compare using the optional *'statistic_of_interest'* parameter.

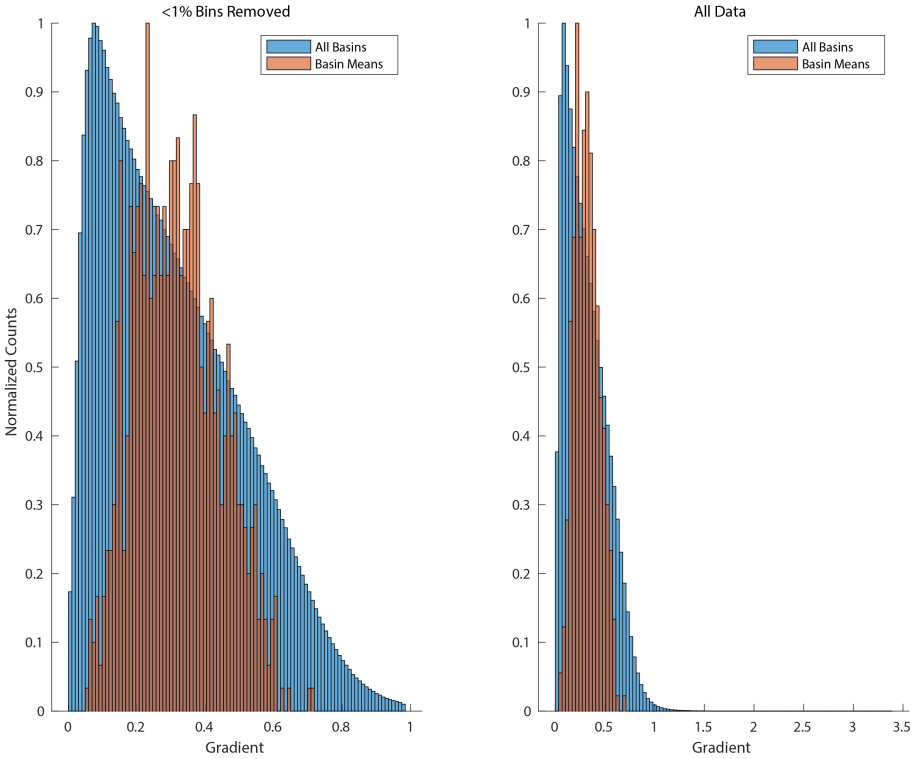


Figure 23: Sample output of *'compare_mean_and_dist'* plot from *BasinStatsPlots* comparing the distribution of gradient across all nodes in all basins (blue) to the distribution of mean gradients across all basins (orange). Data is from the Southern California sample dataset using the basins > 100 km^2 filtered by trunk confluences > 5 km^2.

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 24). 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.

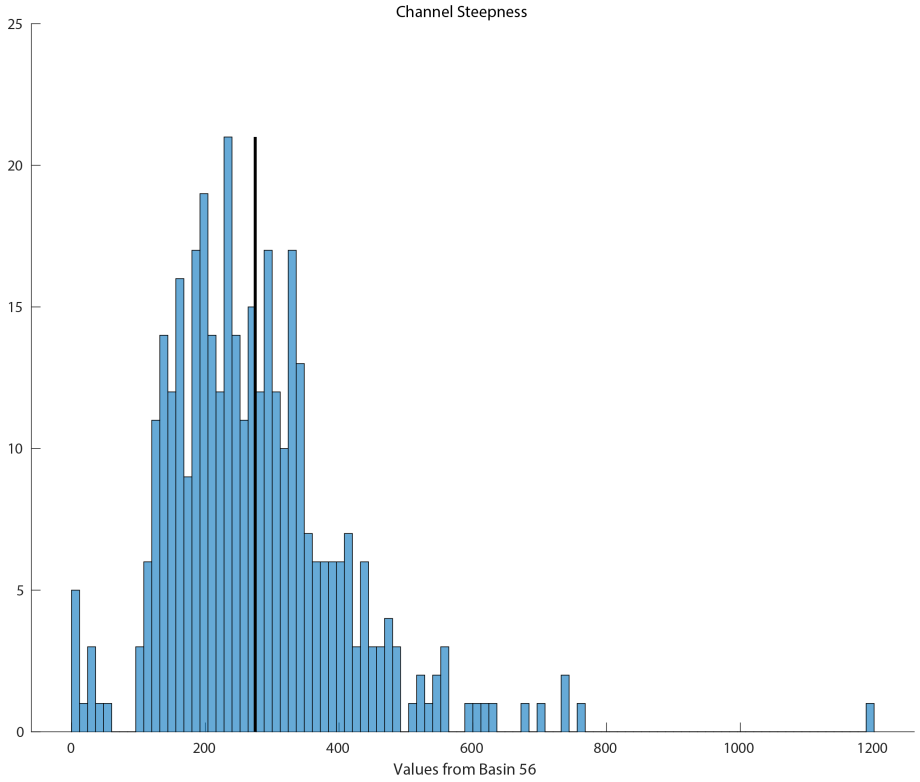


Figure 24: Sample output of *'compare_mean_and_dist'* plot from *BasinStatsPlots* comparing the distribution of k_{sn} within Basin 56 (in the Southern California dataset) to the mean k_{sn} within that basin (the thick black line).

12.9.10 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 25). 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*.

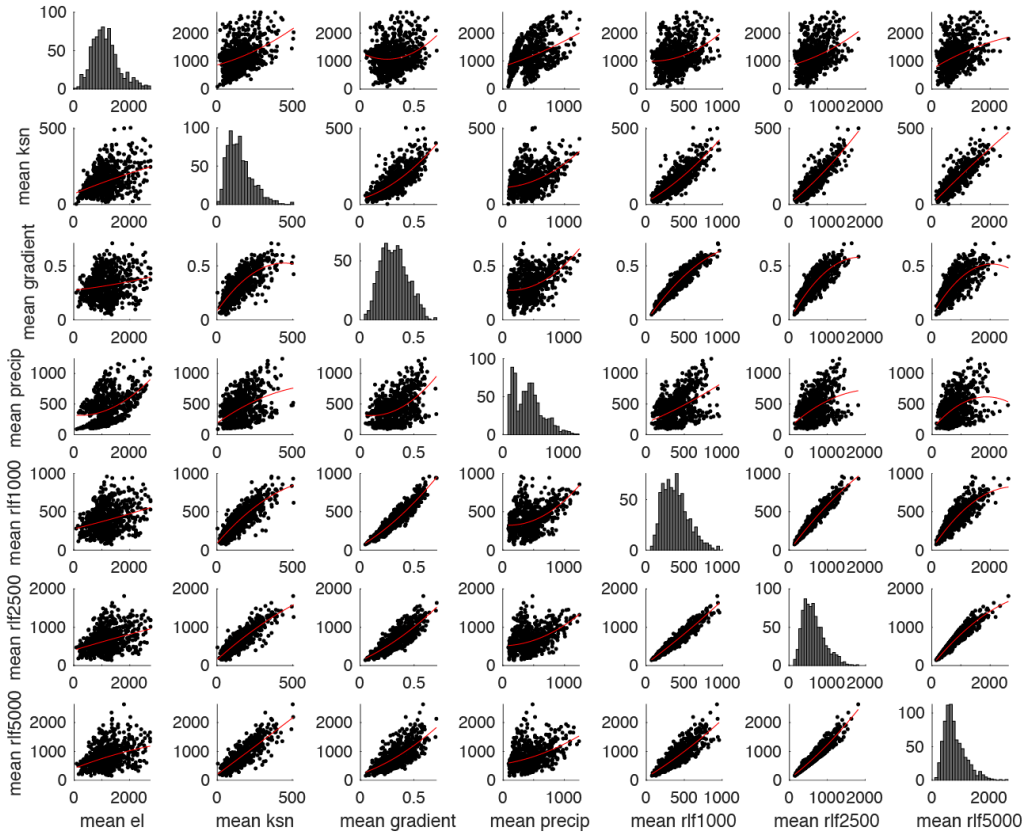


Figure 25: Sample output of *'scatterplot_matrix'* plot from *BasinStatsPlots*. Data is from the Southern California sample dataset using the basins > 100 km^2 filtered by trunk confluences > 5 km^2.

12.9.11 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 19).

13 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 26).

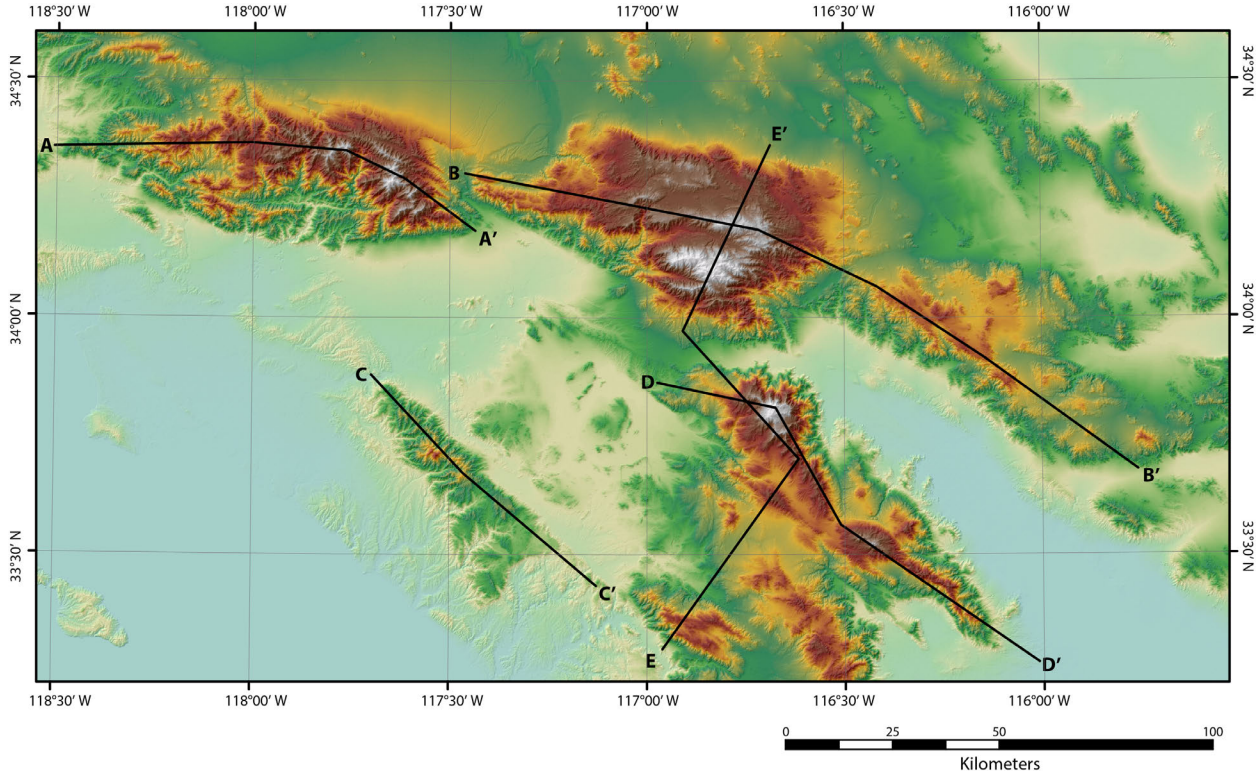


Figure 26: Location of swaths used to demonstrate uses of *MakeTopoSwath* and *MakeCombinedSwath*.

13.1 *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 27).

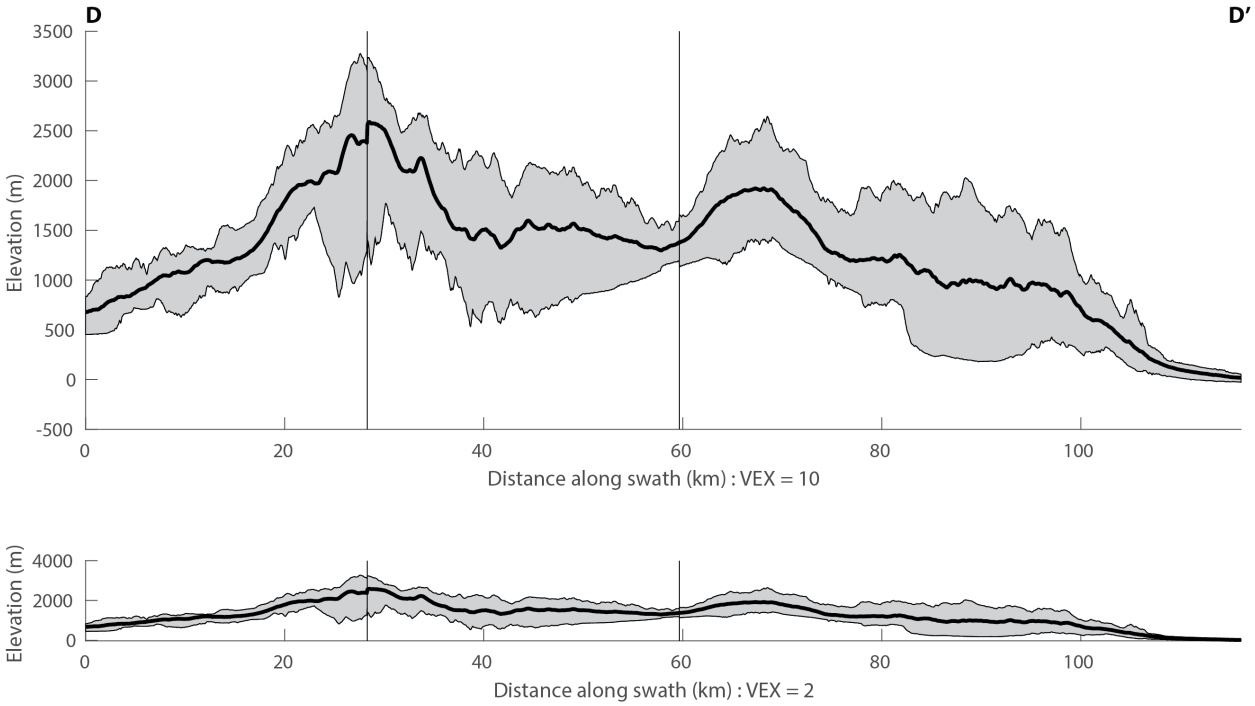


Figure 27: Basic swaths output from *MakeTopoSwath* highlighting the function of the *'vex'* parameter to control the vertical exaggeration.

In addition to the basic swath figure, with *MakeTopoSwath*, you can also display the output swath as an array of points (Figure 28) by setting *'plot_as_points'* to true or a heatmap (Figure 29) by setting *'plot_as_heatmap'* to true.

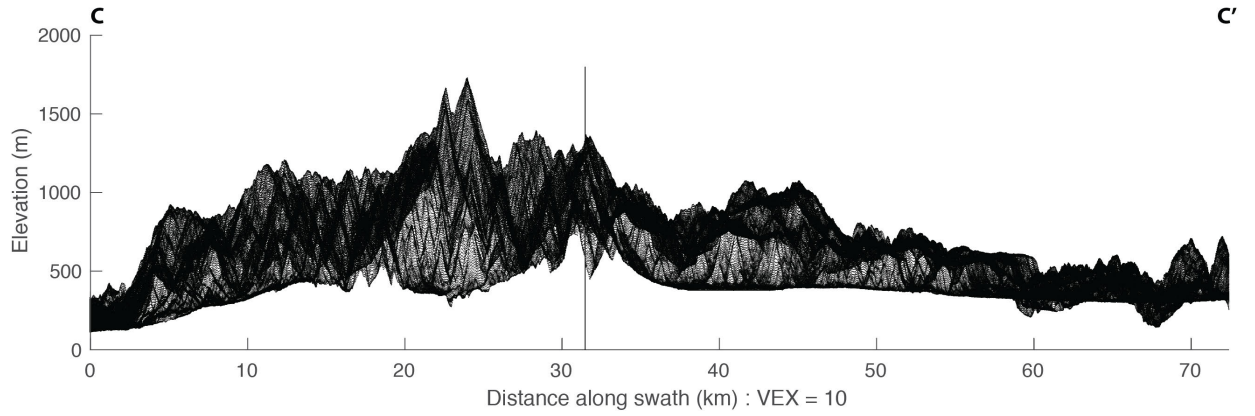


Figure 28: Swath output from *MakeTopoSwath* showing the result of plotting as points.

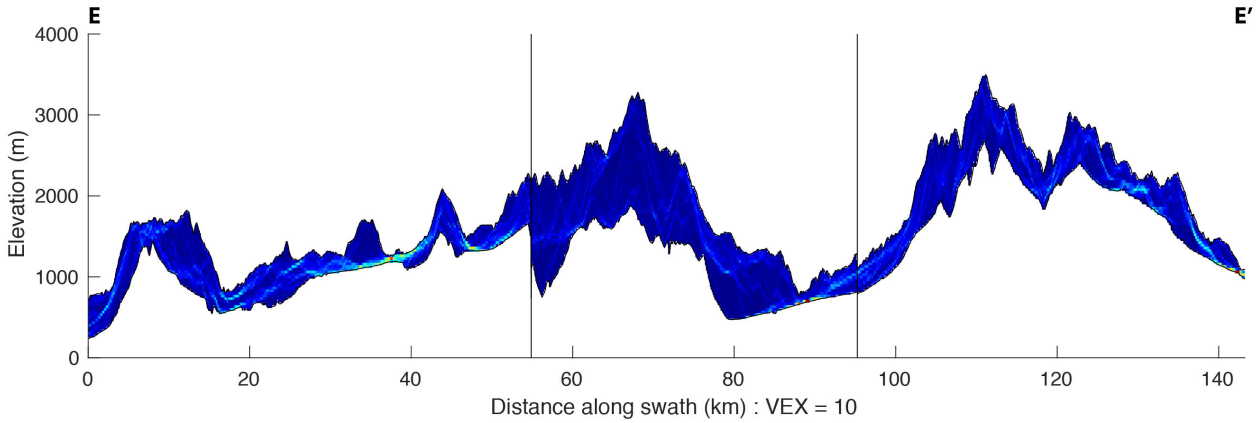


Figure 29: Swath output from *MakeTopoSwath* showing the result of plotting as a heatmap colored by the density of points at a given elevation.

