GEOSPATIAL MODELLING ENVIRONMENT

Version: 0.7.2 RC2

www.spatialecology.com/gme

HAWTHORNE L. BEYER

Don't miss:

1. The page-linked keyword index at the back.
2. Section 2.3: Automation and batch processing.
3. The Hawthstools to GME conversion table in the Appendix.
4. The page-linked Table of Contents at the beginning.

IMPORTANT NOTICE

This is a beta version of the new Geospatial Modelling Environment, the next generation of HawthsTools. All of the commands listed in this document have been tested to the "Beta 2" level: that means they have passed a basic level of testing and consistency checks. However, it is highly recommended that you inspect the output from these commands carefully to ensure it is logical and consistent with your expectations.

Please do report any bugs you encounter! Ideally your email will include a zipped sample of data that I can use to replicate the problem. At the very least please copy and paste the entire contents of any error messages received. My email is: hawthorne@spatialecology.com. Thanks for your help in identifying problems.

Contents

1 INTRODUCING THE GEOSPATIAL MODELLING ENVIRONMENT

1.1 Overview

The promise of GIS has always been that it would allow us to obtain *better* answers to our questions. But this is only possible if we have tools that allows us to perform rigorous quantitative analyses designed for spatial data. The Geospatial Modelling Environment (GME) is a platform designed to help to facilitate rigorous spatial analysis and modelling.

GME provides you with a suite of analysis and modelling tools, ranging from small 'building blocks' that you can use to construct a sophisticated work-flow, to completely self-contained analysis programs. It also uses the extraordinarily powerful open source software R as the statistical engine to drive some of the analysis tools. One of the many strengths of R is that it is open source, completely transparent and well documented: important characteristics for any scientific analytical software.

GME incorporates most of the functionality of its predecessor, HawthsTools, but with some important improvements. It has a greater range of analysis and modelling tools, supports batch processing, offers new graphing functionality, automatically records work-flows for future reference, supports geodatabases, and can be called programmatically.

GME is under active development and I am always grateful for suggestions about how to improve the software, or recommendations of new tools to add. Thank you in advance for your feedback (email: hawthorne@spatialecology.com).

If you find this software useful, please consider providing financial support for this project.

1.2 Design philosphy

A number of years ago I published a free extension (HawthsTools) that contained a somewhat eclectic collection of tools designed to facilitate certain spatial analysis and modelling tasks. While I received a great deal of positive feedback on the tools (thanks to all of you who provided feedback) there were a number of fundamental limitations with the design of this software: it could not be automated, it took too long to develop and maintain tools, the tools could not be chained together very effectively, it was time consuming to support, etc.

The next generation of these tools (the Geospatial Modelling Environment) resolves many of the limitations in the original implementation and adds greatly enhanced new functionality. Here I outline the driving motivations in the design philosophy of the new tools:

1. Rigorous statistical analysis. I use the open source and extraordinarily powerful statistical software R to drive statistical analyses in ESRI ArcMap. I have long felt that the analytical capabilities of GIS software have been grossly inadequate. The promise of GIS has always been that it will allow us to obtain better answers to our questions, but this is facilitated by the analytical capabilities of the software. While much effort has been invested in the more graphical and technical aspects of GIS (displaying data, map making, data storage, movie making, etc), the analytical capabilities have been relatively neglected. I use R to begin to facilitate rigorous statistical analysis in a GIS environment. I have also

developed tools to facilitate stochastic simulations, bootstrapping and randomization testing using spatial data. I feel these are underutilized but key tools for spatial analysis.

2. Automation. In order to be useful for the widest possible range of applications I provide simple methods of automating the running of these tools. The new interface is entirely command line driven. This allows users to string together tools/commands as part of a larger work-flow. I also provide simple programming structures (e.g. a for... loop) to further automate repetitious work-flows. The command line interface also provides a straightforward method for calling these tools from other applications. There is therefore much more scope for interoperability and automation in this new version of the tools.

3. Functionality. The new design makes it quicker and easier to add new tools, which benefits both the developer and the user. It also facilitates the addition of much more sophisticated (higher order) tools, and makes it easier to maintain code each time a new version of ArcGIS is released. As a developer I want to spend less time maintaining code and more time adding new functionality.

4. Graphs. Graphs are extremely useful tools for exploring data and conceptualising relationships in data. I use R to provide graphing functions in ArcGIS (scatterplots, boxplots, histograms, etc).

5. Recording a work-flow. For scientific applications it is important to maintain a record of the steps in a work-flow so that the analysis can be appropriately described and repeated if necessary. The GME automatically records every command that is run and the result of that command as an HTML file so users do not have to spend time recording their work-flow elsewhere.

6. Interface flexibility. In this interface the output window is a web browser. This means that it can accommodate many types of graphical output (text, pictures, movies, dynamic HTML, etc), it can be subsequently viewed without using special software (just a web browser), it allows me to colour code output, and it makes it easy for users to adjust (e.g. making the text larger for those of us with fading eyesight).

Starting in GME version 0.6.0 you can run tools using a graphical user interface or the command line interface. The GUI can also be uesd to build commands that you then run using the command line interface. The GUI is convenient for running one-off commands, but for developing workflows that you may need to repeat I recommend the command line interface, which makes it straightforward to re-run a complex work-flow. Furthermore, once a command string is created it is easy to modify it and rerun the command (as opposed to GUI forms where you have to reset all the options again).

2 HOW TO USE GME

2.1 Instructions and tips for using this interface

GME command can be run using either a graphical user interface (GUI) or a command line interface. It is also possible to use the GUI interface to build commands, and then run them using the command line interface.

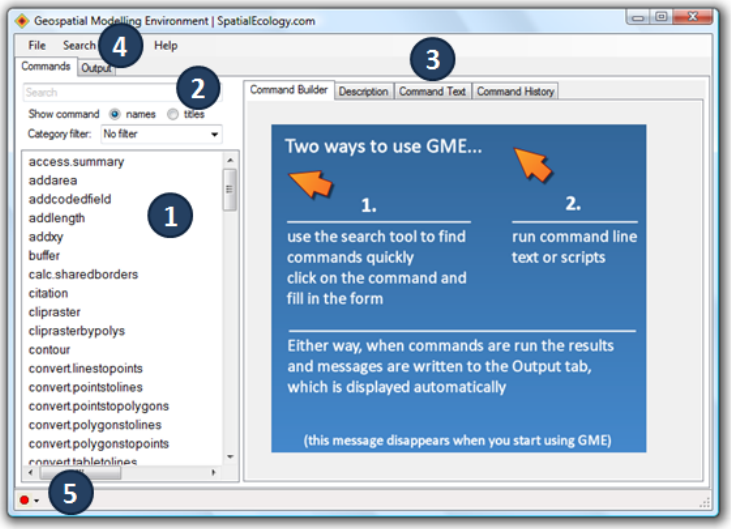

Figure 1: The GME interface. (1) Select a command from the list, and complete the form that is displayed to run the command (see next figure for an example). (2) Search for commands using a keyword or filtering the commands by category. (3) Alternatively, run commands from the command line using the Command Text tab. (4) Either way, when you run a command, processing results are displayed in the Output window, which is displayed automatically. (5) Use the red button if you wish to cancel processing.

There are a variety of resources to help you to find and specify commands:

1. The search box on the left side of GME: type a keyword (e.g. random) or even a few key letters (e.g. gen) to see the commands that contain this word in the command name, title or description.

2. Use the command category filter on the left side of GME: this filters the command list to show only the commands that are members of that category. This works in conjunction with the search tool, so set the category filter back to "No filter" if you wish to see/search all commands again. Many commands are members of more than one category.

3. Search the full help documentation (either the website or the PDF) using standard search tools in your web browser or PDF viewer. The "Commands" page on the web site is particularly useful for this.

For the command line interface it is highly recommended you use a text editor like Notepad++ to keep a record of your commands. You fill often find it convenient to copy

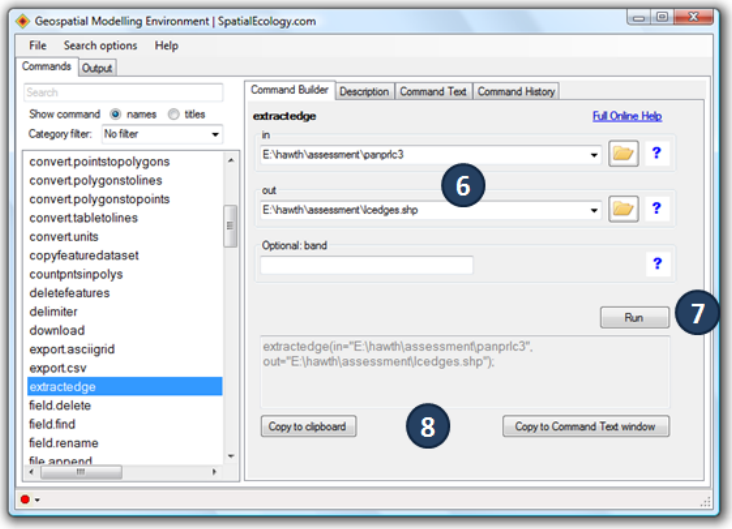

Figure 2: (6) Use the form to specify the parameters for the command you have selected. You do not need to specify values for optional parameters unless you wish to. The blue question mark provides a description of what each parameter means. As you specify parameters the command text is automatically updated at the bottom of the form. (7) If you wish to run the command immediately, press the Run button. (8) Alternatively, copy the command to the clipboard or the Command Text window for further editing or script development.

and paste a previously used command from notepad (where you may only need to modify it slightly). There are three important command line syntax rules: 1) you must use quote marks when supplying text, 2) always use a semi-colon to separate multiple commands, 3) avoid the use of special characters in your file and folder names. Note also that all the field and dataset naming rules that apply in ArcMap also apply here: a good rule of thumb is to keep all field and dataset names short and simple.

If you start a command that takes a long time to run you can cancel it using the red button in the lower left. Note that when you click 'Stop Processing' the interface may take a short time to obey as it only stops at sensible places in the code. Most tools can be cancelled in this way.

The result of the commands that are run are written to the output window. Often this will just be a report of how many records were successfully processed, but this could also include graphical or tabular output. If the tool failed to run then you will receive an error message that explains the nature of the problem encountered.

Note that the text in the output window is colour coded. Please pay special attention to the red and orange messages (error and warning messages respectively).

2.2 Working with geodatabases

GME supports reading and writing vector data using both the personal (Microsoft Access) geodatabase and file geodatabase formats. The syntax for specifying a geodatabase data source is: the path of the folder and the name of the geodatabase, an exclamation mark, and the name of the feature data source (the feature class). If this feature class is contained

within a 'feature dataset' within the geodatabase (a similar idea to a subfolder) then you would also include the feature dataset name followed by an exclamation mark.

An example of the specification of a feature class called boundaries within a personal geodatabase (admin.mdb): C:\data\europe\admin!boundaries

An example of a feature class called roads in the same geodatabase, but in the 'transport' feature dataset: C:\data\europe\admin!transport!roads

When writing data to a geodatabase, you do not have to pre-create the geodatabase, feature dataset, or feature class, but the folder you want it to reside in should exist. For instance, if C:\data\analysis is an empty folder, the output of GME commands can be directed to a new geodatabase using: C:\data\analysis\climate!admin!vectorgrid. GME will automatically create the new geodatabase (as a file geodatabase by default), then create the admin feature dataset, and then write the vectorgrid feature class. If you want the output geodatabase to be a personal geodatabase, then you need to create the empty geodatabase yourself using ArcCatalog.

Geodatabases are a convenient way of organizing and storing feature datasets, File geodatabases in particular can be highly efficient when working with very large datasets. But there are two issues you should consider. First, the geodatabase format is not very portable (unlike the shapefile format). So if you want to use your data in other applications, or share you data with other people who do not have the same software, then shapefiles may be the better choice. Second, Access files have a 2GB limit, so if you are working with a very large dataset then select the file geodatabase. (Running the Compact and Repair Database from within Access is also highly recommended after editing personal geodatabases).

Rasters stored in geodatabases have not been fully enabled in GME. I recommend you do not store your raster data in geodatabases.

2.3 Automation and batch processing

Using the GUI interface is useful for one-time tasks because it is simpler than writing command strings. For any iterative analysis, however, the only efficient approach is scripting. GME provides a suite of functionality specifically designed for this purpose (see the 'Strategic commands' section for a brief overview).

My recommendation for people who have large processing jobs is to write scripts in your favourite text editor (Notepad++ is fantastic and free), save them as a text file, and then use the run() command to run them from GME, or copy and paste them into the command window. Remember to always separate commands using a semi-colon. It does not matter if each command starts on a new line or not, but for the sake of readability you may wish to do that too.

You are also able to run multiple sessions of GME at the same time, thereby taking better advantage of your processing resources. If you do this, you should be careful to ensure that different sessions are not reading/writing the same data source at the same time.

Here, I describe two key scripting approaches to automation in more detail.

Automating commands with for loops in GME

The for loop is essential for efficient processing of many datasets. The few lines of code in a for loop can generate thousands of commands that are run. The premise of this approach is that you wish to iterate over a sequence of continuous numbers, or over a discontinuous or arbitrary sequence of values in a list, running a sequence of commands at each iteration. Simple examples will make this distinction clear.

Example 1: Iterating over a continuous series of numbers

If you have a telemetry dataset with 50 animals that are numbered starting at 101 and ending at 150, then to calculate a kernel density estimate for each animal the for loop would be:

```
for (i in 101:150){
kde(in="telemetry.shp", out=paste("kde_", i, ".img"), bandwidth="SCV", cellsize=20,
where=paste("ANIMALID=", i));
}
```

This would create output rasters named kde_1.img, kde_2.img, etc. This form of for loop is straightforward but rather limited because values of interest are often not continuous series.

Example 2: Iterating over a list of values

For loops can also iterate over an arbitrarily ordered list of values provided they have been defined as variables in GME. For instance:

```
ids <- c("M01","F01","M02","F02","J01");
```

We could write a for loop that simply specified the number of items in the list: for(i in 1:6){ ... }, but it is generally more useful to write flexible loops that determine the length of the list dynamically:

```
for (i in 1:length(ids)){
kde(in="telemetry.shp", out=paste("kde_", ids[i], ".img"), bandwidth="SCV", cellsize=20,
where=paste("ANIMALID='", ids[i], "'"));
}
```

Note that inside the for loop we now use ids[i] instead of just i. We could perhaps have just used i to name the output rasters in this particular case, but you will generally want to refer back to the value of the item in the list, not the value of the current index.

Often you may wish to perform some operation on many datasets stored in different folders, or on sets of unique values in an ID field. GME provides you with tools to automatically generate these lists. See these commands for further information: uniquevalues, list.raster, list.vector. For instance, list.raster can be used to find every raster dataset in a folder and all subfolders matching a specified naming convention. This can be very useful for automation.

Generating GME commands using R

R is a more flexible environment than GME for manipulating strings and constructing GME commands in an automated way. The idea here is that you either copy and paste these commands from R into GME, or write the commands to a text file and run them as a script from GME.

The R cat() and paste() functions are the key to generating GME commands in R. The important points to note are that:

1. use a slash-double quote combination (\") to write a double quote in the cat function
2. end each line with "\n" (this tells R to start a new line)
3. make sure you specify the sep="" option for that cat function because the default is a space

Example 1: writing KDE commands for each individual when the unique ID field is a text field

```
uniqueids <- unique(ANIMALID)
for (i in 1:length(uniqueids)){
cat("kde(in=\"telemetry.shp\", out=\"kde_", uniqueids[i], ".img\", bandwidth=\"SCV\", cellsize=20,
where=\"ANIMALID='", uniqueids[i], "'\");\n", sep="")
}
```

If the uniquevalues vector was c("M01","F01","M02","F02","J01"), for instance, that would result in these commands being written to the R window from where they can be copied and pasted into GME and run:

```
kde(in="telemetry.shp", out="kde_M01.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M01'");
kde(in="telemetry.shp", out="kde_F01.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F01'");
kde(in="telemetry.shp", out="kde_M02.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M02'");
kde(in="telemetry.shp", out="kde_F02.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F02'");
kde(in="telemetry.shp", out="kde_J01.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='J01'");
```

Example 2: writing KDE commands for each individual and season (this is a more complicated version of Example 1 that requires nested for loops)

This example assumes there is a season field coded with numbers 1-4 representing 4 different seasons, and we want to calculate different kernel density estimates for each animal in each season. Let's also count the records to ensure there are enough data points, and only run the kde command if there are more than 30.

```
uniqueids <- unique(ANIMALID)
```

```
for (i in 1:length(uniqueids)){
for (j in 1:4){
recs <- which(ANIMALID==uniqueids[i] & SEASON==j)
if (length(recs) > 30) {
cat("kde(in=\"telemetry.shp\", out=\"kde_", uniqueids[i], "seas", j, ".img\", bandwidth=\"SCV\",
cellsize=20, where=\"ANIMALID='", uniqueids[i], "' AND SEASON=", j, "\");\n", sep="")
}}}
```

If the uniquevalues vector was c("M01","F01","M02","F02","J01"), for instance, but that only animal M01 had any data in season 4, that would result in these commands being written to the R window from where they can be copied and pasted into GME and run:

```
kde(in="telemetry.shp", out="kde_M01seas1.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M01' AND SEASON=1");
kde(in="telemetry.shp", out="kde_M01seas2.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M01' AND SEASON=2");
kde(in="telemetry.shp", out="kde_M01seas3.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M01' AND SEASON=3");
kde(in="telemetry.shp", out="kde_M01seas4.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M01' AND SEASON=4");
kde(in="telemetry.shp", out="kde_F01seas1.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F01' AND SEASON=1");
kde(in="telemetry.shp", out="kde_F01seas2.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F01' AND SEASON=2");
kde(in="telemetry.shp", out="kde_F01seas3.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F01' AND SEASON=3");
kde(in="telemetry.shp", out="kde_F01seas4.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F01' AND SEASON=4");
kde(in="telemetry.shp", out="kde_M02seas1.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M02' AND SEASON=1");
kde(in="telemetry.shp", out="kde_M02seas2.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M02' AND SEASON=2");
kde(in="telemetry.shp", out="kde_M02seas3.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M02' AND SEASON=3");
kde(in="telemetry.shp", out="kde_M02seas4.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='M02' AND SEASON=4");
kde(in="telemetry.shp", out="kde_F02seas1.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F02' AND SEASON=1");
kde(in="telemetry.shp", out="kde_F02seas2.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F02' AND SEASON=2");
kde(in="telemetry.shp", out="kde_F02seas3.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F02' AND SEASON=3");
kde(in="telemetry.shp", out="kde_F02seas4.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='F02' AND SEASON=4");
kde(in="telemetry.shp", out="kde_J01seas1.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='J01' AND SEASON=1");
kde(in="telemetry.shp", out="kde_J01seas2.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='J01' AND SEASON=2");
```

```
kde(in="telemetry.shp", out="kde_J01seas3.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='J01' AND SEASON=3");
kde(in="telemetry.shp", out="kde_J01seas4.img", bandwidth="SCV", cellsize=20,
where="ANIMALID='J01' AND SEASON=4");
```

R is a more powerful but complex solution for generating GME commands. It is recommended primarily for people with automation requirements that cannot be readily handled within GME directly.

2.4 Projection definition files

Projection files are simply text files that define a projection and the associated parameters. GME uses these projection definition files to work with projections. This section provides key information on how to work with these files, how to create them, and where to find them. Some GME commands require you to reference a prj file, so it is important to know how to work with these files.

ArcGIS has dozens of pre-defined projections. In ArcGIS 10 they are located in the installation folder in the 'Coordinate Systems' folder, and are organized in sub-folders representing different families and collections of projections. (E.g. on my computer they are here: C:\Program Files\ArcGIS\Desktop10.0\Coordinate Systems). This is one important source for projection files as you can find the projection file corresponding to your projection of interest, and copy it to your working folder. You will probably be familiar with the directory structure because it is the same one that you must navigate when using the projection tools in ArcGIS. For instance, I often work with the UTM projection, and the Zone 17N projection file is found here:
C:\Program Files\ArcGIS\Desktop10.0\Coordinate Systems\Projected Coordinate Systems\UTM\WGS 1984\Northern Hemisphere\WGS 1984 UTM Zone 17N.prj.
You can copy these files (copy, not move!) to other working folders if you wish (it is often more convenient to have them in your working folder when using GME).

Alternatively, if a projection file is associated with a particular dataset, it will have the same name as the dataset and end in a .prj extention. For instance, a shapefile called roads.shp will have a roads.prj file in the same location as the .shp file. If the .prj file is missing this implies that the projection is undefined (it is wise to ensure projections are always defined). Thus, if you know a dataset is in the projection you wish to work with, you can reference that prj file in GME commands. Or you can copy that prj file and rename it with the name of the projection (so that it is easily identifiable in the future).

All of these prj files can be opened in Notepad (or another text editor), although the information is stored in a format that is not easily readable. It is strongly recommended that you never manually edit a prj file.

How can you create custom projection files? Perhaps the easiest way is to use ArcCatalog to create a new shapefile: in ArcCatalog, right-click on a folder and select New, then Shapfile. Give it a name, and set it as point, line or polygon - it does not matter. In the Spatial Reference section, select Edit, and then in the new form that appears New (you have to say Projected or Geographic at this point). Continue through the process of defining your

projection. At the end of this process your new shapefile will have a .prj file associated with it. I suggest you now copy this prj file, and give it a new name that can be easily identified as your custom projection.

2.5 Specifying statistical and empirical distributions

Several of the commands in this interface accept statistical distributions as arguments (e.g. the movement modelling simulation commands). This section describes the distributions that are available, and how to specify them.

There are 13 statistical distributions available, and there is also the ability to specify an empirical distribution. The statistical distributions are all enabled via R:

1. BETA. The R command 'rbeta' is used to generate beta distributed values based on the two shape parameters specified. Syntax: c("BETA", shape1, shape2), e.g. c("BETA", 1.5, 0.2).

2. BINOMIAL. The R command 'rbinom' is used to generate binomially distributed discrete values based on a specified size (number of trials) and probability parameter. Syntax: c("BINOMIAL", size, prob), e.g. c("BINOMIAL", 20, 0.543).

3. CAUCHY. The R command 'rcauchy' is used to generate Cauchy distributed values based on a specified location and scale parameter. Syntax: c("CAUCHY", location, scale), e.g. c("CAUCHY", 0.1, 0.987).

4. EXPONENTIAL. The R command 'rexp' is used to generate exponentially distributed values based on a single rate parameter. Syntax: c("EXPONENTIAL", rate), e.g. c("EXPONENTIAL", 0.25).

5. GAMMA. The R command 'rgamma' is used to generate gamma distributed values based on the two shape parameters specified. Syntax: c("GAMMA", shape1, shape2), e.g. c("GAMMA", 0.001, 0.001).

6. GEOMETRIC. The R command 'rgeom' is used to generate geometric distributed discrete values based on a single probability parameter. Syntax: c("GEOMETRIC", prob), e.g. c("GEOMETRIC", 0.25).

7. LOGNORMAL. The R command 'rlnorm' is used to generate log normally distributed values based on a specified mean and standard deviation. Syntax: c("LOGNORMAL", log mean, log standard deviation), e.g. c("LOGNORMAL", 1.234, 0.987).

8. NEGBINOMIAL. The R command 'rnbinom' is used to generate negative binomially distributed discrete values based on a specified size (number of trials) and probability parameter. Syntax: c("NEGBINOMIAL", size, prob), e.g. c("NEGBINOMIAL", 20, 0.543).

9. NORMAL. The R command 'rnorm' is used to generate normally (Gaussian) distributed values based on a specified mean and standard deviation. Syntax:

c("NORMAL", mean, standard deviation), e.g. c("NORMAL", 0, 1) or c("NORMAL", 53.234, 3.678).

10. POISSON. The R command 'rpois' is used to generate Poisson distributed discrete values based on a single parameter lambda. Syntax: c("POISSON", lambda), e.g. c("POISSON", 5).

11. UNIFORM. The R command 'runif' is used to generate a uniform distribution of values between a specified minimum and maximum value. Syntax: c("UNIFORM", minimum, maximum), e.g. c("UNIFORM", 0, 1) or c("UNIFORM", 0, 3124.232).

12. WEIBULL. The R command 'rweibull' is used to generate Weibull distributed values based on the specified shape and scale parameters. Syntax: c("WEIBULL", shape, scale), e.g. c("WEIBULL", 1.234, 2.345).

13. WRAPPEDCAUCHY. The R command 'rwrpcauchy' in the CircStats package is used to generate Wrapped Cauchy distributed values based on a specified location and rho parameter. Syntax: c("WRAPPEDCAUCHY", location, rho), e.g. c("WRAPPEDCAUCHY", 0.1, 0.987).

Alternatively an empirical distribution can be used to represent any other distribution (with a small margin of error). Empirical distributions are represented as frequency tables in a comma delimited format. Each row in the table contains a minimum value, a maximum value, and the frequency of values in that interval in the empirical distribution (this frequency can either be expressed as a percentage, a count, or a probability). Draws are made from an empirical distribution using a rejection algorithm: an interval is randomly selected from the table (all intervals have equal probability of selection), a random number from a uniform distribution is generated and used to determine if a value from that interval will be generated or not (if the random number is less than or equal to the specified probability value then a value is generated), and finally a value is generated from the interval by drawing from a uniform distribution using the specified minimum and maximum values for that interval.

The smaller the intervals for each of the bins in the frequency table, the closer the empirical distribution will match the real distribution). Note that the intervals do not have to be constant among all bins. In portions of the distribution where the density is effectively constant (a flat portion of the density curve) then a wide interval can be used to represent that portion of the distribution. Conversely, where the density changes rapidly (the steeper portions of the density curve) then the bins should be narrow.

The empirical distribution should be written to a comma delimited text file with no header row (no column headings or labels), and three columns containing the minimum, maximum and frequency values respectively. For example, this is an empirical distribution of turn angles for a movement path:

-180,-160,0.048623563
-160,-140,0.04777977
-140,-120,0.043349858
-120,-100,0.049256408
-100,-80,0.052104208

-80,-60,0.05600675
-60,-40,0.063917308
-40,-20,0.068980065
-20,0,0.080898639
0,20,0.078894631
20,40,0.07319903
40,60,0.062440671
60,80,0.054846535
80,100,0.042506065
100,120,0.042400591
120,140,0.044615547
140,160,0.042506065
160,180,0.047674296

In this example the frequency column contains probabilities, and the probabilities sum to 1 across all rows. Note that if you provide counts or percentages, the program will automatically convert this to probabilities.

2.6 Using GME with Python (and ArcToolbox tools)

GME is not formally integrated with ArcToolbox, but does allow integration with ArcToolbox / Model Builder models via Python. Although Python functionality is currently rather basic, I do plan to enable Python users to call GME commands via COM. However, for now what I offer is the ability to call GME commands via the command prompt (issuing a system command from Python).

To call GME commands from Python you need to issue a system command to the GME executable SEGME.exe, with the command(s) you wish to execute included as a command line argument. If you want to call a series of commands then it is recommended that you write those to a text file (with a semicolon to end each line), and call SEGME.exe with the 'run' command, referencing that text file.

If you want the GME interface to close when it has finished processing those commands then include the switch -c as the first argument (see examples below). This would allow you to call GME repeatedly, for instance in a loop, without command windows building up with every call to SEGME.

You must first load the subprocess Python library (using: import subprocess as subp) at the beginning of your Python session. To call GME from the command line you use the following generic syntax:
os.system(r'path\SEGME.exe commands')
where path is the full directory path to the SEGME command, usually:
subp.call(r'C:\Program Files\SpatialEcology\GME\SEGME.exe commands')
and commands is the list of commands you wish to execute. Note that the r at the beginning of that path name is not a mistake: it tells Python to interpret the following text literally. Please note that you need to add a backslash before all quote marks in the commands text, otherwise it is the same syntax as GME. For instance, here is a Python call to GME that

tells it to run the GME script called "E:\data\gmescript.txt":
subp.call(r'C:\Program Files\SpatialEcology\GME\SEGME.exe -c
run(in=\"E:\data\gmescript.txt\");');
(This command has been displayed on two lines here because it is too long for a single line, though in Python it would be a single line). Note that by including -c directly after SEGME.exe, this will force the GME interface to close when it has finished running the specified commands.

3 COMMAND REFERENCE

3.1 Strategic Commands

Strategic commands

There are 5 important features of GME that I strongly recommend you learn about before using GME. **They will change the way you use GME, design geospatial analyses, organise your data, and share analysis ideas with others.** They are fundamental tools for efficient data analysis. If you use R, no doubt you will recognize the value of these features already. While some of them can be used individually, exploiting the power of GME involves using them together. The few minutes it takes you to read this section is time well spent. I summarize the features here, but more detailed syntax help is provided for each one in the Command Reference section.

1. **For loops**. These are a basic programming construct that allows you to repeat one or more commands many times. An "index variable" (usually a single letter) is used to control the number of iterations. Importantly, this index variable can be used within the block of commands in the loop to alter the arguments to the commands. For loops can be nested. An example is provided below.

2. **Variable assignment for simple data types**. This allows you to define your own variables, and set them to equal a simple data type: text (strings), integers, double precision numbers, or one dimensional vectors of strings or numbers. This feature enables you to write easily re-usable scripts: you only need to set the variables once at the beginning of your script, and then you can refer to the variable name in all subsequent commands. One obvious way that this is useful is if you want to run the same set of commands on similar datasets in many folders, or on different drives or computers - you only need to change the path variable once rather than search and replace the incorrect path in every single command. (Also see the ls() command when using variable assignment).

3. **r.eval() function**. This function evaluates R code, retrieves the output of that code, and embeds it into the GME command string. This provides you with a way of linking the phenomenal power of R with GME. Sometimes the r.eval() functions is simply a matter of convenience , for instance, rather than typing a 50 value sequence by hand, e.g. c(0, 0.02, 0.04 ... 0.98, 1), we could type r.eval(seq(0,1,0.02)). More importantly, it allows you to access much more powerful R code to parameterize your GME commands. For instance, you might use the optimisation commands in R to generate a maximum likelihood parameter value.

4. **paste() function**. This function is similar to the R paste function. It allows you to construct strings (text) dynamically. This is particularly important within for loops. For instance, assuming we have set the variable called path (path <- "C:\data\rasters\"), and we have written a for loop as for(i in 1:3){...}, then this paste function: paste(path, "tmband", i, ".tif") evaluates to: "C:\data\rasters\tmband1.tif" in the first loop. A more complete example is provided below.

5. **The "where" clause**. This allows you to specify a subset of records/vector features to process based on a simple logical statement (an SQL statement). This makes data management easier (you don't have to split datasets apart to run analyses), and more importantly

it provides you with an important mechanism for repeating an analysis on different subsets of data.

Example

Here is an example that brings all these elements together. Imagine you wish to create kernel density estimates for each individual in each month, based on a database of GPS telemetry locations. In this example we assume there are 10 animals with ID numbers 1001-1010, and we want to process points for each month: inpath <- "C:\data\"

```
outpath <- "C:\output\"
for (i in 1001:1010) {
for (j in 1:12) {
kde(in=paste(inpath, "telemetry.shp"), out=paste(outpath, "kde_an", i, "_m", j, ".img"),
bandwidth=1000, cellsize=r.eval((250000-120000)/2000), where=paste("ANIMALID=", i, " AND
MONTH=", j));
};
};
```

These few lines of code generate 120 kde commands. How does this work? There are two loops: one loop that corresponds to the animal ID numbers as is identified with the variable 'i', and a second nested loop corresponding to the month with variable 'j'. Note that there are three 'paste' commands embedded within the kde command, two of which use the two index variables i and j. The r.eval() statement is simplistic and contrived, but does illustrate how it can be used. In the first iteration of the loop, i=1001 and j=1, and the resulting command becomes: kde(in="C:\data\telemetry.shp", out="C:\output\kde_an1001_m1.img",

```
bandwidth=1250, cellsize=65, where="ANIMALID=1001 AND MONTH=1");
```

The second iteration of the loop would result in: kde(in="C:\data\telemetry.shp",

```
out="C:\output\kde_an1001_m2.img", bandwidth=1250, cellsize=65, where="ANIMALID=1001
AND MONTH=2");
```

and so on, until the final iteration, which yields: kde(in="C:\data\telemetry.shp",

```
out="C:\output\kde_an1010_m12.img", bandwidth=1250, cellsize=65, where="ANIMALID=1010
AND MONTH=12");
```

Do you want even more power to construct GME scripts? Consider using R (or Python, or any other programming/scripting language) directly to write GME commands to a text file, and run them using the 'run' command. For instance, R has a vast array of commands that

can be used to manipulate and create text. **Useful tip**. There is one other feature of GME that is very useful if you are using any of the above strategic commands: the setparameter command (see syntax help in Command Reference section) can be used to tell GME to interpret command text but not to run it. The interpreted commands are displayed in the output window, so you can check for errors in your script.

3.2 access.summary

Access Database Summary: Creates a simple summary description of an Access database (number of tables, number of columns and records in each table, a list of column names, etc).

Description

This tool creates a simple summary description of an Access database (number of tables, number of columns and records in each table, a list of column names, etc). It is intended as a tool to facilitate data exploration and report generation.

