Last Updated: 16 February 2023

Prepared by: Kevin McGarigal

Tutorial 1. Setting Up Software and Inspecting Grids

In this tutorial, you will inspect the grids (using either ArcMap, a text editor, or R) to be analyzed in the subsequent tutorials.

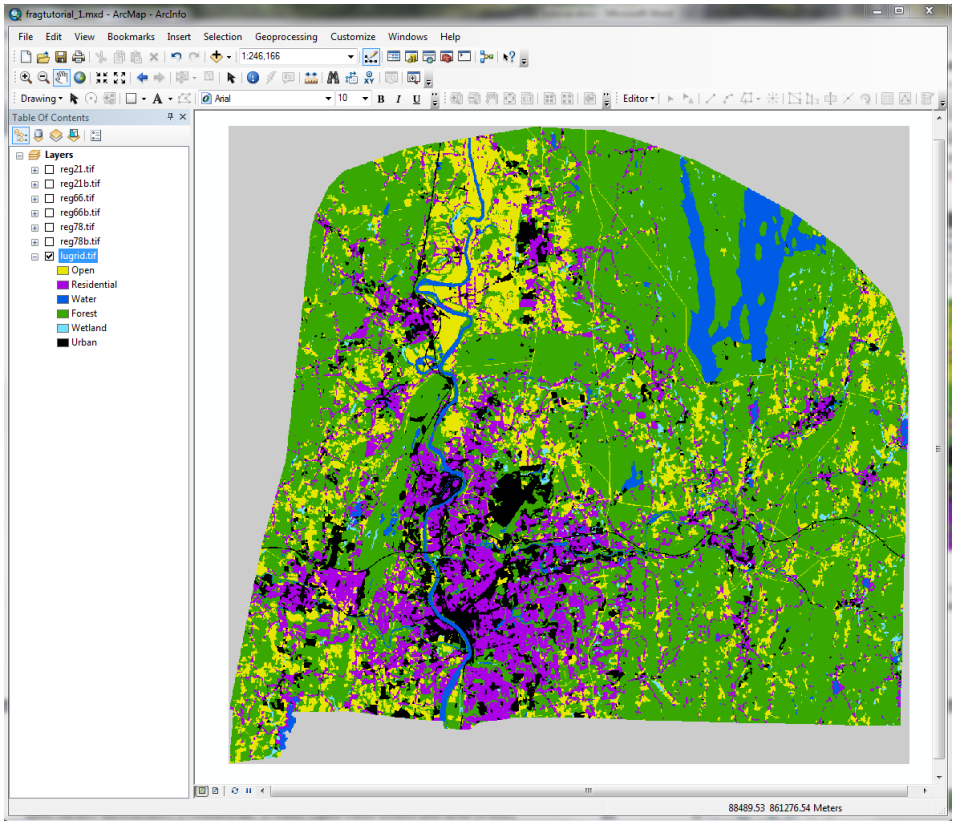
1. Download and install FRAGSTATS

First, if you haven't already done so, download FRAGSTATS 4.x and run the setup utility to install the software on your computer.

2. Inspect Geotiff grids in ArcMap

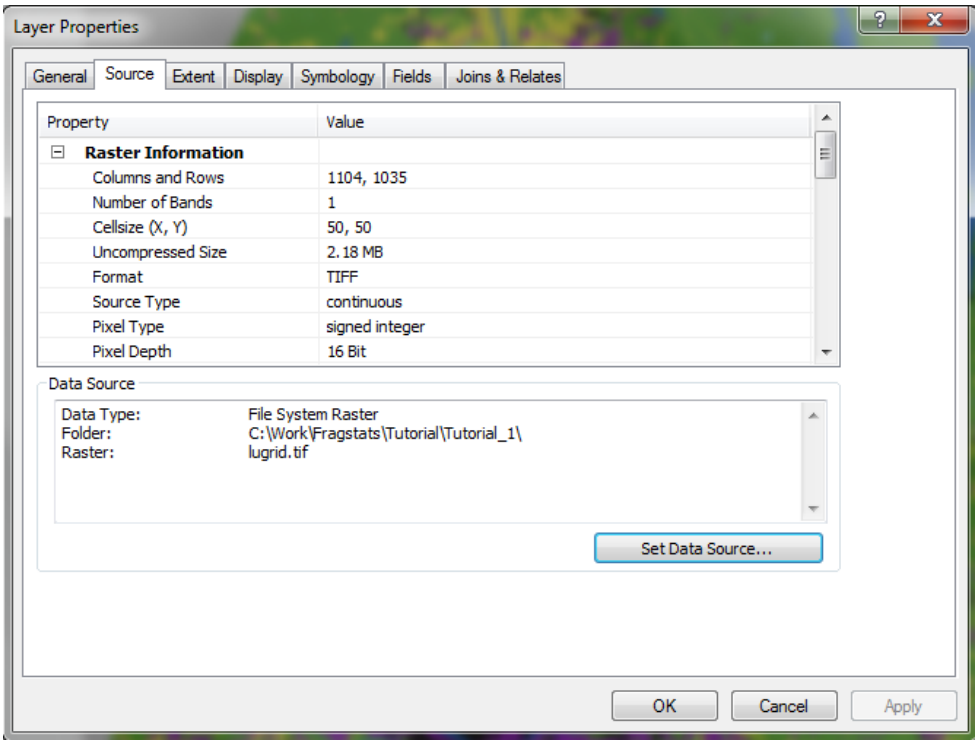
Next, inspect the grids to make sure you understand the landscape definition before analyzing them, since the results of the analysis can only be interpreted in the context of the landscape definition.