13.2 *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 30).
- *'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 31).
- *'basin_knicks'* - will project knickpoint locations as produced by *FindBasinKnicks*.

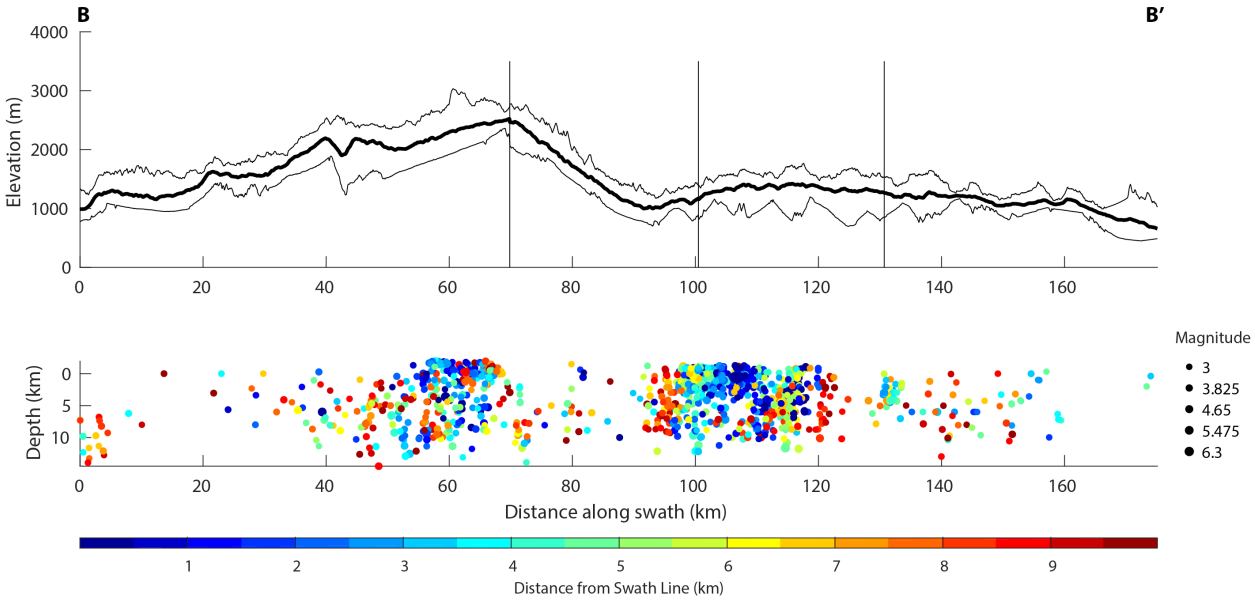


Figure 30: Swath output from *MakeCombinedSwath* projecting earthquake data

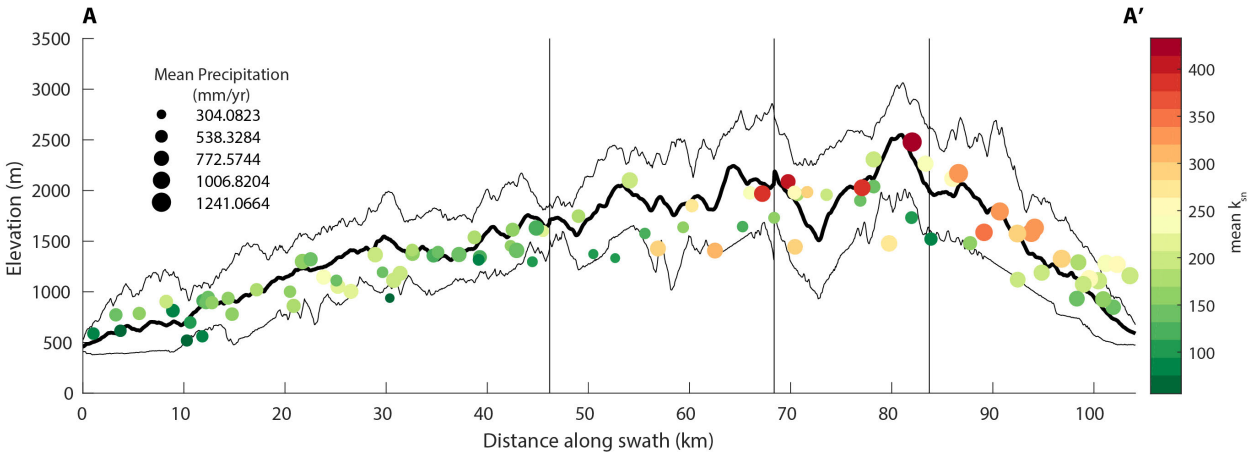


Figure 31: Swath output from *MakeCombinedSwath* projecting basin locations from *CompileBasinStats* and colored by mean k_{sn} and scaled by mean precipitation.

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.

13.3 *MakeSerialSwath*

In some cases, it is useful to produce a series of swath profiles that are semi-parallel to each other or follow a particular path. The *MakeSerialSwath* function is designed to make producing such a product simpler. As with *MakeTopoSwath*, you must provide a base *GRIDobj* from which to produce the swaths and a series of x-y points that will define the serial swaths, but in this case the swaths will be oriented perpendicular to the provided line. You can alternatively provide an empty array to this *'points'* input to initiate a graphical choice within Matlab to create the base line. Through required and optional inputs you can control the length of individual swaths, the width of the swaths either by directly specifying the width in map units or by specifying the number of equal width swaths to create along the base line, and the orientation of the swaths with respect to the baseline (i.e should the swaths be centered along the baseline as in Standard graphic output from *MakeSerialSwath* with a series of 50 equal width, 20 km long swaths, centered along the provided line (dark black line in the upper panel). The middle panel shows the minimum, mean, and maximum swath values for each swath (keyed by their color) and the bottom panel shows the average of all the swaths. or aligned along the 'top' or 'bottom' of the baseline).

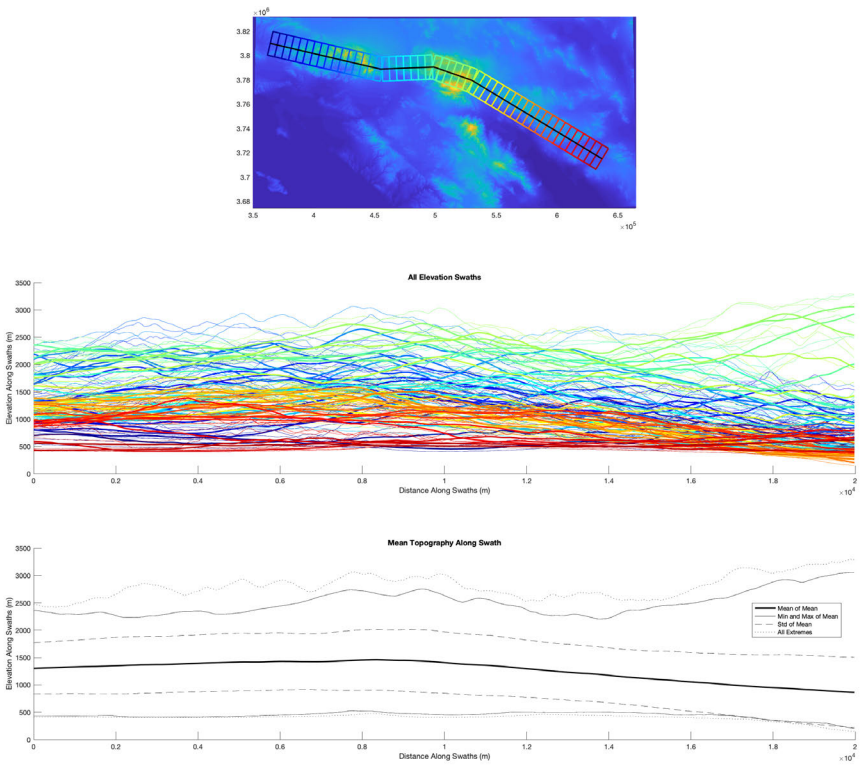


Figure 32: Standard graphic output from *MakeSerialSwath* with a series of 50 equal width, 20 km long swaths, centered along the provided line (dark black line in the upper panel). The middle panel shows the minimum, mean, and maximum swath values for each swath (keyed by their color) and the bottom panel shows the average of all the swaths.

The primary output of the *MakeSerialSwath* function is a cell array containing individual *SWATHobjs* created along the baseline. These individual *SWATHobjs* are suitable for use in other functions, e.g. *MakeCombinedSwath* or plotting with built in *TopoToolbox* function like *plotdz*. Specifically, if you wanted to use the output of *MakeSerialSwath* to make a series of combined swaths with *MakeCombinedSwath*, this can be easily accomplished with a simple loop:

```
% Generate serial swaths based on provided point array
[SWcell,~]=MakeSerialSwath(DEM,points,100,5000);
% Generate a combined swath for each serial swath a generic nx3 point dataset
    (points3) and save out figures for each with a unique name and save the
    projected data into another cell array
% Preallocate the output array
outData=cell(numel(SWcell),1);
for ii=1:numel(SWcell)
        % It is necessary to extract the points and width from each serial
            swath

        [~,~,~,outData{ii}]=MakeCombinedSwath(DEM,SWcell{ii}.xy0,SWcell{ii}.
            width,'points3',points3,SWcell{ii}.width*5,'save_figure',true,'
            file_name_prefix',['Comb_' num2str(ii)]);
end
```

13.4 *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. Note that there is no compiled version of *ProjectOntoSwath*.

13.5 *ProjectSmallCircleOntoSwath*

This function is similar to *ProjectOntoSwath*, but instead of projecting points along paths perpendicular to the swath, this function takes an input coordinate (in projected coordinates) to use as the center of a family of small circles along which to project the provided x-y points. This function can be used in combination with *BestFitSmallCircle* if you need to first determine a small circle center that would be appropriate. As with *ProjectOntoSwath*, the function outputs the location of points along the swath based on their projected positions. The function also calculates the distance from the point along the swath both in map units as measured along the small circle arc between the point and the swath center line and in degrees.

13.6 *ProjectGPSOntoSwath*

The *ProjectGPSOntoSwath* behaves identically to *ProjectOntoSwath* in terms of the projection of the provided x-y location onto the swath and similarly provides distance along and from the swath of those points, but it also takes velocity and associated uncertainties (assumed to be in the standard north and east components) and projects those onto the swath. The output of the function gives the magnitude of the projected GPS vector, i.e. the magnitude of

the velocity if resolved onto a plane parallel to the *SWATHobj* and the associated uncertainty, i.e. the radius of a slice through the error ellipse at the orientation of the *SWATHobj*. The function also provides north and east components for this projected vector which would be suitable inputs to the *quiver* function for plotting.

13.7 *CrossSwath*

Often, it is useful to precisely determine where a particular linear feature crosses a *SWATHobj* for plotting purposes. This function takes a *SWATHobj* and a line (defined by a series of x-y points at the vertices of the line in question) and calculates the positions along a swath where the line crosses the bounds of the swath (in both x-y coordinates and distance along the swath) and the mean crossing location of the line.

14 Miscellaneous

14.1 *BestFitSmallCircle*

Function to find the best fit small circle for a set of provided x-y points. In addition to the x and y coordinates, you are expected to provided the projection information for the x and y coordinates as a structure. Assuming you have a *GRIDobj* with which these x-y points are associated, the necessary projection information (i.e. the input for the *'proj'* argument) is stored in the *DEM.georef.mstruct* or alternatively, if you are using an older version of TopoToolbox, *DEM.georef*. This information can also be directly extracted from a GeoTIFF with the *geotiffinfo* function. If you provide an empty array to the textit'proj' argument, the function assumes that the provided x-y points are already geographic coordinates. The function will output the center of the best fit small circle (in geographic coordinates), the radius of the small circle (in degrees), and a series of points along the perimeter of the small circle (in projected coordinates unless the original points are provided as geographic)

```
% Produce a best fit small circle from a series of x and y coordinates
    extracted from a particular DEM
[clat,clon,crad,cpx,cpy]=BestFitSmallCircle(x,y,DEM.georef.mstruct);
% Alternatively, grab the projection info directly from a geotiff
proj=geotiffinfo('srtm_bigtujunga30m_utm11.tif');
[clat,clon,crad,cpx,cpy]=BestFitSmallCircle(x,y,proj);
% You can use the built in projinv and projfwd functions to convert the clat
    and clon or cpx and cpy outputs
[center_x,center_y]=projfwd(proj,clat,clon); % Convert center to projected
[cp_lat,cp_lon]=projinv(proj,cpx,cpy); % Convert perimeter to geographic
```

14.2 *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 31 for an example of the color map. Note that there is no compiled version of *ksncolor*.

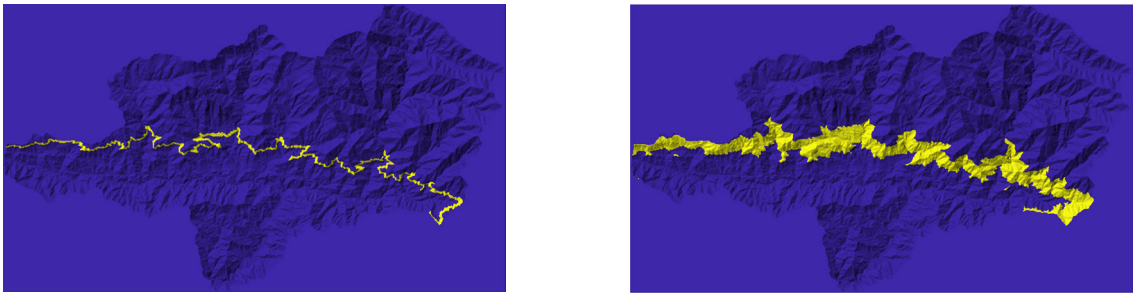
14.3 *PlotKsn*

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 *Process-*

RiverBasins 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.

14.4 *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 33). 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 33: 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.

14.5 *HackRelationship*

The *HackRelationship* function can be used to estimate the so-called Hack coefficient and exponent for a portion of a landscape. This refers to 'Hack's Law', originally described by Hack [1957] which related the change in drainage area along a stream to distance along the stream such that:

$$x = C * da^h \tag{16}$$

where x is distance measured in the downstream direction in map units, da is drainage area in map units squared, C is the Hack coefficient, and h is the Hack exponent. This same relationship has been presented in an alternative form by Whipple and Tucker [1999]:

$$da = C * x^h \tag{17}$$

where the values are the same, but obviously the values of C and h are interpreted differently. This function allows you to fit the relationship with either form using the optional *'relation'* parameter, with the *'original'* (Equation 16) or *'inverse'* (Equation 17) form. To compute the relationship, the function will calculate a value of C and h for each drainage basin, defined by the number and location of outlets in the provided *STREAMobj* and also calculates a 'global' value of C and h from all relevant points in the landscape. The function has a variety of options to control what data is used to fit. For example, for a given watershed, the function can measure distances from the channel heads to the

outlets or from the divides to the outlets. The function can also do the fit using only data along the trunk stream in a watershed, along all streams in a watershed, or using all contributing pixels in a watershed.

14.6 *Mat2Arc*

Mat2Arc is a helper function that takes any matfile as an input and will search its contents and produce either ESRI ascii grids or geotiffs and shapefiles from any valid datasets. Specifically, will make ascii grids or geotiffs of any *GRIDobj*, will convert any *FLOWobj* to an Arc style flow raster an export as an ascii grid or geotiff, and output any *STREAMobj* or recognizable geographic data structure as a shapefile. Whether rasters are saved as ESRI ascii grids or geotiffs is controlled with the optional *'raster_type'* parameter.

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*, 7:87–95, 2019. doi: 10.5194/esurf-7-87-2019.

J T Hack. Studies of longitudinal stream profiles in Virignia and Maryland. Technical report, 1957.

Alan D Howard. Optimal angles of stream junction: Geometric, stability to capture, and minimum power criteria. *Water Resources Research*, 7(4):863–873, 1971.

Alan D Howard. Badland morphology and evolution: Interpretation using a simulation model. *Earth Surface Processes and Landforms*, 22:211–227, 1997.

W.R. James and W.C. Krumbein. Frequency Distributions of Stream Link Lengths. *The Journal of Geology*, 77(5): 544–565, 1969.

William B Ouimet, Kelin X Whipple, and Darryl E Granger. Beyond threshold hillslopes: Channel adjustment to base-level fall in tectonically active mountain ranges. *Geology*, 37(7):579–582, 2009.

J Taylor Perron, William E Dietrich, and James W Kirchner. Controls on the spacing of first-order valleys. *Journal of Geophysical Research*, 113:F04016, 2008. doi: 10.1029/2007JF000977.