Syntax

```
access.summary(file);
 file   the full path to the input database file (*.mdb)
```

Example

```
access.summary(file="C:\data\mydata.mdb");

Database: C:\data\mydata.mdb
Table count: 5

Table: Aquatic Ecosystem Health
Record count: 12
Field count: 9
Fields: Contact, Email, FileName, Group, ID, LayerName, MapName, Metadata, SubGroup

Table: Base Maps
Record count: 80
Field count: 9
Fields: Contact, Email, FileName, Group, ID, LayerName, MapName, Metadata, SubGroup

Table: Groups
Record count: 3
Field count: 2
Fields: Group Name, ID

Table: Land Use and Development
Record count: 48
Field count: 9
Fields: Contact, Email, FileName, Group, ID, LayerName, MapName, Metadata, SubGroup

Table: MapFilesInfo
Record count: 185
Field count: 10
```

Fields: Contact, Email, FileName, Group, ID, LayerName, MapName, Metadata, SubGroup, Visibility

Warning: Table names with spaces were detected. It is strongly recommended that you avoid the use of spaces in table names.

Warning: Column names with spaces were detected. It is strongly recommended that you avoid the use of spaces in column names.

3.3 addarea

Add Area And Perimeter Fields To Table: Adds a new area and/or permeter field to a polygon data source.

Description

This command adds a new area and/or perimeter field to a polygon data source. The tool can also perform on-the-fly unit conversion if the coordinate system of the data source is defined. The default is to write the area or length values in the coordinate system units (e.g. meters for UTM data). This default option does not require that a coordinate system is defined. The unit conversions available are area unit conversions between meters, feet, hectares, acres, kilometers and miles, and the length unit conversions available are between meters, kilometers, miles and feet.

It is not appropriate to use this tool with data in a geographic coordinate system (spherical coordinates require different algorithms to calculate areas and distances). If the tool detects a geographic coordinate system it will raise an error and will not process the command. However, if the coordinate system is not defined the tool is unable to determine whether it is a geographic coordinate system or not, so will process the command even though it creates nonsensical results. It is highly recommended that you reproject your data if you wish to use any of these tools (none of them are designed to accommodate spherical coordinates).

Note that if the area/perimeter field(s) specified already exist then an error will be returned unless you have specified the update=TRUE option.

See also: field.delete

Syntax

addarea(in, [area], [perim], [areaunits], [perimunits], [update], [where]);

in	the input polygon data source
[area]	the name of the area field to add or replace
[perim]	the name of the perimeter length field to add or replace
[areaunits]	the area units (see help for details); the coordinate system must be defined to use this option (default=csu (coordinate system units); options: csu, m^2, km^2, mi^2, ft^2, hect, acre)
[perimunits]	the perimeter length units (see help for details); the coordinate system must be defined to use this option (default=csu (coordinate system units); options: csu, m, km, mi, ft)
[update]	(TRUE/FALSE) if TRUE and you specify an existing field the existing field will be updated rather than generating an error message (default=FALSE); warning: this option will result in overwriting of existing data and is therefore potentially dangerous.
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

addarea(in="C:\data\lakes.shp", area="AREA");
addarea(in="C:\data\lakes.shp", area="ACRES", perim="PERIM_KM",
areaunits="acre", perimunits="km", where="COUNTY='WOOD' AND MONTH=7",
update=TRUE);

3.4 addcodedfield

Add Coded Field To Table: Adds a new field to a table and automatically codes it with a specified pattern of values.

Description

This command adds a new field to a table and automatically codes it with a specified pattern of values. The options for value generation are: i) constant: a constant value or text string is applied to all records, ii) sequence: the tool populates records starting with a specified initial value, incrementing it by the specified increment value after each record, iii) repeat: the tool draws from a sequence of values, in order and looping back to the beginning when the end is reached, until all records have been populated, iv) normal: populates records with random draws from a normal distribution of a specified mean and standard deviation, v) uniform: populates records with random draws from a uniform distribution of a specified minimum and maximum.

The code will attempt to coerce the values it generates into the field type specified in the command. An appropriate field type must therefore be specified. Use the SHORT or LONG

field types when generating integers (use LONG if the values exceed about +/- 32000), and use the DOUBLE field type for real numbers. Although the STRING field type can be used when generating any numeric values, it is inefficient to store numbers as strings and truncation of the values may occur. It is recommended that the STRING option only be used with the 'constant' option, where the constant provided is a text string.

The parameters can be combined in various ways to achieve a greater variety of functions. For instance, if you specify the field type as LONG, and use the sequence option with a start value of 1 and an increment value of 0.2, then the records are coded: 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, etc. Thus, the truncation that results from attempting to write a decimal point value to an integer field can be used to your advantage.

Note that if the specified field already exists then an error will be returned unless you have specified the update=TRUE option.

See also: field.delete

Syntax

addcodedfield(in, field, fieldtype, [constant], [sequence], [repeat], [normal], [uniform], [update], [where]);

in	the input feature/table data source
field	the name of the field to add or replace
fieldtype	the field data type - short integer, long integer, double precision real number, or string (options: SHORT, LONG, DOUBLE, STRING)
[constant]	codes the new field with the specified constant value (a single value expected), e.g.: 3.14 or "RIVER"
[sequence]	codes the new field based on a start value and increment value (a start value and increment value expected), e.g.: c(1000, 1)
[repeat]	codes the new field by cycling through the specified list of values (a list of values expected), e.g. c(1,2,3,4,5)
[normal]	codes the new field by drawing random values from a normal distribution with a specified mean and standard deviation (a mean and standard deviation expected), e.g. c(100,5)
[uniform]	codes the new field by drawing random values from a uniform distribution with a specified minimum and maximum (a minimum and maximum value expected), e.g. c(0,100)
[update]	(TRUE/FALSE) if TRUE and you specify an existing field, the existing field will be updated rather than generating an error message (default=FALSE); warning: this option will result in overwriting of existing data and is therefore potentially dangerous.
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
addcodedfield(in="c:\data\pnts.shp", field="NEWCONST", fieldtype="SHORT",
constant=100);
addcodedfield(in="c:\data\pnts.shp", field="MARKER", fieldtype="STRING",
constant="SIGNS");
addcodedfield(in="c:\data\pnts.shp", field="MyUID", fieldtype="LONG",
sequence=c(1000, 1));
addcodedfield(in="c:\data\pnts.shp", field="MyGROUP", fieldtype="LONG",
repeat=c(1, 2, 3, 4, 5));
addcodedfield(in="c:\data\pnts.shp", field="RAMP", fieldtype="DOUBLE",
sequence=c(0, 0.01))
addcodedfield(in="c:\data\pnts.shp", field="NORMVALS", fieldtype="DOUBLE",
normal=c(100, 3.6));
addcodedfield(in="c:\data\pnts.shp", field="UNIFVALS", fieldtype="DOUBLE",
uniform=c(0, 100), where="COUNTY='WOOD' AND MONTH=7", update=TRUE);
```

3.5 addlength

Add Length Field To Table: Adds a new length field to a polyline data source.

Description

This command adds a new length field to a polyline data source. The tool can also perform on-the-fly unit conversion if the coordinate system of the data source is defined. The default is to write the length values in the coordinate system units (e.g. meters for UTM data). This default option does not require that a coordinate system is defined. The unit conversions available are between meters, kilometers, miles and feet.

It is not appropriate to use this tool with data in a geographic coordinate system (spherical coordinates require different algorithms to calculate distances and lengths). If the tool detects a geographic coordinate system it will raise an error and will not process the command. However, if the coordinate system is not defined the tool is unable to determine whether it is a geographic coordinate system or not, so will process the command even though it creates nonsensical results. It is highly recommended that you reproject your data if you wish to use any of these tools (none of them are designed to accommodate spherical coordinates).

Note that if the length field specified already exists then an error will be returned unless you have specified the update=TRUE option.

See also: field.delete

Syntax

addlength(in, field, [units], [update], [where]);

in	the input line data source
field	the name of the length field to add or replace
[units]	the length units (see help for details); the coordinate system must be defined to use this option (default=csu (coordinate system units); options: csu, m, km, mi, ft)
[update]	(TRUE/FALSE) if TRUE and you specify an existing field, the existing field will be updated rather than generating an error message (default=FALSE); warning: this option will result in overwriting of existing data and is therefore potentially dangerous.
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

addlength(in="C:\data\roads.shp", field="LTEST1");
addlength(in="C:\data\roads.shp", field="LTEST2", units="km",
where="COUNTY='WOOD' AND MONTH=7", update=TRUE);

3.6 addxy

Add XY Coordinates To Table: Adds new x and y coordinate fields to a feature data source table (points, lines, polygons).

Description

This command adds new x and y coordinate fields to a feature data source table. The coordinates added to the table depend on the type of input feature (points, lines or polygons) and the options specified by the user.

For points the user specifies the input data source, and optionally a prefix that is used to name the fields (default field names are X and Y, and if a prefix is specified then X and Y are added to the prefix). There are no options for point data sources (the simplest case).

If the input is a polygon dataset the user can specify the 'env' option, and/or the 'centroid' option, and/or the 'label' option. The 'env' option adds a series of fields (see below) based on the rectangular envelope that contains the polygon. Note that in the case of multipart features (polygons consisting of multiple parts that may be disjoint in space) then the envelope will be the rectangle that contains all of the parts. The centroid option add the point that is the centre of gravity of the shape (although the algorithm used to publish this is not clear, so this statement should not be taken too literally). The centre of gravity of the shape is not guaranteed to fall inside the bounds of the polygon, and in the case of multipart features is unlikely to do so. The 'label' option adds the coordinates of what ESRI defines a label point:

a point that is guaranteed to fall inside the polygon somewhere. The algorithm used to make this calculation is not published so it is not clear how this location is determined.

If the input is a line (polyline) dataset the user can specify the 'env' and/or 'ends' options. The 'env' option is described in the previous paragraph. The 'ends' option adds the coordinates of the start and end points of the line.

Note that the default field names (see below) have been devised so that if you specify multiple options (e.g. the centroid and label options in the case of a polygon data source) then there is no naming conflict, even when a prefix is specified.

In all cases, if a prefix is specified the prefix will precede all the default field names described below. Prefix names should not contain special characters and should be short (6 characters or less - longer prefix names will raise an error).

Note that if the fields specified already exist then the values in those fields are overwritten.

See also: field.delete

Syntax

addxy(in, [env], [centroid], [label], [ends], [prefix]);

in	the input feature data source
[env]	(TRUE/FALSE) (applies to lines and polygons only) adds the min x, max x, min y, max y, center x, center y of the rectangular envelope bounding the feature (default=FALSE)
[centroid]	(TRUE/FALSE) (applies to polygons only) adds the coordinates of the centroid of the polygon (default=FALSE)
[label]	(TRUE/FALSE) (applies to polygons only) adds the coordinates of the label point of the polygon (default=TRUE)
[ends]	(TRUE/FALSE) (applies to lines only) adds the coordinates of the start and end points of the line (default=TRUE)
[prefix]	the prefix to be applied to the default field names (should not exceed 6 characters - see help for further details)

Example

(points): addxy(in="C:\data\wellsites.shp", prefix="COORD");
(polygons): addxy(in="C:\data\myGDB!parcels", env=TRUE, label=TRUE);
(lines): addxy(in="C:\data\roads.shp", env=TRUE, ends=TRUE);

3.7 buffer

Buffer Features: Buffers features using a constant or field based buffer distances, and optionally copies attributes.

Description

This command buffers features using a constant buffer distance, or using buffer distances specified in a field in the feature attribute table. The tool will also optionally copy the attributes from the input attribute table to the output attribute table.

The tool can also perform on-the-fly unit conversion if the coordinate system of the data source is defined. The default is to assume the buffer distance is specified in the same units as the coordinate system (e.g. meters for UTM data). This default option does not require that a coordinate system is defined. The unit conversions available are between meters, kilometers, miles and feet.

If the input data source is polygons, then you can use a negative buffer distance to create an inside buffer. If this results in an empty geometry (because the input polygon is too small to create an inside buffer) then no record is written to the output.

If the input features are polygons and the buffer distance is positive, then there is an additional option (the 'donut' parameter) to only output the portion of the buffer occurring outside of the original polygon. The donut parameter is ignored for all geometries other than polygons, and for zero or negative buffer distances.

Note that if you copy fields to the output table, the values may not appropriately reflect the properties of the buffered polygon. An obvious example is an AREA field: the value that is copied to the output file will reflect the area of the original polygon, not the buffered polygon. It is therefore important to delete or update fields that are copied but do not correctly reflect the buffered polygon.

Syntax

buffer(in, out, distance, [units], [copyfields], [donut], [where]);

in	the input feature data source
out	the output polygon data source
distance	the buffer distance value, or a field name in the input layer that contains these values
[units]	the units of the buffer distance (see help for details); the coordinate system must be defined to use this option (default=csu (coordinate system units); options: csu, m, km, mi, ft)
[copyfields]	(TRUE/FALSE) if TRUE the attribute fields from the input dataset are copied to the output dataset (default=FALSE)
[donut]	(TRUE/FALSE) if TRUE only the portion of the buffer occurring outside of the buffered polygon is written to the output dataset (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
buffer(in="C:\data\wells.shp", out="C:\data\wellbuff1km.shp", distance=1000);
buffer(in="C:\data\wells.shp", out="C:\data\wellbuffs.shp", distance="BUFFDIST");
```

buffer(in="C:\data\wells.shp", out="C:\data\wellbuff1km.shp", distance=1, units="km", copyfields=TRUE);
buffer(in="C:\data\lakes.shp", out="C:\data\lakeshore.shp", distance=100, donut=TRUE, copyfields=TRUE);

3.8 calc.sharedborders

Calculate Shared Polygon Borders: Calculates the shared border line between two adjacent (or approximately adjacent) polygons, and writes the polyline and unique polygon ID's of the polygons to the output file.

Description

This tool identifies the shared border between two adjacent polygons. This is not as straightforward as it sounds. Few polygon datasets explicitly model topology, so each polygon is entirely independent of its neighbours. Hence polygons in, for instance, a shapefile format can overlap each other and often have imperfectly shared borders even if you intend them to be identical (the snapping settings help to resolve this to some extent).

This tool takes each polygon, buffers it by the tolerance you specify, then finds all other polygons that overlap that buffer, and for each one clips it with the buffered polygon to identify the 'shared border' between the polygons. Note that the tolerance and buffering is needed to resolve the problem of polygons that are very close together but not quite touching. You must determine a tolerance value that is large enough to resolve this gap issue, but not so large that the tool identifies spurious shared borders.

Note that a shared border is recorded only once, even though the shared border may be slightly different depending upon which polygon the program encounters first. This is an unfortunate characteristic, but one that arises from the fact that the polygon boundaries themselves are imprecise.

Syntax

calc.sharedborders(in, uidfield, out, tol);

in	the input polygon data source
uidfield	the unique polygon ID field
out	the output line data source
tol	the tolerance distance in coordinate system units for including non-overlapping boundary lines

Example

calc.sharedborders(in="C:\data\counties.shp", uidfield="CNTYID", out="C:\data\cnty_borders.shp", tol=500);

3.9 citation

Citation: Recommends a citation for GME and R.

Description

This command reports the recommended citation for GME and R, specific to the version you have installed.

Syntax

citation);

3.10 clipraster

Clip Raster: Clips an input raster using a reference data source to define the clip boundary or using user defined coordinates defining a clip rectangle.

Description

This tool clips an input raster using a reference data source to define the clip boundary or using user defined coordinates defining a clip rectangle. The clip data source can be a point, polyline or polygon feature data source, or a raster data source. By default the raster is clipped to the rectangle that bounds all of the features in the dataset, or to the boundary of the reference raster layer of the clip layer is a raster. However, if the clip layer contains polygons then you may use the 'extent' option to clip the raster to the bounds of the polygons themselves (this is not the default option - use the extent=FALSE option to clip to polygon boundaries). Clipping to polygon boundaries is much slower than clipping using the extent option.

This tool is designed to work with these three raster formats: grids, TIFF/GeoTIFF, and ERDAS Imagine rasters. Note that not all raster formats support all datatypes. When you are clipping a raster it is recommended you consider two strategies to avoid these pixel data type problems: 1) ensure the output format matches the input format, or 2) always use the Imagine img format as the output format as this supports all the data types. The output format is specified by adding the appropriate file extension to the file name. No extension is interpreted as the grid format, the '.tif' extension is the GeoTIFF format, and the '.img' extension is the Imagine format.

Note that all clips will preserve the cell alignment of the input raster (no shifting of pixels will occur at all). However, the display properties of the input raster are not transferred to the output raster, so if you are clipping digital photos or satellite images you should expect the appearance of the clipped images to differ from that of the original image.

Syntax

clipraster(raster, clip, out, [extent], [where]);

raster	the input raster data source
clip	the reference layer to clip to (a vector or raster layer), or the coordinates of the rectangle to clip to (min x, max x, min y, max y)
out	the output raster data source
[extent]	(TRUE/FALSE) if TRUE, results in a clip to the rectangular envelope containing the input features, whereas if FALSE clips the raster to the polygon boundaries (default=TRUE); if the clip layer is a point or line vector layer, or a raster layer, the clip is always to the rectangular envelope
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

clipraster(raster="C:\data\landcov.img", clip="C:\data\fields.shp",
out="C:\data\lc_clip.img");
clipraster(raster="C:\data\landcov.img", clip=c(428000, 436000, 5960000, 5970000),
out="C:\data\lc_clip.img");

3.11 cliprasterbypolys

Clip Raster By Polygons: Clips an input raster to each polygon in a polygon data source resulting in one new raster per polygon.

Description

This tool clips an input raster using the polygons in a polygon data source. The command cycles through each polygon, clips the raster if there is overlap, and writes one new raster image per polygon. The extent of the new raster is the intersection of the extent of the polygon and the extent of the raster.

This tool is designed to work with these three raster formats: grids, TIFF/GeoTIFF, and ERDAS Imagine rasters. Note that not all raster formats support all datatypes. When you are clipping a raster it is recommended you consider two strategies to avoid these pixel data type problems: 1) ensure the output format matches the input format, or 2) always use the Imagine img format as the output format as this supports all the data types. The output format is specified by adding the appropriate file extension to the file name. No extension is interpreted as the grid format, the '.tif' extension is the GeoTIFF format, and the '.img' extension is the Imagine format.

Note that all clips will preserve the cell alignment of the input raster (no shifting of pixels will occur at all). However, the display properties of the input raster are not transferred to

the output raster, so if you are clipping digital photos or satellite images you should expect the appearance of the clipped images to differ from that of the original image.

Syntax

cliprasterbypolys(raster, poly, uidfield, out, [format], [prefix], [where]);

raster	the input raster data source
poly	the input polygon data source
uidfield	the unique ID field of the input feature data source
out	the output folder to which the raster clips will be written
[format]	the output raster format (default=GRID; options: GRID, TIF, IMG)
[prefix]	the prefix to be used to name the clipped rasters, and to which the unique ID value of the polygon is appended as the suffix; note that for Grids the name cannot exceed 14 characters (default="clip")
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

cliprasterbypolys(raster="C:\data\landcov.img", poly="c:\data\fields.shp", uidfield="FLDID", out="C:\data\lc_clips", format="IMG");
cliprasterbypolys(raster="C:\data\landcov.img", poly="c:\data\fields.shp", uidfield="FLDID", out="C:\data\lc_clips", prefix="FLD_", format="GRID", where="FLDID < 100");

3.12 contour

Contour: Creates contours based on continuous raster data.

Description

This tool creates contours (lines) based on a raster dataset representing continuous data (e.g. a digital elevation model), and a set of user-defined levels representing the z-values at which to generate the contours. The output is a line (polyline) dataset, with the z-values for each contour line recorded in the attribute table.

This algorithm produces high-quality contour lines at the expense of processing time. It interprets the raster as a triangular regular network and performs a plane intersection to determine the paths of the contour lines. It does not perform any sort of smoothing on the line, which I have reservations about. I have not been able to test this algorithm widely, so if you are able to find degenerate examples where the algorithm performs poorly, I would be grateful to hear about them so that the algorithm can be improved.

Although a 'band' option is included so that users can identify which band to process in the case of multiband images, it is likely that this tool will most often be run with continuous, single-band raster data (like DEM's).

Syntax

```
contour(in, out, levels, [band]);
 in       the input integer raster data source
 out      the output line data source
 levels   the z-values at which to generate contours. e.g. 100 or c(10,20,30)
 [band]   the input band (default=1)
```

Example

```
contour(in="C:\data\dem", out="C:\data\contours.shp",
levels=c(1000,2000,3000,4000,5000));
contour(in="C:\data\dem", out="C:\data\contours.shp", levels=0.05);
contour(in="C:\data\dem", out="C:\data\contours.shp",
levels=r.eval(seq(100,10000,100)));
```

3.13 convert.linestopoints

Convert Lines To Points: Converts lines (polylines) to points by extracting the vertices of the polylines.

Description

This tool converts the lines (polylines) in a polyline data source to points in a new or existing point data source by extracting all of the vertices that define the polyline. This works with multipart polylines. The unique polygon ID is written to the point attribute table. Note that the resulting points will overlap where two line vertices have the same coordinates (this algorithm does not simplify the resulting point data to remove points where there is precise overlap).

Syntax

```
convert.linestopoints(in, uidfield, out, [where]);
 in         the input line data source
 uidfield   the unique ID field of the input feature data source
 out        the output point data source
 [where]    the selection statement that will be applied to the feature data source to
            identify a subset of features to process (see full Help documentation for
            further details)
```

Example

```
convert.linestopoints(in="C:\data\rivers.shp", uidfield="RIVERID",
out="C:\data\riverpoints.shp");
convert.linestopoints(in="C:\data\roads.shp", uidfield="RDID",
out="C:\data\rdpoints.shp");
```

3.14 convert.pointstolines

Convert Points To Lines: Converts points to lines (polylines).

Description

This tool converts the points in a point data source to lines (polylines) in a new or existing line data source. An integer unique ID field in the point attribute table is required to distinguish between different output lines, and another integer field is required that defines the order in which points will be connected. This order field does not have to start at 1 for each line, so if your point attribute table is already ordered you can either use the FID field, or better yet add a suitable order field using the addcodedfield command.

The split option controls whether a single polyline is written for each unique ID (the default), or whether each line segment connecting a pair of sequential points is written as a separate line in the output data source.

Syntax

convert.pointstolines(in, uidfield, orderfield, out, [split], [where]);

in	the input point data source
uidfield	the unique ID field in the point data source that identifies collections of points that make up each line)
orderfield	the point attribute field that can be used to sort the points into the correct order (must be numeric or date, but not a date stored as a text field)
out	the output line data source
[split]	(TRUE/FALSE) if TRUE, writes each line segment connecting a pair of sequential points as a separate line (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

convert.pointstolines(in="C:\data\telemetry.shp", uidfield="ANID", orderfield="STEP", split=TRUE, out="C:\data\anpaths.shp");
convert.pointstolines(in="C:\data\samplelocs.shp", uidfield="SAMPID", orderfield="ORDER", out="C:\data\transects.shp");

3.15 convert.pointstopolygons

Convert Points To Polygons: Converts an ordered seqence of points to polygons.

Description

This tool converts the points in a point data source to polygons in a new or existing line data source. An integer unique ID field in the point attribute table is required to distinguish between different output polygons, and another integer field is required that defines the order in which points will be connected. This order field does not have to start at 1 for each polygon, so if your point attribute table is already ordered you can either use the FID field, or better yet add a suitable order field using the addcodedfield command. The last point in the sequence does not have to be the same as the first point (this tool will attempt to close the polygon by connecting the first and last points).

Syntax

convert.pointstopolygons(in, uidfield, orderfield, out, [where]);

in	the input point data source
uidfield	the unique ID field in the point data source that identifies collections of points that make up each line)
orderfield	the point attribute field that can be used to sort the points into the correct order (must be numeric or date, but not a date stored as a text field)
out	the output line data source
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
convert.pointstopolygons(in="C:\data\plotcorners.shp", uidfield="PLOTID",
orderfield="SEQ", out="C:\data\plotpolygons.shp");
convert.pointstopolygons(in="C:\data\boundarylocs.shp", uidfield="BNDID",
orderfield="ORDER", out="C:\data\burns.shp");
```

3.16 convert.polygonstolines

Convert Polygons To Lines: Converts polygons to lines (polylines).

Description

This tool converts the polygons in a polygon data source to lines (polylines) in a new or existing line data source. This works with multipart polygons, and with any inner rings or holes in the polygon: all the component lines that make up the polygon are included in the output line. The unique polygon ID is written to the line attribute table. Note that the resulting lines will overlap where two polygons share the same boundary (this algorithm does not simplify the resulting line data to remove segments from different polygons where there is precise overlap).

Syntax

convert.polygonstolines(in, uidfield, out, [where]);

in	the input polygon data source
uidfield	the unique ID field of the input feature data source
out	the output line data source
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

convert.polygonstolines(in="C:\data\plots.shp", uidfield="PLOTID",
out="C:\data\bndlines.shp");
convert.polygonstolines(in="C:\data\parcels.shp", uidfield="PARCELID",
out="C:\data\bndlines.shp");

3.17 convert.polygonstopoints

Convert Polygons To Points: Converts polygons to points by extracting the vertices of the polygons.

Description

This tool converts the polygons in a polygon data source to points in a new or existing point data source by extracting all of the vertices that define the polygon. This works with multipart polygons, and with any inner rings or holes in the polygon: all the vertices are extracted. The unique polygon ID is written to the point attribute table. Note that the resulting points will overlap where two polygons share the same boundary (this algorithm does not simplify the resulting point data to remove points from different polygons where there is precise overlap).

Syntax

convert.polygonstopoints(in, uidfield, out, [where]);

in	the input polygon data source
uidfield	the unique ID field of the input feature data source
out	the output point data source
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

convert.polygonstopoints(in="C:\data\plots.shp", uidfield="PLOTID",
out="C:\data\bndpoints.shp");

convert.polygonstopoints(in="C:\data\parcels.shp", uidfield="PARCELID", out="C:\data\bndpoints.shp");

3.18 convert.polygonstoraster

Converts Polygons to Raster: Converts a polygon dataset to raster format.

Description

This command converts a polygon dataset to raster format based on a user-specified field in the attribute table of the polygon dataset. This command works on both numeric and string (text) fields. If the field type is numeric, the output raster pixel type will be matched to the field type. If the field type is string, then the output raster will be integers representing unique values based on the string field. The string values will be written to the attribute table of the raster so that you may easily relate the raster integers to the corresponding string.

The output raster is assigned the same spatial reference as the input polygon layer. The cellsize must be specified in coordinate system units (e.g. meters for UTM).

Note that if the field data type is double precision, the output format is forced to be Imagine Image (.img) format as GeoTIFF and Grid formats do not support double precision values.

Syntax

convert.polygonstoraster(in, field, out, cellsize);

in	the input polygon data source
field	the field in the polygon data source upon which the new raster values will be based
out	the output raster dataset to create
cellsize	the cell size resolution in coordinate system units

Example

convert.polygonstoraster(in="C:\data\soils.shp", field="SOILTYPE", out="C:\data\soiltype.img", cellsize=50);

3.19 convert.tabletolines

Convert Table To Lines: Creates lines from pairs of x, y coordinates in a table.

Description

This tool converts pairs of xy coordinates in a table into to lines (polylines) in a new or existing line data source. The unique ID field of the input table is required so that you can join the line attribute table back to the source table.

This command is equivalent to the 'Add XY Line Data' tool in Hawthstools.

Syntax

convert.tabletolines(in, uidfield, fromx, fromy, tox, toy, out, [where]);

in	the input table data source
uidfield	the unique ID field of the input data source
fromx	the name of the field containing the from-x coordinate
fromy	the name of the field containing the from-y coordinate
tox	the name of the field containing the to-x coordinate
toy	the name of the field containing the to-y coordinate
out	the output line data source
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

convert.tabletolines(in="C:\data\coords.dbf", uidfield="LINEID", fromx="FROMX", fromy="FROMY", tox="TOX", toy="TOY", out="C:\data\lines.shp");
convert.tabletolines(in="C:\data\coords.dbf", uidfield="LINEID", fromx="X1", fromy="Y1", tox="X2", toy="Y2" out="C:\data\lines.shp", where="LINEID<1000");

3.20 convert.units

Convert Units: Converts length or area values from one unit to another (an interactive command line tool).

Description

This tool converts length or area values from one unit to another. It is designed simply as a tool to assist users in constructing other command strings. For instance, you might wish to buffer features by 1 mile but the coordinate system is in meters, so this tool could be used to calculate that conversion.

Syntax

convert.units(value, units);

value	the numerical value to convert
units	the units of the value, and the units to convert to (you can list more than one type of units to convert to), e.g. c("M²","A"); supported units: METERS (M), FEET (FT), KILOMETERS (KM), MILES (MI), METERS SQUARED (M^2), FEET SQUARED (FT^2), KILOMETERS SQUARED (KM^2), MILES SQUARED (MI^2), HECTARES (H), ACRES (A)

Example

convert.units(value=12.3456, units=c("A", "H"));
convert.units(value=12.3456, units=c("M", "FT", "MI", "KM"));

The second example would result in the following:
12.3456 METERS = 40.50393706944 FEET
12.3456 METERS = 0.0076712001879552 MILES
12.3456 METERS = 0.0123456 KILOMETERS

3.21 copyfeaturedataset

Copy Feature Dataset: Copies features from one data source to a new data source.

Description

This command copies the features in the input data source to the specified output data source, retaining all attribute information in the table. This provides an easy way of loading shapefile data into geodatabases, making backup copies of datasets, and, if the 'where' clause is used, writing subsets of data to different output data sources. It could also be used to merge different data sources. If the output data source does not exist, it is created (this is the intended primary use of the tool). If the output data source exists the tool will attempt to load the features and tabular data into this existing data source.

The 'where' clause allows you to process a subset of features. It is highly recommended that you learn about the 'where' clause as it is one of the more powerful aspects of GME. See the 'Strategic commands' section and the 'where' section for further details.

Syntax

copyfeaturedataset(in, out, [where]);

in	the input feature data source
out	the output feature data source
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

copyfeaturedataset(in="C:\data\counties.shp", out="C:\bkup\counties.shp");
copyfeaturedataset(in="C:\data\homeranges.shp",
pnt="C:\data\female_homeranges.shp", where="SEX='F'");

3.22 countpntsinpolys

Count Points In Polygons: Counts the number of points that overlap each polygon, and writes the result to a field in the polygon attribute table.

Description

This tool counts the number of points that overlap each polygon, and writes the result to a field in the polygon attribute table. Each polygon is processed consecutively and independently of any other overlapping polygons. Thus, if there are overlapping polygons a single point may contribute to the count of more than one polygon, and the count for a polygon that is 'underneath' an overlapping polygon will not be influenced by this overlap.

This tool will also optionally weight the count by a value in a field in the point attribute table. In other words, rather than simply count the number of points, the tool will sum the weights and write that value to the polygon attribute table rather than the count. If the tool is writing a count the new field that is added will be an integer field, but if the tool is writing a weight value the new field will be a double precision field type.

Note that if the output field specified already exists then an error message is raised and the program will stop.

Syntax

countpntsinpolys(poly, pnt, field, [weight], [where]);

poly	the input polygon data source
pnt	the input point data source
field	the name of the new field to add (must not already exist)
[weight]	if specified the point counts are weighted by the values in this field in the point attribute table (i.e. the weights are summed)
[where]	the selection statement that will be applied to the polygon feature data source to identify a subset of polygons to process (see full Help documentation for further details)

Example

countpntsinpolys(poly="C:\data\counties.shp", pnt="C:\data\wells.shp", field="WELLCNT");
countpntsinpolys(poly="C:\data\homeranges.shp", pnt="C:\data\killsites.shp", field="KILLCNT", weight="KILLTYPE");

3.23 deletefeatures

Delete Features: Deletes the features from a vector data source that meet the criteria specified using a 'where' clause.

Description

This tool deletes one or more features from an input feature data source. The features to be deleted are specified using a 'where' clause. See the 'where' section for further details on how to formulate the where clause.

This tool is dangerous if not used with caution. Deleted features cannot be recovered. You should familiarise yourself with how this command operates on a copy of the dataset. You can use the copyfeaturedataset command to create a backup of the target data source before running this tool (even in a script). You can also test your where clause in ArcGIS using the 'Select by attributes' command and inspecting the resulting selection to ensure it is appropriate.

Syntax

deletefeatures(in, where);

in	the input feature data source
where	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

deletefeatures(in="C:\data\plotlayer.shp", where="COUNTY='WOOD' AND MONTH=7");
deletefeatures(in="C:\data\samples.shp", where="AREA < 100");
deletefeatures(in="C:\data\roads.shp", where="FID=20123");

3.24 delimiter

Set Delimiter: Changes the delimiting character used when reading/writing tabular text output such as comma delimited text files.

Description

This tool allows you to specify the delimiting character used when writing tabular text output. Most frequently you will use one of these keywords: COMMA, SPACE, NONE, TAB, SEMICOLON, or COLON. You may specify other delimiting characters literally (e.g. char=.). It is recommended that you avoid the use of any other symbols or punctuation marks. It is probably extremely unwise to use the NONE option, but I have included it here in case there is a need for it I have not anticipated.

This delimiter is also used when reading delimited input files (e.g. the 'reclassify' command). If you set the delimiter you should be very careful about using that delimited consistently, or changing it to the appropriate character before issuing a command that uses it.

Syntax

delimiter(char);
 char the delimiting character(s) (default="COMMA") - normally one of these keyword options: COMMA, SPACE, NONE, TAB, SEMICOLON, COLON

Example

delimiter(char="TAB");
delimiter(char=".");

3.25 download

Download File: Downloads files from websites via an HTTP connection.

Description

This tool downloads files from websites via an HTTP connection. It is designed to assist with the automation of workflows that require retrieval of data or programs from a website. If the file already exists in the specified destination folder the tool will not overwrite it unless you have explicitly used the 'overwrite=TRUE' option. You can optionally also run the downloaded file automatically upon download.

Note that the first time the tool is run it can take several seconds to establish an internet connection, but all subsequent calls will be much faster.

Syntax

download(url, folder, [overwrite], [run]);
 url the full URL of the file to download
 folder the destination folder for the downloaded file
 [overwrite] (TRUE/FALSE): automatically replace a file if it already exists? (default=FALSE)
 [run] (TRUE/FALSE) run the file after download? (default=FALSE)

Example

download(url="http://www.spatialecology.com/download/htools.txt", folder="C:\tmp");

3.26 export.asciigrid

Export To ASCII Grid: Exports a raster to ASCII grid format (text file format with header data).

Description

This tool exports a raster to ASCII grid file format. The first 6 lines of the output text file contain header data in the following format:
ncols 43200
nrows 18000
xllcorner -180
yllcorner -60
cellsize 0.0083333337679505
NODATA_value -9999
-9999 -9999 ...

If you do not specify a nodata value, the command will attempt to acquire it automatically from the raster. It is probably wise to specify your own value, and -9999 is a frequently used value.

After the header data, the tool simply writes one line for each row in the input raster, with raster values delimited by spaces.

Syntax

export.asciigrid(in, out, [band], [nodata]);

in	the input raster data source
out	the output ASCII grid file to create, with the extension (usually .asc, .txt, or .grd)
[band]	the band to export in the case of a multiband image (default=1)
[nodata]	the NoData value to write to the text file; default is the rasters NoData value though it is recommended you explicitly specify a suitable value, e.g. -9999

Example