If you are planning on working with Geotiffs (preferred format) and wish to inspect the grids using ArcMap, open up the provided **fragtutorial_1.mxd** project in ArcMap. The project contains several data layers, as listed in the table of contents, including a landcover grid (**lugrid.tif**) for an arbitrary extent in western Massachusetts, and all are Geotiffs. Note, if the paths to the layers are broken, you may have to repair the data source and point to the layers in the appropriate folder.



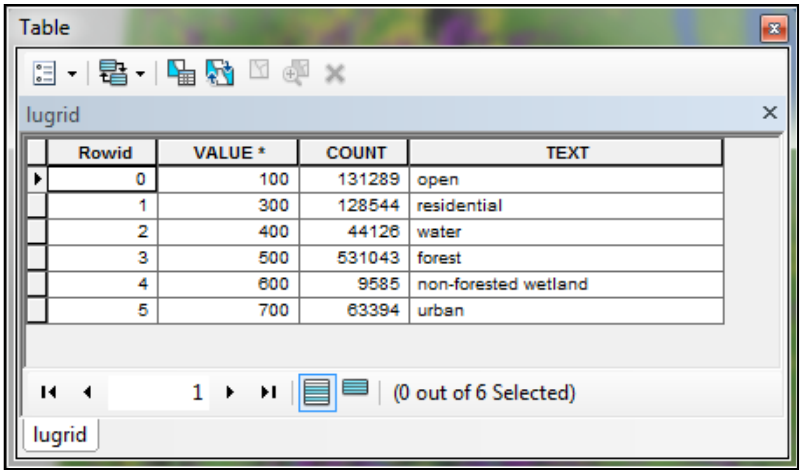
As you can see from the legend, the **lugrid.tif** contains six landcover classes, including: 1) open (largely agriculture), 2) residential, 3) water (open water bodies and large rivers), 4) forest, 5) wetland, and 6) urban.

Open the lugrid layer properties and inspect the grid properties on the Source tab. In particular, note the grid dimensions (1104 columns by 1035 rows), cell size (50 m), format (GRID), and pixel type (signed integer). The *signed* integer pixel type is necessary if the landscape has a border; i.e., strip of classified cells outside the landscape boundary and assigned negative class values. If the landscape does not contain a border, then an *unsigned* integer type is OK. In this case, the lugrid does not contain a border, but the sub-landscapes (below) do, so the pixel type has been set to signed integer.