Joshua J Roering, J Taylor Perron, and James W Kirchner. Functional relationships between denudation and hillslope form and relief. *Earth and Planetary Science Letters*, 264:245–258, 2007.

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.

Wolfgang Schwanghart and Dirk Scherler. Bumps in river profiles: Uncertainty assessment and smoothing using quantile regression techniques. *Earth Surface Dynamics*, 5(4):821–839, 2017. ISSN 2196632X. doi: 10.5194/esurf-5-821-2017.

Noah P Snyder, Kelin X Whipple, Gregory E Tucker, and Dorothy J Merrits. Importance of a stochastic distribution of floods and erosion thresholds in the bedrock river incision problem. *Journal of Geophysical Research*, 108(B2), 2003. doi: 10.1029/2001JB001655.

Kelin X Whipple and Gregory E Tucker. Dynamics of the stream-power river incision model: Implications for height limits of mountain ranges, landscape response timescales, and research needs. *Journal of Geophysical Research*, 104(B8):17661–17674, 1999.

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.

A Headers for Compiled Functions

The following section reproduces the headers of each compiled function. Please refer to the Compiled Functions section of the user manual for additional information on how to use the compiled versions of the functions.

A.1 AutoKsnProfiler

Description: Prototype function for automated ksn fits similar to what is done with KsnProfiler. Function uses the MATLAB 'ischange' function to find breaks in binned ksn values. The sensitivity of the function, i.e. how small a change in binned ksn value is interpreted as a breakpoint, is controlled by the threshold_ratio parameter. Users can explore the affect of different values of threshold ratio, along with reference concavity and segment length, with the optional 'explore_param' option.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- thresh_ratio - value between 0 and 1 that controls how sensitive the function is to change in the running mean of binned ksn vs distance. Values close to 0 result in a lower threshold and more break points.

Optional Inputs:

- prev_param [] - option to use the parameters from a previous run of 'AutoKsnProfiler' stored in the output '*_params.mat' where the prefix is controlled by the 'file_name_prefix' parameter. This will override any other user provided or default values for 'thresh_ratio', 'ref_concavity', and 'segment_length'. You can still run 'explore_param' to subsequently change the values stored in the input to 'prev_param', but the idea behind this is to make it easy to run the AutoKsnProfiler on multiple datasets with the exact same options.
- ref_concavity [0.50] - reference concavity (as a positive value) for calculating ksn
- calc_concavity [false] - logical flag to turn on calculation of a best fit concavity for each stream segment. This will slow down the function considerably so it's only recommended that you set this to true if you are explicitly interested in the concavity of individual segments.

- segment_length [1000] - length in map units over which to bin ksn and distance data over which the mean ksn values will be fit. Also used as the length scale over which to average values when producing the output map structure
- plot_example [false] - logical flag to turn on (true) a plot of results in profile view a single stream in the network. By default, the selected stream will be the longest trunk stream in the provided network. You can override by this providing a valid input to 'channeloi'. Provided as a static way to visualize effect of the choice of 'thresh_ratio', 'ref_concavity', and 'segment_length' on the result. See 'explore_param' for a more dynamic way to explore the effect of these values.
- explore_param [false] - logical flag to turn on (true) a plot of results in profile view a single stream in the network and the option to change the values for 'thresh_ratio', 'ref_concavity', and 'segment_length' interactively. By default, the selected stream will be the longest trunk stream in the provided network. You can override by this providing a valid input to 'channeloi'.
- channeloi [] - 1 x 2 array containing the x-y coordinates of a channel head of a stream to use as the selected stream if either 'plot_example' or 'explore_param' is set to true. If neither of those are true, this input has no effect on the behavior of the code.
- conditioned_DEM [] - option to provide a hydrologically conditioned DEM for use in this function, expects the mat file as saved by 'cmpConditionDEM' See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)
- file_name_prefix ['auto'] - name prefix for the shapefiles, textfiles, and matfiles to be export, must have no spaces to be a valid name for ArcGIS and should NOT include any file extensions.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
AutoKsnProfiler /path/to/wdir Topo.mat 0.5
AutoKsnProfiler /path/to/wdir Topo.mat 0.5 prev_param auto_params.mat
AutoKsnProfiler /path/to/wdir Topo.mat 0.25 ref_concavity 0.4 explore_param
    true
```

A.2 Basin2Raster

Description: Function takes outputs from 'ProcessRiverBasins' function and produces a single GRIDobj with individual drainage basins (as selected by 'ProcessRiverBasins' and 'SubDivideBigBasins') assinged various values

Required Inputs:

- wdir - full path of working directory
- MakeStreamsMat - full path of the matfile provided as an input to cmpProcessRiverBasins
- valueOI - value to assign to basins, acceptable inputs are:
  - 'ksn' - mean ksn value of basin

  - 'gradient' - mean gradient of basin
  - 'elevation' - mean elevation of basin
  - 'relief' - mean relief of basin (must specify the radius of interest with the 'relief_radius' parameter)
  - 'chir2' - R^2 value of chi-z fit (proxy for disequilibrium)
  - 'drainage_area' - drainage area in km2 of basin
  - 'hypsometric_integral' - hypsometric integral of basin
  - 'id' - basin ID number (i.e third column RiverMouth output)
  - 'theta' - best-fit concavity resultant from the topo toolbox chiplot function
  - 'NAME' - where name is the name provided for an extra grid (i.e. entry to second column of 'add_grid' or entry to third column of 'add_cat_grid'), value input will be mean for additional grid names or mode for additional categorical grid names

- location_of_data_files - full path of folder which contains the mat files from 'ProcessRiverBasins' as a string

Optional Inputs:

- file_name_prefix ['basins'] - prefix for outputs, will automatically append the type of output, i.e. 'ksn', 'elevation', etc
- location_of_subbasins ['SubBasins'] - name of folder that contains subbasins of interest (if you created subbasins using "SubDivideBigBasins"), expected to be within the main Basin folder provided with "location_of_data_files".
- method ['subdivided'] - method used for subdividing watersheds. If you used 'ProcessRiversBasins' and then 'SubDivideBigBasins' or if you only used 'ProcessRiverBasins' but did not pick any nested catchments, i.e. none of the river mouths supplied to 'ProcessRiverBasins' were within the catchment boundaries of other watersheds for which you provided river mouths, then you should use use 'subdivided' which is the default so you do not need to specify a value for this property. If you picked nested catchments manually and then ran 'ProcessRiverBasins' you should use 'nested'.
- relief_radius [2500] - relief radius to use if 'valueOI' is set to 'relief'

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
Basin2Raser /path/to/wdir MakeStreams.mat ksn MainBasins
Basin2Raster /path/to/wdir MakeStreams.mat ksn MainBasins
    location_of_subbasins MySubbasins file_name_prefix Test
```

A.3 Basin2Shape

Description: Function to take the outputs from 'ProcessRiverBasins' and 'SubDivideBigBasins' and produce a single shapefile showing the outlines of polygons and with commonly desired attributes from the results of 'ProcessRiverBasins' etc. See below for a full list of fields that the output shapefile will include. If additional grids were provided to 'ProcessRiverBasins', mean and standard error values for those grids will be auto-populated in the shapefile and the name of the fields will be the character array provided in the second column of additional grids input. This function also allows you to input a list of additional fields you wish to include (see Optional Inputs below). If you would rather create a GRIDobj with specified values, use 'Basin2Raster'.

Required Inputs:

- wdir - full path of working directory
- MakeStreamsMat - name of the matfile provided as an input to cmpProcessRiverBasins
- location_of_data_files - name of folder which contains the mat files from 'ProcessRiverBasins'

Optional Inputs:

- location_of_subbasins ['SubBasins'] - name of folder that contains subbasins of interest (if you created subbasins using "SubDivideBigBasins"), expected to be within the main Basin folder provided with "location_of_data_files". Note that if you do not provide the correct directory name for the location of the subbasins, subbasin values will not be included in the output regardless of your choice for the "include" parameter.
- shape_name ['basins'] - name for the shapefile to be export, must have no spaces to be a valid name for ArcGIS and should NOT include the '.shp'
- include ['all'] - parameter to specify which basins to include in building the shapfile. The default 'all' will include all basin mat files in the folder you specify. Providing 'subdivided' will check to see if a given main basin was subdivided using 'SubdivideBigBasins' and then only include the subdivided versions of that basin (i.e. the original main basin for those subbasins will not be included in the shapefile). Providing 'bigonly' will only include the original basins produced by 'ProcessRiverBasins' even if 'SubDivideBigBasins' was run. If 'SubDivideBigBasins' was never run, result of 'all' and 'bigonly' will be the same.
- extra_field_values [] - name text file of extra field values you wish to include. The first column in this file must be the river basin number (i.e. the identifying number in the third column of the RiverMouth input to ProcessRiverBasins or the number generated for the basin in SubDivideBigBasins). Only one row per river basin number is allowed and ALL river basin numbers in the basins being processed must have a value associated with them. Additional columns are interpreted as the values with which you wish to populate the extra fields. These can either be character arrays or numbers, other values will results in an error. The function will use the header names within this file to name fields in the output shapefile
- new_concavity [] - a 1 x m array of concavity values to recalculate normalized channel steepness statistics (mean, standard error and/or standard deviation) using the provided concavities.
- uncertainty ['se'] - parameter to control which measure of uncertainty is included, expects 'se' for standard error (default), 'std' for standard deviation, or 'both' to include both standard error and deviation.
- populate_categories [false] - logical flag to add entries that indicate the percentage of a watershed occupied by each category from a categorical grid, e.g. if you provided an entry for 'add_cat_grids' to ProcessRiverBasins that was a geologic map that had three units, 'Q', 'Mz', and 'Pz' and you set 'populate_categories' to true there will be field names in the resulting shapefile named 'Q', 'Mz', and 'Pz' and the values stored in those columns will correspond to the percentage of each basin covered by each unit for each basin. Setting populate_categories to true will not have any effect if no entry was provided to 'add_cat_grids' when running ProcessRiverBasins.

Output:

- Saves a shapefile with the following default fields:
  - river_mouth - river mouth number provided to ProcessRiverBasins

  - drainage_area - drainage area of basin in km^2
  - center_x - x coordinate of basin in projected coordinates
  - center_y - y coordinate of basin in projected coordinates
  - outlet_elevation - elevation of pour point in m
  - mean_el - mean elevation of basin in meters
  - max_el - maximum elevation of basin in meters
  - mean_ksn - mean channel steepenss
  - mean_gradient - mean gradient
- Either standard errors, standard deviations or both will be populated for elevation, ksn, and gradient depending on value of 'uncertainty'
- Mean and standard error / standard deviation / both values will be populated for any additional grids

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
Basin2Shape /path/to/wdir MakeStreams.mat MainBasins
Basin2Shape /path/to/wdir MakeStreams.mat MainBasins location_of_subbasins
    MySubbasins include subdivided
```

A.4 BasinPicker

Description: Function takes results of makes streams and allows for interactive picking of basins (watersheds). Function was designed intially for choosing basins suitable for detrital analyses (e.g. Be-10 cosmo). Displays two panel figure with topography colored by elevation and local relief on which to pick individual basins. After the figure displays, it will wait until you press enter to begin the watershed picking process. This is to allow you to zoom, pan, etc to find a stream you are interested in. When you click enter, cross hairs will appear in the elevation map so you can select a pour point. Once you select a pour point, a new figure will display this basin and stream to confirm that's the watershed you wanted (it will also display the drainage area). You can either accept this basin or reject it if it was misclick. If you accept it will then display a new figure with the chi-z and longitudinal profiles for that basin. It will then give you a choice to either save the choice or discard it. Finally it will ask if you want to keep picking streams, if you choose yes (the default) it will start the process over. Note that any selected (and saved) pour point will be displayed on the main figure. As you pick basins the funciton saves a file called 'Outlets.mat' that contains the outlets you've picked so far. If you exit out of the function and restart it later, it looks for this Outlets file in the current working directory so you can pick up where you left off.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'

Optional Inputs:

- ref_concavity [0.50]- reference concavity for chi-Z plots
- rlf_radius [2500] - radius in map units for calculating local relief
- extra_grid [] - sometimes it can be useful to also view an additional grid (e.g. georeferenced road map, precipitation grid, etc) along with the DEM and relief. The arugment provided should be the name of the extra grid of interest. This grid can be a different size or have a different cellsize than the underlying dem (but still must be the same projection and coordinates system!), it will be resampled to match the provided DEM.
- cmap ['landcolor'] - colormap to use for the displayed maps. Input can be the name of a standard Matlab colormap
- conditioned_DEM [] - option to provide a hydrologically conditioned DEM for use in this function, expects the mat file as saved by 'cmpConditionDEM' See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)
- plot_type ['vector'] - expects either 'vector' or 'grid', default is 'vector'. Controls whether all streams are drawn as individual lines ('vector') or if the stream network is plotted as a grid and downsampled ('grid'). The 'grid' option is much faster for large datasets, but can result in inaccurate site selections. The 'vector' option is easier to see, but can be very slow to load and interact with.
- threshold_area [1e6] - used to redraw downsampled stream network if 'plot_type' is set to 'grid'
- refine_positions [] - expects the name of a text file containing a m x 2 array of x y positions that are near stream networks that you want to manually snap to the appropriate stream network. An example would be a series of GPS positions of river samples that don't quite lie on the stream network as determined by flow routing. If you provide an entry for 'refine_positions', the code will iteratively work through the provided points, displaying their location one point at a time along with a zoomed inset window (size controlled by 'window_size') on which you can precisely position the river mouth location.
- window_size [1] - size of inset window (in km) if you have provided an entry for 'refine_positions'

Outputs:

- Outlets - n x 3 matrix of sample locations with x coordinate, y coordinate, and basin ID number (valid input to 'ProcessRiverBasins' as 'river_mouths' parameter) saved as a mat file (for continuing a cmpBasinPicker run) and a text file

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
BasinPicker /path/to/wdir Topo.mat
BasinPicker /path/to/wdir Topo.mat conditioned_DEM CondDem.mat extra_grid
    precip.tif
```

A.5 BasinStatsPlots

Description: Function to take the complied outputs from 'ProcessRiverBasins' and 'SubDivideBigBasins' and produce various plots of aggregated basin values.

Required inputs:

- wdir - full path of working directory
- basin_table - name of the BasinTable.mat file output from 'cmpCompileBasinStats'
- plots - Type of plot you want to produce, valid inputs are:
  - 'grd_ksn' - plot of mean basin gradient vs mean basin channel steepness (e.g. see Forte et al, 2016, Earth and Planetary Science Letters for discussion of use of these plots)
  - 'grd_rlf' - similar to 'grd_ksn' but uses local relief instead of ksn, requires that relief was calculated when running ProcessRiverBasins. Assumes relief radius is 2500 (can set alternative radii with 'rlf_radius' optional parameter)
  - 'rlf_ksn' - plot of mean basin relief vs mean basin channel steepness
  - 'compare_filtered' - plot comparing mean values vs filtered mean values if you ran 'CompileBasinStats' and filtered by a category
  - 'category_mean_hist' - if you calculated 'means_by_category' when running 'CompileBasinStats', you can plot distributions of the means by category as histograms using this option. Requires an input to 'cat_mean1'
  - 'category_mean_compare' -if you calculated 'means_by_category' for more than one value (e.g. both gradient and ksn), you can compare the mean values by category using this plot. Requires inputs to both 'cat_mean1' (value that will be plotted on x axis) and 'cat_mean2' (value that will be plotted on y axis)
  - 'stacked_hypsometry' - plot hypsometries for the basins
  - 'compare_mean_and_dist' - plots a histogram of values within a selected basin or across all basins for a statistic of interest to compare to the mean value, accepts an input for 'statistic_of_interest' and 'basin_num'.
  - 'scatterplot_matrix' - matrix of scatterplots and histograms, designed to be sort of similar to 'lattice' plots in R. Providing a table for which you calculated filtered means and leaving 'use_filtered' set to false may produce a large matrix
  - 'xy' - generic plot, requires entries to optional 'xval' and 'yval' inputs

Optional Inputs:

General Parameters

- uncertianty ['se'] - uncertainty to value use for plots, valid options are 'se' (standard error), 'std' (standard deviation), or 'none'. Providing 'none' indicates you do not want to plot errorbars. Behavior of this option will depend on how you ran ProcessRiverBasins, e.g. if you only calculated standard deviations when running ProcessRiverBasins but supply 'se' here, the code will ignore your choice and use the standard deviation values.
- use_filtered [false] - logical flag to use filtered values for 'grd_ksn', 'grd_rlf', 'rlf_ksn', or 'scatterplot_matrix'. Will only work if you calculated filtered values when running 'CompileBasinStats'.