```
export.asciigrid(in="C:\data\dem.img", out="C:\data\elevation.asc", nodata=-9999);
```

3.27 export.csv

Export Table To Delimited Text File: Exports tables to a delimited textfile format.

Description

This tool exports tables to a delimited text file format (by default a comma delimited text file). If the output file already exists, the file will not be overwritten and an error is generated.

The 'transpose' option rotates the data (rows become columns, and columns become rows). If you have a large number of rows in the input table you will not be able to view the table in a program like Excel because of the limit to the number of columns that can be displayed, but you could still use the transposed table in a program that was designed to accept a large number of columns.

The delimiting character can be specified. See the 'delimiter' command for instructions on how to change the delimiting character on all tabular text output.

Syntax

```
export.csv(in, out, [transpose]);
 in           the input feature or table data source
 out          the output delimited text file
 [transpose]  (TRUE/FALSE) if TRUE transposes the table (default=FALSE)
```

Example

```
export.csv(in="C:\data\locs.shp", out="C:\data\locs.csv");
export.csv(in="C:\data\locs.shp", out="C:\data\locs.csv", transpose=TRUE);
```

3.28 extractedge

Extract Edge From Raster: Creates lines along the boundary between two pixels that have different values.

Description

This tool creates lines along the boundary between two pixels that have different values. It is intended to be used with categorical raster data like landcover rasters. The output polyline attribute table contains three fields that identify the cell values of the two pixels on each side of the line, and a text field that combines the two values in a single field (this is useful for identifying and displaying edges between particular combinations of values).

It is common for the output polyline data source to contain large numbers of features (because rasters tend to contain large numbers of cells and edges). This can make the output cumbersome to work with. It is recommended that you reclassify your raster (to simplify it) before running this tool so that only the edges of interest are identified in the output. This collection of tools includes a raster reclassification tool.

If you wish to simplify the output you could consider running the Dissolve command (in ArcToolbox), referencing the text field as the dissolve field. This would merge all edges of the same type into a single multipart polyline feature, thereby greatly reducing the number of records in the table.

Although a 'band' option is included so that users can identify which band to process in the case of multiband images, it is recommended that you do not run this tool with continuous raster data (like DEM's, or unclassified satellite imagery). This would result in the creation of edges at almost every cell boundary.

Syntax

```
extractedge(in, out, [band]);
 in       the input integer raster data source
 out      the output line data source
 [band]   the input band (default=1)
```

Example

```
extractedge(in="C:\data\landcov", out="C:\data\lcov_edges.shp");
```

3.29 field.delete

Delete Fields: Deletes one or more fields from a table.

Description

This tool deletes one or more fields from a table. If more than one field is specified, but one of the fields does not exist, the tool will continue to attempt to delete the other fields. An asterisk (*) can be used at the start and/or end of the field name as a wildcard, but it cannot be used in the middle of the field name. Also, the asterisk wildcard is only recognized when a single string is provided to the 'fld' parameter, but not in an array of string. For example, these are all valid uses of an asterisk: fld=AR*, or fld=*SIZE, or fld=*AA*. These are NOT valid uses of an asterisk: fld=A*BC, or fld=*A*B*, or fld=c(AREA, PERIM*).

Using wildcards to delete fields can be dangerous! Take great care in how the wildcard text is specified. There is no undo, so the delete is permanent. As a precaution this tool will never delete a geometry field or any of the special unique ID fields (thus the FID field is protected, but typically the ID field is not as it is commonly just an integer field).

Syntax

```
field.delete(in, field);
 in      the input feature data source
 field   the names of the field(s) to delete (use the c() format to specify more
         than one field - see example below)
```

Example

```
field.delete(in="C:\data\referencelayer.shp", field="MYFLD1");
field.delete(in="C:\data\roads.shp", field=c("NAME", "PROV", "SURFACE"));
field.delete(in="C:\data\roads.shp", field="LEN*");
```

3.30 field.find

Find Mininum/Maximum Among Fields: Searches fields to find the minimum and/or maximum value in each record.

Description

This tool searches a collection of fields to find the minimum and/or maximum value among those fields, and writes the result of the search to a new field. In the case of a tie, the tool reports the first field encountered with the minimum or maximum value.

The input fields can be specified either as a vector of field names (see example below) or providing the prefix of field names that will be used in the search. For instance, using the isectpolyrst tool with the thematic option to summarize a categorical raster dataset can produce numerous fields that share the same prefix. This prefix option is therefore convenient for searching this sort of data to find, for instance, the dominant land cover type in each polygon.

The data written to new field is based on the field name of the field that met the search criteria. By default it is the field name of the field that met the minimum or maximum criteria. However, you can also use a 'remove' option to specify the number of characters that are removed from the beginning of the field name before writing to the output. Thus, if I want to record the dominant land cover code, and all by fields begin with the prefix 'LCV', then setting remove=3 would strip the prefix and write only the landcover value itself.

In general, if you are working with inconsistently named fields you will provide the field names as a vector, and write the output as a string with no 'remove' option. If you are working with consistently named fields you will typically specify the prefix of the fields, set the remove option to the length of the prefix, and write the output as an integer or string as appropriate.

Note that the field.rename tool is useful here to rename any fields that do not conform to an otherwise consistent naming convention.

Syntax

field.find(in, fields, [max], [outmax], [min], [outmin], [type], [remove]);

in	the input table or feature data source
fields	list of field names as an array, or a single string with the prefix of all fields to include in the search
[max]	(TRUE/FALSE) if TRUE, finds the field with the maximum value among the fields specified (default=TRUE)
[outmax]	if max=TRUE, the name of the maximum output field to create
[min]	(TRUE/FALSE) if TRUE, finds the field with the minimum value among the fields specified (default=FALSE)
[outmin]	if min=TRUE, the name of the minimum output field to create
[type]	the output field type, options: STRING (default), LONG (long integer), DOUBLE (double precision real number)
[remove]	an integer representing the number of characters to remove from the beginning of the field name before writing it to the output field (default=0)

Example

field.find(in="C:\data\plots.shp", fields=c("LCV1", "LCV2", "LCV5", "LCV99"), max=TRUE, outmax="DOMINVAL");
field.find(in="C:\data\plots.shp", fields="LCV", max=TRUE, outmax="DOMINVAL", min=TRUE, outmin="MINVALFLD", type="LONG", remove=3);

3.31 field.rename

Rename Field: Renames a field in a table.

Description

This tool renames fields in a table. It does this by creating a new field with the desired name, copying values from the old field into the new field, and then deleting the old field. The position of the field therefore changes. The field type of the new field is based on the input field.

This tool will only work with the simple field data types: short and long integers, double precision numbers, strings, etc. If you specify a field that is an object ID field or a geometry field then the tool will return an error message and no renaming will occur.

Syntax

field.rename(in, field, out);

in	the input table or feature data source
field	the name of the field to rename
out	the new field name

Example

field.rename(in="C:\data\plots.shp", field="LCOV1999", out="LANDCOV99");

3.32 file.append

Append Text Files: Appends text files located in the same folder.

Description

This tool appends text files located in the same folder. It is likely that you will want to use both of the optional settings for this tool. The first option, 'match', is a search string filter: the tool will only process files with names that match the specified search string. This is useful because it provides a means of preventing other files that may be in the specified folder from being erroneously included in the appended output file. The second option, 'skip', is a boolean setting that controls whether the first line is skipped in all appended files except the

first one. This is useful because many data files have a header line that contains the column headings, but you would only with to include a header line once in the appended file.

Note that the file extension does not matter. Text files can have many different extensions (txt, dat, csv, etc.) so the tool does not automatically filter files based on the extension unless you use the 'match' expression (e.g. match=*.txt"). Nor does the tool automatically ignore binary files. You must either ensure that only text files are in the specified folder, or use the match option to ensure that only the appropriate text files are processed. If a binary file is processed the output will contain strange characters and will be useless.

The standard search pattern rules for Windows apply to the match option. Some examples are: *.txt returns all files with a txt extension, rain*.dat returns all files that begin with 'rain' and have a dat extension, and *data* return all files with word 'data' somewhere in the filename.

An example serves to illustrate the use of this tool. Many datasets are delivered in simple text file format (because it is platform and software independent) and can be downloaded from a website. These data are usually delivered in manageable chunks: divided into regions or time periods or both. Importing them into a useful format would take a great deal of time if they were processed individually. Given that we only want to run the import procedure once, this tool provides a means of appending all of the files into a single input file that we can then import.

A number of tools have been developed to manipulate text files. They were developed to make it more efficient to import datasets delivered in text file format (e.g. weather datasets that are delivered as separate files for each year, or XML files containing spatial data), but are likely to be useful for a wide range of tasks. See the commands beginning with 'file' for further information.

Syntax

file.append(folder, out, [match], [skip]);

folder	the path of the folder containing the text files to append
out	the full path to the new output text file
[match]	a search string filter: only files matching this search string will be processed (use the wildcard character * in the expression - see the help documentation for further details)
[skip]	(TRUE/FALSE) if TRUE skips the first line of all files apart from the first one (default=FALSE)

Example

```
file.append(folder="C:\data", out="C:\data\appended.txt", match="RAIN",
skip=TRUE);
```

3.33 file.countlines

Count Lines In Text File: Counts the number of lines in a text file.

Description

This tool counts the number of lines in a text file, with the option of only counting lines that contain a specified search string. This can be useful when dealing with large XML files for instance when you might wish to determine the number of features the XML file contains. It is also useful when used in conjunction with the file.readlines and file.extractlines tool when you are interested in viewing or extracting data from the end of a large text file.

The 'match' option requires that you specify a search string. If a line in the input file contains this string anywhere within that line it is included in the count. Note that the search string is interpreted literally, and does not interpret any characters as wildcards (e.g. *).

A number of tools have been developed to manipulate text files. They were developed to make it more efficient to import datasets delivered in text file format (e.g. weather datasets that are delivered as separate files for each year, or XML files containing spatial data), but are likely to be useful for a wide range of tasks. See the commands beginning with 'file' for further information.

Syntax

file.countlines(file, [match]);
file the full path to the input text file
[match] if specified counts only the the lines containing this string

Example

```
file.countlines(file="C:\data\roads.gml");
file.countlines(file="C:\data\roads.gml", match="ENDOFRECORD");
```

3.34 file.extractlines

Extract Lines From Text File: Extracts a specified range of lines from a textfile and writes them to a new file.

Description

This tool extracts a specified range of lines from a text file and writes them to a new file. It was designed to provide a way of inspecting text files (e.g. XML or GML files) that are so large they cannot be conveniently opened in other software. The user specifies a start and end line that defines the range of lines to extract. If the end line specified is larger than the number of lines in the file the tool will extract all lines from the specified start line to the end of the file. The file.countlines tool can be used to determine the total number of lines in the text file, which is useful if you wish to extract the last line in the file. This tool is also useful for determining if text data files have header lines (e.g. extract the first 10 lines).

A number of tools have been developed to manipulate text files. They were developed to make it more efficient to import datasets delivered in text file format (e.g. weather datasets

that are delivered as separate files for each year, or XML files containing spatial data), but are likely to be useful for a wide range of tasks. See the commands beginning with 'file' for further information.

Syntax

file.extractlines(file, newfile, start, end);

file	the full path to the text file
newfile	the full path to the new text file to create
start	the line number to start reading from the input file
end	the line number to stop reading from the input file

Example

file.extractlines(file="C:\data\roads.gml", newfile="C:\data\roads.txt", start=100, end=250);

3.35 file.readlines

Read Lines From Text FIle: Extracts a specified range of lines from a textfile and writes them to the output window.

Description

This tool extracts a specified range of lines from a text file and writes them to the output window in this tool. It was designed to provide a way of inspecting text files (e.g. XML or GML files) that are so large they cannot be conveniently opened in other software. The user specifies a start and end line that defines the range of lines to read. If the end line specified is larger than the number of lines in the file the tool will read all lines from the specified start line to the end of the file. The file.countlines tool can be used to determine the total number of lines in the text file, which is useful if you wish to read the last lines in the file. This tool is also useful for determining if text data files have header lines (e.g. read the first 10 lines).

Note that this tool is similar to the file.extractlines tool but writes the lines to the output window rather than a different file. The advantage of this is that it allow faster inspection of a file, but if the file contains special characters then there is a risk it will not be displayed properly in the output window. The file.extractlines tool is better if you want a precise extraction whereas this tool is more useful for a quick but potentially imprecise investigation. XML and GML files can be particularly problematic with this tool, but data files are usually fine.

A number of tools have been developed to manipulate text files. They were developed to make it more efficient to import datasets delivered in text file format (e.g. weather datasets that are delivered as separate files for each year, or XML files containing spatial data), but are likely to be useful for a wide range of tasks. See the commands beginning with 'file' for further information.

Syntax

file.readlines(file, start, end);
 file the full path to the text file
 start the line number to start reading
 end the line number to end reading

Example

file.readlines(file="C:\data\roads.gml", start=1, end=100);

3.36 file.split

Split Text File: Splits a text file into multiple, approximately equally sized, smaller text files.

Description

This tool splits a text file into multiple, approximately equally sized, smaller text files. There are two optional arguments that make this a particularly useful command in certain situations. First, the 'hdr' option allows you to specify the number of lines that make up the header of the file that will also be written as the header of split files (thus preserving the header among all the files). For many data files this will just be 'hdr=1', but for some XML and GML file format the headers can occupy several dozen lines.

The second optional argument is 'match', which ensures a file is only split immediately following lines containing the match search string. In data files this will not normally be useful (because each line represents a different record and it does not matter where the break in lines occurs). But in XML or GML files a single record can occupy multiple lines, so it is important that breaks occur only at appropriate places.

If the output folder does not exist, the tool will attempt to create it. If any of the output files already exist, an error is generated and the tool stops.

A number of tools have been developed to manipulate text files. They were developed to make it more efficient to import datasets delivered in text file format (e.g. weather datasets that are delivered as separate files for each year, or XML files containing spatial data), but are likely to be useful for a wide range of tasks. See the commands beginning with 'file' for further information.

Syntax

file.split(file, folder, prefix, cnt, [hdr], [match]);

file	the full path to the input text file
folder	the output folder
prefix	the file name prefix of the output files
cnt	the number of output files to create (the input file is split between them)
[hdr]	the number of header lines from the beginning of the file to include in each output file (default=0)
[match]	breaks the file only on lines containing this string (unspecified by default)

Example

file.split(file="C:\data\roads.gml", folder="C:\data\newroads", prefix="roads", cnt=5);
file.split(file="C:\data\roads.gml", folder="C:\data\newroads", prefix="roads", cnt=5, hdr=7, match="ENDOFRECORD");

3.37 for

For Loop: A programming device that facilitates iterative repetition of commands.

Description

A for loop is a programming device that allows you to repeat a set of commands many times with only a few lines of code. The format of the for loop in this software is similar to that of R. The word 'for' indicates that you are starting a loop. The next expression controls the number of iterations: for(i in 1:100) implies that the loop is going to repeat itself 100 times. The letter 'i' is the index variable name - it could be any simple string you wish to use (another letter, or even a text string like 'num'). You can use this variable in subsequent commands to construct command arguments using the paste() function.

The set of commands that are to be repeated must be contained by a set of braces { }, and each command must end with a semi-colon to distinguish it from the next command. Nested for loops are supported, but it is essential to use a different index variable for each nested loop (e.g. i, j, k for three nested loops, for instance). For loops can be included within scripts that you call with the 'run' command.

If you want to iterate across a non-continuous range of numbers, or a sequence of arbitrary values, then you can combine the use of a GME variable with a for loop. For example, given a list of animal ID numbers, i.e. uid <- c(12, 14, 19, 21);, a for loop could be specified as for(i in 1:length(uid)) { ... }, and the id numbers can be retrieved from within the loop using uid[i] whereever needed.

An advanced problem is building the for(i in 1:100) statement using dynamic variables. For instance, rather than specify 1 and 100, you might want to use variables you have previously defined (see 'Strategic commands, functions and clauses' section). This is possible, however: you must ensure the variables are defined (and visible using the ls() command) prior to submitting the for loop command text. I.e. you cannot define the variables and use them in

a for loop in the same block of text that is submitted to GME. This is a command interpreter limitation that I intend to resolve in a future update. This should not be a limitation to the majority of users.

See the Automation and Batch Processing section near the beginning of the manual for more detailed examples and explanation of how to use for loops.

See also: paste, r.eval, list.vector, list.raster

Syntax

for(i in x:y) ...commands... ;

i in x:y	i is the variable name that is incremented within the for loop (it can be any simple string), x is an integer at which the looping begins and y is another integer (greater than or equal to x) at which the looping ends, e.g. for(i in 1:100). The command is followed by a series of other commands enclosed within braces and separated by semicolons that are executed at each iteration of the loop: { ...commands... }

Example

```
for(i in 1:5) {
genrandompnts(poly="C:\data\fields.shp", sample=100, out=paste("C:\data\samples", i,
".shp"));
addxy(in=paste("C:\data\samples", i, ".shp"), prefix="COORD");
r.plotxy(in=paste("C:\data\samples", i, ".shp", xfield="COORDX", yfield="COORDY");
};
```

Results in the generation of 5 point datasets, with COORDX, COORDY fields in the attribute table, and five R graphs of the x, y coordinates.

3.38 gencirclesinpolys

Generate Circles In Polygons: Generates a regular arrangement of circles within the bounds of polygons.

Description

This tool generates a regular arrangement of circles within the bounds of polygons. The user specifies the radius of the circles (a constant) and the spacing between the centres of the circles, which controls the amount of overlap in adjacent circles and the amount of interstitial space. Only circles that are completely contained by the polygon are retained. If this tool does not suit your purpose you may find the 'genshapes' command more useful - it will also generate a regular arrangement of circles but without the constraint that the circles must be completely contained by the polygon. Also, the genhexagonsinpolys command serves a similar function to this tool but using hexagons instead of circles.

There is an element of randomisation built into this tool: the overall placement of the grid of circles is determined randomly so that if you run the tool twice, you will get a different result. This is important because it allows you to run the tool many times in an attempt to find the maximum number of circles that can fit in a given polygon. This will depend on the complex shape of the polygon and generally cannot be determined analytically.

Note the default spacing option allows just enough overlap of circles to eliminate interstitial spaces between circles. If you prefer that there is no overlap between circles then set the spacing option to two times the radius. This ensures that the circles are touching, but there is no overlap.

One possible application of this tool might be estimating the number of viable home ranges that an area might contain.

Syntax

gencirclesinpolys(in, uidfield, out, radius, [spacing], [where]);

in	the input polygon data source
uidfield	the unique ID field of the input feature data source
out	the output polygon data source
radius	the radius of the circles (a constant specified in the coordinate system units of the reference layer)
[spacing]	the distance between adjacent circle centres (defaults to 2*radius*cos(PI/6), which is the largest possible spacing that creates no interstitial spaces between circles)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

gencirclesinpolys(in="C:\data\referencelayer.shp", out="C:\data\myoutput.shp", radius=100);
gencirclesinpolys(in="C:\data\boundaries.shp", out="C:\data\myoutput.shp", radius=10.567, spacing=30);

3.39 gencondrandompnts

Generate Conditional Random Points: Generates a sample of random points around each input point, designed for a conditional logistic regression (case controlled) style of analysis.

Description

This tool generates a sample of random points that are associated with each point in an input point data source. The command options can be combined in a variety of ways to produce

different sampling outcomes. Although this tool is therefore very flexible, care must be taken to ensure that the parameters specified to not result in a problem that has no solution.

For each point in a point data source this tool will generate a sample of random points based on that points spatial location and a sampling distribution you specify. The number of points generated can be a constant or variable between points. In the latter case the number of points to generate per polygon must be stored in a field in the point attribute table.

Points are generated by drawing from a bivariate distribution. Two distributions are currently supported: BVUNIFORM, and BVNORMAL. In the case of BVUNIFORM, if you specify a single parameter this is assumed to represent the radius of the circle centred on the source point within which uniformly distributed sample points are generated (this distance should be specified in coordinate system units, e.g. meters for UTM). However, if two parameters are specified then points are generated within a 'donut' shape centred on the source point, i.e. the first parameter defines the radius of the circle that is the inside of the donut, and the second value defines the radius of the circle that is the outside of the donut.

In the case of the BVNORMAL distribution the single parameter represents the standard deviation of the distribution in both the x and y directions (assumed to be the same, with no covariance). The 'mvrnorm' function in the MASS library in R is used to generate these bivariate normal values. To specify the distributions you would either type 'c("BVUNIFORM", value)' or 'c("BVUNIFORM", value, value)' or 'c("BVNORMAL", value)', where the word value is replaced by the radius or variance respectively.

The 'mindist' option will ensure that all output points are this minimum distance apart. This value should be specified in coordinate system units (e.g. meters for UTM). If this value is too large then the problem may have no solution. For example. it is not possible to generate 100 points in a square kilometer where the minimum distance between points is 500m.

The 'excl' option can be used to prevent points from being generated within the polygons of this dataset. For example, if you were to generate forest sampling points using a stand polygon layer, you might use a second polygon layer representing a 100m buffer of roads as the exclusion layer to prevent points from occurring too close to roads. Again, there is a risk here of creating an unsolvable problem: if the exclusion polygons cover all of the input polygons then no random points can be generated.

Syntax

gencondrandompnts(in, uidfield, sample, distrib, out, [mindist], [excl], [where]);

in	the input point data source
uidfield	the name of the unique point ID field in the input data source
sample	the sample size: either a number representing the constant sample size per input point, or the field name of field containing the sample size for that point
distrib	the sampling distribution, either c("BVUNIFORM", radius) or c("BVUNIFORM", radius1, radius2) or c("BVNORMAL", variance) (see full help documentation for details)
out	the output point data source
[mindist]	if specified, points are prevented from occurring within this minimum distance of each other (specified in coordinate system units of the reference layer); this option can be dangerous - see the help documentation for details
[excl]	the polygon data source containing exclusion polygons: points are prevented from being generated within these polygons; this option can be dangerous - see the help documentation for details
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

gencondrandompnts(in="C:\data\locations.shp", uidfield="LOCID", sample=10,
distrib=c("BVUNIFORM",100), out="C:\data\condsamplepnts.shp");
gencondrandompnts(in="C:\data\locations.shp", uidfield="LOCID",
sample="SAMPSIZE", distrib=c("BVNORMAL",29.6),
out="C:\data\condsamplepnts.shp");

3.40 generalizeregions

Generalize Raster Regions: Generalizes / simplifies a thematic raster dataset based on the size of groups of contiguous pixels of the same value.

Description

This tool generalizes / simplifies a thematic (categorical) raster dataset by recoding the smallest contiguous groups of pixels of the same value with the pixel value of the largest neighbouring group. You must run the regiongroup command before running this command as the output of the regiongroup command serves as the input to this command.

The cell neighbourhood that is evaluated can be all eight cells that surround a cell (this is the default option), or can be limited to only the four cardinal cells using the 'diag=FALSE'

option, which prevents diagonal connections being evaluated when defining region membership. You will probably want to use use the same option here that was used with the regiongroup command.

This is an iterative command whereby the maximum allowed number of iterations is controlled by the limit option. If your raster is simple (relatively homogeneous) then only one or two iterations may be required. However, If portions of the raster are highly heterogeneous, and many small groups of pixels are mixed together, then more iterations will be required to arrive at a solution that no longer changes. As a general rule, 20 iterations is a reasonable starting point, but it is recommended that you inspect the raster to ensure the command has produced consistent results. Watch the number of groups to classify decrease in the progress bar as it processes. If it reaches 0 then it has reached an endpoint that cannot be further improved.

Syntax

generalizeregions(in, table, limit, out, [diag]);

in	the input raster region data source
table	the input table data source resulting from the regiongroup command
limit	the minimum region size expressed as the minimum number of connected raster pixels
out	the output raster data source
[diag]	(TRUE/FALSE) allow cells to be diagonally connected when generalizing regions? (default=TRUE)

Example

generalizeregions(in="C:\data\lcovregions.img", table="C:\data\lcovregions.dbf", limit=10, out="C:\data\lcovgen");
generalizeregions(in="C:\data\lcovrg", table="C:\data\lcovrg.dbf", limit=20, out="C:\data\lcovgeneralized.tif", diag=FALSE);

3.41 genhexagonsinpolys

Generate Hexagons In Polygons: Generates a regular arrangement of hexagons within the bounds of polygons.

Description

This tool generates a regular arrangement of hexagons within the bounds of each polygon in a polygon data source. The user specifies the dimension of the hexagons (a constant) and the amount of spacing between adjacent hexagons. Only hexagons that are completely contained by each polygon are retained. Note that if the polygons overlap, then the resulting hexagons are also likely to overlap. If this tool does not suit your purpose you may find the 'genshapes' command more useful - it will also generate a regular arrangement of hexagons but without the constraint that the hexagons must be completely contained by the polygon. Also, the

gencirclesinpolys command serves a similar function to this tool but using circles instead of hexagons.

There is an element of randomisation built into this tool: the overall placement of the hexagon grid is determined randomly so that if you run the tool twice, you will get a different result. This is important because it allows you to run the tool many times in an attempt to find the maximum number of hexagons that can fit in a given polygon. This will depend on the complex shape of the polygon and generally cannot be determined analytically.

Note that hexagons mesh together perfectly to completely cover space with no overlap and with no interstitial spaces between hexagons (unlike circles). However, the 'spacing' option does allow you to add overlap (a negative value) or increase the spacing between hexagons (a positive value).

One possible application of this tool might be estimating the number of viable home ranges that an area might contain.

Syntax

genhexagonsinpolys(in, uidfield, out, dimension, [spacing], [where]);

in	the input polygon data source
uidfield	the unique ID field of the input feature data source
out	the output polygon data source
dimension	the distance from the centre of the hexagon to the middle of one of the six edges that define the hexagon (a constant specified in the coordinate system units of the reference layer)
[spacing]	the additional distance to add between hexagons (default= 0, which implies no interstitial spaces between hexagons; a negative value results in hexagon overlap)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

genhexagonsinpolys(in="C:\data\referencelayer.shp", out="C:\data\myoutput.shp", dimension=100);
genhexagonsinpolys(in="C:\data\boundaries.shp", out="C:\data\myoutput.shp", dimension=100, dimension=30);

3.42 genmcp

Generate Minimum Convex Polygons: Generates minimum convex polygons (MCP's) based on a set if input points.

Description

This tool generates minimum convex polygons (MCP) based on the set of input points specified by the user. If a unique ID field is specified, one MCP is generated for each unique ID. If no unique ID field is specified, a single MCP polygon is generated based on all the points in the input data source.

Syntax

genmcp(in, [uidfield], out, [where]);

in	the input point data source
[uidfield]	the input unique ID field (one MCP polygon is created for each unique ID) - if omitted, one MCP is created for all the points in the dataset
out	the output polygon data source
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

genmcp(in="C:\data\samplepnts.shp", out="C:\data\samplemcp.shp");
genmcp(in="C:\data\telemetry.shp", out="C:\data\telem_mcp.shp",
uidfield="ANIMALID");

3.43 genpointinpoly

Generate Points In Polygons: Generates a single point for each polygon in the input data source.

Description

This tool generates a single point for each polygon in the input data source. The point can be positioned using one of two settings. The LABEL setting, which is the default, generates a point somewhere inside the boundary of the polygon. The algorithm ESRI uses to determine the location of the label point is not published, but the point is guaranteed to be inside the boundary of the polygon. The CENTROID position places the point in the geometric centre of the polygon (the centre of gravity), which may not fall within the bounds of the polygon depending upon its shape.

The attribute fields for the polygons can be copied to the new point dataset using the 'copyfields' setting. By default the fields are not copied.

Syntax

genpointinpoly(in, out, [position], [copyfields], [where]);

in	the input polygon data source
out	the output point data source
[position]	if CENTROID the points are generated at the centre of gravity of the polygon (which may be outside of the polygon); if LABEL the point is guaranteed to be inside the polygon but may not represent the centre of gravity (default=LABEL; options: CENTROID, LABEL)
[copyfields]	(TRUE/FALSE) if true the attribute fields from the input dataset are copied to the output dataset (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
genpointinpoly(in="C:\data\boundaries.shp", out="C:\data\bndcentroid.shp",
position="CENTROID", copyfields=TRUE);
genpointinpoly(in="C:\data\boundaries.shp", out="C:\data\bndlabels.shp");
```

3.44 genrandompnts

Generate Random Points: Generates a sample of random points, with several optional constraint parameters.

Description

This tool generates a sample of random points. The command options can be combined in a variety of ways to produce different sampling outcomes. Although this tool is therefore very flexible, care must be taken to ensure that the parameters specified to not result in a problem that has no solution.

The spatial sampling domain can be defined in several ways. If the 'poly' option is used, points are only generated where there are polygons (but this is not a stratified sampling design - see the genstratrandompnts command for that option). If the 'raster' option is used, points are generated within the bounds of the raster. By default, the tool will prohibit points from occurring in NoData cells, although this behaviour can be overridden with the 'pnd' option). Finally, the 'extent' option can be used to define a rectangle within which points are generated. The extent can either be a feature data source (the extent of all the features is used), or as a set of four values that define the minimum x, maximum x, minimum y, and maximum y of the envelope.

The number of points to generate is defined by the 'samples' option. Points are generated using a simple rejection method algorithm: potential points are generated within the rectangular boundary that defines the feature space, raster space, or extent based on a bivariate

uniform random distribution. They are then accepted if they conform to the acceptance criteria the user has defined.

The 'mindist' option will ensure that all output points are this minimum distance apart. This value should be specified in coordinate system units (e.g. meters for UTM). If this value is too large then the problem may have no solution. For example. it is not possible to generate 100 points in a square kilometer where the minimum distance between points is 500m.

The 'excl' option can be used to prevent points from being generated within the polygons of this dataset. For example, if you were to generate vegetation sampling points using a reference landcover raster, you might use a polygon layer representing lakes as the exclusion layer to prevent points from occurring in water. Again, there is a risk here of creating an unsolvable problem: if the exclusion polygons cover all of the reference polygons or reference raster then no random points can be generated.

The 'pnd' option only applied when using a raster reference data source (i.e the 'raster' option). It controls whether points can be generated within NoData cells. By default, this is not allowed but you can override this behaviour by setting pnd=FALSE. Note that not all raster formats support NoData values. The grid format does, as do geodatabase rasters. But the GeoTIFF and Imagine formats do not. In these cases there are no NoData values, so the pnd option has no effect. You can determine if your raster has a NoData value by looking at the properties for that layer in ArcMap.

The 'probsurf' option determines whether the raster layer is interpreted as a probability density surface or not. The default is that rasters are not interpreted in this way. If you set probsurf=TRUE, then the cell value is interpreted as the probability of a random point being generated within that cell. The program will automatically rescale the raster values to a probability (by dividing by the maximum value), thus your raster does not necessarily have to be scaled in the range 0-1 prior to running the tool.

Syntax

genrandompnts([poly], [raster], [extent], sample, out, [mindist], [excl], [pnd], [probsurf], [where]);

[poly]	the input reference polygon data source (points are only generated within polygons)
[raster]	the input reference raster data source
[extent]	the input reference data source, or coordinates, that defines the extent within which points are generated (without regard to the distribution of features or raster values within that extent)
sample	the sample size
out	the output point data source
[mindist]	if specified, points are prevented from occurring within this minimum distance of each other (specified in coordinate system units of the reference layer); this option can be dangerous - see the help documentation for details
[excl]	the polygon data source containing exclusion polygons: points are prevented from being generated within these polygons; this option can be dangerous - see the help documentation for details
[pnd]	(TRUE/FALSE) in the case of raster data sources, if TRUE points are prevented from being placed in NoData cells (i.e. pnd=prevent NoData) (default=TRUE)
[probsurf]	(TRUE/FALSE and applies only to the raster argument) if TRUE the raster is treated as a probability surface (see help documentation for further details) (default=FALSE)
[where]	(only applies to the 'poly' option) the filter/selection statement that will be applied to the feature class to identify a subset of features to process

Example

```
genrandompnts(poly="C:\data\fields.shp", sample=1000, out="C:\data\samplepnts.shp");
genrandompnts(poly="C:\data\fields.shp", sample=1000, out="C:\data\samplepnts.shp",
mindist=50, excl="C:\data\lakes.shp");
genrandompnts(raster="C:\data\landcov", sample=5000, out="C:\data\samplepnts.shp");
genrandompnts(raster="C:\data\rsf", sample=500, out="C:\data\samplepnts.shp",
probsurf=TRUE);
genrandompnts(extent="C:\data\roads.shp", sample=50000,
out="C:\data\samplepnts.shp");
genrandompnts(extent=c(346000,450600,4956300,5204300), sample=100000,
out="C:\data\samplepnts.shp");
```

3.45 genregionsampleplots

Generate Sample Plots: A feature-rich tool for generating sampling plots (and zones) within a specified area.

Description

This tool generates a sampling grid: a series of regularly sized and spaced square or rectangular polygons. Unlike the simpler 'genvecgrid' command, however, this command provides options for generating both plots and zones (plots are the smallest sampling unit, and zones are groups of plots), for characterizing the plots and zones based on a suitability raster, and for taking into account barriers in the landscape. For instance, this tool can be used to generate sampling plots that do not cross rivers.

The extent of the sampling grid can be specified by referencing a feature data source, or by directly specifying the minimum and maximum x and y coordinates. An ideal choice of reference layer would be a polygon dataset containing a study area boundary. Sampling plots are only generated in areas covered by the features in this layer. Alternatively, if you specify the extent of a rectangle then plots are generate throughout that area.

The primary output of this tool is a plot polygon layer. You may optionally also generate zones (see the 'outzones' and 'dimzones' command options). Zones would be useful if you wish to perform stratified random sampling in order to cluster sample plots in zones. For instance, if the extent of the area you are sampling is very large then pure random sampling of plots may create a distribution of sample plots that is impractical to visit in reality. A more detailed discussion of sampling designs and the issues that motivate different designs is included in the sampling chapter.

The dimensions of the plots (and zones) are specified in coordinate system units (e.g. meters for the UTM projection). It is recommended that you do not use this tool with unprojected data (geographic coordinate systems). The dimension of the zones should be exactly divisible by the dimension of the plots. For instance, plots of 100m and zones of 1km is suitable, but plots of 30m and zones of 1km is not.

The snap option forces the spatial position of the grid to be aligned with the underlying coordinate system. For instance, specifying snap=1000 would force the vector grid lines to be aligned with the exact 1km positions in a UTM grid. For most sampling purposes this is not an important consideration.

The 'raster' option refers to a suitability raster that classifies the landscape into areas that can be sampled (1) and those that cannot (0). For instance, if you are sampling forest habitat in a landscape that is a mixture of forest and open areas, then the raster would depict all forested areas as 1, and all open areas as 0. This raster is used to calculate the proportion and total area of suitable habitat in each plot (and zone), stored in two fields in the attribute table.

This tool adjusts the plots and zones to reflect barriers in the landscapes, which can be depicted as polygons and lines. For the polygons, the tool will remove all plots that are completely contained by barrier polygons, and will clip plots that overlap the barrier polygons. Barrier lines are used to split plots into components that do not cross the barrier lines. You can specify barriers as polygons only, lines only, or both lines and polygons.

After the geometry of the plots has been modified by these barriers, an area threshold (expressed as a proportion of the plot area) is used to identify plot fragments that will be merged with adjacent plots. Any fragments greater than this threshold are considered to be close enough to a full plot size that they can be retained without merging in the plot dataset.

Each of the smaller fragments is merged with the neighbouring plot with which it shares the longest boundary, provided this does not involve crossing a barrier.

The final barrier-related parameter is a tolerance that defines the distance from the edge of the plot that barrier polygons or lines can be ignored. This parameter is needed because, with certain data formats, there can be very small movements of vertices following the splitting procedure (as a result of coordinate precision). Usually this is only a few centimetres and is negligible, but does create a problem when resolving merges of adjacent plot polygons. The default tolerance is 1, and the units for this tolerance are the coordinate system units (e.g. meters for UTM). If you use a barrier line dataset that contains overshoots, then you may need to increase this tolerance to allow the merging algorithm to resolve acceptable merges. It is particularly important when using a barrier line dataset that 1) lines that are supposed to cross do actually cross (e.g. roads at a T junction), and 2) that overshoots at such crossings are not too long. The former issue results in plots that fail to be split by a barrier line, while the latter issue can result in a failure of the algorithm to deal with plot fragments.

Syntax

genregionsampleplots(extent, out, dim, [outzones], [dimzones], [snap], [raster], [barrierpoly], [barrierline], [minarea], [tol]);

extent	the reference layer that defines the extent of the vector grid, or a set of four values that define the extent (min x, max x, min y, max y)
out	the output polygon data source for the plots
dim	the dimensions of the plots in coordinate system units, e.g. 100 or c(100,200); specifying one value results in square plots
[outzones]	the output polygon data source for the zones (if unspecified no zone output is produced; if specified, dimzones must also be specified)
[dimzones]	the dimensions of the zones in coordinate system units (this value should be an exact multiple of the plot dimensions), e.g. 1000 or c(1000,2000); specifying one value results in square zones
[snap]	a value > 0; controls whether the vector grid is aligned with a major coordinate system interval (supplying a value of 1000 will result in the grid being aligned to the 1000-mark intervals of the coordinate system); default=0 (no snap)
[raster]	a suitability (1/0) raster that is used to characterize plots (see help documentation for further details)
[barrierpoly]	a polygon data source representing barriers that cannot be sampled (see help documentation for further details)
[barrierline]	a line data source representing barriers that cannot be crossed (see help documentation for further details)
[minarea]	the threshold, expressed as a proportion of the plot area, that is used to determine whether a plot fragment will be retained as a fragment, or merged (default=0.8, see help documentation for further details)
[tol]	the tolerance that is used in the merging of plot fragments resulting from barriers (see help documentation for further details)

Example

```
genregionsampleplots(extent="C:\data\regions.shp", out="C:\data\plots.shp", dim=200,
outzones="C:\data\zones.shp", dimzones=1000, raster="C:\data\suit.tif");
genregionsampleplots(extent="C:\data\regions.shp", out="C:\data\plots.shp", dim=200,
outzones="C:\data\zones.shp", dimzones=1000);
genregionsampleplots(extent="C:\data\regions.shp", out="C:\data\plots.shp", dim=200,
raster="C:\data\suit.tif", barrierpoly="C:\data\lakes.shp",
barrierline="C:\data\rivers.shp", minarea=0.7, tol=5);
```

3.46 genregularpntsinpolys

Generate Regular Points In Polygons: Generates a regular grid of sample points within a polygon given a specified spacing and rotation.

Description

This tool generates samples of regularly spaced points within polygons. The spacing in the x and y directions, and the rotation angle for the orientation of the sampling grid, can be adjusted for each polygon or can be set as constants for every polygon. The x and y distance parameters (xdist, ydist) can, therefore, be set either to a single value, or can reference fields in the input polygon data source that contain the appropriate spacing values for each polygon. The x spacing refers to the axis perpendicular to the rotation angle, and the y spacing refers to the axis parallel to the rotation angle. Thus, if there is no rotation (rot=0), the x distance refers to the east-west axis.

Note that the rotation angle must be specified in radians, not degrees. To convert from degrees to radians use the formula: radians = degrees * pi / 180, where pi = 3.141529654.

The 'excl' option can be used to prevent points from being generated within the polygons of this dataset. For example, if you were to generate vegetation sampling points you might use a polygon layer representing ponds as the exclusion layer to prevent points from occurring in water.

Syntax

genregularpntsinpolys(in, uidfield, xdist, ydist, out, [rot], [excl], [random], [where]);

in	the input reference polygon data source (points are only generated within polygons)
uidfield	the input polygon unique ID field
xdist	the x-axis sampling distance: either a number representing the distance, or the field name of field containing this value, e.g. 100 or "XDIST"
ydist	the y-axis sampling distance: either a number representing the distance, or the field name of field containing this value, e.g. 100 or "YDIST"
out	the output point data source
[rot]	the rotation angle of the sampling axis: either a number representing the angle in radians, or the field name of field containing this value (default=0)
[excl]	the polygon data source containing exclusion polygons: points are prevented from being generated within these polygons; this option can be dangerous - see the help documentation for details
[random]	(TRUE/FALSE) randomize the alignment of the grid - if false, always aligns the grid in reference to the upper left corner of the polygon envelope (default=TRUE)
[where]	the filter/selection statement that will be applied to the polygon feature class to identify a subset of features to process

Example

genregularpntsinpolys(in="C:\data\stands.shp", uidfield="StandID", xdist=100, ydist=200, rot="Direction", out="C:\data\samplepnts.shp");
genregularpntsinpolys(in="C:\data\lakes.shp", uidfield="LAKEID", xdist="SpacingX", ydist="SpacingY", rot=0, out="C:\data\samplepnts.shp", where="AREA>100000");

3.47 genshapes

Generate Shapes: Generates shapes (polygons) centred on a set of input points, currently supported shapes are: circles, diamonds, hexagons, squares, triangles (equilateral), rectangles and ellipses.

Description

This tool generates shapes (polygons) centred on the input points specified by the user. The currently supported shapes are: circles, diamonds, hexagons, squares, triangles (equilateral), rectangles and ellipses. The dimensions for the shapes can either be specified as a constant that is applied to all shapes, or as a field name that contains variable dimensions. The shapes are all created in a reference to the x axis, but you can specify a rotation parameters (again, either as a constant or a field name) to rotate the resulting shapes.

The input point always defines the centre of the shape. The dimension values expected depend upon the shape. CIRCLES expect a single dimension value representing the radius. DIAMONDS expect two parameters: one is the distance from the centre-point to the tips of the diamond that lie on the x axis, the second dimension is the distance from the centre-point to the tips of the diamond that lie on the y axis. HEXAGONS, SQUARES, and TRIANGLES expect a single dimension value that represents the distance from the centre-point to the middle of one of the edges. RECTANGLES expect two parameters: one is the distance from the centre-point to the middle of the edges the lie along (but perpendicular to) the x axis, the second dimension is the distance from the centre-point to the middle of the edges the lie along (but perpendicular to) the y axis. ELLIPSES expect two dimensions: one that is the distance from the centre-point to the edge of the ellipse along the x axis, and the second is the distance from the centre-point to the edge of the ellipse along the y axis.

Note that the rotation angle should be specified in degrees, not radians.

Syntax

genshapes(in, shape, dim, out, [rot], [where]);

in	input points defining the center of the shapes to generate
shape	the shape to generate (options: CIRCLE, DIAMOND, HEXAGON, SQUARE, TRIANGLE, RECTANGLE, ELLIPSE)
dim	the shape dimension constant(s) that are applied to all shapes, or the name of the field(s) in the point attribute table containing the numerical shape dimension values (see full help documentation for details on what these values refer to)
out	the output polygon data source
[rot]	the rotation angle constant (degrees) that is applied to all shapes, or the name of the field in the point attribute table containing the rotation angle (degrees) (default=0)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
genshapes(in="C:\data\spnts.shp", shape=1, dim=120, out="myoutput.shp");
genshapes(in="C:\data\mygdb!mypnts, shape=6, dim=c(120, 40), rot=45,
out="myoutput.shp");
genshapes(in="C:\data\spnts.shp", shape=1, dim="MYFLD1", rot=180,
out="myoutput.shp");
genshapes(in="C:\data\referencelayer.shp", shape=6, dim=c("MYFLD1", "MYFLD2"),
out="myoutput.shp");
```

3.48 genstratrandompnts

Generate Stratified Random Points: Generates a sample of stratified random points (where strata are polygons), with various optional constraint parameters.

Description

This tool generates a sample of stratified random points where the strata are polygons. The command options can be combined in a variety of ways to produce different sampling outcomes. Although this tool is therefore very flexible, care must be taken to ensure that the parameters specified to not result in a problem that has no solution.

For each polygon in a polygon data source this tool will generate a sample of random points. The number of points generated can be a constant or variable between polygons. In the latter case the number of points to generate per polygon must be stored in a field in the polygon attribute table.

Points are generated using a simple rejection method algorithm: potential points are generated within the rectangular boundary that defines each polygon based on a bivariate uniform random distribution. They are then accepted if they conform to the acceptance criteria the user has defined.

The 'mindist' option will ensure that all output points are this minimum distance apart. This value should be specified in coordinate system units (e.g. meters for UTM). If this value is too large then the problem may have no solution. For example. it is not possible to generate 100 points in a square kilometer where the minimum distance between points is 500m.

The 'excl' option can be used to prevent points from being generated within the polygons of this dataset. For example, if you were to generate forest sampling points using a stand polygon layer, you might use a second polygon layer representing a 100m buffer of roads as the exclusion layer to prevent points from occurring too close to roads. Again, there is a risk here of creating an unsolvable problem: if the exclusion polygons cover all of the input polygons then no random points can be generated.

Syntax

genstratrandompnts(in, uidfield, sample, out, [mindist], [excl], [where]);

in	the input polygon data source
uidfield	the name of the unique polygon ID field
sample	the sample size: either a number representing the sample size per strata, or the field name of field containing the sample size per polygon, e.g. 100 or "NSAMPLES"
out	the output point data source
[mindist]	if specified, points are prevented from occurring within this minimum distance of each other (specified in coordinate system units of the reference layer); this option can be dangerous - see the help documentation for details
[excl]	the polygon data source containing exclusion polygons: points are prevented from being generated within these polygons; this option can be dangerous - see the help documentation for details
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

genstratrandompnts(in="C:\data\fields.shp", sample=1000, uidfield="FLDID", out="C:\data\samplepnts.shp");
genstratrandompnts(in="C:\data\plots.shp", sample=1000, uidfield="PLOTID", out="C:\data\samplepnts.shp", mindist=50, excl="C:\data\lakes.shp");

3.49 genvecgrid

Generate Vector Grid: Generates a vector grid (a series of regularly sized and spaced square/rectangular polygons).

Description

This tool generates a 'vector grid': a series of regularly sized and spaced square or rectangular polygons, lines or points. The extent of the vector grid can be specified by referencing a feature data source (from which the extent of the dataset is extracted), or by directly specifying the minimum and maximum x and y coordinates. The user also specifies the dimensions of the vector cells (which can be square or rectangular). The tool then fills the extent with cells of this size.

Note that if the output geometry is polygons and the vector cell size does not result in an exact fill of a row or column then the tool will always extend slightly beyond the defined extent to ensure all space is filled, rather than leave some space within the extent unfilled. Thus the extent of the output dataset may not match the extent you specified, but all space within the specified extent is filled.

There is an option to only keep the vector cells that overlap with the features in another layer. This can be particularly useful for generating sampling grids where sample plots are only needed/allowed in places where there is something to sample. Note that using this option can result in considerably longer processing times, especially if there are large numbers of features in the layer you specify for the overlap test.

The snap option forces the spatial position of the grid to be aligned with the underlying coordinate system. For instance, specifying snap=1000 would force the vector grid lines to be aligned with the exact 1km positions in a UTM grid.

The line geometry option results in what are effectively graticule lines. The point geometry option generates a grid of points, with the additional option that alternate rows can be offset (resulting in a honeycomb pattern rather than a chessboard pattern).

Syntax

genvecgrid(extent, dim, out, [geometry], [snap], [testoverlap], [offset]);

extent	the reference layer that defines the extent of the vector grid, or a set of four values that define the extent (min x, max x, min y, max y)
dim	the dimensions of each vector grid cell (either a single value representing the cell width and height, or two values representing the width and height respectively if they are not the same dimension)
out	the output feature data source
[geometry]	the type of output geometry (default=POLYGON, options: POINT, LINE, POLYGON)
[snap]	a value > 0; controls whether the vector grid is aligned with a major coordinate system interval (supplying a value of 1000 will result in the grid being aligned to the 1000-mark intervals of the coordinate system); default=0 (no snap)
[testoverlap]	only vector grid cells that overlap features in this feature data source will be retained
[offset]	(TRUE/FALSE; only applies to the geometry="POINT" option) offsets alternate rows of points by half of the x spacing (default=FALSE)"

Example

```
genvecgrid(extent=c(100000, 220000, 2000000, 2300000), dim=c(1000, 2000),
out="myoutput.shp");
genvecgrid(extent="C:\data\referencelayer.shp", dim=1000, out="myoutput.shp",
geometry="LINE", snap=100, testoverlap="C:\data\roads.shp");
genvecgrid(extent="C:\data\locs.shp", dim=c(0.05,0.1), out="myoutput.shp", snap=0.1,
usesel=TRUE, testoverlap="C:\data\rivers.shp");
genvecgrid(extent="C:\data\county.shp", dim=1000, out="pntgrid.shp",
geometry="POINT", offset=TRUE);
```

3.50 geom.clip

Clip Geometries: Clips the geometry of the input feature data source to polygons in the clip feature data source.

Description

This tool clips the features in the input feature data source to the polygons in the clip feature data source. All attribute data in the input table is copied to the output table.

The 'where' clause can be used to define a subset of polygon clip features to use. See the 'where' section for further details on how to formulate a where clause.

Syntax

geom.clip(in, clip, out, [where]);

in	the input feature data source
clip	the input polygon data source that is used to clip the input feature data source
out	the output feature data source
[where]	the selection statement that will be applied to the clip polygon feature data source to identify a subset of polygons to process (see full Help documentation for further details)

Example

```
geom.clip(in="C:\data\stands.shp", clip="C:\data\plots.shp",
out="C:\data\standsclipped.shp");
geom.clip(in="C:\data\stands.shp", clip="C:\data\plots.shp",
out="C:\data\standsclipped.shp", where="YEAR=2010");
```

3.51 geom.difference

Difference Between Geometries: Calculates the geometric difference between the input features and the polygons in a clip feature data source.

Description

This tool calculates the geometric difference of the features in the input feature data source to the polygons in the clip feature data source. All attribute data in the input table is copied to the output table. The geometric difference retains the feature data that is outside of the clip polygon boundaries, and is thus the opposite of the clip command.

This tool modifies the source data, and is therefore dangerous if not applied with caution. It is advised that you use the copyfeaturedataset command to make a backup of your input feature data source before using this command. As a safety precaution, you must set the

modifysource command option to TRUE to run this command. If you do not explicitly set this option then the command will not run.

By default the tool will write multipart geometries as output. However, if you specify the singlepart=TRUE option then any multipart geometries will be split into singlepart geometries and written back to the input data source as separate records.

The 'where' clause can be used to define a subset of polygon clip features to use. See the 'where' section for further details on how to formulate a where clause.

Syntax

geom.difference(in, clip, modifysource, [singlepart], [where]);

in	the input feature data source (WARNING: the input data source is modified; use copyfeaturedatasource first if you need to preserve the original)
clip	the input polygon data source that is used to clip the input feature data source
modifysource	(TRUE/FALSE; default=FALSE) a safety measure that forces you to explicitly authorize the source input data to be modified (the command does not run if set to false)
[singlepart]	(TRUE/FALSE; default=FALSE) if TRUE, forces the output geometries to be singlepart geometries
[where]	the selection statement that will be applied to the clip polygon feature data source to identify a subset of polygons to process (see full Help documentation for further details)

Example

geom.difference(in="C:\data\fields.shp", clip="C:\data\plots.shp", modifysource=TRUE);
geom.difference(in="C:\data\fields.shp", clip="C:\data\plots.shp", modifysource=TRUE, singlepart=TRUE, where="YEAR=2010");

3.52 geom.extractpolygoncomponents

Extract Polygon Components: Extracts the exterior or interior components of polygons containing holes.

Description

This tool allows you to create a new polygon layer consisting of only the exterior or interior components of the input polygons. Thus, it is only usefully applied to polygons containing 'holes' when you either want the outer edges of the polygon without preserving the holes, or you want to convert the holes to new polygons.

The 'where' clause can be used to define a subset of polygons to process. See the 'where' section for further details on how to formulate a where clause.

Syntax

geom.extractpolygoncomponents(in, out, extract, [copyfields], [where]);

in	the input polygon data source
out	the output polygon feature data source
extract	the keyword defining what components to extract (options: EXTERIOR, INTERIOR)
[copyfields]	(TRUE/FALSE): if TRUE the fields in the input attribute table are copied to the output attribute table (default=FALSE)
[where]	the selection statement that will be applied to the line feature data source to identify a subset of lines to process (see full Help documentation for further details)

Example

geom.extractpolygoncomponents(in="C:\data\lakes.shp", out="C:\data\shore.shp", extract="EXTERIOR");
geom.extractpolygoncomponents(in="C:\data\lakes.shp", out="C:\data\islands.shp", extract="INTERIOR", copyfields=TRUE, where="AREA > 10000");

3.53 geom.polygonfetch

Calculate Fetch In Polygons: Determines the longest line that can be position within the bounds of a polygon, crossing no edges.

Description

This tool estimates the longest straight line that can be positioned within a polygon, without crossing any edges. It corresponds to the 'fetch' of a lake: the longest stretch of water over which waves can build up as a function of wind action. The calculation is based on evaluating the lines created by connecting all pairs of non-neighbouring vertices and retaining the longest line that does not cross any interior or exterior boundaries of the polygons. This is a brute force algorithm that can take a long time for very complex polygons (e.g. polygons with more than 1000 vertices).

The 'where' clause can be used to define a subset of polygons to process. See the 'where' section for further details on how to formulate a where clause.

Syntax

geom.polygonfetch(in, uidfield, out, [where]);

in	the input polygon data source
uidfield	the unique ID field of the input feature data source
out	the output line data source
[where]	the selection statement that will be applied to the line feature data source to identify a subset of lines to process (see full Help documentation for further details)

Example

```
geom.polygonfetch(in="C:\data\lakes.shp", out="C:\data\fetch.shp",
uidfield="LAKEID");
geom.polygonfetch(in="C:\data\lakes.shp", out="C:\data\fetch.shp",
uidfield="LAKEID", where="AREA > 10000");
```

3.54 geom.splitpolysbylines

Split Polygons By Lines: Splits input polygons using lines.

Description

This tool splits the polygon features in the input data source based on the lines in another feature data source. All attribute data in the input table is copied to the output table. If an input polygon is not intersected by any lines, it is written to the output data source without modification. If a split results in multiple polygon fragments, each fragment is written as a new records in the output data source.

The 'where' clause can be used to define a subset of line features to use to split polygons. See the 'where' section for further details on how to formulate a where clause.

Syntax

```
geom.splitpolysbylines(in, line, out, [singlepart], [where]);
```

in	the input polygon data source
line	the input line data source that is used to split the input polygons
out	the output polygon feature data source
[singlepart]	(TRUE/FALSE; default=FALSE) if TRUE, forces the output geometries to be singlepart geometries
[where]	the selection statement that will be applied to the line feature data source to identify a subset of lines to process (see full Help documentation for further details)

Example

```
geom.splitpolysbylines(in="C:\data\plots.shp", line="C:\data\roads.shp",
out="C:\data\splitplots.shp");
geom.splitpolysbylines(in="C:\data\plots.shp", line="C:\data\roads.shp",
out="C:\data\splitplots.shp", where="TYPE='HWY'");
```

3.55 graph.createfrompoints

Generate Spatial Graph From Points: Creates a spatial graph by connecting points based on a distance threshold, and exports node and edge data that can be imported into R.

Description

This tool converts a point dataset representing graph nodes to a spatial graph by connecting nodes automatically using a distance threshold. The edges are output as a line feature dataset, and the nodes and edges are export as two delimited text files that can be easily read into R. The unique point ID field is preserved in the exported node data so that the connection with the original point data can be preserved. An optional node wieght field can also be specified, which results in an extra column of data being written to the export file.

The graph can be created in R using the igraph library as follows.

```
setwd("C:\\gme\\mygraphdata")
require(igraph)
edata <- read.csv("exportedges.csv")
ndata <- read.csv("exportnodes.csv")
graph1 <- graph.data.frame(edata[,-1], directed=FALSE, vertices=NULL)
```

Syntax

graph.createfrompoints(in, uidfield, distance, outline, exportnode, exportedge, [nodeweightfield], [where]);

in	the input point data source that represents graph nodes
uidfield	the unique feature ID field for the input data source
distance	the threshold distance for connecting nodes
outline	the output line feature data source
exportnode	the graph node export csv file
exportedge	the graph edge export csv file
[nodeweightfield]	a field name containing node weights to be included in the export file)
[where]	the selection statement that will be applied to the input feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

graph.createfrompoints(in="C:\data\points.shp", uidfield="PNTID",
outline="C:\data\graphedges.shp", exportnode="C:\data\exportnodes.csv",
exportedge="C:\data\exportedges.shp");
graph.createfrompoints(in="C:\data\patchpolygons.shp", uidfield="PATCHID",
outline="C:\data\graphedges.shp", exportnode="C:\data\exportnodes.csv",
exportedge="C:\data\exportedges.shp", nodeweightfield="POPULATION",
where="POPULATION > 0");

3.56 graph.createfrompolygons

Generate Spatial Graph From Polygons: Creates a spatial graph by connecting polygons based on a distance threshold, and exports node and edge data that can be imported

into R.

Description

This tool creates a spatial graph based on discontinuous (non-touching) polygons. (To create a graph based on polygon adjacency see the calc.sharedborders command.) To create the nodes a single point is generated to represent each polygon using either the centroid or label point methods (see the genpointinpoly command for a description of these terms). Nodes are connected if the shortest distance between the polygons they represent is no larger than the distance threshold specified. The command generates two types of edge outputs: the straight lines connecting the node points, and the lines (links) representing the shortest distance between polygons. The length of both edges and links is written to both attribute tables, and the edge export file.

Nodes and edges are export as two delimited text files that can be easily read into R. The unique point ID field is preserved in the exported node data so that the connection with the original point data can be preserved. An optional node wieght field can also be specified, which results in an extra column of data being written to the export file.

The graph can be created in R using the igraph library as follows.

```
setwd("C:\\gme\\mygraphdata")
require(igraph)
edata <- read.csv("exportedges.csv")
ndata <- read.csv("exportnodes.csv")
graph1 <- graph.data.frame(edata[,-1], directed=FALSE, vertices=NULL)
```

Syntax

graph.createfrompolygons(in, uidfield, distance, outline, outpoint, outlink, exportnode, exportedge, [nodeweightfield], [centroid], [where]);

in	the input polygon data source that represents graph nodes
uidfield	the unique feature ID field for the input data source
distance	the threshold distance for connecting nodes
outline	the output line feature data source
outpoint	the output point (node) feature data source
outlink	the output link (line) feature data source
exportnode	the graph node export csv file
exportedge	the graph edge export csv file
[nodeweightfield]	a field name containing node weights to be included in the export file)
[centroid]	(TRUE/FALSE) if TRUE uses the centroid point of the polygon for the node, if FALSE uses the label point)
[where]	the selection statement that will be applied to the input feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

graph.createfrompolygons(in="C:\data\points.shp", uidfield="PNTID",
outline="C:\data\graphedges.shp", outlink="C:\data\graphlinks.shp",
exportnode="C:\data\exportnodes.csv", exportedge="C:\data\exportedges.shp");
graph.createfrompolygons(in="C:\data\patchpolygons.shp", uidfield="PATCHID",
outline="C:\data\graphedges.shp", outlink="C:\data\graphlinks.shp",
exportnode="C:\data\exportnodes.csv", exportedge="C:\data\exportedges.shp",
nodeweightfield="AREA", where="POPULATION > 0");

3.57 import.asciigrid

Import Raster From ASCII Grid: Imports an ASCII grid file to a binary raster format (GRID, TIFF, IMG).

Description

This tool imports an ASCII grid file to a binary raster format (Grid, GeoTIFF, or Imagine Image format). The ASCII raster file must conform to certain standards, although the tool does allow some flexibility in the specification of the header information.

ASCII raster files begin with 5 or 6 lines of header information that determine the extent and orientation of the raster. For example:
ncols 43200
nrows 18000
xllcorner -180
yllcorner -60
cellsize 0.0083333337679505
NODATA_value -9999
-9999 -9999 ...

The Nodata line is optional, although it is wise to specify it to prevent default values from being used that may conflict with your data. The header lines can also be specified with each line bracketed within <...>, e.g. <ncols 43200>. White space (spaces, tabs) on the header lines should not influence the ability of the tool to extract the header data.

To determine if your ASCII file meets the appropriate formatting criteria, you can open the ASCII grid file in a text editor (e.g. Notepad++) if it is small enough (no more than a few megabytes usually). For larger files the GME file.readlines command is an easy way of viewing the header. E.g.:
file.readlines(file="C:\data\climate.asc", start=1, end=6);

You must specify the appropriate output raster pixel type. The options are double precision (range: any real number), long integer (range: +/- 2.1 billion), short integer (range: +/- 32768), or byte (range: 0-255). You should be able to determine the appropriate pixel type from the metadata associated with the ASCII grid file. Failing that, you can inspect the file to attempt to determine the range. Again, the GME file.readlines command is useful for inspecting small portions of even very large files. E.g.:

file.readlines(file="C:\data\climate.asc", start=5000, end=5001);
Note that you will usually only want to inspect one line at a time using this command (each line contains a lot of data). If the output type is DOUBLE, then you must specify the output as a IMG format raster.

Ideally, you should select the smallest output pixel data type for your date (byte < short < long < double in terms of the number of bytes of storage space required to store a single value. Although double is the most flexible format in that it can store almost any number, it also requires several times more disk space than the other types.

The projection information for the data must also be provided. You need to reference an ESRI .prj file. You can either reference an prj file from a dataset you know to be in the same projection, or copy the appropriate prj file from the
C:\Program Files\ArcGIS\Desktop10.0\Coordinate Systems
folder. It is useful to copy the file locally to the same folder as the grid files. See the Projection Definition File section for more details on working with and creating projection files.

Syntax

import.asciigrid(in, prj, out, type, [delimiter]);

in	the input ASCII grid file
prj	the file containing the projection data of the input raster
out	the output raster, including the extension in the case of an IMG or TIF formats
type	the output raster pixel data type (options: DOUBLE, LONG, SHORT, CHAR)
[delimiter]	the delimiting character(s) - normally one of these keywords: SPACE (default), COMMA, TAB, SEMICOLON, or COLON (see full help documentation for further details)

Example

import.asciigrid(in="C:\data\climate.asc", prj="C:\data\WGS 1984.prj",
out="C:\data\climate.img", type="LONG");

3.58 import.hadisst

Import Raster From HADISST: Imports seas surface temperature rasters from HDF format to an ArcGIS supported raster format.

Description

This is a highly specialised tool that imports seas surface temperature rasters from HDF format to a raster format that is supported by ArcGIS. This website describes the dataset this tool is targeted at: http://badc.nerc.ac.uk/data/hadisst.

To use this tool you should download all of the monthly global datasets you are interested in (that may be several hundred) into a single folder. This tool will process all of the HDF files it finds in the specified folder, import the data to grid format, georeference the rasters, and build the statistics on each raster so that they display correctly.

Syntax

import.hadisst(in, out);
in the input folder containing the HadiSST HDF files
out the output folder for the imported rasters (a new, empty folder is strongly recommended)

Example

import.hadisst(in="C:\data\hadisst", out="C:\data\imported");

3.59 isectfeatures

Intersect Features: Calculates intersections between geometries (points, lines or polygons).

Description

This tool calculates intersections between geometries (points, lines or polygons). If you specify a single input feature data source, then it calculates intersections between geometries within that dataset. If you specify two input datasets then the tool calculates intersections between each feature in the first dataset with the features in the second dataset.

The 'dimension' option allows you to control the dimension of the geometry returned: 0 refers to points, 1 refers to lines, and 2 refers to polygons. An intersection between points and any other geometry can only result in point geometries. Lines intersecting with lines can return either point geometries (where two lines cross), or line geometries (where two lines overlap exactly for a non-zero distance). Polygons intersecting with lines can result in either points (where the boundary of the polygon overlaps with the lines) or lines (the portion of the line that is contained by the polygon). And polygons intersecting with polygons can result in polygons (where the two polygons overlap) or points (there the boundaries of the polygons cross).

The output layer will contain the unique ID numbers of each of the geometries that were used to calculate the intersections, hence the need to specify a unique ID field for the input data sources. This means that you can join the attributes of the original tables to the output of this tool if you need to.

For within-layer intersections, the tool will only calculate an intersection once. If a layers has two features, A and B, the tool will start by calculating the intersection between A and B, but will then not calculate the intersection between B and A as this would create duplication in the output layer.

You may get unexpected results from this tool. I have noticed that calculations between two layers returns the geometries you would typically expect, but within-layer intersections sometimes do not. This is perhaps most apparent when calculating the point intersections within a single line layer (e.g. a roads layer). It appears that when the endpoints of two lines are exactly aligned the ESRI Intersect calculation will sometimes identify this as an intersection and return a point, but sometimes will not. It is not clear to me why this is the case, but it is recommended that you inspect the output of this tool carefully to determine if it meets with your expectations.

Intersections can be computationally expensive, therefore it is recommended that you consider taking steps to increase processing speed, especially for large datasets. Ensuring that your feature dataset has a spatial index can improve search speed (see the Add Spatial Index tool in ArcToolbox). You might also consider eliminating any features that are known to have no overlap with the features in another layer. For instance, the ArcMap Select By Location tool allows you to easily select overlapping features between two layers, which you can then export into two new (temporary) layers that are the input to this tool. Reducing the number of features in the layers will thus also improve search speeds.

Syntax

isectfeatures(in, uidfield, out, [in2], [uidfield2], [dimension], [where]);

in	the input feature data source
uidfield	the unique ID field
out	the output feature data source
[in2]	the second input feature data source (if specified, calculates intersections between layers rather than within a single layer)
[uidfield2]	the unique ID field for the second data source (if specified)
[dimension]	the dimension of the output (0=point, 1=line, 2=poly); default: 0, 0 and 2 for point, line and polygon intersections respectively, and the default cannot be overridden if it is nonsensical, e.g. always 0 for point intersections (options: 0, 1, 2)
[where]	the filter/selection statement that will be applied to the first ('in', not 'in2') polygon feature class to identify a subset of polygons to process

Example