- color_by [] - value to color points by, valid for 'grd_ksn','grd_rlf','rlf_ksn', and 'xy', either the name of a column in the provided table or a m x 1 array of numeric values the same length as the provided table
- cmap [] - colormap to use if an entry is provided to 'color_by', can be the name of a standard colormap or a nx3 array of rgb values to use as a colormap.
- define_region [] - set of coordinates to define a rectangular region to draw data from, expects a four element matrix (row or column) that define the minimum x, maximum x, minimum y, and maximum y coordinate to include OR define as true to bring up a plot of all basin centers for you to select a region by drawing a rectangle. Works with all plots.
- rlf_radius [2500] - radius of relief used when plotting relief related values
- save_figure [false] - logical flag to save pdfs of all figures produced

xy plot

- xval [] - value to plot on x axis for plot type 'xy' provided as name of column as it appears in the provided table or a a m x 1 array of numeric values the same length as the provided table
- yval [] - value to plot on y axis for plot type 'xy' provided as name of column as it appears in the provided table or a a m x 1 array of numeric values the same length as the provided table

compare_mean_and_dist plot

- statistic_of_interest ['ksn'] - statistic of interest for plotting histogram to compare with mean value. Valid inputs are 'ksn', 'gradient', 'elevation', 'relief' (if you provide relief, the code will look for relief calculated at the radius specified with the optional 'rlf_radius' parameter), or the name of an additional grid provided to 'ProcessRiverBasins', e.g. if you provided a precipitation grid and provided the name 'precip' and a column named 'mean_precip' exists in the table, then 'precip' would be a valid input to this parameter.
- basin_num [] - number of basin (as it appears in the ID column of the table) to use for 'compare_mean_and_dist', if empty, 'compare_mean_and_dist' will use all basins.

category_mean_hist OR category_mean_compare

- cat_mean1 [] - category to use for plotting, see 'category_mean_hist' or 'category_mean_compare', valid inputs are 'ksn', 'rlf', 'gradient', or the name of an additional grid provided to ProcessRiverMeans.
- cat_mean2 [] - category to use for plotting, 'category_mean_compare' , valid inputs are 'ksn', 'rlf', 'gradient', or the name of an additional grid provided to ProcessRiverMeans.

Fit Gradient-Ksn Relationship

- fit_grd_ksn [false] - logical flag to initiate fitting of gradient - ksn relationship. Setting this flag to true only produces a result if the plot type is set to 'grd_ksn'. The relationship is a fit using a power law relationship between erosion rate and channel steepness and erosion rate and mean hillslope gradient. See Forte et al, 2016, Earth and Planetary Science Letters for further discussion. The fit optimizes values of hillslope diffusivity (D), fluvial erodibility (K), and threshold gradient (Sc). The best fit values for these will be printed to the console.
- start_diffusivity [0.01] - starting value for optimization of hillslope diffusivity parameter.

- start_erodibility [1e-7] - starting value for optimization of fluvial erodibility parameter.
- start_threshold_gradient [0.8] - starting value for optimzation of threshold hillslope gradient parameter.
- n_val [2] - n value on slope parameter, this is not a free parameter in the fit.

Fit Relief-Ksn Relationship

- fit_rlf_ksn [false] - logical flag to initiate simple linear fit of relief-ksn relationship (expectation is a linear relationship). Setting this flag to true only produces a result if the plot type is set to 'rlf_ksn'.

Fit Filtered Data

- fit_filtered [false] - logical flat to initiate a simple linear fit to filtered data. Setting this to true only produces a result if plot type is set to 'compare_filtered'.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
BasinStatsPlots /path/to/wdir BasinTable.mat grd_ksn
BasinStatsPlots /path/to/wdir BasinTable.mat xy xval mean_ksn yval center_x
```

A.6 ClassifyKnicks

Description: Function to iterate through a set of bounds (i.e. knickpoints) selected while running 'KsnProfiler'. The function will display a long profile and chi - elevation plot for individual stream segments and will iterate through each bound point you selected in KsnProfiler. The code expects you to input a number or character (at the command prompt) to categorize the knickpoint higlighted in red. You must be consistent in your choice (i.e. you must either use numbers for all of the classifications or characters for all the classifications within a given run), mixing numbers and characters will result in an error at the end of the run. For entering characters, it's recommended you keep these short strings without spaces (i.e. entries supported into a shapefile a attribute table), e.g. knick or bound

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins' that was used to run 'cmpKsnProfiler'
- KsnProfilerMat - Full path of matfile output from 'cmpKsnProfiler'

Optional Inputs:

- shape_name ['ksn'] - name for the shapefile to be export, must have no spaces to be a valid name for ArcGIS and should NOT include the '.shp'

Outputs: saves a shapfile of knickpoints including the classification you assign using this tool

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
ClassifyKnicks /path/to/wdir Topo.mat KsnProfiler.mat
ClassifyKnicks /path/to/wdir Topo.mat KsnProfiler.mat shape_name my_knicks
```

A.7 CompileBasinStats

Description: Function to take the outputs from 'ProcessRiverBasins' and 'SubDivideBigBasins' and produce a Matlab table that summarizes the results of ProcessRiverBasins and optionally SubDivideBigBasins. This table is a required input for 'BasinStatsPlots'. If additional grids were provided to 'ProcessRiverBasins', mean and standard error values for those grids will be included in the table. This function also allows you to input a list of additional fields you wish to include (see Optional Inputs below). There are also a variety of additional parameters / quantities that can be calculated if you provided a categorical grid to 'ProcessRiverBasins'.

Required Inputs:

- wdir - full path of working directory
- location_of_data_files - full path of folder which contains the mat files from 'ProcessRiverBasins'

Optional Inputs:

- location_of_subbasins ['SubBasins'] - name of folder that contains subbasins of interest (if you created subbasins using "SubDivideBigBasins"), expected to be within the main Basin folder provided with "location_of_data_files". Note that if you do not provide the correct directory name for the location of the subbasins, subbasin values will not be included in the output regardless of your choice for the "include" parameter.
- include ['all'] - parameter to specify which basins to include in building the shapfile. The default 'all' will include all basin mat files in the folder you specify. Providing 'subdivided' will check to see if a given main basin was subdivided using 'SubdivideBigBasins' and then only include the subdivided versions of that basin (i.e. the original main basin for those subbasins will not be included in the table). Providing 'bigonly' will only include the original basins produced by 'ProcessRiverBasins' even if 'SubDivideBigBasins' was run. If 'SubDivideBigBasins' was never run, result of 'all' and 'bigonly' will be the same.
- file_name_prefix ['Basins'] - parameter to specify a file name prefix for the output tables
- extra_field_values [] - name of text file of extra field values you wish to include. The first column in this file must be the river basin number (i.e. the identifying number in the third column of the RiverMouth input to ProcessRiverBasins or the number generated for the basin in SubDivideBigBasins). Only one row per river basin number is allowed and ALL river basin numbers in the basins being processed must have a value associated with them. Additional columns are interpreted as the values with which you wish to populate the extra fields. These can either be character arrays or numbers, other values will results in an error. The function will use the header names within this file to name fields in the output shapefile
- new_concavity [] - a 1 x m array of concavity values to recalculate normalized channel steepness statistics (mean, standard error and/or standard deviation) using the provided concavities.
- uncertainty ['se'] - parameter to control which measure of uncertainty is included, expects 'se' for standard error (default), 'std' for standard deviation, or 'both' to include both standard error and deviation.

- dist_along_azimuth [] - option to calculate distances along a given azimuth for all basins. Expects an single numeric input, interpreted as an azimuth in degrees
- filter_by_category [false] - logical flag to recalculate selected mean values based on filtering by particular categories within a categorical grid (provided to ProcessRiverBasins as 'add_cat_grids'). Requires entries to 'filter_type', 'cat_grid', and 'cat_values'. Will produce filtered values for channel steepness, gradient, and mean elevation by default along with any additonal grids present (i.e. grids provided with 'add_grids' to ProcessRiverBasins).
- filter_type ['exclude'] - behavior of filter, if 'filter_by_categories' is set to true. Valid inputs are 'exclude', 'include', or 'mode'. If set to 'exclude', the filtered means will be calculated excluding any portions of grids have the values of 'cat_values' in the 'cat_grid'. If set to 'include', filtered means will only be calculated for portions of grids that are within specified categories. If set to 'mode', filtered means will be calculated based on the modal value of the categorical grid by basin, e.g. if the mode of basin 1 is 'grMz' and the mode of basin 2 is 'T', then the filtered mean will be calculated based on nodes that are 'grMz' in basin 1 and are 'T' in basin 2. The idea behind this filter is if you wish to find characteristic stats for particular categories. If filter type is 'mode' then an entry for 'cat_values' is not required.
- cat_grid [] - name of categorical grid to use as filter, must be the same as the name provided to ProcessRiverBasins (i.e. third column in the cell array provided to 'add_cat_grids').
- cat_values [] - name of text file containing single row of comma separated categorical values of interest to use in filter. These must match valid categories in the field of interest in the shapefile provided to PrepareAddCatGrids to prepare the grid name in the 'cat_grid' function
- populate_categories [false] - logical flag to add entries that indicate the percentage of a watershed occupied by each category from a categorical grid, e.g. if you provided an entry for 'add_cat_grids' to ProcessRiverBasins that was a geologic map that had three units, 'Q', 'Mz', and 'Pz' and you set 'populate_categories' to true there will be field names in the resulting shapefile named 'Q', 'Mz', and 'Pz' and the values stored in those columns will correspond to the percentage of each basin covered by each unit for each basin. Setting populate_categories to true will not have any effect if no entry was provided to 'add_cat_grids' when running ProcessRiverBasins.
- means_by_category [] - method to calculate means of various continuous values within by categories. Requires that a categorical grid(s) was input to ProcessRiverBasins. Expects a text file containing a single row with comma separated entries, where the first entry is the name of the category to use (i.e. name for categorical grid you provided to PrepareAddCatGrids) following entries are names of grids you wish to use to find means by categories, e.g. an example single row in an input table would be 'geology,ksn,rlf2500,gradient' (without the quotes) if you were interested in looking for patterns in channel steepness, 2.5 km^2 relief, and gradient as a function of rock type/age. Valid inputs for the grid names are:
  - ksn - uses channel steepness map structure with user provided reference concavity
  - gradient - uses gradient grid
  - rlf##### - where #### is the radius you provided to ProcessRiverBasins (requires that 'calc_relief' was set to true when running ProcessRiverBasins
  - NAME - where NAME is the name of an additional grid provided with the 'add_grids' option to ProcessRiverBasins

Output:

- Outputs a table as a text with the following default fields:
  - river_mouth - river mouth number provided to ProcessRiverBasins
  - drainage_area - drainage area of basin in km^2
  - out_x - x coordinate of basin mouth
  - out_y - y coordinate of basin mouth
  - center_x - x coordinate of basin in projected coordinates
  - center_y - y coordinate of basin in projected coordinates
  - outlet_elevation - elevation of pour point in m
  - mean_el - mean elevation of basin in meters
  - max_el - maximum elevation of basin in meters
  - mean_ksn - mean channel steepenss
  - mean_gradient - mean gradient
- Either standard errors, standard deviations or both will be populated for elevation, ksn, and gradient depending on value of 'uncertainty'
- Mean and standard error / standard deviation / both values will be populated for any additional grids

Also saves a matfile for use in 'cmpBasinStatsPlots' or 'cmpMakeCombinedSwath'

Notes If you use 'filter_by_category' to create filtered means and uncertainites, note that the filtered value for channel steepness is calcuated using the interpolated 'KsnOBJc', not the stream values like the the value reported in mean_ksn in the output table.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
CompileBasinStats /path/to/wdir basin_dir
CompileBasinStats /path/to/wdir basin_dir location_of_subbasins subbasinsv1
    include subdivided
```

A.8 ConditionDEM

Description: Wrapper around the variety of methods provided by TopoToolbox for smoothing a stream profile. With the exception of 'quantc_grid' and 'mingrad' these methods will only modify elevations along the stream network provided to the code. See the relevant parent functions for a more in depth description of the behavior of these individual methods. Produces one figure that compares the long profile of the longest stream within the dataset using the uncondtioned and conditioned DEM to provide a quick method of evaluating the result. These methods vary in their complexity and processing times so it is recommended you understand your choice. Using the 'mincost' method is a good starting place before exploring some of the more complicated methods. Outputs the resulting DEM as an ascii file for use in other 'cmp*' codes our outside GIS applications.

Required Inputs:

- wdir - full path of working directory
- MatFile - Name of matfile output from 'cmpMakeStreams'
- method - method of conditioning, valid inputs are as follows:
  - 'mincost' - uses the 'mincosthydrocon' function, valid optional inputs are 'mc_method' and 'fillp'.
  - 'mingrad' - uses the 'imposemin' function, valid optional inputs are 'ming'. Note that providing a large minimum gradient to this code can carve the stream well below the topography.
  - 'quantc' - uses the 'quantcarve' function (for STREAMobjs), valid optional inputs are 'tau','ming', and 'split'. Requires the Optimization Toolbox and if 'split' is set to true, requires Parallel Processing Toolbox.
  - 'quantc_grid' - uses the 'quantcarve' function for (GRIDobjs), valid optional inputs are 'tau'. Requires the Optimization Toolbox. This is a computationally expensive calculation and because it operates it on the whole grid, it can take a long time and/or fail on large grids. The 'quantc' method which only operates on the stream network is significantly fasters and less prone to failure.
  - 'smooth' - uses the 'smooth' function, valid optional inputs are 'sm_method','split','stiffness','stiff_tribs', and 'positive' depending on inputs to optional parameters may require Optimization Toolbox ('sm_method'='regularization' and 'positive'=true) and Parallel Processing Toolbox ('split'=true).
  - 'crs' - uses the 'crs' function, valid optional inputs are 'stiffness', 'tau', 'ming', 'stiff_tribs', 'knicks', and 'split'. Requires Optimization Toolbox.
  - 'crslin' - uses the 'crslin' function, valid optional inputs are 'stiffness', 'stiff_tribs', 'ming', 'imposemin', 'attachtomin', 'attachheads', 'discardflats','precisecoords'

Optional Inputs:

- file_name ['cond_DEM'] - name for output ascii file containing conditioned DEM
- new_stream_net [] - option to provide name of a matfile containing a new stream network as as output from another function (e.g. cmpFindThreshold) to use instead of the stream network saved in the MatFile provided to the function. This new stream network must have been generated from the DEM stored in the provided MatFile
- mc_method [interp] - method for 'mincost', valid inputs are 'minmax' or 'interp'
- fillp [0.1] - scalar value between 0 and 1 controlling the ratio of carving to filling for 'mincost'
- ming [0] - minimum gradient [m/m] in downslope direction, used in 'mingrad','quantc','crs','crslin'
- tau [0.5] - quantile for carving, used in 'quantc', 'quantc_grid', 'crs'.
- split [true] - logical flag to utilized parallel processing to independently process tributaries, used in 'quantc_grid', 'smooth', and 'crs'
- sm_method ['regularization'] - method for 'smooth', valid inputs are 'regularization' and 'movmean'.
- stiffness [10] - scalar positive value for stiffness penalty, used in 'smooth', 'crs', and 'crslin'
- stiff_tribs [true] - logical flag to relax the stiffness penalty at tributary junctions, used in 'smooth', 'crs', and 'crslin'

- knicks [] - nx2 matrix of x and y locations of knickpoints where stiffness penalty should be relaxed, used in 'crs' and 'crslin'
- imposemin [false] -logical flag to preprocess DEM with imposemin during crslin
- attachtomin [false] - logical flag to prevent elevations from going below profile minima, used in crslin
- attachheads [false] - logical flag to fix the channel head elevations, used in crslin
- discardflats [false] - logical flag to discard flat portions of profiles, used in crslin
- maxcurvature [] - maximum convex curvature at any vertices along profile, used in crslin
- precisecoords [] - nx3 matrix with x, y, and z coordinates of points that the smoothed profile must past through, used in crslin

Outputs:

- georeferenced ascii file of the processed dem

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
ConditionDEM /path/to/wdir Topo.mat mincost
ConditionDEM /path/to/wdir Topo.mat mincost fillp 0.5
```

A.9 DippingBedFinder

Description: Function to determine the expected location of a planar dipping bed within a landscape based on an input coordinate

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- xy - 1 x 2 vector with the x and y coordinate (i.e. easting and northing) of the location of interest, if you provide an empty vector you will be given the opportunity to pick a location on the DEM
- hght_abv_base - height of the outcrop of interest above the base of the bed of interest (i.e. positin of the outcrop in the section)
- thickness - thickness of the bed (hght_abv_base must be smaller than total thickness)
- strike - strike of bed, report with right hand rule
- dip - dip of bed

Output: Code will produce a figure showing expected location of bed and will an ascii text file with expected location of the bed (1 where the bed should appear, 0 where it should not)

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
DippingBedFinder /path/to/wdir Topo.mat [25600 234500] 250 500 100 10
DippingBedFinder /path/to/wdir Topo.mat [] 250 500 100 10
```

A.10 EroGrid

Description: Function to produce an erosion rate map based on an empirical relationship between normalized channel steepness (ksn) and erosion rate (E) defined in the form of ksn = C*E^{phi}. This relationship would likely come from a series of catchment averaged cosmogenic erosion rates and basin averaged normalized channel steepness. Function can produce an erosion rate map for a unique relationship between ksn and E or a series of relationships based on different regions defined in a separate grid provided as the optional 'VAL' input. Function is also able to incorporate uncertainty in the interpolated KSN map and/or uncertainty on the fit parameters (i.e. C and phi) to calculate min and max erosion rate estimates based on these uncertainties.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- KsnFile - ascii grid of continous ksn (e.g. as produced by KsnChiBatch with product set to 'ksngrid') or a shapefile of ksn (e.g. as produced by KsnChiBatch with product set to 'ksn'). Input to KsnFile must have either a '.txt' or '.shp' file extension
- C - coefficient of power law relationship between ksn and E (ksn = C*E^{phi}), can be a single value or an array of values. If an array of values is provided, it is assumed that these are multiple coefficients that refer to different relationships based on spatial subsetting that will be defined with inputs to the optional 'edges' and 'VAL' inputs
- phi - exponent of power law relationship between ksn and E (ksn = C*E^{phi}), can be a single value or an array of values. If an array of values is provided, it is assumed that these are multiple exponents that refer to different relationships based on spatial subsetting that will be defined with inputs to the optional 'edges' and 'VAL' inputs

Optional Inputs:

- radius [5000] - radius for creating a spatially averaged ksn grid if the entry to KSN is a mapstruct
- KSNstd - ascii grid of standard deviation of continous ksn (e.g. as produced by KsnChiBatch with product set to 'ksngrid') to incoporate uncertainty in ksn smoothing into the production of erosion rate maps. If you are providing a mapstructure to the 'KSN' argument and you wish to use the KSNstd that will be calculated to consider how uncertainty in the ksn value leads to uncertainty in the erosion rate map, use the 'use_ksnstd' logical flag
- use_ksnstd [false] - logical flag to calculate and use the standard deviation of ksn in estimating erosion rates, in the event that you are providing a shapefile of Ksn values to the 'KsnFile' input.
- C_std - standard deviation / uncertainty on coefficient of power law. If an entry is provided to C_std (and phi_std), then these uncertainties will be incorporated into the erosion grid outputs. There must be the same number of entires to C_std as there is to 'C'.

- ph_std - standard deviation / uncertainty on the exponent of power law. If an entry is provided to phi_std (and C_std), then these uncertainties will be incorporated into the erosion grid outputs. There must be the same number of entires to phi_std as there is to 'phi'.
- VAL [] - ascii grid (.txt) defining the regions by which to index the KSN grid to establish different empirical relationships between ksn and E. For example, VAL might be a precipitation grid if you have reason to believe there are different ksn - E relationships depending on precipitation. If you provide an input to VAL, you must also provide an input to 'edges' that define the bounds on the differnet values within VAL that define the different regions, i.e. VAL and edges must be in the same units and edges must cover the full range of values within VAL
- edges [] - array that define the edges to index the GRIDobj provided to VAL. There must be n+1 entries for n entries to 'C' (and 'phi'). It's also assumed that entries for 'C' and 'phi' are in the same order as the bins defined by 'edges'.
- resample_method ['nearest'] - method to resample the provided VAL GRIDobj if it is a different dimension or pixel size than the input DEM. Valid options are 'nearest', 'bilinear', and 'bicubic'. Default is 'nearest'.

Note: The function does not explicitly depend on the 'KSN' arugment actually being KSN. E.g., if you have established a relationship between erosion rate and local relief in the form rlf = $C*E^{phi}$, then this function will work equally well.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
EroGrid /path/to/wdir Topo.mat batch_ksngrid.txt 100 0.5
EroGrid /path/to/wdir Topo.mat batch_ksngrid.txt 100 0.5 KSNstd
    batch_ksngridstd.txt
EroGrid /path/to/wdir Topo.mat batch_ksngrid.txt [100 316 1000] [0.5 0.5 0.5]
    VAL precip.txt edges [0 1 2 4]
```

A.11 FindBasinKnicks

Description: Function for manually selecting knickpoints within a Basin_Data_File (i.e. result of ProcessRiverBasins). Choose knickpoints on Chi-Elevation plot with mouse clicks and press return when you have selected all the knickpoints for a given stream segment. As you progress through, knickpoints you have already picked (i.e. on shared portions of river profiles) will be displayed as red dots. If you're interested in trying out an automated method of finding knickpoints, try 'knickpointfinder' included with TopoToolbox. If you choose to classify knickpoints ('classify_knicks' = true) The code expects you to input a number or character to categorize the knickpoint higlighted in red. You must be consistent in your choice (i.e. you must either use numbers for all of the classifications or characters for all the classifications within a given run), mixing numbers and characters will result in an error at the end of the run. For entering characters, it's recommended you keep these short strings without spaces (i.e. entries supported into a shapefile a attribute table), e.g. knick or bound

Required Inputs:

- wdir - full path of working directory
- basin_dir - name of the folder containing the basin files
- Basin_Data_File - name of a basin file result from the ProcessRiverBasins script
- plot_result - logical flag to either plot the results (true) or not (false)

Optional Inputs:

- classify_knicks [false] - logical flag to provide a classification for each chosen knickpoint
- ref_concavity [0.5] - reference concavity for chi calculation
- shape_name [] - character string to name output shapefile (without .shp), if no input is provided then no shapefile is output

Outputs: Saves a matfile containing the KnickTable - table with one row for each selected knickpoints. If classify_knicks is false, will have with columns x_coord, y_coord, elevation, distance, and chi. If classify_knicks is true, will have a sixth column containing the classification of the knickpoints. Also saves the KnickTable as a text file Will output a shapefile as well if an argument is provided for the 'shape_name' parameter

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
FindBasinKnicks /path/to/wdir Basins Basin_56_Data.mat true
FindBasinKnicks /path/to/wdir Basins Basin_56_Data.mat true classify_knicks
    true
```

A.12 FindThreshold

Description: Function to interactively select an appopriate threshold area for a given stream network. Function will either have you iterate through a number of single streams, controlled by the number passed to 'num_streams', extracted from the drainage divide or all streams within the provided drainage network if you provide 'all' to 'num_streams'. If 'num_streams' is numeric, then the function will use the average of the user selected minimum threshold areas to define a new stream network. If 'num_streams' is set to 'all', the function will use the user selected minimum threshold areas to define a new stream network for each individual stream (i.e. the minimum threshold area will be different for each stream base on your selections). You can use either chi-elevation or slope-area plots (the default), both plots will be displayed regardless of choice, to visually select where channels begin. Function also outputs the lists of selected threshold areas and distance from channel head to divide.

Required Inputs:

- wdir - full path of working directory
- MatFile - Name of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- num_streams - Number of stream profiles to view and select threshold areas, if you wish to manually select threshold areas for all streams in the provided network, provide 'all' instead of a number

Optional Inputs:

- ref_concavity [0.50] - refrence concavity used to generate the chi-elevation plot
- pick_method ['slope_area']- Type of plot you wish to choose the threshold area on, valid options are:
  - 'chi' - Choose threshold areas on a chi elevation plot
  - 'slope_area' - Choose threshold areas on slope-area plot

Outputs:

- thres_table.txt - text file containing a list of the threshold areas and xds for each stream
- thresh_streams.shp - shapfile of new stream network
- thresh_streams.mat - mat file containing new stream network for use with other cmp* codes.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
FindThreshold /path/to/wdir Topo.mat 25
FindThreshold /path/to/wdir Topo.mat all
FindThreshold /path/to/wdir Topo.mat 25 pick_method chi
```

A.13 HackRelationship

Description: Function to find the Hack relationships for all watersheds in a given landscape. Assumes that the provided STREAMobj terminates at the outlets of the watersheds for which you wish to calculate Hack coeffecients and exponents.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'

Optional Inputs:

- method ['trunks'] - optional parameter for controlling which data is used to fit the Hack relationship. Options are 'trunks' (default), 'streams', or 'grids'. If 'trunks', only values in trunk streams will be used. If 'streams' only values in stream networks (as defined by the input STREAMobj) will be used. If 'grids', all pixels in watersheds upstream of outlets of the provided STREAMobj will be used.
- relation ['original'] - optional parameter to control the form of the Hack relationship. The 'original' option will fit the original relationship as presented by Hack, e.g. length = C * drainage area^h. The 'inverse' will instead fit the form used by Whipple and Tucker, 1999, drainage area = C * length^h.
- include_hillslope [false] - logical flag to either include or not include (default) the portions of the channels between the channelheads in the provided STREAMobj and the drainage divide. Regardless of the value of this parameter, distances within the channel will be relative to the divide unless changed with 'measure_from' parameter, but if left false, low drainage area values physically above channelhead will not be included in fits. This parameter is ignored if 'method' is set to 'grids'.
- measure_from ['divide'] - optional parameter to control how distances ares measured, either measured from the drainage divides when set to 'divide' or from the channelheads (as defined in the provided STREAMobj) if set to 'channelheads'. If 'method' is set to 'grids', then this parameter will be set to 'divide' regardless of user input.
- draw_fig [true] - logical flag to display a plot of the coefficient (C) and exponent (h) as a function of drainage area for all basins
- save_fig [false] - logical flag to save the output figure

Outputs:

- 'HackTable.mat' with columns for x coordinate and y coordinate of each basin, drainage area for each basin, and the best fit 'C' and 'h' values for each basin
- 'GlobalHack.mat' with columns for the C and h values fit from all relevant points in the landscape

A.14 InspectJunction

Description: Function for visualizing the fit of up- and downstream links to evaluate the values for the junction angles. It will display both a zoomed view of the junction, displayed in polar coordinates, and the location of the junction on a map of the stream network.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins', in this case it must be the same matfile you provided to JunctionAngle
- JunctionMatFile - Full path of matfile output from JunctionAngle
- num - junction number, referenced from JunctionAngle table. If an empty array is provided, you will be prompted with a map of the stream network to select the junction of interest.

Optional Inputs:

- fit_distance - distance in map units along stream to do fit of selected junction. Distance must be less than the maximum distance used when JunctionAngle was run. This input (and the related 'num_nodes') is provided if you want to visually explore the sensitivity of the fit distance to junction angles
- num_nodes - number of stream nodes to do fit of selected junction. Number of nodes must be less than the number of nodes extracted when JunctionAngle was run.
- save_fig [false] - logical flag to toggle saving the figure output of this function as a PDF.

Note: You cannot provide entries to both fit distance and num_nodes

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
InspectJunction /path/to/wdir Topo.mat Topo_junctions.mat 50
InspectJunction /path/to/wdir Topo.mat Topo_junctions.mat 50 fit_distance 500
InspectJunction /path/to/wdir Topo.mat Topo_junctions.mat 50 num_nodes 10
```

A.15 JunctionAngle

Description: Function calculates the angle of stream junctions. The basic strategy and nomenclature used in this function is adapted from Howard, 1971, 'Optimal angles of stream junction: Geometric, stability to capture, and minimum power criteria'. The code first fits the upstream and downstream links that meet at junctions with linear fits. The lengths of stream used to perform these fits is controlled with the 'fit_distance' parameter. Junction angles are calculated as a pair of angles (e1 and e2) between each tributary (tributary 1 and tributary 2) and the upstream projection of the downstream link. The total junction angle is calculated as the angle between the two upstream links, which in well behaved junctions should be the sum of e1 and e2, but is not always the case (i.e. for some junctions both upstream links are on the same side of the upstream projection of the downstream link, in these cases the junction angle will not be the sum of the e1 and e2 angles, but instead will be the true angle between link 1 and link 2, whereas the values of e1 ane e2 will be measured from the upstream plane).

The function also calculates predicted junction angles based on the simple geometric criteria described in Howard, 1971, and the handedness of junction angle as defined by James and Krumbein, 1969, 'Frequency distributions of stream link lengths'. It is important to note that angles related to junctions at which more than two upstream links meet or for which there are no downstream nodes (i.e. the junction is an outlet) are set to NaN as these are undefined in the Howard criteria.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- fit_distance - distance in stream distance (map units) for fitting stream links, if more than one value is provided to fit_distance, it is assumed you want to calculate junction angles fitting stream segments with the range of distances. The provided value(s) can be interpeted as the number of stream nodes over which to fit the stream links if the optional 'use_n_nodes' is set to true

Optional Inputs:

- file_name_prefix ['topo'] - prefix for outputs shapefiles, textfiles, and mapfiles
- use_n_nodes [false] - logical flag to interpret the value(s) provided to fit_distance as the number of stream nodes to extract up and downstream as opposed to streamwise distance
- previous_IX [] - Calculation of the 'IX' output is the most time consuming aspect of the calculation, so if you wish to rerun the function with different parameters (e.g. recalculating predicted angles with a different reference concavity or different method) the presupplying the IX result from a previous run can be useful. In the compiled version this accomplished by supplying the name of a matfile generated during a previous run of the compiled version of junction angle. It is important that this IX result was produced in a run using the same MatFile inputs and that the maximum value provided to fit distance does not exceed the maximum fit distance that was provided when you initially ran the function to produce the provided IX input. The code will check that the fit distances are compatible, but will not explicity check that S, A, and the DEM are the same.
- predict_angle_method ['area'] - method to use to calculate predicted junction angles. Both methods are simple geometric predictions from Howard, 1971. Valid inputs are 'slope', 'area', or 'both'. If pred_angle_method is

set to 'area' (default), then you should make sure the ref_concavity is valid for the network you are analyzing, default value is 0.5. For the 'slope' method, the slope of the two upstream and one downstream link will be the mean of the gradient of those links over the length of stream sampled (controlled by the values provided to fit_distance). Providing 'both' will calculate both the slope and area predicted angles.

- ref_concavity - [0.5] - reference concavity for calculating predicted junction angles using Howard, 1971 area relation. If no value is provided, default value of 0.5 will be used.
- verbose [false] - logical flag to report progress through function. This can be useful because certain steps of the process are very time consuming.

Outputs: Outputs a table as a text file. If one value is provided to fit_distance then there will be one table. If multiple values are provided to fit_distance, then there will be a table for each fit distance, in order of increasing distances. Columns of the table are:

- junction_number - ID number of junction
- junction_x - x coordinate of junction
- junction_y - y coordinate of junction
- junction_angle - angle in degrees between tributary 1 (e1) and tributary 2 (e2)
- handedness - handedness of junction as defined by James and Krumbein, 1969, options are Right, Left, or Undefined, stored as a categorical
- split - logical indicating whether upstream projection of downstream link lies between tributary 1 and tributary 2 (true) or does not (false)
- e1_obs_angle - angle between tributary 1 and upstream projection of downstream link
- e1_Apred_angle - predicted angle between tributary 1 and upstream projection of downstream link based on Howard 1971 area relations (will appear if 'predict_angle_method' is 'area' or 'both')
- e1_Spred_angle - predicted angle between tributary 1 and upstream projection of downstream link based on Howard 1971 slope relations (will appear if 'predict_angle_method' is 'slope' or 'both')
- e1_rotation - orientation of tributary 1 with respect to the upstream projection of the downstream link, options are CCW (counter-clockwise), CW (clockwise), or Undefined (more than two tributaries meet at junction). If split is true, one of the upstream links will be CW and one will be CCW.
- e1_shreve - shreve order of tributary 1
- e1_direction - flow azimuth of tributary 1
- e1_distance - stream distance used to fit tributary 1
- e1_num - number of upstream nodes used to fit tributary 1
- e1_R2 - R2 value on linear fit of tributary 1
- e2_obs_angle - angle between tributary 2 and upstream projection of downstream link

- e2_Apred_angle - predicted angle between tributary 2 and upstream projection of downstream link based on Howard 1971 area relations (will appear if 'predict_angle_method' is 'area' or 'both')
- e2_Spred_angle - predicted angle between tributary 2 and upstream projection of downstream link based on Howard 1971 slope relations (will appear if 'predict_angle_method' is 'slope' or 'both')
- e2_rotation - orientation of tributary 2 with respect to the upstream projection of the downstream link, options are CCW (counter-clockwise), CW (clockwise), or Undefined (more than two tributaries meet at junction)
- e2_shreve - shreve order of tributary 2
- e2_direction - flow azimuth of tributary 2
- e2_distance - stream distance used to fit tributary 2
- e2_num - number of upstream nodes used to fit tributary 2
- e2_R2 - R2 value on linear fit of tributary 2
- e3_direction - flow azimuth of downstream link
- e3_distance - stream distance used to fit downstream link
- e3_num - number of downstream nodes used to fit downstream link
- e3_R2 - R2 value of linear fit of downstream link

If multiple fit_distances are provided a separate '*_mean_junctions.txt' file will be saved, which will be a table containing mean and standard deviations across the angles calculated using the different fit distances, including the mean total junction angle, standard deviation of the total junction angle, mean angle between tributary 1 and the upstream projection of the downstream link (e1), standard deviation of e1, mean angle between tributary 2 and the upstream projection of the downstream link (e2), standard deviation of e2. If the predict_angle_method is set to either 'slope' or 'both', this will also include the mean and standard deviations of the predicted e1 and e2 angles based on the slope method. Means and standard deviations are not calculated for the area method because these will not vary as a function of fit distance.

Note: For all outputs, the tributaries are sorted so that tributary 1 is always the 'larger' of the two tributaries. The size of the tributaries are based on the shreve order of the two tributaries. In the event that the two tributaries have the same shreve order then the drainage area is used to determine respective size.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
JunctionAngle /path/to/wdir Topo.mat 1000
JunctionAngle /path/to/wdir Topo.mat [1000 2000 5000]
JunctionAngle /path/to/wdir Topo.mat 10 use_n_nodes true
```

A.16 JunctionLinks

Description: Function for identifying the links within a stream network. Results will be similar (but not identical) to networksegments function. Classifies links according to James and Krumbein, 1969, 'Frequency distributions of stream link lengths', classifying links as either interior or exterior (exterior links have channel heads as their upstream ends) and either cis or trans (cis links have the same handedness on both ends of the link, trans links have the oppostite handedness on the upstream and downstream links). Exterior links, interior links that connect to an outlet, links for which one or more of the junctions at the end have more than two upstream links, and interior links for which there is not clear handedness (i.e. the incoming streams have the same Shreve order) will be undefined in terms of a cis or trans classification.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins', in this case it must be the same matfile you provided to JunctionAngle
- JunctionMatFile - Full path of matfile output from JunctionAngle

Optional Inputs:

- file_name_prefix ['topo'] - prefix for outputs shapefiles, textfiles, and mapfiles
- make_shape [false] - logical flag to generate a shapefile containing the stream network broken into segments defined by junctions and categorized in terms of link_type and link_side (see outputs). Production of the mapstructure used to create shapefile can be time consuming

Outputs: '*_junction_links.txt' - table containing the information for all the stream links:

- link_number - ID number of link
- downstream_x - x coordinate of downstream end of link
- downstream_y - y coordinate of downstream end of link
- downstream_IX - index into GRIDobj of downstream end of link
- upstream_x - x coordinate of upstream end of link
- upstream_y - y coordinate of upstream end of link
- upstream_IX - index into GRIDobj of upstream end of link
- link_type - classification of link is either Exterior (upstream end of link is channelhead) or Interior
- downstream_junc_num - junction number of downstream end of link referencing the junction table provided to J, a NaN indicates that the downstream end of the link is not a junction
- upstream_junc_num - junction number of upstream end of link referencing the junction table provided to J, a NaN indicates that the upstream end of the link is not a junction
- link_side - classifcation of link as either Cis (both ends of link are same handedness), Trans (ends of link are opposite handedness), or Undefined if one or both of the ends of the link do not have a handedness classification

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
JunctionLinks /path/to/wdir Topo.mat Topo_junctions.mat
JunctionLinks /path/to/wdir Topo.mat Topo_junctions.mat make_shape true
```

A.17 KsnChiBatch

Description: Function to produce channel steepness, chi maps or chi grids for all channels within a DEM

Reqiured Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- product - switch to determine which products to produce:
  - 'ksn' - ksn map as a shapefile
  - 'ksngrid' - ascii file with ksn interpolated at all points in a grid
  - 'chimap' - ascii file with chi calculated in channel networks
  - 'chigrid' - ascii file with chi calculate at all points in a grid
  - 'chi' - results for both chimap and chigrid
  - 'all' - ksn, ksngrid, chimap, and chigrids

Optional Inputs:

- conditioned_DEM [] - option to provide a hydrologically conditioned DEM for use in this function, expects the mat file as saved by 'cmpConditionDEM' See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- new_stream_net [] - option to provide full path of a matfile containing a new stream network as as output from another function (e.g. cmpFindThreshold) to use instead of the stream network saved in the MatFile provided to the function. This new stream network must have been generated from the DEM stored in the provided MatFile
- file_name_prefix ['batch'] - prefix for outputs, will append the type of output, i.e. 'ksn', 'chimap', etc
- smooth_distance [1000] - distance in map units over which to smooth ksn measures when converting to shapefile
- ref_concavity [0.50] - reference concavity (as a positive value) for calculating ksn
- ksn_method [quick] - switch between method to calculate ksn values, options are 'quick', 'trunk', or 'trib', the 'trib' method takes 3-4 times longer than the 'quick' method. In most cases, the 'quick' method works well, but if values near tributary junctions are important, then 'trib' may be better as this calculates ksn values for individual channel segments individually. The 'trunk' option calculates steepness values of large streams independently (streams considered as trunks are controlled by the stream order value supplied to 'min_order'). The 'trunk' option may be of use if you notice anomaoloulsy high channel steepness values on main trunk streams that can result because of the way values are reach averaged.

- min_order [4] - minimum stream order for a stream to be considered a trunk stream, only used if 'ksn_method' is set to 'trunk'
- output_level_method [] - parameter to control how stream network base level is adjusted. Options for control of output elevation are:
  - 'elevation' - extract streams only above a given elevation (provided by the user using the 'min_elevation' parameter) to ensure that base level elevation for all streams is uniform. If the provided elevation is too low (i.e. some outlets of the unaltered stream network are above this elevation) then a warning will be displayed, but the code will still run.
  - 'max_out_elevation' - uses the maximum elevation of all stream outlets to extract streams only above this elevation, only valid for options that operate on streamlines only (i.e. will not work with 'ksngrid' or 'chigrid').
- min_elevation [] - parameter to set minimum elevation for base level, required if 'base_level_method' is set to 'elevation'
- complete_networks_only [true] - if true (default) the code will only populate portions of the stream network that are complete. Generally, this option should probably be left as true (i.e. chi will not be accurate if drainage area is not accurate), but this can be overly agressive on certain DEMs and when used in tandem with 'min_elevation', it can be slow to calculate as it requires recalculation of the FLOWobj.
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)

Notes: Please be aware that the production of the chigrid can be time consuming, so be patient...

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
KsnChiBatch /path/to/wdir Topo.mat ksn
KsnChiBatch /path/to/wdir Topo.mat ksn conditioned_DEM DEMcond.mat
    smooth_distance 500
```

A.18 KsnProfiler

Description: Function to interactively select channel heads and define segements over which to calculate channel steepness values. This function is designed to be similar to the operation of Profiler_51, with some improvements. Function will display map with the stream network and expects the user to select a location near a channel head of interest. The user will be then prompted to confirm that the defined stream is the desired choice. Finally, displays of the chi-z and longitudinal profile of the selected river will appear and the user is expected to define (with mouse clicks) any obvious segments with different channel steepness (or concavity) on either the chi-z plot or the stream profile (see 'pick_method' option). When done selecting press enter/return. The user will be prompted whether they wish to continue picking streams or if they are done. When done picking streams, the function will output three different products (see below) and produce a shapefile of the selected streams with ksn, concavity, area, and gradient.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'

Optional Inputs:

Restart Picking

- restart [] - providing an entry to this parameter allows the user to restart a run, either a run that you succesfully completed but want to restart or a run that failed part way through either because of an error or because you aborted out. While the code is running, it will save data necessary to restart in a mat file called '*_restart.mat'. If the code succesfully completes, this '*_restart.mat' file will be deleted. DO NOT DELETE THIS FILE WHILE THE CODE IS RUNNING OR IF THE CODE FAILS AND YOU WISH TO SALVAGE THE RUN. You can also call use restart if you just wish to restart picking streams from a previously completed run. If you run the code with an 'input_method' other than 'interactive' and the code succesfully completes (i.e you fit all the streams selected via the input method you choose and you did not stop the code early) then running with restart will not do anything. If you wish to restart, you do not need to define any of the original parameters, these are saved in the output files and will be loaded in, you only need to provide the four required inputs (see example) along with the restart parameter. Valid inputs to restart are:
  - 'continue' - will restart the run. If used with a completed or failed 'interactive' run will repopulate the map with already picked streams and you can continue picking. If using with a non interactive input method that either failed or you aborted, will start on the next stream in the sequence.
  - 'skip' - only a meaningful input for a non interactive run. This will skip the next stream segment in the sequence. This would be useful if a particular stream segment causes the code to error, this way you can skip that stream in a restart without having to modifying the stream network.

Main Options

- input_method ['interactive'] - parameter which controls how streams of interest are supplied:
  - 'interactive' - user picks streams of interest by selecting channelheads on a map, this option will also iteratively build a channel steepness map as the user picks more streams.
  - 'all_streams' - will use the supplied STREAMobj and iterate through all channel heads. There is an internal parameter to avoid selecting streams that are too short to properly fit (mostly relevant if 'junction method' is set to 'check'). The default value is 4 * the DEM cellisze, the user can change this value by providing an input for the optional parameter 'min_channel_length', input should be in map units and greater than the default. You can use a code like 'SegmentPicker' to select portions of a STREAMobj
  - 'stream_length' - will use supplied STREAMobj and entry to 'min_length_to_extract' to iterate through all streams that are longer than the length provided to 'min_length_to_extract'. There is an internal parameter to avoid selecting streams that are too short to fit (mostly relevant if 'junction method' is set to 'check'). The default value is 4 * the DEM cellsize, the user can change this value by providing an input for the optional parameter 'min_channel_length', input should be in map units and greater than the default.

  - 'channel_heads' - will use a supplied list of coordinates of channel heads to select and iterate through streams of interest. If this option is used, the user must provide an input for the optional 'channel_head_list' parameter.

- pick_method ['chi'] - choice of how you want to pick stream segments. The diagram within which to pick based on your selection will be outline in red. Valid inputs are:
  - 'chi' - select segments on a chi - z plot (recommended and default)
  - 'stream' - select segments on a longitudinal profile
  - 'slope_area' - select segments on a slope area plot

- junction_method ['check'] - choice of how to deal with stream junctions:
  - 'check' - after each choice, will check whether downstream portions of the selected stream have already been fit, and if it has, the already fit portion of the stream will not be displayed or refit
  - 'ignore' - each stream will be displayed from its head to mouth independent of whether portions of the same stream network have been fit

- concavity_method ['ref']- options for concavity:
  - 'ref' - uses a reference concavity, the user can specify this value with the reference concavity option (see below)
  - 'auto' - function finds a best-fit concavity for each selected stream, if used in conjunction with 'junction_method','check' this means that short sections of streams picked will auto fit concavity that may differ from downstream portions of the same streams

Input Method Options

- min_channel_length [] - minimum channel length for consideration when using the 'all_streams' method of input, provide in map units.

- channel_head_list [] - m x 2 array of x and y coordinates of channel heads OR the name / location of a point shapefile of channel heads, one of these is required when using 'channel_heads' method of input, must be in the same coordinate system as the input DEM etc. The code will attempt to find the nearest channel head to the coordinates you provided, so the closer the provided user coordinates are to channel heads, the more accurate this selection method will be.

- min_length_to_extract [] - minimum stream length (in map units) to extract streams if 'input_method' is set to 'stream_length'.

Redefine Threshold Area Options

- redefine_threshold [false] - logical flag to initiate an extra step for each stream where you manually define the hillslope-fluvial transition (this will result in overriding the threshold area you used to generate the supplied STREAMobj, and it will also produce a STREAMobj with a variable threshold area for channel definition). See additional optional input 'rd_pick_method'.

- rd_pick_method ['slope_area'] - plot to use to choose new threshold area if 'redefine_threshold' is set to true. Valid inputs are 'slopearea' and 'chi'.

Stream Network Modification Options

- complete_networks_only [false] - if true, the code will filter out portions of the stream network that are incomplete prior to choosing streams
- min_elev [] - minimum elevation below which the code stops extracting channel information (no action if left empty)
- max_area [] - maximum drainage area above which the code stops extracting channel information (in square map units, no action if left empty)

Hydrological Conditioning Options

- conditioned_DEM [] - option to provide a hydrologically conditioned DEM for use in this function (do not provide a conditoned DEM for the main required DEM input!) which will be used for extracting elevations. See 'ConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM). Values closer to 0 tend to 'carve' more, whereas values closer to 1 tend to fill. See info for 'mincosthydrocon'

Replace Stream Network

- new_stream_net [] - option to provide full path of a matfile containing a new stream network as as output from another function (e.g. cmpFindThreshold) to use instead of the stream network saved in the MatFile provided to the function. This new stream network must have been generated from the DEM stored in the provided MatFile

Display Options

- display_slope_area [false] - logical flag to display slope area plots. Some people love slope area plots (like one of the authors of the supporting paper), some people hate slope area plots (like the other author of the supporting paper), so you can either not draw them at all (false - default) or include them (true). This will automatically be set to true if you select 'slope_area' as the 'pick_method'.
- plot_type ['vector'] - expects either 'vector' or 'grid', default is 'vector'. Controls whether all streams are drawn as individual lines ('vector') or if the stream network is plotted as a grid and downsampled ('grid'). The 'grid' option is much faster on large datasets, but can result in inaccurate channel head selection. The 'vector' option is easier to see, but can be very slow to load and interact with on large datasets.

Constants

- ref_concavity [0.50] - refrence concavity used if 'theta_method' is set to 'ref'
- smooth_distance [1000] - distance in map units over which to smooth ksn measures when converting to shapefile
- max_ksn [250] - maximum ksn used for the color scale, will not effect actual results, for display purposes only
- threshold_area [1e6] - used to redraw downsampled stream network if 'plot_type' is set to 'grid'

Output Options

- stack_method ['stack'] - if 'junction_method' is set to 'ignore', this parameter will control how the function deals with overlapping sections of stream networks when generating the shapefile. Valid inputs are 'stack' (default) and 'average'. If set to 'stack', the output shapefile will have multiple stacked polylines in overlapping portions of networks. This is similar to how Profiler51 worked. If set to 'average', the function will average overlapping portions of networks on a node by node basis. Note that if 'junction_method' is set to 'check', then this parameter is ignored.
- shape_name ['ksn'] - name for the shapefile to be export, must have no spaces to be a valid name for ArcGIS and should NOT include the '.shp'
- save_figures [false] - logical flag to either save figures showing ksn fits (true) or to not (false - default)

Outputs:

- *_KsnFit.txt - n x 12 array of node list for selected stream segments, columns are x coordinate, y coordinate, drainage area, ksn, negative ksn error, positive ksn error, reference concavity, best-fit concavity, mininum threshold area, gradient, fit residual, and an identifying number. Note that if using the code in 'concavity_method','auto' mode then the reference concavity and best-fit concavity columns will be the same.
- *_KsnBounds.txt - n x 4 array of selected bounds for fitting ksn, columns are x coordinate, y coordinate, elevation, and the stream identifying number (this could be thought of as a list of knickpoints), also output as a seperate shapefile. If x y and z values appear as NaN, this indicates that bounds for this stream were not selected.
- *.shp - shapefile of the stream network containing the KsnFit outputs as fields
- *_knicks.shp - shapefile of ksn fit boundaries

* is controlled by 'shape_name'

Notes

- If no boundaries/knickpoints are selected for any of the streams selected, then a '*_knicks.shp' shapefile will not be produced.
- The '*_profiler.mat' that is saved out contains additional files besides the formal outputs of the code. These additional variables are necessary to be able to restart a run using the 'restart' option.
- f you have set 'save_figures' to true, DO NOT close figures manually as this will cause the code to error.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
KsnProfiler /path/to/wdir Topo.mat
KsnProfiler /path/to/wdir Topo.mat redefine_threshold true
KsnProfiler /path/to/wdir Topo.mat conditioned_DEM CondDEM.mat
```

A.19 MakeCombinedSwath

Description: Function to plot various additional data onto a swath profile.

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins' points - name of text file containing n x 2 matrix of x,y points for swath, minimum are two points (start and end points). First row contains starting point and proceeds down rows, additional points besides a start and end are treated as bends in the swath. Coordinates for points must be in the same coordinate system as DEM and must lie within the DEM (cannot be coordinates on the very edge of the DEM).
- width - width of swath in map units
- data_type - the type of additional data you are providing to plot along with the swath, supported inputs are:
  - 'points3' - generic point dataset, expects a n x 3 matrix with values of x, y, and z stored in a text file
  - 'points4' - generic point dataset, expects a n x 4 matrix with values of x, y, z, and extra value stored in a text file. Dots will be colored by this extra value
  - 'points5' - generic point dataset, expects a n x 5 matrix with values of x, y, z, and two extra values stored in a text file. Dots will colored by the first extra value (column 4) and scaled by the second extra value (column 5).
  - 'eqs' - earthquakes, expects a n x 4 matrix with x, y, depth, and magnitude stored in a text file. Points will be scaled by magnitude and colored by distance from swath line. Expects depth to be positive.
  - 'gps' - gps velocity vectors, expects a n x 6 matrix, with x, y, north component, east component, north uncertainty, and east uncertainty. See 'ProjectGPSOntoSwath' for additional details
  - 'STREAMobj' - will project portions of selected stream profiles (as points) onto a swath. Expects a matfile containing a STREAMobj that was generated from the provided DEM (can be the same input as MatFile, but you must provide it again)
  - 'ksn_chandata' - will plot swath through ksn values, expects full path of a *chandata.mat file as output from old Profiler51 code (just in case you have some sitting around)
  - 'ksn_shape' - will plot swath through ksn values, expects the shapefile output from 'cmpKsnChiBatch' or 'cmpKsnProfiler' function 'basin_stats' - will plot swath through selected mean basin values as calculated from 'cmpProcessRiverBasins', expects matfile output from 'cmpCompileBasinStats' and requires an entry to optional input 'basin_value' and accepts optional input to 'basin_scale'. Will place point for basin at mean elevation and projected location of the basin centroid, will color by value provided to 'basin_value' and will optionall scale the point by the value provided to 'basin_scale'
  - 'basin_knicks' - will plot swath through knickpoints as chosen by 'cmpFindBasinKnicks'. For 'data' provide name of folder within working directory to find knickpoint files saved as a result of running 'FindBasinKnicks' on a series of basins selected from 'ProcessRiverBasins'
- data - input data, form varies depending on choice of data_type
- data_width - width in map units of swath through provided data. Values greater than data_width/2 from the center line of the toposwath will not be plotted

Optional Inputs:

- sample [] - resampling distance along topographic swath in map units, if no input is provided, code will use the cellsize of the DEM which results in no resampling.
- smooth [0] - smoothing distance, width of filter in map units over which to smooth values, default (0) results in no smoothing
- vex [10] - vertical exaggeration for the topographic swath. Note that because matlabs controls on physical axis dimensions are problematic, the vertical exaggeration controls don't work on plots that have two panels (e.g. 'ksn_batch', 'ksn_profiler', 'ksn_chandata', and 'eqs')
- basin_value [] - required for option 'basin_stats', name (as it appears in the provided table provided to 'data') of the value you wish to color points by
- basin_scale [] - optional input for option 'basin_stats', name (as it appears in the provided table provided to 'data') of the value you wish to scale points by
- plot_map [true] - logical flag to plot a map displaying the location of the topographic swath and the additional data included in the swath (red dots) and those not (white dots) based on the provided data_width parameter.
- cmap ['parula'] - valid name of colormap (e.g. 'jet')
- save_figure [false] - logical flag to save the swath figure as a pdf

Outputs:

- SwathArray.txt - n x 6 array containing x coordinate, y coordinates, distance along the swath, min elevation, mean elevation, max elevation
- SwathBends.txt - distances along swath of any bends, 0 if no bends
- SwathBounds.shp - polyline shapefile showing outline of swath for both topo and data and center line of swath
- SwathProjectedData.txt - data for plotting the swath through the provided data, distances that area 'NaN' indicate those data do not fall on the swath line provided. Form of output depends on data_type:
  - 'points3' - distances, elevation, distance from base line, x coordinate, y coordinate
  - 'points4' - distances, elevation, value, distance from base line, x coordinate, y coordinate
  - 'eqs' - distances, depth, magnitude, distance from base line, x coordinate, y coordinate
  - 'STREAMobj' - distances, elevation, distance from base line, x coordinate, y coordinate
  - 'ksn_chandata' - distances, elevation, ksn, distance from base line, x coordinate, y coordinate
  - 'ksn_shape' - distances, ksn, distance from base line, x coordinate, y coordinate
  - 'basin_stats' - distances, mean basin elevation, 'basin_value', 'basin_scale' (if provided), distance from base line, x coordinate, y coordinate

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
MakeCombinedSwath /path/to/wdir Topo.mat points.txt 10000 points3 data_points.
    txt 20000
MakeCombinedSwath /path/to/wdir Topo.mat points.txt 10000 basin_stats
    BasinTable.mat 20000 basin_value mean_ksn
```