```
isectfeatures(in="C:\data\roads.shp", uidfield="ROADID",
out="C:\data\rdintersections.shp");
isectfeatures(in="C:\data\roads.shp", uidfield="ROADID", in2="C:\data\rivers.shp",
uidfield2="RIVERID", out="C:\data\rd_river_crossings.shp");
isectfeatures(in="C:\data\plots.shp", uidfield="PLOTID", in2="C:\data\lakes.shp",
uidfield2="LAKEID", out="C:\data\plot_water.shp", dimension=2);
```

3.60 isectlinerst

Intersect Lines With Raster: Creates a statistical or frequency summary of raster data along a line (polyline) based on a raster layer.

Description

This tool creates a summary for a line (polyline) based on a raster layer. The output consists of summary fields that are added to the line attribute table. It processes thematic (i.e. categorical) and continuous rasters in different ways. For thematic rasters, a new field is added for each unique value in the raster and is populated with the length of each line passing though raster cells of that type. The tool also records the cell values for the start and end of the line. For continuous rasters the summary includes the length weighted mean of raster values along the line, the minimum and maximum values encountered, and the cell values for the start and end of the line.

Note that the tool cannot distinguish between thematic and categorical rasters automatically. By default it assumes the raster represents continuous data, so if the raster is a thematic raster you must explicitly use the 'thematic=TRUE' option. For thematic rasters there is an option to report the proportion of the line that falls within each of the cell values instead of the length of the line (use the 'proportion' option). If you use the 'proportion' option with a continuous raster then it is ignored.

NoData cells are ignored. Therefore, of the line crosses NoData cells this may influence the accuracy of the statistical summary. It is highly recommended you ensure that raster data exists for the entire area covered by the lines. Also, any lines that are entirely or partially outside of the extent of the raster will not be processed - they will be coded with NoData values (-2147483648) so should be easily identifiable.

Syntax

isectlinerst(in, raster, prefix, [thematic], [proportion], [where]);

in	the input line data source
raster	the input raster data source
prefix	a short prefix to use in the summary statistic fields that are added, which end with MIN, MAX, LWM, BEG, END for continuous raster data, and V# for thematic raster data where # is each of the unique cell values. The prefix should be no longer than 6 characters, and should be related in some way to the raster dataset.
[thematic]	(TRUE/FALSE) controls whether the raster should be treated as a thematic (categorical) raster (default=FALSE)
[proportion]	(TRUE/FALSE) only applies to thematic rasters: records the proportion of the line in each cell value instead of the length of the line (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

isectlinerst(in="C:\data\lines.shp", raster="C:\data\landcov", prefix="LCOV", thematic=TRUE);
isectlinerst(in="C:\data\lines.shp", raster="C:\data\landcov", prefix="LCOV", thematic=TRUE, proportion=TRUE);
isectlinerst(in="C:\data\lines.shp", raster="C:\data\dem", prefix="DEM");

3.61 isectpntpoly

Intersect Points With Polygons: Acquires attribute data from a polygon data source and writes it to the input point data source.

Description

This tool writes data from a polygon attribute table to the attribute table of the input point data source. The tool finds the polygon that intersects each point, and acquires the attribute data from that polygon. If polygons overlap and a point intersects multiple polygons the tool just acquires the data from the first polygon it finds.

The tool can acquire data from more than one field in the polygon table at one time. If a single field name is provided (e.g. field="NAME") then only that one field is recorded. To specify multiple fields use the list format, e.g. field=c("NAME", "HEIGHT", "COST"). This tool does not support geometry or blob fields, and if you specify a unique ID field (like FID or OID) then it will be written as a long integer field and renamed so that it does not conflict with the unique ID fields already in the point attribute table.

It is important that the spatial references are defined and are identical for both layers.

Syntax

isectpntpoly(in, poly, field, [where]);

in	the input point data source
poly	the polygon data source
field	the field(s) in this polygon layer to acquire (either one field or a list of fields)
[where]	the filter/selection statement that will be applied to the point feature class to identify a subset of points to process

Example

isectpntpoly(in="C:\data\samples.shp", poly="C:\data\soils.shp", field="SOILTYPE");
isectpntpoly(in="C:\data\locs.shp", poly="C:\data\forestunits.shp", field=c("PRIMARY","SECONDARY"));
isectpntpoly(in="C:\data\wells.shp", poly="C:\data\census.shp", field="POPSIZE");

3.62 isectpntrst

Intersect Points With Raster: Intersect points with a raster layer and writes the cell value of the cell containing each point to the attribute table of the point data source.

Description

This tool intersect points with a raster layer and writes the cell value of the cell containing each point to the attribute table of the point data source. If the specified raster contains multiple bands, the tool returns one field for each band.

Note that points that fall outside the bounds of the raster layer will be coded with the rasters NoData value. If this is not defined the cell will be coded with the smallest value supported by the pixel data type, which may be zero. It is therefore highly recommended that you explicitly address the issue of points that fall outside the bounds of the raster to ensure that these NoData points do not bias any subsequent analysis you might do. Zero values can be mistaken for true values in some cases, so please pay special attention to this problem.

The fully supported common raster types are Grid, Imagine image (.img) GeoTIFF (.tif), and geodatabase rasters. Other image types are not supported by ArcGIS as 'raster datasets' and are unlikely to work with this tool (e.g. JPEG, PNG, MrSID, etc) even though they can be displayed in ArcGIS.

Syntax

isectpntrst(in, raster, field, [where], [update]);

in	the input point data source
raster	the input raster data source
field	the new field name to write to the point attribute table and populate with raster values (if the raster has multiple bands the field name is treated as a prefix to which B# is appended, where # is the band number)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)
[update]	(TRUE/FALSE) if TRUE and you specify an existing field, the existing field will be updated rather than generating an error message (default=FALSE); warning: this option will result in overwriting of existing data and is therefore potentially dangerous.

Example

isectpntrst(in="C:\data\locs.shp", raster="C:\data\landcov", field="LCOV");
isectpntrst(in="C:\data\locs.shp", raster="C:\data\landcov", field="LCOV");
isectpntrst(in="C:\data\locs.shp", raster="C:\data\landsattm.img", field="TM5",
where="COUNTY='WOOD' AND MONTH=7", update=TRUE);

3.63 isectpolypoly

Intersect Polygons With Polygons: Generate summary statistics for one polygon dataset based on another polygon dataset.

Description

For each polygon in a 'zonal' polygon dataset that defines the area over which a summary is to be made, this tool summarizes a thematic or continuous attribute field in another polygon dataset and writes the results to the attribute table of the zonal polygon dataset. Examples of zonal polygon data include sampling plots, counties, school catchment areas, watershed, etc. The other polygon data contains the data in an attribute field. Examples of continuous data include soil pH in a soils dataset or population size in a census dataset. Examples of thematic (categorical or ordinal) data include landcover code or zoning code.

For continuous data this tool creates a statistical summary of the field including the area-weighted mean (AWM), minimum (MIN), maximum (MAX), and area weighted sum (AWS). If you only need a subset of these statistics then you can use the optional awm, min, max and aws parameters to enable/disable reporting of each of these summary statistics. It is unlikely that all four summary statistics will be appropriate for any particular analysis. By default all four are enabled. For thematic data, the tool reports the area of each category present in the zonal polygon. Because the tool cannot automatically distinguish between continuous and thematic data, you must use the 'thematic' parameter if you wish to obtain the thematic summary; by default the tool assumes that the field represents continuous data. For thematic summaries you also have the option of reporting the proportion of the zonal polygon occupied by each category rather than the area (see the 'proportion' parameter).

If v is the continuous value specified by the 'field' parameter, c is the area of the (clipped) continuous data polygon falling within the zonal polygon, and A is the total area of the (unclipped) continuous data polygon, then the area weighted mean is calculated as sum(v*c)/sum(c), and the area weighted sum is calculated as sum(v*c/A). In these formulae sum iterates over the set of polygons that overlap that zonal polygon.

Syntax

isectpolypoly(in, poly, field, prefix, [thematic], [proportion], [where], [awm], [min], [max], [aws]);

in	the input zonal polygon data source
poly	the polygon data source containing the quantitative data to summarize
field	the field in this second polygon layer to process
prefix	a short prefix to use in the summary statistic fields that are added, which end with AWM, MIN, MAX for continuous data, and V# for thematic data where # is each of the unique values. The prefix should be no longer than 6 characters.
[thematic]	(TRUE/FALSE) controls whether the polygon data should be treated as thematic (categorical) (default=FALSE)
[proportion]	(TRUE/FALSE) only applies to thematic data: records the proportion of the polygon in each unique value instead of the area of the polygon (default=FALSE)
[where]	the filter/selection statement that will be applied to the first ('in', not 'poly') polygon feature class to identify a subset of polygons to process
[awm]	(TRUE/FALSE) applies only to statistical summaries, if false, the area weighted mean statistic is not added to the attribute table (default=TRUE)
[min]	(TRUE/FALSE) applies only to statistical summaries, if false, the minimum statistic is not added to the attribute table (default=TRUE)
[max]	(TRUE/FALSE) applies only to statistical summaries, if false, the maximum statistic is not added to the attribute table (default=TRUE)
[aws]	(TRUE/FALSE) applies only to statistical summaries, if false, the area weighted sum statistic is not added to the attribute table (default=FALSE)

Example

isectpolypoly(in="C:\data\fields.shp", poly="C:\data\soils.shp", field="SOILTYPE", prefix="SOIL", thematic=TRUE);
isectpolypoly(in="C:\data\plots.shp", poly="C:\data\lcov.shp", field="LANDCOVER", thematic=TRUE, proportion=TRUE);
isectpolypoly(in="C:\data\counties.shp", poly="C:\data\census.shp", field="POPSIZE", awm=FALSE, min=FALSE, max=FALSE, aws=TRUE);

3.64 isectpolyrst

Intersect Polygons With Raster: Creates summaries for each polygon based on the values in a raster layer.

Description

This tool summarizes the raster cell values that are contained by a polygon. The output consists of summary fields that are added to the polygon attribute table. It processes thematic (i.e. categorical) and continuous rasters in different ways. For thematic rasters, a new field is added for each unique value in the raster and is populated with either a count of the

number of cells of each raster value within that polygon, or the proportion of cells of each raster value. For continuous rasters a statistical summary is produced (the mean, median, minimum, maximum, standard deviation, count, and the 95% quantiles).

Polygons are processed sequentially, so are not affected in any way by overlapping polygons (in contrast to related tools in ArcGIS). This means that it can take a considerable time to process many polygons.

This tool cannot distinguish between thematic and categorical rasters automatically. By default it assumes the raster represents continuous data, so if the raster is a thematic raster you must explicitly use the 'thematic=TRUE' option.

NoData cells are ignored. Therefore, if the polygon contains NoData cells this may influence the accuracy of the statistical summary. It is highly recommended you ensure that raster data exists for the entire area covered by the polygons. By default, any polygons that are entirely or partially outside of the extent of the raster will not be processed - they will be coded with NoData values (-2147483648) so should be easily identifiable. This behaviour is a precaution against potentially biased estimates, but can be overridden using the allowpartialoverlap parameter. Setting this parameter to TRUE will force all polygons to be processed, even if they only partially overlap the extent of the raster dataset.

Note that this single command provides the same functionality as both the 'Zonal Statistics ++' and 'Thematic Raster Summary' tools in HawthsTools.

Recent changes to the interface with R mean that the median and quantiles options (calculated of medquant=TRUE) will tend to run quite slowly. I am working on resolving this speed issue for a future update.

Syntax

isectpolyrst(in, raster, prefix, [thematic], [proportion], [metrics], [allowpartialoverlap], [medquant], [where]);

in	the input polygon data source
raster	the input raster data source
prefix	a short prefix to use in the summary statistic fields that are added, which end with MN, MIN, MAX, STD, CNT, MED for continuous raster data, and V# for thematic raster data where # is each of the unique cell values. The prefix should be no longer than 6 characters, and should be related in some way to the raster dataset.
[thematic]	(TRUE/FALSE) controls whether the raster should be treated as a thematic (categorical) raster (default=FALSE)
[proportion]	(TRUE/FALSE) only applies to thematic rasters: records the proportion of the polygon in each cell value rather than the area of the polygon (default=FALSE)
[metrics]	the statistical metric(s) to calculate, expressed as a single value (e.g. "MIN") or as a vector of metrics (e.g. c("MN","STD","CNT")); options are MN/MEAN, MIN, MAX, STD, SUM, CNT, MED, QUPP, QLOW (see help for details)
[allowpartialoverlap]	(TRUE/FALSE) if TRUE, will process polygons that only partially overlap the extent of the raster dataset (default=FALSE)
[medquant]	(TRUE/FALSE) controls whether the R derived median and quantiles are included in the statistical summary statistics (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

isectpolyrst(in="C:\data\fields.shp", raster="C:\data\landcov", prefix="LCOV", thematic=TRUE);
isectpolyrst(in="C:\data\plots.shp", raster="C:\data\landcov", prefix="LCOV", thematic=TRUE, proportion=TRUE);
isectpolyrst(in="C:\data\parcels.shp", raster="C:\data\dem", prefix="DEM");

3.65 isopleth

Isopleths: Generares isopleth lines based on a raster representing a probability surface.

Description

This tool creates isopleth lines based on a raster dataset representing a probability surface. Isopleths represent the boundary lines that contain a specified volume of a surface. For instance, the 0.95 isopleth represents the line containing 95% of the volume of the surface. The isopleths are specified as 'quantiles' that can be expressed either as a proportion, or as a percentage. If the list of quantiles contains any values greater than 1, the list is interpreted as containing percentages. If all the values are in the range 0-1, the quantiles are interpreted as proportions.

This algorithm produces higher-quality isopleth lines at the expense of processing time, and is the same algorithm that is used in the contour command. It interprets the raster as a triangular regular network and performs a plane intersection to determine the paths of the contour lines. It does not perform any sort of smoothing on the line.

While lines are always output from this command, you can also optionally specify that polygons should be created. This does not, however, guarantee that a polygon will be created for every isopleth. For a polygon to be created, the line must form a complete loop. For the larger quantiles the isopleths may not form complete loops if the raster has been generated over too small an extent. It would be inappropriate for a tool to close those isopleths based on the boundary of the raster extent, as these boundaries do not represent the true position of those isopleths. Rather than close isopleth lines inappropriately, the tool only converts the complete loops to polygons and reports a warning message that some lines were not closed. If you receive that message it is recommended that you regenerate the raster surface over a larger extent. I have added an 'edgeinflation' parameter to the kde command to assist with this problem.

Two types of polygon outputs are available: full polygons or donut polygons. For full polygons the smaller isopleth polygons will overlap (and be contained by) the polygons for larger isopleth values. In the case of donut polygons, only the portion of the polygon representing the interval between two isopleths is retained, and there are no overalapping polygons. Always specify different output polygon datasets for the poly and donut parameters (never write both to the same dataset).

Although a 'band' option is included so that users can identify which band to process in the case of multiband images, it is likely that this tool will most often be run with continuous, single-band raster data.

Syntax

isopleth(in, out, quantiles, [band], [poly], [donut]);

in	the input integer raster data source
out	the output line data source
quantiles	the quantiles (expressed as a proportion or percentage) at which to generate isopleths, e.g. 0.95 or c(0.5, 0.9, 0.95)
[band]	the input band (default=1)
[poly]	an output polygon data source (the isopleths are converted to polygons; refer to full documentation for details)
[donut]	an output polygon data source to which the donut polygons are written (the isopleths are converted to polygons; refer to full documentation for details)

Example

```
isopleth(in="C:\data\kde", out="C:\data\isopleths.shp", quantiles=c(0.5, 0.9, 0.95));
isopleth(in="C:\data\kde", out="C:\data\isopleths.shp",
poly="C:\data\isopleths_poly.shp", quantiles=0.99);
isopleth(in="C:\data\kde", out="C:\data\isopleths.shp", quantiles=r.eval(seq(0,100,10)));
```

3.66 julian

Julian Day: Returns the Julian day for a specified date.

Description

This very simple tool returns the Julian day for a specified date. The date can be specified in a number of formats (see example below).

Syntax

julian(date);
date the date to convert to julian day (this can be expressed in a vaiety of ways), e.g. "10 Jan 2012"

Example

julian(date="15/2/2007");
julian(date="October 15 1999");
julian(date="4 April 2007");

Result of the third example:
Wednesday, April 04, 2007 is Julian day 94

3.67 kde

Kernel Density Estimation: Calculates kernel density estimates based on a set of input points.

Description

This tool calculates kernel density estimates based on a set of input points. This tool implements three types of kernel: Gaussian (bivariate normal), quartic, and uniform. The quartic kernel is an approximation to the Gaussian kernel that is used because it is computationally simpler and faster. However, I would suggest that for most scientific applications there is little justification for using the quartic kernel over the Gaussian kernel. The Gaussian kernel is the default in this tool, although the quartic kernel has been included in order to allow users to make comparisons with software packages that calculate the quartic kernel.

The bandwidth you provide will depend on the type of kernel used in the calculation. If the kernel is bivariate normal the bandwidth is the covariance matrix for a bivariate normal distribution. Although this is a 2x2 matrix, you need only provide three parameters because the two parameters representing the covariance between x and y are identical. The three parameters needed are thus: the standard deviation for x, the standard deviation for y, and the covariance. Note that some software packages require you to provide a bandwidth parameter, h, while others require h^2. Although h is smaller than h^2 and therefore easier to work with, h^2 is the correct representation for the covariance matrix. It is important to

be aware of how the bandwidth is represented when comparing the output from different software packages.

The command also implements several bandwidth estimation algorithms in the 'ks' library in R, which will estimate an optimized bandwidth matrix for you. The algorithms available are the plug-in estimator (bandwidth="PLUGIN"), smoothed cross validation (bandwidth="SCV"), biased cross-validation (bandwidth="BCV"), a second BCV algorithm (bandwidth="BCV2"), and least squares cross validation (bandwidth="LSCV"). Note that there can be large differences among the different bandwidth estimators, so it is recommended that you try several of them in order to determine which is most biologically relevant to your data and question. I have found the plug-in and SCV algorithms seem to perform very well. Processing times may be long with large numbers of points (thousands). Note also that some algorithms (e.g. LSCV) are sensitive to points with identical coordinates. You might consider adding a small amount of random noise to your points in R if this is an issue with your dataset. The bandwidths are calculated using the default settings for these estimators in the ks library in R.

If the kernel is the quartic approximation to the bivariate normal distribution, then you only specify a single value that represents the radius beyond which the density estimate for the kernel is 0. Thus, the quartic kernel bandwidth parameters corresponds to a real distance on the ground, unlike the bandwidth for the bivariate normal kernel which is a covariance matrix. Thus, these two bandwidths do not directly map. You cannot, for instance, estimate the optimal bandwidth using a bivariate normal kernel algorithm (like least squared cross validation) and then use it in a quartic kernel calculation: the optimal bandwidth for the quartic kernel will be very different.

The uniform kernel corresponds to what is also sometimes referred to as 'simple density'. The bandwidth represents the radius of a circle within which points are counted around each cell. The density value is simply n / pi*h^2, where n is the nunmber of points in the circle.

It takes some experience to learn what suitable cell size values are. A cell size that is too large will result in a 'blocky' output raster that is a poor statistical approximation to a continuous surface. A cell size that is too small will result in a very large output raster (many cells) that takes a long time to calculate. I suggest the following rule of thumb to calculate a reasonable bandwidth: take the square root of the x or y variance value (whichever is smaller) and divide by 5 or 10 (I usually round to the nearest big number - so 36.7 becomes 40). Before using this rule of thumb value calculate how many cells this will result in for the output (take the width and height of you input points, divide by the cell size, and multiply the resulting numbers together). If you get a value somewhere between 1-20 million, then you have a reasonable value. If you have a value much larger then 20 million cells then consider increasing the cell size.

A scaling factor is often used in KDE calculations to prevent a loss of precision in density values. Point density values are often very small numbers, and some raster formats do not support double-precision values (the Imagine img format is the only format that does, and for that reason I recommend it as the format for the output raster). The scaling factor is just a value that the point density values are multiplied to make them larger. The default is 1000000. Again, scaling factors may vary between software packages and this is something that must be considered when making comparisons.

Weights can be optionally specified for each point via a field in the attribute table. The kernel based on each point is weighted by the value in this field, and the density estimates are standardized by dividing by the sum of the weight values. Thus, you do not need to standardize the weights yourself. By default (when no weight field is specified) all points are weighted equally.

By default the output extent is automatically calculated as the extent of the input point dataset plus a suitable buffer distance that ensures the density surface is not unduly truncated at the edges. However, you may override this extent using the 'ext' option which requires that you specify either a reference geospatial layer (vector or raster dataset), or the minimum x, maximum x, minimum y, and maximum y coordinates of the desired extent.

Syntax

kde(in, out, bandwidth, cellsize, [weightfield], [scalingfactor], [kernel], [ext], [edgeinflation], [where]);

in	the input point data source
out	the output raster data source
bandwidth	the bandwidth (see the help documentation for details); bandwidth estimators include: SCV, BCV, BCV2, PLUGIN, LSCV, CVh
cellsize	the cell size dimension of the output raster
[weightfield]	the name of a field in the input table that contains the weighting value for each point (the default is no field and equal weighting of all points)
[scalingfactor]	multiplies densities by this value - see help for details (default=1000000)
[kernel]	kernel type (default=GAUSSIAN; options: GAUSSIAN, QUARTIC, UNIFORM)
[ext]	the input reference data source (vector or raster dataset), or coordinates, that defines the analysis extent
[edgeinflation]	the extra distance to add to each side of the output raster extent when determining raster dimensions (default=0)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

kde(in="C:\data\locs.shp", out="C:\data\kdeloc1.img", bandwidth=c(10000,10000,0), cellsize=20);
kde(in="C:\data\locs.shp", out="C:\data\kdeloc2", bandwidth=10000, cellsize=20, weightfield="WEIGHTS");
kde(in="C:\data\locs.shp", out="C:\data\kdeloc3", bandwidth=40000, cellsize=20, kernel="UNIFORM");
kde(in="C:\data\locs.shp", out="C:\data\kdeloc4.tif", bandwidth=500, cellsize=50, kernel="QUARTIC", edgeinflation=200);

3.68 kmeans

K-means Classification: Performs k-means clustering on a set of fields in a feature data source table.

Description

This tool performs k-means clustering on a set of fields in a feature data source table (e.g. a point attribute table). K-means clustering is a statistical method of grouping data. The user specifies the number of groups and the fields that contain the relevant data, and the algorithm groups similar records into each of those groups. You can specify any number of fields to be used in the clustering process. Although this is not a spatial algorithm, you can supply spatial data to the algorithm. For instance, given a point data set the x and y coordinates of the points could be supplied to the algorithm and the result would be spatial groupings of locations.

It is a stochastic process so may not yield exactly the same results each time you run it unless there are clear and obvious groupings in your data that correspond to the number of groups you have identified. The stochasticity arises because the initial 'centres' of each of the clusters are generated randomly in n-dimensional space. The 'iters' parameter corresponds to the 'iters.max' parameter in the R kmeans command, and controls the maximum number of iterations that are attempted when searching for a better model.

The output comma delimited text file this command creates contains the summary of the statistical output of the tool: the number of records in each group, the within group sums of squares, and the coordinates of the centres of the clusters in n-dimensional space.

For further information on the R kmeans command, type '? kmeans' at the R prompt, and press Enter.

This command is driven by R. Type 'citation' to see the suggested citation for R.

Syntax

kmeans(in, k, flds, outfld, [iters], [outfile], [where]);

in	the input feature data source
k	the number of groups into which the data is partitioned
flds	a list of the numerical fields that the algorithm is based on
outfld	the output field that will contain the group membership ID numbers
[iters]	the maximum number of iterations allowed
[outfile]	if specified the cluster size and within group sums of squares are written to this delimited text file
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

kmeans(in="C:\data\plots.shp", k=10, flds=c("X","Y","NDVI","Slope"),
outfld="CLUSTER");
kmeans(in="C:\data\mypoints.shp", k=100, flds=c("X","Y"), outfld="CLUSTER",
iters=100, outfile="C:\data\kmeansdata.csv");

3.69 licensestatus

License Status: Reports the current availability and product level of ESRI ArcGIS licenses.

Description

This command reports the current availability and product level of ESRI ArcGIS licenses on your system. GME only requires an ArcView license, but will search for any license that is available on a users system in this order: ArcView, ArcEditor, ArcInfo, ArcServer, ArcEngine. It will use the first one from this list that is available. If none are available then you will not be able to run GME.

I do not know whether GME will run with an ArcServer or ArcEngine license (I suspect not). But I have no means of testing this and it seemed prudent to at least try in the code. I would be interested in hearing from you if you can test this.

Syntax

licensestatus);

Example

licensestatus

3.70 lineofsight2d

Line Of Sight (2D): Generates polygons representing visible areas from source viewpoints given a polygon barrier dataset.

Description

This tool defines the area that is visible from a source point given a set of barriers (polygons) that block the line of sight. The outer limits of the visible area are defined by a radius the user specifies. If multiple viewpoints are specified, then the output can be specified as independent polygons, the union of these polygons, or the intersection of these polygons. This facilitates answering questions such as 'What area can be seen from any viewpoint?' and 'What area can be seen from all viewpoints simultaneously?'.

This is a two dimensional vector model of visibility in contrast to the three dimensional raster DEM based model of visibility. There are advantages and disadvantages to both approaches. The advantage of this 2D vector model is that it may be more precise (often the horizontal extent of objects is mapped more precisely than the vertical dimension). The disadvantage is that it does not account for elevation, rather it assumes that all barriers block visibility.

Syntax

lineofsight2d(poly, radius, out, [pnt], [source], [independent], [union], [intersect], [where]);

poly	the barrier polygon data source
radius	the maximum radius around source points out to which the visibility polygons is calculated
out	the output polygon data source
[pnt]	input point data source representing the locations from which visibility polygons are calculated (see the 'source' option as an alternative to this parameter)
[source]	the coordinates of the location from which the visibility polygon is calculated specified using the c(x, y) syntax (see the 'pnt' for an alternative way of specifying this location
[independent]	(TRUE/FALSE) if true, one polygon is produced for each input point regardless (polygons are independent of one another and may overlap) (default=TRUE)
[union]	(TRUE/FALSE) if true, one polygon is produced as output by unioning all the visibility polygons, thereby representing all areas that can be seen from at least one source point (default=FALSE)
[intersect]	(TRUE/FALSE) if true, one polygon is produced as output by intersecting all the visibility polygons, thereby representing the areas that can be seen from all source points (default=FALSE)
[where]	the filter/selection statement that will be applied to the input polygon data source to identify a subset of features to process

Example

lineofsight2d(poly="C:\data\islands.shp", radius=20000, out="C:\data\visible.shp", pnt="C:\data\observers.shp", union=TRUE);
lineofsight2d(poly="C:\data\obstacles.shp", radius=1000, out="C:\data\visible.shp", source=c(123456.1,9876543.9));
lineofsight2d(poly="C:\data\barriers.shp", radius=50000, out="C:\data\visible.shp", pnt="C:\data\targets.shp", intersect=TRUE);

3.71 list.raster

List Raster Datasets: Creates a list of raster datasets in folders and subfolders matching user-specified search criteria.

Description

The command searches a folder (and optionally all subfolders and geodatabases) for raster datasets, storing the results in a variable in GME (a list of strings containing the full path to each raster dataset, with a default name of "rasterlist"). If a 'match' string is specified, only raster datasets with names matching this search string are returned (e.g. if match="b*" only raster datasets beginning with the letter b are returned).

When using the list of datasets in scripting and automation it is often useful to have a corresponding list that contains only the name of the raster dataset. If a value is specified for the namevariable parameter, a second variable in GME is created that contains only the dataset name (with no extension). For instance, if the first dataset in the returned list is "C:\data\landcover.img", the corresponding value in the namevariable list is "landcover". This means that the raster dataset name can be easily retrieved and used in naming output files in a script.

This command applies only to Grids, GeoTIFFs (*.tif), Imagine Image (*.img), and geodatabase raster formats. By default, geodatabases are not searched (as this takes much more time, and storing raster datasets in geodatabases is both rare and undesirable).

The command will print the resulting list to the output window. For very large lists this may be unwanted, and it can be switched off by setting print=FALSE.

Syntax

list.raster(in, [match], [subfolders], [geodatabases], [variable], [namevariable], [print]);

in	the folder to search
[match]	only include datasets that match this text (wildcards allowed)
[subfolders]	(TRUE/FALSE) If TRUE, also searches all subfolders (default=FALSE)
[geodatabases]	(TRUE/FALSE) If TRUE, also searches all geodatabases (default=FALSE)
[variable]	the full path of returned raster datasets are stored in a variable of this name (default=rasterlist)
[namevariable]	if specified, an additional variable containing only the names of the raster datasets (without file extensions) is created
[print]	(TRUE/FALSE) If TRUE, the list of datasets is printed to the output window (default=TRUE)

Example

list.raster(in="C:\data");
list.raster(in="C:\data", match="*.img", subfolder=TRUE, variable="imglist");
list.raster(in="C:\data", match="NDVI*", geodatabase=TRUE, subfolder=TRUE, namevariable="rasternames", print=FALSE);

3.72 list.vector

List Vector Datasets: Creates a list of vector datasets in folders and subfolders matching user-specified search criteria.

Description

The command searches a folder (and optionally all subfolders and geodatabases) for vector datasets, storing the results in a variable in GME (a list of strings containing the full path to each vector dataset, with a default name of "vectorlist"). If a 'match' string is specified, only vector datasets with names matching this search string are returned (e.g. if match="b*" only vector datasets beginning with the letter b are returned). The search can also be restricted to specific geometry types (point, line or polygon) using the type parameter (by default, vector datasets of any geometry type are returned).

When using the list of datasets in scripting and automation it is often useful to have a corresponding list that contains only the name of the vector dataset. If a value is specified for the namevariable parameter, a second variable in GME is created that contains only the dataset name (with no extension). For instance, if the first dataset in the returned list is "C:\data\soil.shp", the corresponding value in the namevariable list is "soil". This means that the vector dataset name can be easily retrieved and used in naming output files in a script.

The command will print the resulting list to the output window. For very large lists this may be unwanted, and it can be switched off by setting print=FALSE.

Syntax

list.vector(in, [match], [type], [subfolders], [variable], [namevariable], [print]);

in	the folder to search
[match]	only include datasets that match this text (wildcards allowed)
[type]	only list datasets of this geometry type (options: ANY, POLYGON, POLYLINE, POINT)
[subfolders]	(TRUE/FALSE) If TRUE, also searches all subfolders (default=FALSE)
[variable]	the full path of returned raster datasets are stored in a variable of this name (default=vectorlist)
[namevariable]	if specified, an additional variable containing only the names of the raster datasets (without file extensions) is created
[print]	(TRUE/FALSE) If TRUE, the list of datasets is printed to the output window (default=TRUE)

Example

```
list.vector(in="C:\data");
list.vector(in="C:\data", match="*.shp",, type="POINT", subfolder=TRUE,
variable="shapefiles");
```

list.vector(in="C:\data", match="samp*", subfolder=TRUE, namevariable="names", print=FALSE);

3.73 listintersectingfeatures

List Intersecting Features: For each feature in an input layer this tool creates a list of other features (in the same layer or a different layer) that intersect it.

Description

This tool creates a delimited text file that summarizes intersections among features. For each feature (point, line or polygon) in a dataset the tool identifies which features from the same or another dataset intersect it, and writes the unique ID numbers to the output file.

By default the output file contains one line per feature of the input data source (not including features with no intersections at all), followed by the count of intersecting features, and then a list of the ID numbers. The alternative 'linear' format consists of one pair of intersecting features per line (only the two ID numbers are shown).

The delimiting character can be specified. See the 'delimiter' command for instructions on how to change the delimiting character on all tabular text output (this is a separate command that you would issue before this command, not a parameter for this command).

Although the default spatial relationship is intersect, you can actually specify the type of relationship to use. The options (and their abbreviations in parentheses) are: INTERSECTS (I), CROSSES (CR), OVERLAPS (O), TOUCHES (T), WITHIN (W), CONTAINS (CO), ENVELOPEINTERSECTS (EI), INDEXINTERSECTS (II). You may also specify a custom relationship using: RELATExxxxxxxxx, where the x's define the custom relationship and can assume values of F, T, or * (e.g. RELATEFFFTTT***). Refer to the ESRI Help documentation for descriptions of these relationships.

Intersections can be computationally expensive, therefore it is recommended that you consider taking steps to increase processing speed, especially for large datasets. Ensuring that your feature dataset has a spatial index can improve search speed (see the Add Spatial Index tool in ArcToolbox). You might also consider eliminating any features that are known to have no overlap with the features in another layer. For instance, the ArcMap Select By Location tool allows you to easily select overlapping features between two layers, which you can then export into two new (temporary) layers that are the input to this tool. Reducing the number of features in the layers will thus also improve search speeds.

Syntax

listintersectingfeatures(in, uidfield, in2, uidfield2, out, [relationship], [linear], [where]);

in	the input feature data source
uidfield	the unique ID field of the first vector data source
in2	the second vector data source
uidfield2	the unique ID field of the second vector data source
out	the full path to the output delimited text file
[relationship]	the spatial relationship to detect features (default=INTERSECTS; options: INTERSECTS, CROSSES, OVERLAPS, TOUCHES, WITHIN, CONTAINS, ENVELOPEINTERSECTS, INDEXINTERSECTS)
[linear]	(TRUE/FALSE) if true, one pair of ID numbers is written per line (default=FALSE)
[where]	the filter/selection statement that will be applied to the first ('in', not 'in2') feature class to identify a subset of features to process

Example

```
listintersectingfeatures(in="C:\data\plots.shp", uidfield="PLOTID",
in2="C:\data\transects.shp", uidfield2="TID", out="C:\data\transinplots.csv",
linear=TRUE);
listintersectingfeatures(in="C:\data\zones.shp", uidfield="ZNID",
in2="C:\data\roads.shp", uidfield2="RDID", out="C:\data\isectfeat.csv");
listintersectingfeatures(in="C:\data\parcels.shp", uidfield="PID",
in2="C:\data\schoolzones.shp", uidfield2="SCHID", out="C:\data\isectfeat.csv",
relationship="OVERLAPS", linear=TRUE);
```

3.74 ls

List GME Variables: List the names and assigned values of the variables the user has defined.

Description

This command lists the names and assigned values of the variables the user has defined. It is similar to the R command ls(), except that this tool also lists the variable values.

Syntax

ls);

Example

ls();

3.75 mergesampleplots

Merge Sample Plots: Merges sample plots based on the area of suitable habitat within plots so that resulting plots contain a minimum area of suitable habitat to sample.

Description

The purpose of this tool is to merge sampling plots (and/or zones) that contain less than a specified area of suitable habitat. It is designed for use with plot and zones layers that have been created with the genregionsampleplots command. This tool can also be applied to a plot layer that has no corresponding zone layer. The difference is that when a zone is also specified, merging of plots occurs preferentially within the same zone.

It is essential that you specify the zone and plot layer that were created simultaneously using the genregionsampleplots tool. If you have run that tool more than once, you must not mix a zone and plot layer from different sessions (the unique ID values are unlikely to match and this will result in this merging tool producing nonsensical data). It is also not advisable to use a previously merged zone/plot layer as the input to this tool.

Define the minimum suitable habitat area in coordinate system units e.g. square meters for UTM. This value corresponds to the values in the ArSuitHab field that were produced by the genregionsampleplots command. Any zones with a value of ArSuitHab less than the value specified will be merged using the merging rule you specify. When a plot is merged, the attribute fields are updated and the polygon it has been merged with is deleted. Note that the new, merged plot is not evaluated to determine if the minimum suitable area criteria has been met. However, this merged polygon is available for other plots to be merged with. Note that after merging the plot IDs will still be unique, but may not be a consecutive series as some of the plots have been deleted. After the plots are merged, the zones are evaluated for merging. If a zone is merged then the plots that are contained within this zone are updated to reflect the new zone ID value.

Merging Rules. For all merging rules the term neighbour refers to the four bordering plots in the four cardinal directions. A plot on the edge of a zone, however, will have fewer than four neighbours because merging of plots across zones is not permitted.

1. Dominant Neighbour Rule. The plot (or zone) is merged with the neighbour that contains the greatest total area of suitable habitat (the maximum ArSuitHab value).

2. Longest Border Rule. The plot (or zone) is merged with the neighbour that shares the border passing through the greatest amount of suitable habitat. The line representing the shared border is intersected with the suitability raster and the proportion of suitable habitat cells intersected by that line is calculated. The border with the greatest value defines which polygons are merged.

3. Simple Contiguity Rule. The plot (or zone) is split into halves: north and south halves, and east and west halves (i.e. two pairs of halves, not four quarters). The half containing the greatest proportion of suitable habitat defines the neighbour that is selected for merging.

In landscapes in which the suitable area is highly fragmented, the Dominant Neighbour Rule is probably the best choice.

Syntax

mergesampleplots(in, out, area, rule, [inzones], [outzones], [areazones], [rulezones], [raster]);

in	the input plot data source that was created using the genregionsampleplots command (it must contain a ArSuitHab field)
out	the new, merged, output plot data source
area	the minimum area of suitable habitat per plot (identifies the plots that will be merged)
rule	the number representing the plot merge rule: 1=Dominant Neighbour, 2=Longest Border, 3=Simple Contiguity (see full help documentation for details)
[inzones]	the input zone data source that was created using the genregionsampleplots command
[outzones]	the new, merged, output zone data source
[areazones]	the minimum area of suitable habitat per plot (identifies the plots that will be merged)
[rulezones]	the number representing the zone merge rule: 1=Dominant Neighbour, 2=Longest Border, 3=Simple Contiguity (see full help documentation for details)
[raster]	a suitability (1/0) raster that is used to characterize plots, required for rules 2 and 3 (see help documentation for further details)

Example

mergesampleplots(in="C:\data\plots.shp", out="C:\data\plotsmerged.shp", area=2500, rule=2, raster="C:\data\suit.tif");
mergesampleplots(in="C:\data\plots.shp", out="C:\data\plotsmerged.shp", area=2500, rule=1, inzones="C:\data\zones.shp", outzones="C:\data\zones.shp", areazones=25000, rulezones=1);
mergesampleplots(in="C:\data\plots.shp", out="C:\data\plotsmerged.shp", area=2500, rule=3, inzones="C:\data\zones.shp", outzones="C:\data\zones.shp", areazones=25000, rulezones=3, raster="C:\data\suit.tif");

3.76 movement.pathmetrics

Calculate Movement Path Metrics: Calculates movement paths metrics (turn angles, step lengths, bearings, time intervals) for a point time series dataset.

Description

This tool calculates turn angles, step lengths, bearings, and time intervals for a point time series dataset. The turn angle is based on the point sequence p(t-1), p(t), p(t+1); the step lengths, bearings and time intervals are based on the sequence p(t), p(t+1). Thus, a NoData value (-999) will be written for the first and last turn angles in a point series, and for the last step length, bearing and time interval records. These NoData values should obviously not be included in subsequent analyses.

If the optional radians parameter is set to TRUE then the turn angles and bearings are specified in radians (the default is degrees). The step length distances are always recorded in coordinate system units (e.g. meters for UTM). If the field name for any of the fields is set to NONE then that field will not be created. If names are not provided for these three fields then default names are used (TURNANGLE, STEPLENGTH, BEARING) and all three fields are calculated. In order to calculate time intervals (default field name: TIMEINTVL) you must specify a data/time field (datetimefield parameter). Shapefiles can be problematic for storing date/time data, so it is recommended that you use the geodatabase format for telemetry datasets.

The tool uses an integer order field to sort the points into the correct order (geodatabases are unordered collections, so must be ordered for time series analysis). Specifying an integer order field is compulsory as it eliminates the risk that incorrect data will be written because the order of the points is incorrect.

If an optional integer group field is specified then the command will sequentually process each unique value in that field, extracting all the points associated with that ID, sorting them, and writing the movement path metrics. The group field would be used when a point dataset contains multiple animals, for instance.

Syntax

movement.pathmetrics(in, uidfield, orderfield, [groupfield], [tafield], [slfield], [brgfield], [radians], [where], [datetimefield], [intervalfield]);

in	the input point data source
uidfield	the name of the unique point ID field in the input data source
orderfield	the name of the integer order field in the input data source
[groupfield]	the name of the integer grouping field representing collections of points (e.g. an animal ID field)
[tafield]	the turn angle field name to add (default=TURNANGLE; use NONE to disable)
[slfield]	the step length field name to add (default=STEPLENGTH; use NONE to disable)
[brgfield]	the bearing field name to add (default=BEARING; use NONE to disable)
[radians]	(TRUE/FALSE) specifies whether the turn angle and bearing are recorded as radians rather than degrees (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)
[datetimefield]	the field name of a date/time field (required if you specify an intervalfield)
[intervalfield]	the time interval field to add (the interval in seconds between this record and the next; requires datetimefield is specified)

Example

movement.pathmetrics(in="C:\data\locs.shp", uidfield="PNTID", orderfield="ORDER");

movement.pathmetrics(in="C:\data\locs.shp", uidfield="PNTID", orderfield="ORDER", groupfield="ANIMALID", radians=TRUE);
movement.pathmetrics(in="C:\data\locs.shp", uidfield="PNTID", orderfield="ORDER", groupfield="ANIMALID", tafield="TURNS", slfield="STEPS", brgfield="NONE", datetimefield="RECDATE", intervalfield="TIMEINTVL", radians=TRUE);

3.77 movement.simplecrw

Simulate Simple Correlated Random Walk: A simple correlated random walk movement path simulator, drawing from statistical or empirical distributions, with a home range boundary constraint option.

Description

This tool is a simple correlated random walk movement path simulator. It simulates paths by making independent random draws from a step length and turn angle distribution that you specify. Based on a set of start locations (which can be randomly generated using other tools in this tool set), the number of steps per path to simulate (this does not have to be a constant value), and the number of iterations per start location, the tool will generate simulated paths and write the output to a point output file (and optionally also a line output file). You may also optionally specify a polygon that defines a reflective boundary for simulated paths (i.e. paths will not be permitted to cross outside of this polygon). This is a flexible tool and can be run in many different ways for different purposes, but it is called a 'simple' CRW simulator because it does not accommodate multiple behavioural states for simulated paths (all steps and turns are drawn from a single pair of distributions).

Step length and turn angle distributions can be specified as either empirical distributions (which are loaded from a text file), or as statistical distributions from which random values are generated using R. Please refer to the section 'Specifying statistical and empirical distributions' for detailed instructions on how to specify distributions.

If you specify an empirical distribution that represents turn angles in radians then you must also set the radians=TRUE option. If you fail to do this the values will be interpreted as degrees and your paths will turn very little. If you specify a probability density function for the turn angle distribution then this is always interpreted as radians and you cannot override this setting using the 'radians' option (it is ignored).

It is important to exercise care when using the reflective polygon boundary option. If you specify a turn angle distribution that is too restrictive (strong directional persistence), or a step length distribution with a mean value that is large relative to the size of the boundary polygon, then the simulations may take a very long time to run. The risk is that the simulated path reaches the edge of the reflective boundary and then the restrictive movement parameters ensure that the vast majority of potential steps are rejected.

Syntax

movement.simplecrw(inpoint, uidfield, tad, sld, nsteps, iterations, outpoint, [bnd], [outline], [radians], [where]);

inpoint	the input start locations (a point feature source)
uidfield	the name of the unique ID field in the input data source
tad	the turn angle distribution (see full help documentation for details)
sld	the step length distribution (see full help documentation for details)
nsteps	the number of steps in the path or a field name in the start location data source containing these values
iterations	the number of paths to generate per input point or input polygon
outpoint	the output point data source to create
[bnd]	a polygon data source containing a single polygon that defines the reflective boundary for simulated paths
[outline]	also generates the output in line format (one line per step) in this data source
[radians]	(TRUE/FALSE) specifies whether the turn angle distribution is in radians (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
movement.simplecrw(inpoint="C:\data\startlocs.shp", uidfield="STARTID",
tad="C:\data\turns.csv", sld="C:\data\steps.csv", nsteps=1000, iterations=100,
outpoint="C:\data\sims.shp");
movement.simplecrw(inpoint="C:\data\startlocs.shp", uidfield="STARTID",
tad="C:\data\turns.csv", sld="C:\data\steps.csv", nsteps="STEPCNT", iterations=1,
outpoint="C:\data\sims.shp", outline="C:\data\simsline.shp",
bnd="C:\data\parkbnd.shp", radians=TRUE);
movement.simplecrw(inpoint="C:\data\startlocs.shp", uidfield="STARTID",
tad=c("WRAPPEDCAUCHY",0,0.3), sld=c("EXPONENTIAL",0.0015), nsteps=20000,
iterations=100, outpoint="C:\data\sims.shp", radians=TRUE);
```

3.78 movement.ssfsamples

Generate Step Selection Function Sample Steps: Generates step selection function (SSF) sampled steps from telemetry locations.

Description

This tool generates sampled steps along a movement path and is designed to faciliate the developement of step selection function (SSF) models (Fortin et al. 2005). Given an observed movement path (e.g. using GPS telemetry collar data), an SSF model characterizes the relative probability of selecting a step based on a set of covariates. It is a use versus availability design in which each observed step is compared to a sample of available steps at each point along the movement path. A 'step' in this context refers to the straight line that connects two consecutive locations. The model does not assume that animals move in a straight line

between consecutive locations, only that the environmental characteristics along that line are correlated with the likelihood of moving to that destination point.

The sampled steps are generated by drawing from an observed (or perhaps theoretical) step length and turning angle distributions. These can either be empirical (essentially binned frequency distributions based on observed movement paths), or fitted statistical distributions such as the wrapped Cauchy distribution for turning angles, or the gamma distribution for step lengths. Please refer to the section 'Specifying statistical and empirical distributions' for detailed instructions on how to specify distributions. If you specify an empirical distribution that represents turn angles in radians then you must also set the radians=TRUE option. If you fail to do this the values will be interpreted as degrees and your sampled steps will turn very little.

The unique ID field and the order field must currently both be integers. The order field is required because geodatabases can return records in a random order. So this tool sorts the locations in ascending order based on this field, which should therefore represent the chronological sequence in the time series.

The 'include' parameter controls whether the observed step is included in the output file or not. It is suggested that you set this to TRUE, because merging the sampled and observed steps at a later time can be tedious. Note that when TRUE, an additional field called 'OBSERVED' is added to the output table to distinguish the real observed steps (1) from the simulated sampled steps (0). This field is needed in the statistical analysis.

References

Fortin, D., Beyer, H. L., Boyce, M. S., Smith, D. W., Duchesne, T. & Mao, J. S. 2005. Wolves influence elk movements: behavior shapes a trophic cascade in Yellowstone National Park. Ecology 86: 1320-1330.

Syntax

movement.ssfsamples(in, uidfield, order, tad, sld, nsamples, out, [include], [radians], [where]);

in	the input location time-series data (a point feature source)
uidfield	the name of the unique ID field in the input data source (unique for each input point; must be integer)
order	the name of the chronological ordering field in the input data source (must be integer)
tad	the turn angle distribution (see full help documentation for details)
sld	the step length distribution (see full help documentation for details))
nsamples	the number of sampled steps to generate at each observed location (see full help documentation for details)
out	the output line data source to create
[include]	(TRUE/FALSE) specifies whether the observed step is included in the output (default=FALSE)
[radians]	(TRUE/FALSE) specifies whether the turn angle distribution is in radians (default=FALSE)
[where]	the selection statement that will be applied to the point feature data source to identify a subset of points to process (see full Help documentation for further details)

Example

movement.ssfsamples(in="C:\data\telemetry.shp", uidfield="PNTUID", order="SEQID", tad="C:\data\turns.csv", sld="C:\data\steps.csv", nsamples=10, out="C:\data\ssfsamples.shp", include=TRUE, where="ANIMALID=1012");
movement.ssfsamples(in="C:\data\telemetry.shp", uidfield="PNTUID", order="SEQID", tad="C:\data\turns.csv", sld="C:\data\steps.csv", nsamples=20, out="C:\data\ssfsamples.shp", include=TRUE, where="ANIMALID=1012", radians=TRUE);
movement.ssfsamples(in="C:\data\telemetry.shp", uidfield="PNTUID", order="SEQID", tad=c("WRAPPEDCAUCHY",0,0.3), sld=c("EXPONENTIAL",0.0015), nsamples=20, out="C:\data\ssfsamples.shp", radians=TRUE, where="ANIMALID=1012");

3.79 movement.ssfsim1

Simulate Step Selection Function Model: Simulates movement paths based on a step selection function (SSF) model.

Description

This tool simulates movement paths based on a step selection function (SSF) model (Fortin et al. 2005). Given an observed movement path (e.g. using GPS telemetry collar data), an SSF model characterizes the relative probability of selecting a step based on a set of covariates. It is a use versus availability design in which each observed step is compared to a sample

of available steps at each point along the movement path. A 'step' in this context refers to the straight line that connects two consecutive locations. The model does not assume that animals move in a straight line between consecutive locations, only that the environmental characteristics along that line are correlated with the likelihood of moving to that destination point.

There are a large number of ways in which SSF models can be formulated, so writing a generic simulation program is difficult. Various assumptions must be made regarding model structure. This simulator is flexible, but it does assume that: 1) step lengths and turn angles can be described by a single pair of distributions (i.e. there is only one movement state), 2) all of the covariates can be described by spatial raster datasets, and 3) these raster datasets are static (they do not change through time).

The model is specified using a 'model file', which describe the covariate raster datasets, the model coefficients, and the summary statistic used. The raster dataset is simply the path to the raster, e.g. C:\data\dem (for a grid) or C:\data\ndvi.img (for a raster in Imagine format). The rasters do not have to be in the same folder, but they should be stored locally (not on a network drive). It is highly recommended that you avoid spaces and all unusual characters in the folder and file names associated with these raster datasets, and that the rasters are not buried in a deep directory structure. The model coefficients should be expressed to the maximum level of precision. The summary statistic is expressed as text and can be one of the following: MEAN, MIN, MAX, START, END, MED, SUM. 'MEAN' refers to the length weighted mean of the covariate along the step. If your covariate represents a dummy variable then the mean corresponds to the proportion of the step that passes through that covariate. 'Min' and 'max' are the minimum or maximum values encountered along the step. 'Start' and 'end' are the value of the covariate at the beginning/end of the step respectively. 'MED' refers to the median value along the step (note that this does not take into account the length of the segment that passes through a cell: all cell values encountered along the step contribute equally to the calculation of the median).

The format of the model file must be strictly observed. All values are separated by commas, and each line must contain only a single covariate description. Blank lines and other comments are not permitted. The first line of the model file will always be 'INTERCEPT,value', where value is the intercept value from the model, e.g. -13.2893934. Each subsequent line will follow the format: 'raster-dataset, value, summary-statistic', where raster-dataset is the full path to the raster dataset, value is the model coefficient, and summary-statistic is one one of the key words described above.

The user is required to prepare the raster datasets representing covariates prior to running this tool. First, all of the raster datasets must be in the same projection. The cell sizes and alignment of cells among the raster datasets can be different. In most cases the raster datasets used to parameterize the model will be used in the simulations. If they are not, then you must take care to ensure that consistent units are maintained. For instance, if a digital elevation model (DEM) that measures elevation meters is used as a covariate in the model, then you must ensure that the units of the DEM used in the simulations is also in meters, not in feet. Changes in units will influence the coefficient values that are estimated by the model, hence the need to ensure units are consistent.

You must also convert any thematic (categorical) rasters to dummy variable rasters. For

instance, if a raster with 5 categorical habitat types is used in the SSF model, you must create 4 separate rasters coded as 1 or 0 to represent those variables in the simulation. (Note that you only need to create 4 rasters even though there are 5 habitat types in the model because one of those habitat types is the 'reference' category and is therefore omitted).

If you have performed any transformations of variables in the model, then you must also apply those transformations to the raster layer before running the simulation. For instance, you might have centred and log transformed a variable prior to fitting the model, in which case you would use Raster Calculator to centre and log transform the raster dataset. Or you might have a quadratic expression for a covariate like slope, in which case you must provide both the slope and slope^2 rasters. The key point is that the raster you use in this simulation must be directly related to the coefficient that the SSF model has estimated.

It is also important that you specify an appropriate boundary polygon. The most important aspect of this boundary is that it does not exceed the limits of any of the covariate raster datasets. Often, raster datasets cover different extents so you must take care to ensure that the polygon you create does not exceed the boundaries of any of these raster datasets. If you are simulating paths within the context of a home range, then the boundary polygon would be the home range polygon, and might only cover a small fraction of your raster datasets. But if you are interested in more of a landscape level simulation in which the simulated paths are not constrained to a pre-defined home range, then the polygon would be the common minimum limit of the raster datasets. The projection of the boundary polygon must also match the raster datasets.

The start locations can be random or based on actual animal locations. In either case, you must ensure that the projection of the start locations is identical to that of the raster datasets and boundary polygon. Note that there are a number of different strategies you can employ with regard to start locations. For instance, if you were generating a total of 10000 simulated paths you might have 1) a single point that all simulated paths start from (iterations=10000), 2) a set of 100 random points from which simulations start from (iterations=100), or 3) 10000 random points from which one path starts from each location (iterations=1). The strategy you adopt will depend on the question you are interested in addressing.

This simulation tool functions as follows. From the start location, a random initial bearing is drawn from a uniform distribution. The code then generates a number of available steps (the number of steps is controlled by the 'nsamples' option), and calculates the likelihood of each step based on the model, i.e. if we let w = exp(beta0 + beta1 * X1 + ... + betaN * XN), then the likelihood is calculated as L=w/(1+w). All available steps must end inside the boundary polygon (sampling continues until a full set of available steps that end inside the boundary polygon is acquired). Of these available steps, a single step is selected as the 'used' step where the probability of selection is proportional to the likelihood. Each of the available steps thus has a chance of being selected (if the likelihood of the step is non-zero).

Step length and turn angle distributions can be specified as either empirical distributions (which are loaded from a text file), or as statistical distributions from which random values are generated using R. Please refer to the section 'Specifying statistical and empirical distributions' for detailed instructions on how to specify distributions.

If you specify an empirical distribution that represents turn angles in radians then you must also set the radians=TRUE option. If you fail to do this the values will be interpreted

as degrees and your paths will turn very little. If you specify a probability density function for the turn angle distribution then this is always interpreted as radians and you cannot override this setting using the 'radians' option (it is ignored).

If you specify a turn angle distribution that is too restrictive (strong directional persistence), or a step length distribution with a mean value that is large relative to the size of the boundary polygon, then the simulations may take a very long time to run. The risk is that the simulated path reaches the edge of the reflective boundary and then the restrictive movement parameters ensure that the vast majority of potential steps are rejected. This will eventually cause the simulator to fail.

It is recommended that you first use the movement.simplecrw command to simulate movement paths based only on the set of start locations, the step length and turn angle distributions, and the boundary polygon. This allows you to check that the distributions you have specified are reasonable before running the more complicated SSF simulation.

References

Fortin, D., Beyer, H. L., Boyce, M. S., Smith, D. W., Duchesne, T. & Mao, J. S. 2005. Wolves influence elk movements: behavior shapes a trophic cascade in Yellowstone National Park. Ecology 86: 1320-1330.

Syntax

movement.ssfsim1(model, inpoint, uidfield, tad, sld, nsamples, nsteps, iterations, outpoint, bnd, [outline], [radians], [where]);

model	the SSF model specification file (see full help documentation for details)
inpoint	the input start locations (a point feature source)
uidfield	the name of the unique ID field in the input data source
tad	the turn angle distribution (see full help documentation for details)
sld	the step length distribution (see full help documentation for details))
nsamples	the number of potential steps to evaluate at each simulated step (see full help documentation for details)
nsteps	the number of steps in the path or a field name in the start location data source containing these values
iterations	the number of paths to generate per input point or input polygon
outpoint	the output point data source to create
bnd	a polygon data source containing a single polygon that defines the reflective boundary for simulated paths
[outline]	also generates the output in line format (one line per step) in this data source
[radians]	(TRUE/FALSE) specifies whether the turn angle distribution is in radians (default=FALSE)
[where]	the selection statement that will be applied to the point feature data source to identify a subset of points to process (see full Help documentation for further details)

Example

movement.ssfsim1(model="C:\data\ssfmodel1.txt", inpoint="C:\data\startlocs.shp", uidfield="STARTID", tad="C:\data\turns.csv", sld="C:\data\steps.csv", nsamples=20, nsteps=1000, iterations=100, outpoint="C:\data\sims.shp", bnd="C:\data\parkbnd.shp");
movement.ssfsim1(model="C:\data\ssfmodel1.txt", inpoint="C:\data\startlocs.shp", uidfield="STARTID", tad="C:\data\turns.csv", sld="C:\data\steps.csv", nsamples=50, nsteps="STEPCNT", iterations=1, outpoint="C:\data\sims.shp", outline="C:\data\simsline.shp", bnd="C:\data\parkbnd.shp", radians=TRUE);
movement.ssfsim1(model="C:\data\ssfmodel1.txt", inpoint="C:\data\startlocs.shp", uidfield="STARTID", tad=c("WRAPPEDCAUCHY",0,0.3), sld=c("EXPONENTIAL",0.0015), nsamples=20, nsteps=20000, iterations=100, outpoint="C:\data\sims.shp", bnd="C:\data\hrbnd.shp", radians=TRUE);

An example of the model file structure:
INTERCEPT, 12.3456789
C:\data\habforest.img, 0.987654, MEAN
C:\data\habmead.img, -1.234567, MEAN
C:\data\slope, 2.9589334, MIN
C:\data\slope2, -0.0290340, MIN
C:\data\biomass, 6.290384, MAX

3.80 neighbourhoodstatistics

Raster Neighbourhood Statistics: Calculates summary statistics in a circular roving window based on raster data.

Description

This tool calculates basic summary statistics (sum, mean, minimum, maximum, standard deviation, median) of raster values within a circular, roving analysis window. This is similar to what is called 'zonal statistics' in some software. The input is a single raster layer representing continuous data (it would not be appropriate to use this command with thematic raster data), and an analysis window radius that is defined in coordinate system units. For a cell to be included in the calculation of the statistics the centre of the cell must fall within the specified distance of the centre of the target cell.

The metrics calculated at the edges of the raster are obviously based on smaller samples than the cells in the interior of the raster (when the analysis window extends beyond the edge of the raster). Similarly, when NoData cells are encontered within the analysis window, they are ignored, and the metrics are calculated on the remaining cells with data values. Although the mean, minimum, maximum and median are probably fairly robust to this reduced sample size issue, it is possible that the standard deviation is underestimated somewhat in these edge cells. The sum metric is the most sensitive to the number of samples, and this metric is adjusted to remove this bias. The observed sum is divided by: the observed number of cells

divided by the total number of cells that would have constituted the analysis window had it been complete. This procedure thus normalizes the sum values so that they are comparable across the entire raster.

Syntax

neighbourhoodstatistics(in, radius, sum, min, max, mean, sd, med, [edge]);

in	the input raster data source
radius	the radius of the neighbourhood analysis window in coordinate system units
sum	the new output raster data source representing the sum of values in the neighbourhood
min	the new output raster data source representing the minimum of values in the neighbourhood
max	the new output raster data source representing the maximum of values in the neighbourhood
mean	the new output raster data source representing the mean of values in the neighbourhood
sd	the new output raster data source representing the standard deviation of values in the neighbourhood
med	the new output raster data source representing the median of values in the neighbourhood
[edge]	(TRUE/FALSE) calculate statistics for edge cells, where the neighborhood is only partially observed (default=TRUE)

Example

```
neighbourhoodstatistics(in="C:\data\dem", radius=2000,
mean="C:\data\demmean.img", sd="C:\data\demsd.img");
neighbourhoodstatistics(in="C:\data\habitat", radius=250, sum="C:\data\habsum.img",
edge=FALSE);
```

3.81 paste

Paste: Concatenates a list of values and/or variables to create a text string.

Description

This function is designed to be embedded within other command text. It creates new strings (text) based on a variety of inputs you define. Those inputs can be other strings (in quotes), numbers (in quotes or not in quotes), index variables from a for loop, or variables that you have defined.

The optional 'sep' argument allows you to define the separator to use between the components. By default this is nothing, but if you are constructing a comma delimited list you would chance this to sep="," for instance.

This paste function is very similar to the paste function in R.

Syntax

paste(..., [sep]);

...	a comma delimited list of components from which the new string is constructed
[sep]	defines the separator character to use between the components in the list (default is no seperator); e.g. to construct a comma delimited list specify sep=”,”

Example

wd <- ”C:\data\”
paste(wd, ”image”, i, ”.img”)
Assuming this paste function is embedded in a command occurring within a for(i in 1:3){...} loop, and the ’wd’ variable has previously been defined, the following strings would result in each of the three loops:
”C:\data\image1.img”
”C:\data\image2.img”
”C:\data\image3.img”

3.82 pointdistances

Distances Among Points: Calculates distances between points, either among points within a point data source, or among points between two point data sources.

Description

This tool calculates distances between points, either among points within a point data source, or among points between two point data sources. It also optionally identifies the N nearest neighbours, and has three output formats to choose from. This tool has been designed to produce flexible distance matrix outputs that would be used in subsequent analyses.

The simplest implementation of the tool is to specify an input point dataset, identify the unique ID field for that dataset, and specify the output file. The tool will calculate distances between each of the points within this dataset and write the result using the default format.

There are three formats to choose from. The first (1D) write a table that has three columns: the unique ID of each of the two points, and the distance between them. This format is the default because it can accommodate very large numbers of points, and it does not report distances twice (i.e. if the tool calculates the distance between points A and B, it will not also report the distance between points B and A). The second option (2D) writes an NxN table (the full distance matrix), where N is the total number of points in the dataset. This option is only recommended with small numbers of points because the output is unlikely to be useful for large numbers of points (you would not be able to open it in Excel or import it into Access for instance). The 2D format reports all values twice (e.g. A and B, and B and

A, and all the cells on the diagonal are 0 (the distance between a point and itself). The final format (SUMMARY) generates a simple statistical summary (mean, minimum, maximum, standard deviation, and optionally median) of the distances between a given point and all the other points. The output table includes these summary statistic fields as columns, and N rows.

Nearest neighbours can be identified at the same time using the 'nearest' and 'nout' parameters. The nearest neighbour output is written to a different table, and includes the ID's of the ordered list of the n nearest neighbours, and the distance to each one.

The 'multiplier' option provides you with a way of changing the units of the distance value. By default the distance is calculated in coordinate system units, e.g. meters for UTM. If you wanted these distances in kilometers then you would set the multiplier to be 0.001.

To change the delimiting character in the output table (if you are in a country that prefers not to use the comma as the delimiting character), please refer to the 'delimiter' command, which must be issued before running this tool. However, you only have to issue that command once per session as it is a global setting.

The median is not calculated by default because it can be a computationally expensive calculation for large numbers of points. The median will be calculated if you specify 'format=SUMMARY' and 'median=TRUE' options.

This tool will attempt to ignore points with null geometries. These records can arise in a number of ways, but are particularly common in the output of address geocoding algorithms (the records that the algorithm failed to resolve). They are also common in GPS telemetry databases. It is advisable to either repair or remove these records prior to running GME commands. You will be warned if this command detects null geometries.

This tool should not be used with data in a geographic coordinate system. The distance calculations here assume the coordinates are Cartesian (i.e. a projected coordinate system), not spherical.

Note that this tool will not overwrite existing files, so you should ensure the output file does not already exist.

Syntax

pointdistances(in, fld, out, [in2], [fld2], [format], [nearest], [nout], [multiplier], [median], [where]);

in	the input point dataset (calculates distances between points within this dataset - but see 'in2')
fld	the unique ID field in the input dataset
out	the output delimited text file to create
[in2]	the second point layer (calculates distances between points between datasets)
[fld2]	the unique ID field in the second input dataset (MUST be specified if 'in2' is specified)
[format]	the structure of the output table: 1D (default, one record per row), 2D (an NxN matrix), SUMMARY (summary statistics for each point only) (options: 1D, 2D, SUMMARY)
[nearest]	the number of nearest neighbours to identify for each input point (default=0); note that 'nout' must also be specified with this option
[nout]	the output delimited text file to create for the nearest neighbour dataset
[multiplier]	multiplies the distance by this value before writing to output (default=1)
[median]	(TRUE/FALSE) determines whether the median is also calculated if the 'summary' format is specified (this can add considerably to processing time for large datasets, default=FALSE)
[where]	the filter/selection statement that will be applied to the first ('in', not 'in2') point feature class to identify a subset of points to process

Example

```
pointdistances(in="C:\data\locs.shp", fld="RECID", out="C:\data\distances.csv",
format="SUMMARY", multiplier=0.001);
pointdistances(in="C:\data\predators.shp", fld="ANID", out="C:\data\distances.csv",
in2="C:\data\prey.shp", fld2="PREYID");
pointdistances(in="C:\data\firestations.shp", fld="STID", out="C:\data\distances.csv",
in2="C:\data\houses.shp", fld2="PROPID", nearest=10,
nout="C:\data\nearestheigh.csv", median=TRUE);
```

3.83 r

Executes An R Command: Executes R commands.

Description

The r() command is different from most other GME commands in that it takes the text within the parentheses and sends it directly to R, without any interpretation of the text. For this reason you should not enclose the R command text within quotes: write the R commands here exactly as you would write them in R. This command does not recover any R objects, but you can use the r.ls() command to view a list of objects in the current R session.

This command is designed to be used in conjunction with the following commands: r.loaddata, r.writedatatofield, r.writedatatoraster. One of the most important uses of this

command will be to run R scripts that you have written and saved in text files.

You must call the r() command multiple times if you have several commands to execute (rather than separate them using semi-colons within a single call to this command). Better yet, write commands to a text file, and then call that as a script.

Note that if your code requires R libraries, then you must also make sure that the libraries are installed, and that you include the R library() or require() commands to load them within your code.

Syntax

```
r(code);
 code   R code to be evaluated (not in quotes)
```

Example

```
r(source(file="C:/scripts/ranalysis.txt"));
```

3.84 r.deldir

Delaunay Triangulation / Dirichlet Tesselation: Computes the Delaunay (Voronoi) triangulation and Dirichlet tessalation of a set of points using the 'deldir' library in R.

Description

This tool uses the 'deldir' library in R to compute the Delaunay triangulation and Dirichlet tessellation of a set of points. Given an input point data source, the user can produce one or both of the two outputs (they are both calculated simultaneously so there is no significant processing cost to producing both) as line layers.

Dirichlet tessellation requires a rectangular boundary that defines the limits of the calculation. By default the deldir library will set this to be plus/minus 10% of the range of x and y point coordinates, but this can be specified explicitly with the 'extent' option.

This tool automatically filters any stacked (identically placed) points before running the computation in R.

For citations and a full description of the deldir library, open R and type: library(deldir), then ?deldir. If the deldir library has not been installed on your system, you must first type: install.pacakges("deldir").