A.20 MakeSerialSwath

Description: Function to create a series of swath profiles perpendicular to a provided line

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins', alternatively, if you wish to generate a swath through an aribtrary raster, you can provide the name of an ascii grid (.txt) to use
- points - name of text file containing n x 2 matrix of x,y points for the line along which series of perpendicular swaths will be generated, minimum are two points (start and end points). First row contains starting point and proceeds down rows, additional points besides a start and end are treated as bends in the line. Coordinates for points must be in the same coordinate system as DEM and must lie within the DEM (cannot be coordinates on the very edge of the DEM). If you provide an empty array this will invoke a display of the DEM that you can use to manually define the line.
- divisions - scalar that controls the number of swaths that will be generated. How this parameter is interpreted depends on value of the optional 'div_type' parameter. If 'div_type' is 'number' then the value provided to 'divisions' will be the total number of swaths created and the width of these swaths will equal the length of the line (defined by 'points') divided by the number of swaths. If 'div_type' is 'width' then the value provided to 'divisions' will be the width of each swath in map units. The code will start producing swaths from the beginning of the line defined by points and will proceed until it can not produce a swath that is 'width' wide (e.g. if the total length of the line defined by points is 11000 meters and 2000 is provided to 'divisions' with 'div_type' set to 'width' then the result will be 5, 2000 meter wide swaths and the last 1000 meter length of the line defined by 'points' will be ignored).
- sw_length - length of individual swaths (perpendicular to the line defined by 'points')

Optional Inputs:

- div_type ['number'] - controls how the 'divisions' parameter is interpreted. Viable entries are 'number' or 'width', see above the 'division' parameter for more info.
- alignment ['center'] - controls how the serial swaths are drawn relative to the line defined by 'points'. Vialble entries are: 'center', 'left', 'right', and 'between'. Behavior is as follows:
  - 'center' - Swaths will be drawn so that their centers intersect the line defined by 'points'.
  - 'right' - Swaths will be drawn to the righthand side of the line. This is based on the order in which points are defined, e.g. if the xy coordinates provided to 'points' define an east-west oriented line and the order of the points go from west to east then a 'right' alignment will draw swaths south of the line.
  - 'left' - Swaths will be drawn to the lefthand side of the line. This is based on the order in which points are defined, e.g. if the xy coordinates provided to 'points' define an east-west oriented line and the order of the points go from west to east then a 'left' alignment will draw swaths north of the line.
  - 'between' - Swaths will be drawn between two lines, one provided to the required 'points' input and one provided to the optional 'points2' input. If 'alignment' is set to 'between' and you provide an input to 'points' but none to 'points2', this will generate an error. If 'alignment' is set to 'between' and you leave

'points' empty, then you can graphically select both lines between which the serial swaths will be drawn. The code uses the shorter of the two provided lines to determine the number of swaths (and the widths of those swaths if 'div_type' is set to 'width'). Note that setting 'alignment' to 'between' means the input to the provided 'sw_length' parameter will be ignored.

- 'points2' [] - option to provide name of text file containing n x 2 matrix of x,y points for defining the other half of the area within which to draw swaths if 'alignment' is set to 'between'.
- sample [cellsize of DEM] - resampling distance along swath in map units, if no input is provided, code will use the cellsize of the DEM which results in no resampling.
- smooth [0] - smoothing distance, width of filter in map units over which to smooth values, default (0) results in no smoothing
- plot_map [true] - logical to turn on plotting of map showing location of all swaths
- plot_individual [false] - logical to turn on plotting of individual maps and swaths (i.e. there will be one figure per swath)
- save_figures [false] -logical to turn on the saving of generated figures as PDFs

Outputs:

- SerialSwath_*.txt - n x 6 array containing x coordinate, y coordinates, distance along the swath, min elevation, mean elevation, max elevation for each serial swath
- SerialSwathBounds.shp - polyline shapefile showing outline of all swaths and the centerline

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
MakeSerialSwath /path/to/wdir topo.mat points.txt 10 5000
MakeSerialSwath /path/to/wdir precip.txt points.txt 10 5000
MakeSerialSwath /path/to/wdir topo.mat [] 10 5000
```

A.21 MakeStreams

Description: Function takes a dem and outputs the necessary base datasets for use in other TopoToolbox functions. Input DEMs with grid resolutions (i.e. cellsizes) that are not whole numbers sometimes cause issues in companion functions. If the provided DEM has a non-whole number for a cellsize, the code will warn the user (but not do anything). If you want to fix the cellsize issue, you can either reproject in a GIS program or you can use this code (with 'resample_grid' set to true) to do it for you.

Required Inputs:

- wdir - full path of working directory
- dem - name of dem file as either an ascii text file (recommended) or geotiff
- threshold_area - minimum accumulation area to define streams in meters squared
- file_name - name for matfile containing the DEM, FD, A, and S (for use in all the other compiled versions of the TAK codes) and the shapfile of the stream network, do not include a file type suffix, this will be added by the function.

Optional Inputs:

- precip_grid [] - optional input of name of a precipitation raster (ascii or geotiff). If you provide an argument for this, the code will use this to produce a weighted flow accumulation grid.
- rr_grid [] - optional input of nameof a runoff ratio raster (ascii or geotiff). If you provide an argument for this, the code will use this, along with the input to 'precip_grid' to produce a weighted flow accumulation grid.
- no_data_exp [] - input to define no data conditions. Expects a string that defines a valid equality using the variable DEM OR 'auto'. E.g. if you wish to define that any elevation less that or equal to 0 should be set to no data, you would provide 'DEM<=0' or if you wanted to set elevations less than 500 and greater than 1000 ot no data, you would provide 'DEM<500 | DEM>1000'. If the expression is not valid the user will be warned, but the code will continue and ignore this continue. If you provide 'auto' the code will use the log of the gradient to identify true connected flats and set these to nan. If you want more control on removing flat ares that are at multiple elevations (e.g. internally drained basins), consider using 'RemoveFlats'.
- min_flat_area [1e8] - minimum area (in m^2) for a portion of the DEM to be identified as flat (and set to nan) if 'no_data_exp' is set to 'auto'. If 'no_data_exp' is not called or a valid logical expression is provided, the input to 'min_flat_area' is ignored.
- resample_grid [false] - flag to resample the grid. If no input is provided for new_cellsize, then the grid will be resampled to the nearest whole number of the native cellsize.
- new_cellsize [] - value (in map units) for new cellsize.

Outputs:

- Saved matfile containing (for use with other 'cmp*' codes):
  - DEM - GRIDobj of the DEM
  - FD - FLOWobj from the supplied DEM
  - A - Flow accumulation grid (GRIDobj)
  - S - STREAMobj derived from the DEM
- Shapefile of streams

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
MakeStreams /path/to/wdir dem.txt 1e6 Topo
MakeStreams /path/to/wdir dem.txt 1e6 Topo no_data_exp auto min_flat_area 1e6
MakeStreams /path/to/wdir dem.txt 1e6 Topo no_data_exp DEM<=0
```

A.22 MakeTopoSwath

Description: Wrapper around TopoToolbox SWATHobj functionality

Required Inputs:

- wdir - full path of working directory
- MatFile - Full path of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- points - name of text file containing n x 2 matrix of x,y points for swath, minimum are two points (start and end points). First row contains starting point and proceeds down rows, additional points besides a start and end are treated as bends in the swath. Coordinates for points must be in the same coordinate system as DEM and must lie within the DEM (cannot be coordinates on the very edge of the DEM).
- width - width of swath in map units

Optional Inputs:

- sample [] - resampling distance along swath in map units, if no input is provided, code will use the cellsize of the DEM which results in no resampling.
- smooth [0] - smoothing distance, width of filter in map units over which to smooth values, default (0) results in no smoothing
- vex [10] - vertical exaggeration for displaying plot.
- plot_as_points [false] - logical flag to switch plot type to distributions of points
- plot_as_heatmap [false] - logical flag to switch plot type to a heat map
- save_figure [false] - logical flag to save the swath figure as a pdf (this will also set 'plot_figure' to true)

Outputs:

- SwathArray.txt - n x 6 array containing x coordinate, y coordinates, distance along the swath, min elevation, mean elevation, max elevation
- SwathBends.txt - distances along swath of any bends, 0 if no bends
- SwathBounds.shp - polyline shapefile showing outline of swath and center line of swath

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
MakeTopoSwath /path/to/wdir Topo.mat points.txt 10000
MakeTopoSwath /path/to/wdir Topo.mat points.txt 10000 vex 5 plot_as_heatmap
    true
```

A.23 Mat2Arc

Description: Function converts all valid topotoolbox files contained within a mat file to Arc compatible outputs. Specifically converts any GRIDobjs to ascii files, any STREAMobjs to shapefiles, any FLOWobjs to ArcGIS flow direction grids saved as an ascii file, and any valid mapstructures to shapefiles.

Required Inputs:

- wdir - full path of working directory
- MatFile - full path to matfile of interest
- file_prefix - characters to add to the front of all output files

Optional Inputs:

- raster_type ['ascii'] - option to specify the format of the raster export, valid inputs are 'tif' or 'ascii'

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
Mat2Arc /path/to/wdir Topo.mat outputs
Mat2Arc /path/to/wdir Topo.mat outputs raster_type tif
```

A.24 PlotIndividualBasins

Description: Function takes outputs from 'ProcessRiverBasins' function and makes and saves plots for each basin with stream profiles, chi-z, and slope area

Required Inputs:

- wdir - full path of working directory
- location_of_data_files - name folder within working directory that contains the mat files from 'ProcessRiverBasins'

Optional Inputs:

- location_of_subbasins ['SubBasins'] - name of folder that contains subbasins of interest (if you created subbasins using "SubDivideBigBasins"), expected to be within the main Basin folder provided with "location_of_data_files"
- bin_size [500] - bin size (in map units) for binning slope area data.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
PlotIndividualBasins /path/to/wdir Basins
PlotIndividualBasins /path/to/wdir Basins locations_of_subbasins MySubBasins
```

A.25 PlotKsn

Description: Function to plot a map of normalized channel steepness on a hillshade colored by elevation.

Required Inputs:

- MatFile - full path to matfile of interest from which the ksn shapefile was created
- ksn - ksn data as a shapefile (as ouput from KsnProfiler, ProcessRiverBasins KsnChiBatch)

Optional Inputs: Can provide name of shapefile (as output by FindBasinKnicks or KsnProfiler) containing knickpoint locations

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
PlotKsn /path/to/wdir Topo.mat ksn.shp
PlotKsn /path/to/wdir Topo.mat ksn.shp knicks.shp
```

A.26 PrepareAddGrids

Description: Function to prepare additional grids for use in 'cmpProcessRiverBasins'

Required Inputs:

- out_file_name - name for the mat file to be produced, do not include the '.mat'
- Additional inputs must be given in groups of twos, and be in the order:
  1. name of the ascii or geotiff of the extra raster data you want to include (must be in the same projection as the original grid you provided to 'cmpMakeStreams' and will use in 'cmpProcessRiverBasins')
  2. a reference name for the produced grid

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
PrepareAddGrids /path/to/wdir AddGrids precip.tif precip
PrepareAddGrids /path/to/wdir AddGrids precip.tif precip ndvi.txt ndvi
```

A.27 PrepareCatAddGrids

Description Function to prepare categorical grids for input to 'cmpProcessRiverBasins'

Required Inputs:

- out_file_name - name for the mat file to be produced, do not include the '.mat'
- MakeStreamMat - full path of the output produced by 'cmpMakeStreams' and that you will be using as an input to 'cmpProcessRiverBasins'
- Additional inputs must be given in groups of threes, and be in the order:
  1. name of the shapefile containing the field you want to convert into a categorical grid
  2. the field name within the shapefile you want to convert into a categorical grid
  3. a reference name for the produced grid

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
PrepareAddCatGrids /path/to/wdir AddCatGrids Topo.mat geo_polygons.shp RTYPE
    rock_type
PrepareAddCatGrids /path/to/wdir AddCatGrids Topo.mat geo_polygons.shp RTYPE
    rock_type geo_polygons.shp UNIT unit_name
```

A.28 ProcessRiverBasins

Description: Function takes grid object outputs from MakeStreams script (DEM,FD,A,S), a series of x,y coordinates of river mouths, and outputs clipped dem, stream network, various topographic metrics, and river values (ks, ksn, chi). Saves mat files for use in other codes, if you want to produce files that are usable in an outside GIS program, be sure to set 'write_arc_files' to true.

Required Inputs:

- wdir - full path of working directory
- MakeStreamsMat - name or location of the mat file saved after running 'cmpMakeStreams'
- river_mouths - locations of river mouths (i.e. pour points) above which you wish to extract basins, can take one of three forms:
    1. name of a text file containing a nx3 array of river mouths with x, y, and a number identifying the stream/basin of interest (must be same projection as DEM) saved as a '.txt'. No extra columns should be present, the code should work with or without headers as long as headers are restricted to a single line. The text file of Outlets saved by BasinPicker, can be used as the river mouths
    2. a single value that will be interpreted as an elevation that the code will use this to autogenerate river mouths at this elevation.
    3. point shapefile with one numeric user input field (e.g. the default 'ID' field generated by ArcGIS) that will be used as the river mouth ID (must be same projection as DEM).
- basin_dir - name of folder to store basin files (if specified folder does not exist in current directory, code will create it)

Optional Inputs:

- conditioned_DEM [] - option to provide a hydrologically conditioned DEM for use in this function, expects an ascii grid as saved by 'cmpConditionDEM' See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)
- threshold_area [1e6] - minimum accumulation area to define streams in meters squared
- segment_length [1000] - smoothing distance in meters for averaging along ksn, suggested value is 1000 meters
- ref_concavity [0.5] - reference concavity for calculating ksn, suggested value is 0.45
- ksn_method [quick] - switch between method to calculate ksn values, options are 'quick', 'trunk', or 'trib', the 'trib' method takes 3-4 times longer than the 'quick' method. In most cases, the 'quick' method works well, but if values near tributary junctions are important, then 'trib' may be better as this calculates ksn values for individual channel segments individually. The 'trunk' option calculates steepness values of large streams independently (streams considered as trunks are controlled by the stream order value supplied to 'min_order'). The 'trunk' option may be of use if you notice anomaloulsy high channel steepness values on main trunk streams that can result because of the way values are reach averaged.

- min_order [4] - minimum stream order for a stream to be considered a trunk stream, only used if 'ksn_method' is set to 'trunk'
- write_arc_files [false] - set value to true to output a ascii's of various grids and a shapefile of the ksn, false to not output arc files
- add_grids [] - option to provide the name of the mat file produced by running 'cmpPrepareAddGrids'. Use this function if you want this function to calculate statistics for additional grids (e.g. precipitation).
- add_cat_grids [] - option provide the name of the mat file produced by running 'cmpPrepareCatAddGrids'. Use this if you want to calculate statisitcs related to categorical data stored in a shapefile (e.g. geologic map).
- resample_method ['nearest'] - method to use in the resample function on additional grids (if required). Acceptable inputs are 'nearest', 'bilinear', or 'bicubic'. Method 'nearest' is appropriate if you do not want the resampling to interpolate between values (e.g. if an additinal grid has specific values that correlate to a property like rock type) and either 'bilinear' or 'bicubic' is appropriate if you want smooth variations between nodes.
- gradient_method ['arcslope'] - function used to calculate gradient, either 'arcslope' (default) or 'gradient8'. The 'arcslope' function calculates gradient the same way as ArcGIS by fitting a plane to the 8-connected neighborhood and 'gradient8' returns the steepest descent for the same 8-connected neighborhood. 'gradient8' will generally return higher values than 'arcslope'.
- calc_relief [false] - option to calculate local relief. Can provide an array of radii to use with 'relief_radii' option.
- relief_radii [2500] - a 1d vector (column or row) of radii to use for calculating local relief, values must be in map units. If more than one value is provided the function assumes you wish to calculate relief at all of these radii. Note, the local relief function is slow so providing multiple radii will slow code performance. Saved outputs will be in a m x 2 cell array, with the columns of the cell array corresponding to the GRIDobj and the input radii.