Syntax

r.deldir(in, dir, del, [extent], [where]);

in	the input point data source
dir	the output line data source for the Dirichlet (Voronoi) tessellation (optional if 'del' is specified)
del	the output line data source for the Delaunay triangulation (optional if 'dir' is specified)
[extent]	the reference layer to clip to (a vector or raster layer), or the coordinates of the rectangle to clip to (min x, max x, min y, max y)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

r.deldir(in="C:\data\firestations.shp", dir="C:\data\dirtess.shp");
r.deldir(in="C:\data\firestations.shp", del="C:\data\deltriang.shp");
r.deldir(in="C:\data\samples.shp", dir="C:\data\dirtess.shp",
del="C:\data\deltriang.shp", extent=c(346000,450600,4956300,5204300));
r.deldir(in="C:\data\trees.shp", dir="C:\data\dirtess.shp", del="C:\data\deltriang.shp",
extent="C:\data\studyarea.shp");

3.85 r.eval

Evaluate R Expression: This function evaluates an expression in R, and retrieves the result.

Description

This function is designed to be embedded within command text. It provides a mechanism for running R code to obtain strings, numbers or arrays that are used as parameters for GME command arguments.

Important note. The r.eval syntax is different than all other GME commands because you must not enclose the R code in quotes. All other GME commands and functions require that you enclose all text in quotes, but the code contained within the r.eval(...) function is sent directly to R. The reason for this is that you may need to use quotes in your R code, which is perfectly acceptable.

Important note. R commands must be separated by semi-colons.

If the R code is single line and simple, then GME will automatically retrieve the resulting value. However, if your R code is complex, and/or multiline, then you must explicitly set a variable called 'gme' within that R code, which GME will then retrieve. If your code is complex, I recommend you save it as an R script file, and then use the r.eval function to call that script.

Note that if your code requires R libraries, then you must also make sure that the libraries are installed, and that you include the R library() command to load them within your code.

Syntax

r.eval(code);
code the R code to evaluate (not in quotes) and the result returned

Example

r.eval(seq(0, 1, 0.02))
r.eval(exp(-0.01*200))
r.eval(a <- rnorm(10, 100, 12.234); gme <- log(a))

3.86 r.graphsettings

Graph Settings: Changes the default graph parameters that are used to generate graphics in R.

Description

This command allows you to change the default graph parameters that are used to generate graphics in R. Some of the parameters only affect the next graphics to be produced, while the other parameters affect all subsequent graphics.

The parameters 'xlab', 'ylab' and 'main' correspond to the x axis title, the y axis title and the main title respectively, and only affect the next graph to be produced, after which they are reset. You must therefore set these for each graph if you wish to customize the labels or title. Unlike R, you should not use quotation marks to set these parameters (see the example below). Note that for the labels, the code will automatically wrap long labels ($>$ 50 characters) onto multiple lines.

The 'width' and 'height' parameters refer to the dimensions in pixels of the graphic that is produced. The default is 500 x 500 pixels. Small dimensions ($<$ 300 pixels) may result in absurd graphics.

The 'pointsize' parameters refers to the size of text. Note that the size of the main title text is always automatically scaled to be larger than the text of the axis labels. Point sizes in the range of 9-16 are likely to produce reasonable graphics.

The 'fillcol' parameters sets the default fill colour (e.g. for bars in a histogram), and 'linecol' sets the default line colour for graph types that display lines. Colours are specified using colour names. See the R Colours appendix for samples of colours and their names.

The 'filetype' parameters controls the output file type. The default is Ping (png), which is a widely used format that most other software you encounter will be able to use. You may change the default to JPEG format (jpg) or Windows Bitmap format (bmp).

This command is driven by R. Type 'citation' to see the suggested citation for R.

Syntax

r.graphsettings([xlab], [ylab], [main], [width], [height], [pointsize], [fillcol], [linecol], [filetype]);

[xlab]	the x axis label for the next plot (overrides the default label)
[ylab]	the y axis label for the next plot (overrides the default label)
[main]	the main title for the next plot (overrides the default title, which is usually no title)
[width]	the width of the output plot in pixels (default=700)
[height]	the height of the output plot in pixels (default=700)
[pointsize]	the font size for text on R graphics (default=12)
[fillcol]	the fill colour for plots like histograms (default=lightgreen) - see the full help documentation for a list of R colours
[linecol]	the line colour for plots (default=lightgreen) - see the full help documentation for a list of R colours
[filetype]	the output file type for R graphics (default=png; options: png, jpg, bmp)

Example

r.graphsettings(xlab="X Coordinate (UTM)", ylab="Y Coordinate (UTM)",
main="Animal locations", width=900, height=900);
r.graphsettings(width=400, height=400, filetype="jpg", pointsize=9, fillcol="blue");

3.87 r.hist

Histogram: Creates a histogram graph using R.

Description

This command creates a histogram graph using R (the 'hist' function in R) and displays it in the output window. The R commands used to create the histogram are also displayed. Type '? hist' at the R command prompt for further details.

This command is driven by R. Type 'citation' to see the suggested citation for R.

Syntax

r.hist(in, field, [breaks], [freq], [where]);

in	the input feature data source
field	the name of field containing the data to plot
[breaks]	the number of bins or cells in the histogram (default=determined automatically)
[freq]	(TRUE/FALSE) if TRUE the histogram represents frequencies/counts, if FALSE the histogram represents probability densities (default=TRUE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

r.hist(in="C:\data\parcels.shp", field="Value08");
r.hist(in="C:\data\county.shp", field="Popn", breaks=30, freq=FALSE);

3.88 r.loaddata

Load Data Into R: Loads data from a vector attribute table into the current R session.

Description

This command will read data from a vector data attribute table and load it into the current R session (every instance of GME has a hidden R session - you can use the r.ls() command to view the objects in that session). The 'fields' parameter is used to define which fields will be read, and the optional 'where' parameter can be used to define a subset of rows. There are two important issues to be aware of when using this tool. First, the order in which records from an attribute table are returned is not always the same, so if this command is called multiple times on the same dataset there is a risk the order of values in the resulting R vectors will be misaligned! Thus, it is important to specify all of the fields you want to load into R in a single call to this command (the 'fields' parameter can be a vector of multiple field names). If you do this you can be sure that the order of values in each vector will be correct.

The second issue to be aware of is the need for an integer unique ID field in the attribute table that you can use to relate the R vectors (or new R vectors) back to the attribute table. If you perform an analysis that generates a new R vector as output, for instance, and you wish to write those values back to the attribute table then this unique ID field is essential for assigning the correct value from an R vector back to the attribute table. So if you need to maintain a relational link between the R data and the original attribute table then be sure to specify an integer unique ID field in the list of fields that will be loaded. See the r.writedatatotable command for an example of how this can be used.

The objects in R will have the same name as the fields from which they were written. Note that R is case sensitive.

Syntax

r.loaddata(in, fields, [where]);

in	the input feature data source
fields	the names of field or fields containing the data to load
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

r.loaddata(in="C:\data\parcels.shp", field="Value08");

r.loaddata(in="C:\data\county.shp", fields=c("FID", "Popn","Area","Name"), where="STATE='WI'");

3.89 r.ls

List R Variables / Objects: Lists the R objects currently present in the GME R session.

Description

This command will display a list of the objects that are currently present (in memory) in the GME R session. Although you cannot see this R session, you can interact with it using several of the GME commands that begin with 'r.'. This command is useful for checking the current state of the R session.

Syntax

r.ls);

Example

r.ls();

3.90 r.plotxy

Scatterplot: Creates a scatterplot graph using R.

Description

This command creates a scatterplot graph using R (the 'plot' function in R) and displays it in the output window. The R commands used to create the scatterplot are also displayed. Type '? plot' at the R command prompt for further details.

This command is driven by R. Type 'citation' to see the suggested citation for R.

Syntax

r.plotxy(in, xfield, yfield, [where]);

in	the input feature data source
xfield	the name of field containing the x axis data to plot
yfield	the name of field containing the y axis data to plot
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

r.plotxy(in="C:\data\points.shp", xfield="UTMX", yfield="UTMY");
r.plotxy(in="C:\data\parcels.shp", xfield="AREA", yfield="COST");

3.91 r.sample

Sample Features: Creates a random or stratified random sample of records in a feature table.

Description

This tool uses the R sample command to create a random or stratified random sample of records in a feature attribute table. Sampled records are coded with a 1 in a field in the attribute table, and all other records are 0. By default this field is named RNDSAMP, although you can specify any other field names using the 'field' parameter. This tool will not automatically overwrite an existing field unless you explicitly specify 'overwrite=TRUE' in the parameter list.

The number of features that are sampled can be specified as a count using the 'size' parameter, or as a proportion of the total number of records using the 'proportion' parameter. In the case of stratified sampling, the size parameter refers to the number of samples that will be selected within each stratum. By default, all strata are processed. If you wish to randomly sample among strata first, and then sample within strata then you must implement the stratalimit parameter (this controls the number of strata that are randomly selected).

For simple random samples a weight field can be specified with values that are proportional to the relative probability of selection. The values cannot be less than 0, but can be greater than 1 because the tool will automatically standardize the values by dividing my the maximum value (this ensures all the values are within the range 0-1). The weight field functionality is not yet implemented for the stratified sampling tool. If no weight field is specified, all records or features have an equal probability of being selected.

When using the size parameter, if you specify a sample size that is greater than the number of available records then a warning message will be generated, but all available features will be marked as being sampled.

This is a stochastic algorithm and is, therefore, very unlikely to yield the same results each time you run it. The 'verbose' option can be useful for checking that this algorithm is working correctly as it provides a detailed report of the sampling process. For further information on the R sample command, type '? sample' at the R prompt, and press Enter.

This command is driven by R. Type 'citation' to see the suggested citation for R.

Syntax

r.sample(in, size, proportion, [field], [weightfield], [stratified], [stratalimit], [overwrite], [verbose], [where]);

in	the input feature data source
size	the number of features to sample (integer); takes precedence over 'proportion' if both are specified
proportion	the proportion of features to sample (0.0-1.0)
[field]	the field that will record the selection (if it exists the program will stop, but see the overwrite option below) (default=RNDSAMP)
[weightfield]	the field that contains relative probabilities of selection (does not apply to stratified samples; see documentation for details)
[stratified]	the field that describes the strata in the data (typically an integer field representing unique group ID's); the count or proportion options are applied at the level of the strata
[stratalimit]	the number of strata to randomly sample (if undefined, all strata are processed)
[overwrite]	(TRUE/FALSE): if TRUE, if the output field already exists it will automatically be deleted and recreated, if FALSE the program stops with an error message if the field exists (default=FALSE)
[verbose]	(TRUE/FALSE): if TRUE, reports the sequence of sampled record numbers in the output window (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
r.sample(in="C:\data\plots.shp", size=100, overwrite=TRUE);
r.sample(in="C:\data\plots.shp", size=100, overwrite=TRUE, weightfield="SELPROB");
r.sample(in="C:\data\locs.shp", field="STRSEL", proportion=0.1,
stratified="ANIMALID", verbose=TRUE);
```

3.92 r.setpath

Set The R Executable Folder: Sets or changes the folder containing the R executable (Rscript.exe) you wish to use.

Description

This tool allows you to set or change the folder containing the R executable (Rscript.exe). When GME starts it will search C:\Program Files\R for the most recent version of R (specifically, a folder containing Rscript.exe). If it does not find it there it will then search C:\Program Files (x86)\R.

Note that you can have multiple versions of R installed on your computer. GME should work with any version from R 2.12 onwards (perhaps even earlier versions). If you wish to override the default behaviour of GME you can change the R folder using this command (either in a script, or in the user interface). You may also use the File menu to do this in

GME (see the 'Set R executable folder' menu item, and use it to navigate to the Rscript.exe file).

Note that the R libraries needed by GME must be installed for each and every copy of R you wish to use with GME (R libraries are not shared among different R versions). It is best to run R as an Administrator to install the libraries. (In Windows XP: use an account with Administrator priveleges. In Vista/Win 7: right click the R icon and select 'Run as Administrator').

Syntax

r.setpath(path);
path the folder path containing the Rscript.exe file

Example

r.setpath(path="C:\Program Files\R\R-2.15.0\bin");

3.93 r.writedatatofield

Write R Vector To Field: Writes an R vector to a vector attribute table.

Description

This command will write values from an R vector to an attribute table using an integer unique ID field to relate the two sets of data. This command is designed to be used in conjunction with the r.loaddata command, and subsequent processing of that data in R using the r() command. If a new R vector results from this processing, it can be added as a new field to the attribute table of the original data source. You should ensure that the lengths of the vectors referenced by the 'ruid' and 'rvector' parameters are identical. Note that R is case sensitive.

Syntax

r.writedatatofield(ruid, rvector, out, uidfield, datafield, fieldtype);

ruid	the R vector containing the integer unique ID values corresponding to the values in 'uidfield'
rvector	the R vector containing the data values to be written
out	the output feature data source
uidfield	the unique ID field name (must be an integer field)
datafield	the name of the new data field to create
fieldtype	the field data type - short integer, long integer, double precision real number, or string (options: SHORT, LONG, DOUBLE, STRING)

Example

r.writedatatofield(ruid="PntID", rvector="weight", out="C:\data\samples.shp", uidfield="PntID", datafield="WEIGHT", fieldtype="DOUBLE");
r.writedatatofield(ruid="FID", rvector="label", out="C:\data\lakes.shp", uidfield="FID", datafield="MAPLABEL", fieldtype="STRING");

3.94 r.writedatatoraster

Write R Matrix To Raster: Writes data in an R matrix to a raster format.

Description

This command will write values from a 2 dimensional R matrix to a raster format. The spatial positioning of the matrix data can be defined in two ways. You can either specify the positioning parameters explicitly, including the coordinates of the lower left corner of the raster (llx, lly parameters), the cellsize, and the file containing the projection definition, or you can reference an existing reference dataset from which these values are acquired. Note that the dimensions of the new raster are based on the dimension of the matrix, not the dimension of the reference raster. When specifying the matrix object in the current R seesion note that R is case sensitive.

Syntax

r.writedatatoraster(rmatrix, out, [reference], [llx], [lly], [cellsize], [prj], [pixeltype]);

rmatrix	the R matrix containing the data values to be written
out	the output raster, including the extension in the case of an IMG or TIF formats
[reference]	a reference raster data source from which the llx, lly, cellsize and prj parameters will be acquired (do not specify the other optional parameters)
[llx]	the x coordinate of the lower left corner of the raster
[lly]	the y coordinate of the lower left corner of the raster
[cellsize]	the raster pixel cell size
[prj]	the file containing the projection data of the input raster
[pixeltype]	the pixel type of the raster (if DOUBLE, must be an IMG raster) (default=DOUBLE; options: SHORT, LONG, DOUBLE, FLOAT)

Example

r.writedatatoraster(rmatrix="f", out="C:\data\randomlandscape.img", reference="C:\data\DEM", pixeltype="DOUBLE");
r.writedatatoraster(rmatrix="f", out="C:\data\rf1", llx=520000, lly=4950000, cellsize=100, prj="C:\data\UTMZn17N_WGS1984.prj", pixeltype="FLOAT");

3.95 raster.profile

Extract Raster Profile Data: Extract profile data for a row or column (all bands) in the raster.

Description

This tool extracts profile data from a raster along a row or column that you specify, creates a graph of the profile, and optionally writes it out as an R object that you can easily read into R using the load() function. It will extract profile data from all bands in the raster. The structure of the R data object created is a matrix in which the first two columns are the x and y coordinates of each cell in the row or column, and then a series of columns (equal to the number of bands) that contain the raster cell value data.

Syntax

raster.profile(in, [row], [col], [out]);

in	the input raster data source
[row]	the row number to extract (range 1:row count)
[col]	the column number to extract (range 1:column count)
[out]	the R object to create that contains the raster profile data

Example

raster.profile(in="C:\data\dem", row=1385, out="C:\data\rprofile.r");
In R: load(file="C:/data/rprofile.r")

3.96 raster.shift

Shift Raster: Performs a simple x, y shift on a raster dataset.

Description

This tool will shifts a raster by the specified x and y distances. This tool is designed as a quick, approximate method of resolving only the most simple raster misalignment problems. Note that for most raster misalignment problems the best solution may be a transformation or more thorough re-registration. The shift distances must be specified in coordinate system units (e.g. meters for UTM, decimal degrees for geographic coordinate systems, etc).

Syntax

raster.shift(in, shiftx, shifty, out);

in	the input raster data source
shiftx	the x-axis shift distance
shifty	the y-axis shift distance
out	the output raster data source to create

Example

raster.shift(in="C:\data\landcov", shiftx=15028.5, shifty=-12.4, out="C:\data\shiftedlc");

3.97 reclassify

Reclassify Raster: Reclassify a raster to create a new raster.

Description

This tool will reclassify a raster data source to create a new raster dataset using the 'recode' table you specify. The recode table is a comma delimited text file and determines how values from the input raster are mapped to the output raster values. Two forms of entry are allowed in the recode table (a mixture is allowed within the same table): 1) from-value : to-value, new-value; 2) input-value, output-value. The first type of entry defines a range of values in the input raster that correspond to a single value in the output raster, and is interpreted as from-value $\leq$ x $<$ to-value –> new value. Note that the to-value is interpreted as 'just less than the to-value'.

To recode cells that are NoData in the input raster, use either the NODATA or NA keywords in the recode table. For instance, including 'NODATA, 0' as a line in the recode file will recode all NoData cells to 0. To code existing value to NoData, use the NODATA or NA keywords as the target value (e.g. '0, NODATA' or '-100:0,NA'). These keywords are case sensitive and must be capitalized.

Note that the recode file must be comma delimited, and must include a header line (the first line containing the column headings). When the tool reads the recode file it always ignores the first line, hence the requirement that this header is present. See below for an example of a recode table.

Note that you can specify real numbers as the new values. If the new values in the recode table are all integers, the output raster will be an integer raster. If the new values in the recode table contain real numbers then the output raster will be a double precision rasters if the Imagine Image format is specified (.img) or a floating point raster for all other formats.

Syntax

reclassify(in, file, out);

in	the input integer raster data source
file	the reclassification table, a delimited text file (with a header row) - see the full help documentation for details.
out	the output integer raster data source

Example

reclassify(in="C:\data\landcov", file="C:\data\lcrecode1.csv", out="C:\data\newlc");
reclassify(in="C:\data\dem.img", file="C:\data\recodedem.csv", out="C:\data\elev");

IN,OUT
100:300,1
300:301,2
301:302,3
303,4
411:500,5
0, NODATA

INPUT,OUTPUT
100:300,1.1
300:301,2.7
301:302,3.00002
303,4.9999
411:500,5.123456789
NA, 0

3.98 reclassifyrecords

Reclassify Records / Features: Issues a series of attribute selection statements to classify tabular records.

Description

This tool provides a mechanism for automatically issuing a series of selection statements, and populating a field with values based on those statements. The selection statements are formulated in exactly the same way as in the 'Select by attributes' tool in ArcMap (SQL-style statements based on field names and values). The benefit of this command is that a set of selection statements can be issued automatically, a field can be automatically populated with values, and this procedure can be conveniently applied to many datasets.

The input is a table or feature data source, and the output is stored in a new or existing field. By default the output field type is the long integer type and the recode values must therefore be integers. However, you can change this to either double or string field types to accomodate other recode values (real numbers or text respectively). If you specify an existing field you must also authorize the tool to overwrite data in this field using the 'update=TRUE' option.

The selection statements are stored in a separate text file (e.g. one that you have created with Notepad). This text file should have no header line, and should have one statement per line. The syntax of the selection statement is exactly as it would appear in the 'Select by attributes' tool in ArcMap. An R style assignment operator (->) is used to define what value is assigned to the records that are selected by a statement. For instance, given a field name called PrSuitHab, which contains real numbers ranging from 0-1, the following three lines would result in all records being classified as either 1, 2 or 3.

"PrSuitHab" $\leq$ 0.2 -> 1 "PrSuitHab" $>$ 0.2 AND "PrSuitHab" $\leq$ 0.7 -> 2 "PrSuitHab" $>$ 0.7 AND "PrSuitHab" $\leq$ 1 -> 3

If you have specified a string output field then you must specify the recode value using double quotes:

"PrSuitHab" $\leq$ 0.2 -> "A"

Syntax

reclassifyrecords(in, file, field, [fieldtype], [update]);

in	the input table or feature data source
file	the file containing the reclassification instructions - see the full help documentation for details
field	the new field that will contain the reclassification values
[fieldtype]	the field data type - short integer, long integer, double precision real number, or string (options: SHORT, LONG, DOUBLE, STRING)
[update]	(TRUE/FALSE) if TRUE and you specify an existing field, the existing field will be updated rather than generating an error message (default=FALSE); warning: this option will result in overwriting of existing data and is therefore potentially dangerous.

Example

reclassifyrecords(in="C:\data\landcov", file="C:\data\lcrecode1.csv",
out="C:\data\newlc");
reclassifyrecords(in="C:\data\dem.img", file="C:\data\recodedem.csv",
out="C:\data\elev");

3.99 regiongroup

Raster Region Group: A raster tool that identifies contiguous blocks of cells of the same value (i.e. regions) and assigns each region a unique ID number that is written to the output raster dataset.

Description

This is a raster tool that identifies contiguous blocks of cells of the same value (i.e. regions) and assigns each region a unique ID number that is written to the output raster dataset. It is designed for use only with thematic (categorical) rasters.

The cell neighbourhood that is evaluated can be all eight cells that surround a cell (this is the default option), or can be limited to only the four cardinal cells using the 'diag=FALSE' option, which prevents diagonal connections being evaluated when defining region membership.

The unique ID numbers in the output raster are arbitrary, but are ordered by when the region is first encountered when moving left to right and top to bottom through the raster dataset. The output also includes a DBASE (dbf) table that summarises the number of cells that belong to each region.

For large raster datasets this tool can take quite some time to run (a few hours for very large rasters). It is therefore recommended that you test the tool on a small raster dataset before applying it to the full dataset.

Syntax

regiongroup(in, out, [diag]);
 in the input integer raster data source
 out the output integer raster data source
 [diag] (TRUE/FALSE) allow cells to be diagonally connected when defining regions? (default=TRUE)

Example

regiongroup(in="C:\data\landcov", out="C:\data\lcovrg");
regiongroup(in="C:\data\landcov", out="C:\data\lcovrg", diag=FALSE);

3.100 reproject.raster

Reproject Raster: Reprojects a raster dataset.

Description

This tool will reproject a raster data source to create a new raster dataset. The projection of the input raster dataset must be pre-defined. If you open this raster in ArcGIS, for instance, and the projection is 'undefined' then you must use the Define Projection tool in ArcToolbox to define the projection before running this command.

The output projection is specified by referencing a projection definition file (*.prj). See the section on 'Projection definition files' for an explanation of where to find these files, and how to create them if you are not already familiar with them.

Syntax

reproject.raster(in, prj, out);
 in the input raster data source
 prj the projection file specifying the new output projection
 out the new output raster to create

Example

reproject.raster(in="C:\data\landcov", prj="C:\data\WGS 1984.prj",
out="C:\data\lcovrp.img");

3.101 run

Run GME Script: Runs the commands contained in the specified script file.

Description

This command runs the commands contained in the specified script file, which is a text file that contains any of the GME commands exactly as you would type them directly into the command window.

Note that every command in the script file must end with a semi-colon, even if you have started each new command on a new line. You can add comments to these files using # (any line beginning with this symbol is ignored).

Syntax

run(in);

in	the full path of the input script file, including the file extension

Example

```
run(in="C:\data\batchfile1.txt");
```

3.102 sample.empirical

Sample From Empirical Distribution: Draws samples from an empirical distirbution and writes the sample to the output window.

Description

This tool draws samples from an empirical distribution. It is primarily intended as a testing tool to verify that the distributions you have specified will produce the desired result when used in conjunction with the movement modelling commands, like movement.simplecrw. Please refer to the 'Distributions' appendix for detailed instructions on how to specify an empirical distribution.

Syntax

sample.empirical(in, out, sample);

in	the input empirical distribution (a character delimited text file with or without header line)
out	the output text file to write the sampled values to
sample	the number of samples to generate

Example

```
sample.empirical(in="C:\data\distrib1.csv", out="C:\data\samples.csv", sample=10000);
```

3.103 sampleperppointsalonglines

Sample Perpendicular Points Along Lines: Creates sets of points sampled at a regular intervals along lines, and perpendicular to them.

Description

This tool creates sets of points sampled at a regular intervals (distance d) along lines. The points are oriented perpendicular to the line at the line sample point, and are placed symmetrically on both sides of the line. The number of perpendicular points to sample, and the distances at which they should be placed (distances p), can be specified by the user. The line sample point that acts as the origin from which the perpendicular points are placed can also optionally be included in the output point data source.

The tool processes lines sequentially. For each line a random start distance in the interval [0,d] is identified (although this can be overridden with the 'start' parameter, which allows you to specify a constant initial distance - often 0). At that point a set of potential sample points is generated based on the set of perpendicular distances specified (p). This set of potential sample points is tested to ensure that each point is at least distance max(p) away from any line other than the current sample line, and distance max(p) away from any other generated sample point. If the user has specified the optional exclusion polygon data source the points are also tested to ensure none fall within one of these polygons. If any point in the set of perpendicular points fails one of these tests, the set is discarded. Otherwise, the points are accepted and written to the output data source. The algorithm then moves to the next point at distance +d along the line (taking into account any vertices and changes in direction along the line), and the process is repeated until the end of the line is reached.

The tool writes several ID fields to the output table that define the different levels of organisation in the data. First, the unique line ID field specified by the user is written to the point dataset so that sample points can be tied to the lines from which they were generated. Second, every output point has a unique integer ID (field PNTID). Third, each complete set of points has a unique ID (field SETID). Fourth, the perpendicular stratum of sampling is also assigned an integer ID number (field STRATUM): 0 for the line sample point (if it is included in the output), and then a sequence of numbers beginning at 1 corresponding to each perpendicular distance. If only one distance is specified, a pair of points is generated, and this unique ID will be 1. If two distances are specified, four points are generated, the inner two with an ID of 1, and the outer two with an ID of 2, and so on.

The screening of sample points to ensure minimum distances are maintained adds substantially to processing times. This feature is disabled by default but can be activited using the 'screen=TRUE' parameter. If you are interested in this screening, it is recommended that you first run the command without the screening to ensure that the output is consistent with your expectations. Then re-run the command with a different output file and the screening switched on.

The speed of processing will benefit from ensuring that i) all input spatial data sources have the spatial references explicitly defined, and ii) a spatial index has been built on the input line data source and the exclusion polygon data source if specified. No attempt should be made to access these input data sources while processing is occurring. Processing speeds

will also be improved by removing any excess lines from the input line data source, or excess polgons from the exclusion polygon data source (if specified). Spatial indices should be built or rebuilt after removing excess polygons.

Syntax

sampleperppointsalonglines(in, uidfield, out, interval, linedistance, [excl], [linepoint], [screen], [start]);

in	the input line data source containing lines along which perpendicular samples are generated
uidfield	the input line data source unique integer ID field
out	the output point data source
interval	the constant interval distance along each line at which sets of sample points are generated, or the name of a field in the input line data source containing this value for each line, e.g. 100 or "SDIST"
linedistance	the perpendicular distance(s) from each line at which sample points are generated
[excl]	a polygon data source within which points are not permitted to be generated
[linepoint]	(TRUE/FALSE) if true, includes the origin point on the input line in the output data point data source (default=FALSE)
[screen]	(TRUE/FALSE) if true, screens sample points to ensure that the maximum linedistance is maintained among points and lines (default=FALSE)
[start]	specifies the start distance of the first sample (default is a random start in the interval 0-interval)

Example

```
sampleperppointsalonglines(in="C:\data\fences.shp", uidfield="FENCEID",
out="C:\data\samples.shp", interval=1000, linedistance=500, linepoint=F, screen=T);
sampleperppointsalonglines(in="C:\data\roads.shp", uidfield="RDID",
out="C:\data\samples.shp", interval=1000, linedistance=c(100,500,1000), linepoint=T,
screen=F);
sampleperppointsalonglines(in="C:\data\fences.shp", uidfield="FENCEID",
out="C:\data\samples.shp", excl="C:\data\ponds.shp", interval=100,
linedistance=c(10,20,30,40,50), linepoint=T, screen=F, start=0);
```

3.104 save

Save Output: Saves the contents of the output window to a new folder.

Description

This system command saves the contents of the output window (the command history and the processing results, in HTML format) to the new folder you specify. You may issue this

command at any time during your session, and then continue using the interface if you wish to.

If you specify this 'save' command with no arguments then the current location of the log file is reported along with the syntax for the command.

Syntax

```
save(out);
 out   the new output folder to save the log to
```

Example

```
save(out="C:\data\mylogs");
```

3.105 setparameter

Sets GME Parameter Values: Provides the user with a way of changing important GME system variables that control the behaviour of GME.

Description

Provides the user with a way of changing important GME system variables that control the behaviour of GME. The available parameters are:

ENUMERATE "interpretonly". If TRUE, the GME interface will interpret command text but will not run the commands. Instead, the interpreted commands are written to the output window. This is useful for testing scripts that use any of the strategic commands, functions and clauses. "maxattempts". The default number of maximum attempts to complete an algorithm is 1000000. For computationally intensive algorithms this means you can wait a long time for a mis-parameterized algorithm to fail. If you reduce this value it is recommended that you do not reduce it below 10000. This settings may be particularly useful when generating random points while enforcing a minimum distance between points (this can lead to a problem with no solution). "debug". If TRUE, you will activate my debugging message system, which provides slightly a slightly more detailed stack trace. This is useful for providing detailed information on a problem you have encountered.

Syntax

```
setparameter([interpretonly], [maxattempts], [debug]);
 [interpretonly]   (TRUE/FALSE) if TRUE, the GME interface interprets command text
                   but does not execute the commands (default=FALSE)
 [maxattempts]     an integer that controls the maximum number of attempts an algorithm
                   will make to solve a problem
 [debug]           (TRUE/FALSE) if TRUE, the debugging message system is activated
                   (default=FALSE)
```

Example

```
setparameter(interpretonly=TRUE);
setparameter(interpretonly=FALSE);
setparameter(maxattempts=10000);
```

3.106 setspatialreference

Set Default Spatial Reference: Sets the default spatial reference to use when creating new feature classes that are not based on existing geodatasets.

Description

This command sets a default spatial reference to use when creating new datasets that are not based on existing datasets. For most GME commands, when new datasets are created the spatial reference of the new layer is derived from the spatial reference of the input layers (e.g. when creating a kernel density raster, the spatial reference of the output raster is acquired from the input point dataset). A few commands, however, can be run without reference to any other datasets (e.g. the genvecgrid command with the extent option). In these cases, if no default spatial reference is already defined then the output dataset is just assigned as an unknown coordinate system. Geodatabases, however, require that a spatial reference is explicitly defined. Setting the default spatial reference with this command allows you to generate output to geodatabases for the subset of commands that cannot automatically determine an appropriate spatial reference from input data.

There are three options for setting the spatial reference (use only one of them). The spatial reference can be acquired from an existing vector dataset (the vector parameter), an existing raster dataset (the raster parameter), or from a projection text file (these are normally found here: C:\Program Files\ArcGIS\Desktop10.0\Coordinate Systems).

Syntax

```
setspatialreference([vector], [raster], [prj]);
 [vector]   acquires the default spatial reference from a vector dataset
 [raster]   acquires the default spatial reference from a raster dataset
 [prj]      acquires the default spatial reference from projection file (*.prj)
```

Example

```
setspatialreference(vector="C:\data\parcels.shp");
setspatialreference(raster="C:\data\landcover.img");
setspatialreference(prj="C:\data\UTMZn13NWGS84.prj");
```

3.107 setwd

Set Working Directory: Sets the current current input and output working directories.

Description

This tool sets the current current input and output working directories. If you specify the default working directories using this command, then you do not have to specify these paths in any other commands as the tool will look for them in the working directory. This concept is very similar to the R 'setwd' command but here you can specify different default input and output directories. Using this command is highly recommended as it can greatly reduce the amount of typing you need to do when writing a command.

Note that these directories must exist before you run this command (this tool does not create new directories if they do not exist).

Syntax

setwd(in, out, all);
 in the path of the default input working directory
 out the path of the default output working directory
 all the path of the default input and output working directory

Example

setwd(in="C:\data\gis");
setwd(out="C:\data\analysis");
setwd(in="C:\data\gis", out="C:\data\analysis");
setwd(all="C:\data\gis");

Current input working directory:
C:\data\gis

Current output working directory:
C:\data\analysis

3.108 shiftrotate

Shift And Rotate Features: Shifts and/or rotates vector features and writes the result to a new output data source.

Description

This tool shifts and/or rotates vector features and writes the result to a new output data source. The shift is specified as a distance each feature is moved in the x and y directions (negative values imply a shift left/down, positive values imply a shift right/up respectively). If you do not specify a shift then no shift is applied (i.e. the default shift is 0,0). If the features need to be shifted by different amounts, then you can specify two fields in the attribute table that contain the x and y shift values for each feature.

The rotation is a more complex transformation to specify because you must define the pivot point around which the rotation is based. The rotation angle is specified in degrees

(positive values correspond to clockwise rotations, negative values to counter-clockwise rotations). The pivot point can be specified as a point that is different for each feature, or is common among all features (a global pivot point). The default is that the pivot point is global. There are five options for where the pivot point is located in relation to the rectangle that bounds the dataset (in the case of a global pivot) or the feature (in the case of local pivots): the upper left (UL), upper right (UR), lower left (LL), lower right (LR) or centre (C). Note that the centre is not the 'centre of gravity', but the geometric centre of the envelope. There is also the option of specifying your own global pivot point by supplying its coordinates, e.g. pivot=c(315000,4950000).

If your goal is to adjust the spatial position of all the features in your dataset you will probably want to use a global pivot point. If your goal is to use this tool in the context of randomization tests then you will probably want to use a local pivot point.

The order in which a shift and rotation are applied can affect the result. By default the rotation is applied before the shift but you can override this using the 'rotfirst' parameter.

Syntax

shiftrotate(in, out, [shift], [rot], [pivot], [global], [rotfirst], [copyfields], [where]);

in	the input feature data source
out	the output feature data source
[shift]	the distances to move features along the x and y axes, or the field names of the fields containing these values (must be in coordinate system units)
[rot]	the rotation angle in degrees, or the field name of the field containing these values
[pivot]	the pivot point for the rotation (upper left: UL or NW, upper right: UR or NE, lower right: LR or SE, lower left: LL or SW, centre: C) or a coordinate pair (default=C)
[global]	(TRUE/FALSE): if TRUE the pivot definition applies to the entire dataset (e.g. the NW corner of the dataset), if FALSE it applies to the current feature (e.g. the centre of the current feature) (default=TRUE)
[rotfirst]	(TRUE/FALSE): if TRUE and both a shift and rotation are specified, the rotation is applied before the shift, if FALSE the shift is applied first (default=TRUE)
[copyfields]	(TRUE/FALSE): if TRUE the fields in the input attribute table are copied to the output attribute table (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
shiftrotate(in="C:\data\plots.shp", out="C:\data\newplots.shp", shift=c(-10000,500),
rot=45, copyfields=TRUE);
shiftrotate(in="C:\data\plots.shp", out="C:\data\newplots.shp",
shift=c("XVAL","YVAL"), rot="ROTVAL", pivot=LL, global=FALSE, rotfirst=FALSE);
```

3.109 simplify

Simplify Lines / Polygons: Simplifies lines or polygons by removing the vertices that contribute least to the shape.