Notes: The code will perform a check of the river_mouths input to confirm that 1) there are no duplicate ID numbers (it will dump your ID numbers and create new ID numbers if this is the case and output a text file containing the river mouth locations with their new ID numbers) and 2) that no provided river mouths are outside the boundaries of the DEM (it will remove these IDs if this the case).

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
ProcessRiverBasins /path/to/wdir Topo.mat river_mouths.txt Basins
ProcessRiverBasins /path/to/wdir Topo.mat river_mouths.shp Basins
ProcessRiverBasins /path/to/wdir Topo.mat 500 Basins
ProcessRiverBasins /path/to/wdir Topo.mat river_mouths.txt Basins
   add_cat_grids AddCatGrids.mat add_grids AddGrids.mat
ProcessRiverBasins /path/to/wdir Topo.mat river_mouths.shp Basins calc_relief
   true relief_radii [1000 2500 5000]
```

A.29 ProjectedIncision

Description: Function for generating maps of projected incision throughout a network based on results of stream projections from SegmentProjector. Code uses the steepness of the fit portion of the stream network and the chi values of the entire network to find the projected elevations of the original stream network and amounts of incision implied by

this, i.e. if you fit portions of streams in SegmentProjector that you interpret as recording a former low relief landscape now uplifted and incised, this code first calculates what the stream elevations would be if no incision (but surface uplift) had occurred. The code then subtracts this projected elevation from the current elevation of the network to estimate the amount of incision this would imply. If you projected multiple streams in SegmentProjector (that are stored in the OUT cell array) then the code will calculate projected elevations and implied incision for each projected stream separately and then find the mean, standard deviation, minimum, and maximum values of the projected elevations and incision for the network in question. Additionally, if the Sc input has multiple outlets, the projected elevation and incision values will only be calculated for portions of the stream network S that are 1) upstream of the outlets in Sc and 2) connected to channels used to project. E.g. if Sc has two outlets, defining two connected stream networks we'll call network A and B, and you projected two streams in network A and three streams in network B, then the portions of the resulting zpOUT and inOUT values that correspond to nodes in network A will only be calculated using the projections from the two streams in network A. This is done in case there is spatial variability in the amount of incision.

Note that negative incision values indicate that the elevations of these portions of the modern stream network are above the projected elevations. It is also important to note that this function explicitly assumes that no change in drainage area / network topology has occurred during surface uplift.

Required Inputs:

- wdir - full path of working directory
- MatFile - Name of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- NewStreamMat - Name of matfile containing the subset of streams used to project segments when running SegmentProjector
- ProjSegmMat - Name of matfile output from SegmentProjector

Optional Inputs:

- shape_name ['bsn'] - prefix on the output shapefiles names
- display_figure [true] - logical flag to display a figure showing the values of mean, max, min, and standard deviation of calculated incision
- save_figure [false] - logical flag to save the displayed figure as a pdf (setting this to true) will force 'display_figure' to be true
- exclude_streams [] - optional input if you wish to exclude any of the projected streams in the OUT cell array from the calculation. Provide a list of stream numbers (e.g. if you wish to exclude the 2nd and 15th stream that you projected, you would give [2 15] as the input to this parameter)
- conditioned_DEM [] - option to provide name of a hydrologically conditioned DEM for use in this function, expects the mat file as saved by 'cmpConditionDEM'. See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)

Outputs: Code saves two shapefiles, '*_Pnts_Used.shp' and '*_Proj_Incision.shp'. '*_Pnts_Used.shp' is a point shapefile that records the channel heads of the projected stream used to calculate the incision amounts. '*_Proj_Incision.shp' is stream shapefile containing the mean, standard deviation, minimum, and maximum of the projected elevations AND the mean, standard deviation, minimum, and maximum of the incision values throughout the stream network

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
ProjectedIncision /path/to/wdir Topo.mat PickedSegments_5.mat
    Topo_Projected_Channels.mat
```

A.30 RemoveFlats

Description: Function takes DEM and attempts a semi-automated routine to remove flat areas with some input from the user to select areas considred to be flat. This function sometimes works reliably, but will never produce as clean a result as manually clipping out flat areas in gis software (but it's a lot faster!)

Required Inputs:

- wdir - full path of working directory
- dem - either full path of dem file as either an ascii text file or geotiff
- strength - integer value between 1 and 4 that controls how aggressively the function defines flat areas, specifically related to the size of the neighborhood the function uses to connect ares of similar elevation. A strength of 1 = a 3x3 neighborhood, 2=5x5, 3=7x7, and 4=9x9. If the results of the function do not capture enough of the flat areas in the MASK, increase the strength and rerun. Similarly, if the function erroneously includes areas that are not part of what you consider the flats, try decreasing the strength.
- file_name - name of output ascii file (without a file type suffix)

Outputs: georeferenced ascii file of the processed dem and mask (mask will have '_mask' appended to the file name you provide).

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
RemoveFlats /path/to/wdir dem.txt 1 dem_rm_flat
```

A.31 SegmentPicker

Description: Function to select a segment of a stream network from the top of the stream, and plot the long profile and chi-Z relationship of that segment,also outputs the extraced portion of the stream network and chi structure (out of 'chiplot'). Allows user to iteratively select different parts of the stream network and display. Keeps running dataset of all the streams you pick and accept.

Required Inputs:

- wdir - full path of working directory
- MatFile - Name of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'
- basin_num - basin number from ProcessRiverBasins for output name or other identifying number for the set of streams you will pick

Optional Inputs:

- conditioned_DEM [] - option to provide full path of a hydrologically conditioned DEM for use in this function, expects the mat file as saved by 'cmpConditionDEM'. See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- new_stream_net [] - option to provide name of a matfile containing a new stream network as as output from another function (e.g. cmpFindThreshold) to use instead of the stream network saved in the MatFile provided to the function. This new stream network must have been generated from the DEM stored in the provided MatFile
- direction ['down'] - expects either 'up' or 'down', default is 'down', if 'up' assumes individual selections are points above which you wish to extract and view stream profiles (i.e. a pour point), if 'down' assumes individual selections are channel heads if specific streams you wish to extract and view stream profiles.
- method ['new_picks'] - expects either 'new_picks' or 'prev_picks', default is 'new_picks' if no input is provided. If 'prev_picks' is given, the user must also supply an input for the 'picks' input (see below)
- plot_style ['refresh'] - expects either 'refresh' or 'keep', default is 'refresh' if no input is provided. If 'refresh' is given, the plots reset after each new stream pick, but if 'keep' is given, all selected streams remain on both the map (as thick red lines) and the chi-z/longitudinal profile/slope-area plots.
- plot_type ['vector'] - expects either 'vector' or 'grid', default is 'vector'. Controls whether all streams are drawn as individual lines ('vector') or if the stream network is plotted as a grid and downsampled ('grid'). The 'grid' option is much faster with large datasets, but can result in inaccurate choices. The 'vector' option is easier to see, but can be very slow to load and interact with.
- calc_full_slope_area [false] - logical flag to either calculate and display the slope area data for just the trunk stream in the network (false, default), or to calculate and display slope area data for all streams in the network (true). If direction is set to 'up' and you are choosing large stream networks, it is strongly recommended that you leave this parameter set to false to speed code completion.
- complete_networks_only [false] - if true, the code will filter out portions of the stream network that are incomplete prior to choosing streams
- picks - expects name of a textfile containing a n x 3 matrix with columns as x coordinates, y coordinates, and an identifying number OR the name of a point shapefile with a single value column of identifying numbers. Will interpret this input as a list of channel heads if 'direction' is 'down' and a list of channel outlets if 'direction' is 'up'.
- ref_concavity [0.50] - reference concavity for calculating Chi-Z, default is 0.50

- min_elev [] - minimum elevation below which the code stops extracting channel information, only used if 'direction' is 'down'
- max_area [] - maximum drainage area above which the code stops extracting channel information, only used if 'direction' is 'down'
- recalc [false] - only valid if either min_elev or max_area are specified. If recalc is false (default) then extraction of streams stops downstream of the condition specified in either min_elev or max_area, but chi is not recalculated and distances will remain tied to the original stream (i.e. distances from the outlet will be relative to the outlet of the stream if it continued to the edge of the DEM, not where it stops extracting the stream profile). If recalc is true, then chi and distance are recalculated (i.e. the outlet as determined by the min_elev or max_area condition will have a chi value of zero and a distance from mouth value of zero).
- threshold_area [1e6] - used to redraw downsampled stream network if 'plot_type' is set to 'grid'
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)
- bin_size [500] - bin size (in map units) for binning slope area data.

Outputs: Saves an output called 'PickedSegements_*.mat' with the provided basin number containing these results:

- StreamSgmnts - Cell array of selected stream segments as STREAMobj
- ChiSgmnts - Cell array of selected chi structures
- SlpAreaSgmnts - Cell array of slope area data
- Sc - Single STREAMobj containing all the streams chosen. and if 'down' is selected:
- Heads - nx3 matrix of channel heads you picked with x cooord, y coordinate, and pick number as the columns and if 'up' is selected:
- Outlets - nx3 matrix of outlets you picked with x cooord, y coordinate, and pick number as the columns (valid input to 'ProcessRiverBasins' as 'river_mouths' parameter)

Also saves a shapefile of the selected stream network

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
SegmentPicker /path/to/wdir Topo.mat 1
SegmentPicker /path/to/wdir Topo.mat 20 direction up picks Mouths.shp
```

A.32 SegmentPlotter

Description: Function to plot all of the chi-Z relationships, longitudinal profiles, and slope area plots from a series of picked segments of river networks that result from the 'SegmentPicker' function.

Required Input:

- wdir - full path of working directory
- basin_nums - row or column vector of basin numbers used for the SegmentPicker you wish to plot together. Code expects that the mat files saved from cmpSegmentPicker are in the present working directory.

Optionl Input:

- separate [false] - logical flag to plot all segments as separate figures
- subset [] - list of specific river numbers (i.e. the third column of either the 'Heads' or the 'Outlets' variable) that you wish to include in the plot. Only valid if you have only provided a single basin number for 'basin_nums'.
- label [false] - logical flag to either label individual streams with the river number (true) or not label them (false, default). If 'separate' flag is true then the input for label is ignored as the stream number will be in the title of the plots
- names [] - option to add an identifying name for streams when 'label' is set to true.

Outputs: saves pdfs of all figures produced

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
SegmentPlotter /path/to/wdir 1
SegmentPlotter /path/to/wdir [1 2 3 6] separate true
```

A.33 SegmentProjector

Description: Function to interactively select segments of a channel profile you wish to project (e.g. projecting a portion of the profile with a different ksn). You can use the 'cmpSegmentPicker' function to interactively choose channels to provide to the StreamProjector function. If the STREAMobj has more than one channel head, this code will iterate through all channel heads (i.e. make sure you're only providing it stream you want to project, not an entire network!). It calculates and will display 95 % confidence bounds on this fit.

Required Inputs:

- wdir - full path of working directory
- MatFile - Name of matfile output from either 'cmpMakeStreams' or the name of a single basin mat file from 'cmpProcessRiverBasins'

Optional Inputs:

- conditioned_DEM [] - option to provide name of a hydrologically conditioned DEM for use in this function, expects the mat file as saved by 'cmpConditionDEM'. See 'cmpConditionDEM' function for options for making a hydrological conditioned DEM. If no input is provided the code defaults to using the mincosthydrocon function.
- new_stream_net [] - option to provide name of a matfile containing a new stream network as as output from another function (e.g. cmpFindThreshold) to use instead of the stream network saved in the MatFile provided to the function. This new stream network must have been generated from the DEM stored in the provided MatFile

- concavity_method ['ref']- options for concavity:
  - 'ref' - uses a reference concavity, the user can specify this value with the reference concavity option (see below)
  - 'auto' - function finds a best-fit concavity for the provided stream
- pick_method ['chi'] - choice of how you want to pick the stream segment to be projected:
  - 'chi' - select segments on a chi - z plot
  - 'stream' - select segments on a longitudinal profile
- ref_concavity [0.50] - refrence concavity used if 'theta_method' is set to 'auto'
- refit_streams [false] - option to recalculate chi based on the concavity of the picked segment (true), useful if you want to try to precisely match the shape of the picked segment of the profile. Only used if 'concavity_method' is set to 'auto'
- save_figures [false] - option to save (if set to true) figures at the end of the projection process
- interp_value [0.1] - value (between 0 and 1) used for interpolation parameter in mincosthydrocon (not used if user provides a conditioned DEM)

Output:

- Projected_Channel_Heads.txt - 2 column text file containing the x and y coordinates of channel heads of projected channels
- Projected_Channel_*.txt - 10 column text file for each projected channel containing the x coordinate, y coordinate, drainage area, chi value, concavity, true elevation, projected elevation, postive uncertainty on projected elevation, and negative uncertainty on projected elevation.

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
SegmentProjector /path/to/wdir Topo.mat
SegmentProjector /path/to/wdir Topo.mat new_stream_net ThresholdStreams.mat
    pick_method stream
```

A.34 SubDivideBigBasins

Description: Function takes outputs from 'ProcessRiverBasins' function and subdvides any basin with a drainage area above a specified size and outputs clipped dem, stream network, variout topographic metrics, and river values (ks, ksn, chi)

Required Inputs:

- wdir - full path of working directory
- basin_dir - full path of folder which contains the mat files from 'ProcessRiverBasins'
- max_basin_size - size above which drainage basins will be subdivided in square kilometers

- divide_method - method for subdividing basins, options are ('confluences' and 'up_confluences' is NOT recommended large datasets):
  - 'order' - use the outlets of streams of a given order that the user can specify with the optional 's_order' parameter
  - 'confluences' - use the locations of confluences (WILL PRODUCE A LOT OF SUB BASINS!). There is an internal parameter to remove extremely short streams that would otherwise result in the code erroring out.
  - 'up_confluences' - use locations just upstream of confluences (WILL PRODUCE A LOT OF SUB BASINS!). There is an internal parameter to remove extremely short streams that otherwise result in the code erroring out.
  - 'filtered_confluences' - use locations of confluences if drainage basin above confluence is of a specified size that the user can specify with the optional 'min_basin_size'
  - 'p_filtered_confluences' - similar to filtered confluences, but the user defines a percentage of the main basin area with the optional 'min_basin_size'
  - 'trunk' - uses the tributary junctions with the trunk stream within the main basin as pour points for subdivided basins. There is an internal parameter to remove extremely short streams that would otherwise result in the code erroring out.
  - 'filtered_trunk' - same as 'trunk' but will only include basins that are greater than the min_basin_size
  - 'p_filtered_trunk' - same as 'filtered_trunk' but 'min_basin_size' is interpreted as a percentage of the main basin area

Optional Inputs:

- SBFiles_Dir ['SubBasins'] - name of folder (within the main Basins folder) to store the subbasin files. Subbasin files are now stored in a separate folder to aid in the creation of different sets of subbasins based on different requirements.
- recursive [true] - logical flag to ensure no that no subbasins in the outputs exceed the 'max_basin_size' provided. If 'divide_method' is one of the trunk varieties the code will continue redefining trunks and further split subbasins until no extracted basins are greater than the 'max_basin_size'. If the 'divide_method' is one of the confluence varities, subbasins greater than 'max_basin_size' will simply no be included in the output. The 'recursive' check is not implemented for the 'order' method.
- threshold_area [1e6] - minimum accumulation area to define streams in meters squared
- segment_length [1000] - smoothing distance in meters for averaging along ksn, suggested value is 1000 meters
- ref_concavity [0.5] - reference concavity for calculating ksn
- write_arc_files [false] - set value to true to output a ascii's of various grids and a shapefile of the ksn, false to not output arc files
- s_order [3] - stream order for defining stream outlets for subdividing if 'divide_method' is 'order' (lower number will result in more sub-basins)

- min_basin_size [10] - minimum basin size for auto-selecting sub basins. If 'divide_method' is 'filtered_confluences' this value is interpreted as a minimum drainage area in km^2. If 'divide_method' is 'p_filtered_confluences', this value is interpreted as the percentage of the input basin drainage area to use as a minimum drainage area, enter a value between 0 and 100 in this case.
- no_nested [false] - logical flag that when used in conjunction with either 'filtered_confluences' or 'p_filtered_confluences' will only extract subbasins if they are the lowest order basin that meets the drainage area requirements (this is to avoid producing nested basins)

Notes:

- Only the 'order', 'trunk', 'filtered_trunk', and 'p_filtered_trunk' divide methods will not produce nested sub-basins.
- The interpolation necessary to produce a continuous ksn grid will fail on extremely small basins. This will not cause the code to fail, but will result in no 'KsnOBJc' being saved for these basins.
- Methods 'confluences', 'up_confluences', and 'trunk' can result in attempts to extract very small basins. There is an internal check on this that attempts to remove these very small basins but it is not always effective and can occasionally result in errors. If you are encountering errors try running the drainage area filtered versions

Examples if running for the command line, minus OS specific way of calling main TAK function:

```
SubDivideBigBasins /path/to/wdir Basins 100 trunk
SubDivideBigBasins /path/to/wdir Basins 100 order SBFiles_Dir MySubBasins
    order 4
```