Description

This tool creates simplified lines or polygons by removing vertices that contribute least to the overall shape of the feature. Specifically, this tool implements the Douglas-Peucker algorithm. There is often considerable redundancy in the definition of spatial features for a given application. Although this may not be an issue for many applications, some computationally intensive applications can achieve significant speed increases using simpler data. The idea here is that by simplifying the geometry of the shape, we facilitate faster analysis speeds while not sacrificing any important functional information contained in the geometry. I have been staggered at how effective this algorithm is at reducing the number of vertices while maintaining the overall shape of the feature.

The tolerance is the key parameter. High tolerances will result in greater simplification. It is worth testing a variety of tolerances to find the level that best suits your application. As a general rule of thumb, you might start by roughly estimating the average distance between vertices, and set your tolerance to be 10x greater than that. For instance, for features like lakes the distance between vertices might be roughly 5m in the UTM projection, and an initial tolerance of 50m is a reasonable starting point. However, the level of simplification that is acceptable depends very much on your application and it is absolutely essential to closely compare the original and simplified datasets to determine if the simplification could have a negative impact on your analysis.

Of particular concern are simplifications of polygons, especially polygons with holes or portions of which are long and thin (rivers, peninsulas, etc). The risk is that if the tolerance is too high then adjacent sides of the polygon will eventually overlap, fundamentally changing the overall shape of the geometry. This is why it is important to inspect the output of the tool closely and use this as the basis for judging an appropriate tolerance.

Syntax

simplify(in, out, tol, [copyfields], [where]);

in	the input line or polygon feature data source
out	the output feature data source
tol	the tolerance distance that controls the degree of simplification (specified in coordinate system units)
[copyfields]	(TRUE/FALSE): if TRUE the fields in the input attribute table are copied to the output attribute table (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)

Example

```
simplify(in="C:\data\rivers.shp", out="C:\data\riverssimple.shp", tol=50,
copyfields=TRUE);
simplify(in="C:\data\lakes.shp", out="C:\data\lakessimple.shp", tol=100,
copyfields=FALSE, where="AREA > 1000");
```

3.110 simulation.gridspread

Grid Spread Simulation: Simulates the spread of an agent (e.g. fire) triggered by random events (e.g. lightening) on a raster landscape in discrete time (a cellular automata model)..

Description

This program simulates the spread of events among neighbouring cells in a raster landscape. It is designed to be as general as possible, though is inspired by the spread of fires that have been started by lightening strikes.

There are two major components to this model: 1) events that occur randomly in time and space and that can trigger a response in a landscape cell (e.g. lightening strikes starting a fire), and 2) the spread of the disturbance (e.g. fire) among neighbouring cells. This is a discrete time, discrete space simulation, and is a form of cellular automata model.

There are two spatial datasets needed to run this model, though the same raster can be used for both purposes. First, a binary (0/1) raster representing the landscape cells in which the random events can trigger a response (e.g. cells that can be ignited by lightening). Second, a binary raster representing the cells in which spread can occur. Although these may be the same raster for some processes, the flexibility exists to specify two different rasters.

There are several parameters that must also be specified. The random event component is driven by a rate parameter ("eventrate"), which must be specified in the same units as the time step of the model, and a parameter ("suscptconst") defining the probability that an event triggers a response on the landscape. The ignition component of the model works as follows. The number of events that occur in a time step is determined at the beginning of each time step by drawing from a Poisson distribution using the specified eventrate parameter. Each of these events are then randomly located on the landscape: a random row and column is selected, and provided the cell is not a NoData cell, then the event is applied to that cell. An event causes a response if a random number in the range [0,1] is less than or equal to the specified susceptibility probability (susceptconst) multiplied by the cell value (0 or 1). As such, if the cell value is 0, no response is triggered, and if the cell value is 1, then a response is triggered proportionally to susceptconst. Events occurring in NoData cells are recycled until they occur in a 0 or 1 cell. You can force an event to occur in the first time step (this is essentially equivalent to starting monitoring when an event first occurs) use the "force" parameter.

The spread model is driven by a spread probability parameter ("spreadconst") in the range [0,1]. In each time step, spread can occur from currently active (e.g. burning) cells and

cells in the spread raster with a value of 1 that are not yet burning. The neighbourhood is defined as the 8 cells surrounding a cell, so spread can occur on diagonals. The probability of spread is proportional to the number of active cells surrounding a cell. A value is drawn from a binomial distribution with probability parameter spreadconst, and a size parameter equal to the number of active neighbouring cells (1-8). This random number corresponds conceptually to the number of times that the event spreads into the current cell from neighbours, and can theoretically range from 0-8. Any value greater than 0 results in spread, and the current cell becomes active in the next time step.

Cells that have just become active, either through initial ignition, or through local spread, cannot themselves spread to other cells until the next time step. The duration of the event (number of time steps) is controlled by the "duration" parameter, and must be at least 1 to allow any local spread to occur. When this duration is exceeded, the event expires, and spread can no longer occur to or from this cell.

The "timesteps" parameter controls how many time steps the simulation is run for, and "iterations" controls the number of independent iterations of the simulation to perform. All output is written to the "out" folder, using the "prefix" you specify. There is both tabular and raster output. The rasters represent the state of cells at a given time step (raster values: (1) no event has taken place, (2) an event is currently active in this cell, (3) an event previously occurred and has now expired). The frequency at which these rasters are written is controlled by the "writefreq" value. By default, only the final raster in the time series is written. If writefreq is 1 then rasters are written at every time step, and if writefreq is 10 then rasters are written every 10th time step. It is recommended you only specify values that are exactly divisible into the number of time steps (e.g. if timesteps is 100, valid values of writefreq include 1, 2, 4, 5, 10, 20, 25, 50). If you view these rasters in ArcMap it is recommended that you switch the symbology to "unique values".

Three statistics files are also written. "prefix_modelsummary.txt" contains basic summary information and records the parameter values used in the simulation. "prefix_rawdata.csv" contains summary statistics for every time step in every iteration. "prefix_meansxtime.csv" contains mean summary statistics for every time step among all iterations. The summary statistics are:

ENUMERATE Iteration: iteration number Timestep: the time step number EventCount: the cumulative number of events that have occurred NewEventCount: the number of new events in the current time step EventReponses: the cumulative number of events that have resulted in a response in a cell SpreadCount: the cumulative number of spread events that have occurred NewSpreads: the number of spread events in the current time step Expired-Count: the cumulative number of expired disturbances (fires go out) NewExpires: the number of expirations in the current time step ActiveCells: the number of active cells from which spread can occur in the current time step

Limitations. This simulation program is limited to rasters that are no larger than 50 million pixels. If you specify a larger raster, it will read the first 50 million pixels and perform the simulation on that portion alone. This can be a computationally intensive program to run with large datasets.

Syntax

simulation.gridspread(susceptibility, susceptconst, spread, spreadconst, eventrate, out, prefix, timesteps, duration, iterations, [force], [writefreq]);

susceptibility	binary raster (0/1) representing the susceptibility of a cell to the random events
susceptconst	a value in the range [0,1] representing the susceptibility of the raster cells to the random events (this value is applied to cells of value 1 in the susceptibility raster)
spread	binary raster (0/1) representing the susceptibility of a cell to spread of events from neighbouring cells
spreadconst	a value in the range [0,1] representing the susceptibility of the raster cells to spread of events from neighbouring cells (this value is applied to cells of value 1 in the spread raster)
eventrate	the rate at which events randomly occur within the defined landscape, expressed in the same units as the specified time step
out	the output folder to which all data is written (a new, empty folder is strongly recommended)
prefix	the file name prefix used when writing all output (a short, simple prefix is recommended)
timesteps	the number of time steps that the simulation is run for
duration	the duration of an event in a single cell (number of time steps)
iterations	the number of iterations to run the simulation
[force]	(TRUE/FALSE) If TRUE, forces an event at the start (default=FALSE)
[writefreq]	The frequency (number of time steps) at which output rasters are written e.g. 1=every time step, 10=every 10th time step (default: writes only the final raster)

Example

simulation.gridspread(susceptibility="C:\data\landscape\susceptible.img", susceptconst=0.5, spread="C:\data\landscape\spread.img", spreadconst=0.1, eventrate=0.76, out="C:\data\landscape\sims", prefix="fires1", timesteps=100, duration=2, iterations=10, force=TRUE, writefreq=25);

3.111 snappoints

Snap Points To Features: Snaps a set of input point points to the features in a vector layer (containing points, lines or polygons) and writes the result to a new point dataset.

Description

This tool snaps a set of input points to the features in a vector layer (containing points, lines or polygons). The output is a new point dataset, and the attributes from the input dataset are copied to the output dataset. You must specify a tolerance that controls whether the

snap is allowed. If snapping a feature would result in moving the point greater than the tolerance distance then the snap does not take place and the original unsnapped point is written to the output dataset. This snap tolerance allows you to control whether only small snaps are allowed (thereby just tweaking an existing dataset) or to snap all the features by setting the tolerance to a large number.

Note that the larger the tolerance value, the longer it will take to process each record (because the algorithm must search through a greater number of features within that tolerance radius to find the closest one). So if you are concerned about processing time a good strategy is to set the tolerance no larger than it really needs to be. For instance, if you know that there are no points greater than about 2500m from a line, then setting the tolerance to 3000 will accomplish the same thing as setting it to 10000, but it will do so more quickly.

It is recommended that you specify a new dataset as the output dataset. Although you can specify an existing dataset, you must ensure that any fields having the same name in both the input point dataset and the output dataset are of the same type.

An additional field (SNAPDST) is added to the output dataset that records the distance the point was moved. If the point was not moved then this field records a value of -999 (NoData). If a field SNAPDST already exists (because it was already in the input point attribute table, perhaps as a result of using the snappoints command previously) then the algorithm will find a new unique SNAPDST field by adding a number to the end (1, 2, ...) until it finds a unique name.

Syntax

snappoints(in, snap, out, tol, [where]);

in	the input point data source
snap	the input feature source to which points are snapped
out	the output point data source
tol	the maximum snap distance in coordinate system units
[where]	the selection statement that will be applied to the point feature data source to identify a subset of points to process (see full Help documentation for further details)

Example

```
snappoints(in="C:\data\locs.shp", snap="C:\data\roads.shp",
out="C:\data\snappedlocs.shp", tol=5000);
snappoints(in="C:\data\locs.shp", snap="C:\data\lakes.shp",
out="C:\data\shorepnts.shp", tol=250);
```

3.112 splitdataset

Split Dataset: Splits a vector dataset into several separate datasets based on an ID value in one of the attribute fields.

Description

This command writes the features and attribute table from a vector data source (a shapefile or geodatabase) and creates multiple new vector data sources based on an ID field that contains unique ID's for each of the new datasets you wish to create. Typically these unique ID values will be integers, although text ID fields are also acceptable.

The new output can be a geodatabase, or even a new feature dataset within a new or existing geodatabase. See the section on 'Working with geodatabases' for further details.

GME has been designed to avoid the problem of having to split datasets. Many of the vector commands support the 'where' clause, which allows you to process a subset of features. It is highly recommended that you learn about the 'where' clause as it is one of the more powerful aspects of GME. See the 'Strategic commands' section and the 'where' section for further details.

Syntax

splitdataset(in, uidfield, outws, prefix);

in	the input feature data source
uidfield	the unique ID field of the input feature data source
outws	the output workspace, which can either be a folder or a geodatabase (if a geodatabase, it must end with an exclamation mark - see full Help documentation for further details)
prefix	the prefix to use when naming the new feature data sources (the unique value is appended to the prefix)

Example

splitdataset(in="C:\data\roads.shp", uidfield="ROADTYPE",
outws="C:\data\roadgdb!bytype!", prefix="RDTYPE");
splitdataset(in="C:\data\telemetry.shp", uidfield="ANID", outws="C:\data\animals",
prefix="ANIM");

3.113 sumlinelengthsinpolys

Sum Line Lengths In Polygons: Sums the lengths of the portions of all lines that intersect each polygon.

Description

This tool sums the lengths of the portions of all lines in the specified line data source that intersect each polygon in the polygon data source, and writes the result to a new field in the polygon attribute table. You may optionally also specify a weight field in the line attribute table that is used to weight the line lengths: the length of the line intersecting the polygon is multiplied by the weight value.

This can be a computationally intensive program to run with large datasets. In such cases ensuring that your input data sources have a spatial index built will improve processing efficiency.

Syntax

sumlinelengthsinpolys(line, poly, field, [weight], [where], [update]);

line	the input line feature source
poly	the input polygon feature source
field	the output field name
[weight]	the name of the field in the line attribute table to weight the line lengths by (default=no weight field)
[where]	the selection statement that will be applied to the polygon feature data source to identify a subset of polygons to process (see full Help documentation for further details)
[update]	(TRUE/FALSE) if TRUE and you specify an existing field, the existing field will be updated rather than generating an error message (default=FALSE); warning: this option will result in overwriting of existing data and is therefore potentially dangerous.

Example

```
sumlinelengthsinpolys(line="C:\data\roads.shp", poly="C:\data\counties.shp",
field="ROADLENGTH");
sumlinelengthsinpolys(line="C:\data\roots.shp", poly="C:\data\plots.shp",
field="LINESUM", weight="WIDTH");
```

3.114 system

Execute System Command: Executes a system command via a command prompt.

Description

This command executes a system command via the command prompt. Like the r() command, the system() command is different from most other GME commands in that it takes the text within the parentheses and executes it directly, without any interpretation of the text. For this reason you should not enclose the command text within quotes: write the system commands here exactly as you would write them at the command prompt. This command waits for the command to finish before continuing, but performs no checks regarding the success or failure of the execution of the system command.

It is recommended that for all but the simplest system commands that you call a batch file.

Syntax

system(command);
command System command to be executed (not in quotes)

Example

system(notepad.exe);
system(notepad.exe "C:\data\instructions.txt");
system(C:\data\script.bat);

3.115 timer

Command Timer: Switches on or off the command timing utility that reports the processing time for each command in the output window.

Description

For planning and scheduling purposes it is sometimes useful to know how long it takes to run commands on your datasets. Although timing data is always recorded and viewable in the GME File – Command History interface, this command will also report timing data directly in the output window (this creates a more permanent timing record that you can view even after closing GME, whereas the Command History data will be lost when GME closes).

The timer in no way consumes processing power or slows down processing. The reported times are only accurate to within 1 second, so the timer is not useful for timing commands that run very quickly.

Syntax

timer(on);
on (TRUE/FALSE) if TRUE, processing times are reported for most commands

Example

timer(on=TRUE);
timer(on=FALSE);

3.116 uniquevalues

Unique Values: Reports the unique values for a specified field in an attribute table.

Description

This command generates a sorted list of unique values for a specified field in an attribute table. If the optional parameter 'rformat' is set to TRUE then the list is reported using R syntax so that the result can be copied and pasted into R, thereby creating a vector in R that contains the unique values. If the optional variablename parameter is specified, a GME variable is also created containing the unique values (this is useful for scripting and automation in GME).

Syntax

uniquevalues(in, field, [rformat], [where], [variablename]);

in	the input vector feature source
field	the field name to process
[rformat]	(TRUE/FALSE) writes the output as in R vector format (default=FALSE)
[where]	the selection statement that will be applied to the feature data source to identify a subset of features to process (see full Help documentation for further details)
[variablename]	if specified, a GME variable containing the unique values is created

Example

```
uniquevalues(in="C:\data\roads.shp", field="ROADNAME");
uniquevalues(in="C:\data\countries.shp", field="COUNTRYNAME",
variablename="countries");
uniquevalues(in="C:\data\telemetry.shp", field="ANIMALID", rformat=TRUE);
```

4 SPATIAL ANALYSIS AND MODELLING TOPICS

4.1 Creating binary and weighted polygon adjecency matrices based on shared borders

Purpose and method

Adjacency matrices are useful for incorporating spatial relationships among features (in this case polygons) into models and statistical analyses. An adjacency matrix simply defines which polygons are adjacent to one another. It can take the form of a binary (1/0) variable, where 1 indicates the polygons touch and 0 indicates they do not, or the matrix can consist of continuous values that quantify not only whether two polygons touch, but how much they touch based on the length of the shared border between them. Here, I describe how to create both of these matrices and save them as R objects.

The starting point is polygon layer. In this example the polygons depict administrative districts. The assumption is that the polygons do not overlap substantially, although this procedure is tolerant of small amounts of spatial inaccuarcy that might result in slight overlap of polygons along boundaries, or slight gaps between polygons that we would really like to think of as adjacent.

If your polygon layer contains multiple, disjunct polygons that represent the same feature, you should run the ArcGIS dissolve command to ensure that these are presented as multipart features, not separate polygons. Let me explain that with an example. One of the districts in my example is bisected by a large river, and as a result the district consists of two disjunct polygons. For my application I wish to quantify adjaceny at the district level and it is not important to consider finer-scale nuances in adjancency that might be caused by component polygons. This is a judgement decision you need to make for your application. What you need to run this analysis is, therefore, a polygon layer that contain exactly one polygon (but it can be a multipart polygon) per level of your variable of interest. So my polygon layer contains exactly one polygon (one record in the table) per district ID.

The first step is to run the GME calc.sharedborders command on the polygon layer. It requires that we have a unique identifier field.

```
setwd(all="C:\data\canada\provinces");
calc.sharedborders(in="quebec_districts.shp", uidfield="CDID", out="qcd_sharedborders.shp",
tol=5);
```

The 'tol' (tolerance) parameters is what gives us some lee-way with inaccuracy in the spatial borders, and it is specified in units of the coordinate system of the data (meters in my example). Typically a value in the range of 1-10 m will be about right, and will have negligable influence on the quantification of adjacency.

Once that is finished, we load the relevant data into GME's R session.

```
r.loaddata(in="quebec_districts.shp", fields=c("CDID"));
r.loaddata(in="qcd_sharedborders.shp", fields=c("SRCUID","NEIGHUID","LENGTH"));
```

Finally, we run an R script to construct and save the matrices. Save the next block of code as a text file (such as adj.txt), and change the first few lines to match your data if you need to. You will probably only need to change the name of the unique ID field (the id variable), and possibly the file names at the end of the script.

```
# change these variables to match your data
id <- CDID
v1 <- SRCUID
v2 <- NEIGHUID
v3 <- LENGTH
n <- length(id)
# create length of shared border matrix (lsbm)
lsbm <- matrix(0, nrow=n, ncol=n)
for (i in 1:length(v1)){
lsbm[which(id==v1[i]),which(id==v2[i])] <- v3[i]
lsbm[which(id==v2[i]),which(id==v1[i])] <- v3[i]
}
# create binary adjaceny matrix (bam)
bam <- matrix(0, nrow=n, ncol=n)
for (i in 1:length(v1)){
bam[which(id==v1[i]),which(id==v2[i])] <- 1
bam[which(id==v2[i]),which(id==v1[i])] <- 1
}
# save these objects so they can be accessed from other R sessions
save(lsbm, file="lsbm.r")
save(bam, file="bam.r")
save(id, file="amid.r")
```

Once you have saved that as a text file, you need to run it from GME.

```
r(setwd("C:\\data\\canada\\provinces"));
r(source("adj.txt"));
```

Check in your working directory to make sure the files exist. To use them in any other R session you can load them using these R commands:

```
setwd("C:\\data\\canada\\provinces")
load("amid.r")
load("lsbm.r")
load("bam.r")
```

Note that we needed to save the id vector so that we know which polygon ID corresponds to which matrix row and column. The matrices will be symmetric. A convenient way of inspecting these matrices is with the R image command.

```
image(bam)
image(lsbm)
```

5 APPENDIX

5.1 Specifying colours in R

There are a number of ways of specifying colours in R, but one of the most inuitive is to simply specify the name of the colour. There are several hundred pre-defined colours in R that you can choose from (see the figures below). Note that the names are spelling and case sensitive. While the grey-scale colours support both international and U.S. spelling (grey and gray respectively), some of the colours do not so particular care must be taken to use the spelling provided (e.g. darkslategray1-4, slategray1-4).

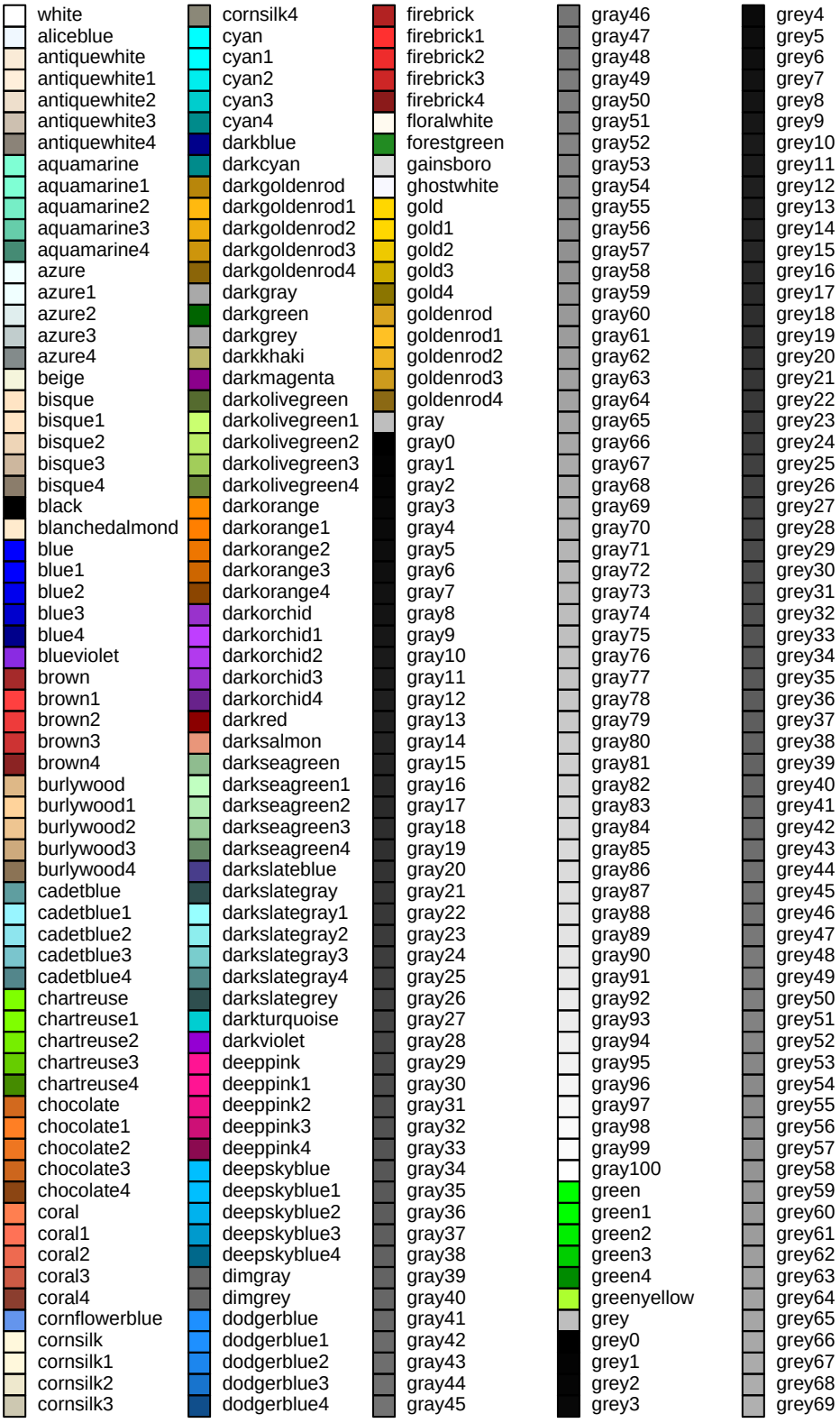

Figure 3: R colour names (part 1 of 2)

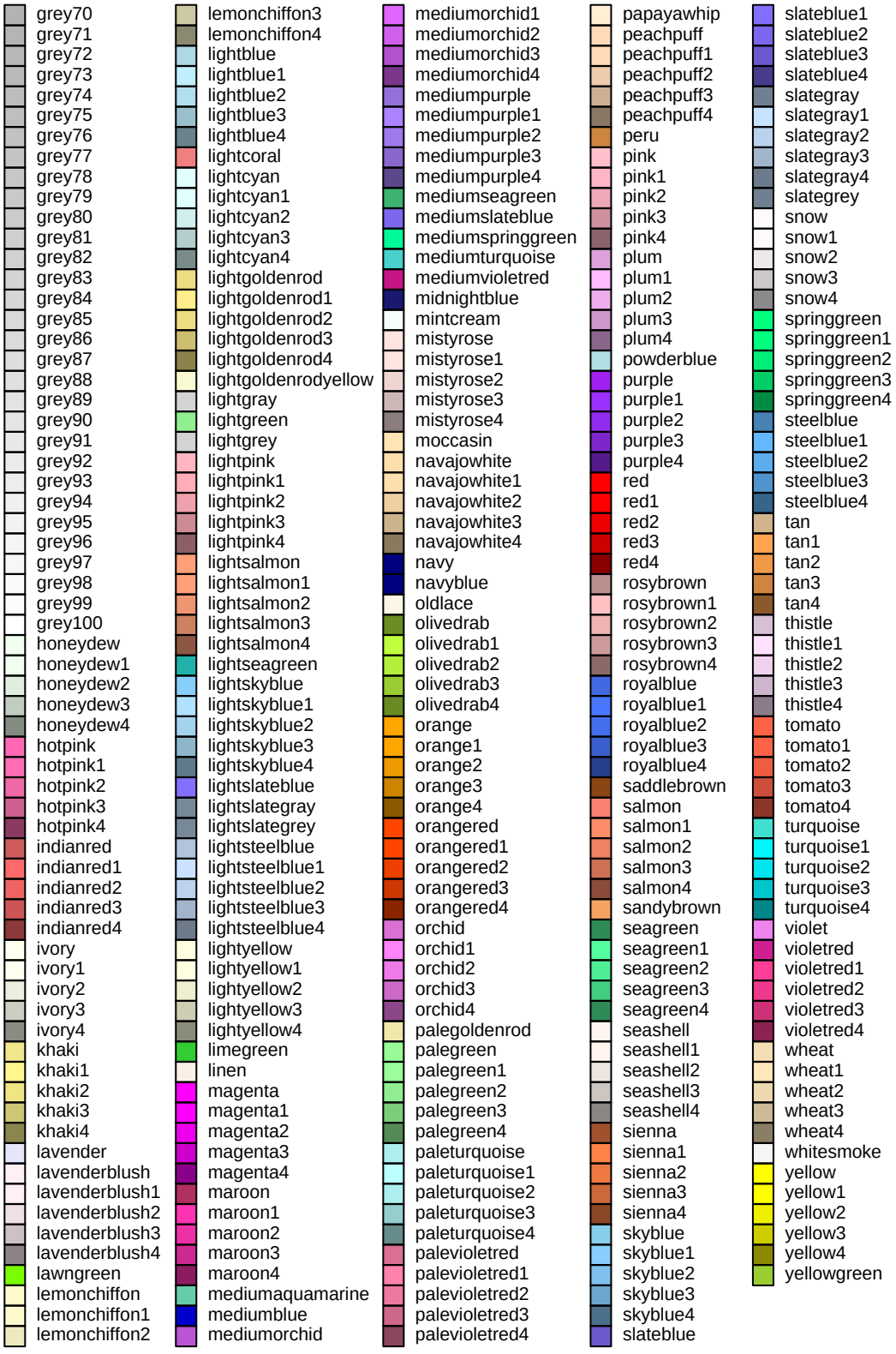

Figure 4: R colour names (part 2 of 2)

5.2 HawthsTools command reference

This table allows you to look up the GME command that corresponds to the functionality of a tool in HawthsTools. Note that there are many more commands in GME than are listed in this table (that correspond to functionality that was not in HawthsTools).

HawthsTool's tool name	GME command
Analysis Tools	
Intersect Point Tool	isectpntrst
Distance Between Points (Within Layer)	pointdistances
Distance Between Points (Betw. Layer)	pointdistances
Count Points In Polygons	countpntsinpolys
Polygon In Polygon Analysis	isectpolypoly
Sum Line Lengths in Polygons	sumlinelengthsinpolys
Line Raster Intersection Statistics	isectlinerst
Enumerate Intersecting Features	listintersectingfeatures
Line Metrics	Functionality not currently available
Sampling Tools	
Create Random Selection	r.sample
Random Selection Within Subsets	r.sample
Generate Random Points	genrandompnts
Generate Regular Points	genvecgrid
Conditional Point Sampling Tool	gencondrandompnts
Create Vector Grid (lines/polygons)	genvecgrid
Create Sample Shapes (various shapes)	genshapes
Generare Random 3D Points	Functionality not currently available
Animal Movements	
Create Minimum Convex Polygons	genmcp
Calculate Movement Parameters	movement.pathmetrics
Convert Locations To Paths	convert.pointstolines
Convert Paths to Points	convert.linestopoints
CRW Simulation I	movement.simplecrw
CRW Simulation II	movement.simplecrw
Kernel Tools	
Fixed Kernel Density Estimator	kde
Batch Fixed Kernel Density Estimator	kde
Percent Volume Contour	isopleth

(Table continues on next page...)

HawthsTool's tool name	GME command
Raster Tools	
Clip Raster	clipraster
Clip Raster By Polygons	cliprasterbypolys
Landscape Characterization (fast)	neighbourhoodstatistics
Extract Raster Edge	extractedge
Thematic Raster Summary (by polygon)	isectpolyrst
Zonal Statistics ++ (by polygon)	isectpolyrst
Spatial Replace Tool	Functionality not currently available
Maximum Grid Separation Tool	Functionality not currently available
Cellular Automata (1D x Time)	Functionality not currently available
Grid Spread (Cellular Automata)	simulation.gridspread
Raster Pixel Type Conversion	Functionality not currently available
Table Tools	
Add Area Field To Table	addarea
Add Length Field To Table	addlength
Add XY To Table	addxy
List Unique Values	uniquevalues
Sum Values	Functionality not currently available
Delete Multiple Fields	deletefield
Add XY Line Data (creates line layer)	convert.tabletolines
CSV Management Tool	see commands 'file.*'
Vector Editing Tools	
Create Buffers (Retain Attributes)	buffer
Vector Rotation and Shifting Tool	shiftrotate
Snap Points To Lines Tool	snappoints
Intersect Lines (Make Points)	isectfeatures
Split Vector Layer By Unique Value Field	splitdataset
Specialist Tools	
River Sample Extraction	Functionality not currently available
Point Redistribution Tool	Functionality not currently available
PLSS Point Finder	Functionality not currently available
Julian Day Lookup	julian

5.3 End User License Agreement

END USER LICENSE AGREEMENT (EULA) FOR THE GEOSPATIAL MODELLING ENVIRONMENT (GME)

Before proceeding with the installation and/or use of this software please read the following terms and conditions of this license agreement. The license agreement applies to both the GME interface and the installation program.

By installing or using this software you indicate your acceptance of this agreement. If you do not accept or agree with these terms, you may not download, install or use it.

1. USE

Permission is hereby granted, free of charge, to use this software and associated documentation files (the "Software") subject to the terms of this EULA. This license does not transmit any intellectual rights on the Software. The Software is protected by copyright.

You, the user, are responsible for ensuring that the Software is used appropriately, and that the output of this Software is accurate, relevant, consistent, and otherwise error-free. The author does not warrant that this software is bug free.

2. COPYING AND DISTRIBUTION

You are free to copy and redistribute this software within your own organization. You may refer other people to the www.spatialecology.com website to facilitate distribution of the software to other organizations. You are expressly forbidden from i) selling, leasing, or in any other way deriving a profit from the distribution of this Software, and ii) making this software available on any website, or public access FTP site.

3. WARRANTY AND LIMITATION OF LIABILITY

THIS SOFTWARE IS PROVIDED BY THE AUTHOR 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index