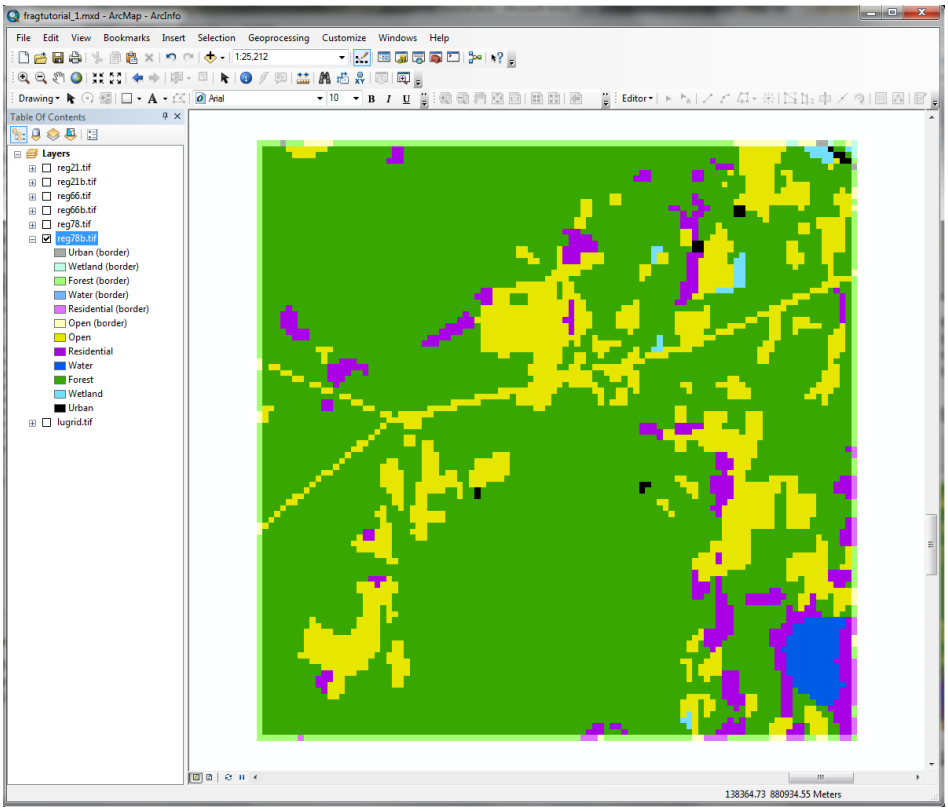


Next, open the lugrid layer attribute table and inspect the class values present on the grid. Note the class values and text description for each class. You will need to know the class values later on.

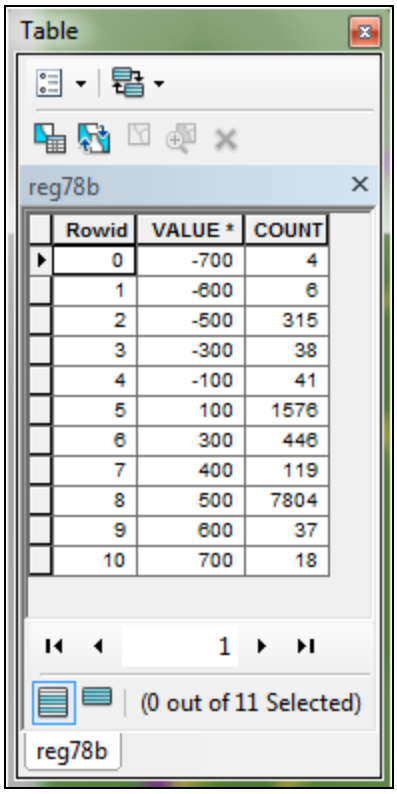
Next, view the **reg78b.tif** grid by selecting it in the table of contexts and zooming to the layer extent. This grid is a randomly selected roughly 5x5 km (25 km^2) square sub-landscape sampled from the lugrid. Expand the legend and note the landcover classes



present; it has the same six landcover classes as before, but with the addition of six "border" classes.



The **border** is simply a strip of classified cells surrounding the landscape of interest that provides information on patch type adjacency along the landscape boundary. In the legend provided, the border classes have been assigned a lighter shade of the color assigned to the corresponding class inside the landscape boundary. Importantly, a border is identified in FRAGSTAST by negatively valued cells. An inspection of the grid attribute table reveals the same six landcover classes as before, but with both a positive (inside the landscape boundary) and negative (border, or outside the landscape boundary) version of each class. Briefly, I created a custom script that clips the lugrid layer with a polygon coverage for one of the sub-landscapes, and then buffers the polygon by 50 m and clips the lugrid layer again but with the buffered (i.e., slightly larger) polygon. Next, the larger grid is multiplied by -1 to convert the cell values to negative. Lastly, the two grids are merged, with the smaller grid (the sub-landscape of interest) on top, resulting in positive values everywhere except the narrow strip of cells in the border. Note, this is a basic geoprocessing script that can be implemented in any number of languages depending on the users preference, and thus is not presented here.



Lastly, view the other sub-landscape grids in the table of contents. There are three different sub-landscapes: reg78, reg66, and reg21, each of which also contains a version with a landscape border: reg78b, reg66b, and reg21b. These landscapes differ largely in the amount of forest landcover.

2. Inspect Ascii grids in text editor

If you are planning on working with Ascii grids and don't want to inspect them using R (see below), you can inspect the grids with a text editor. Ascii files are interpretable. They are not pretty to look at and you can't do too much with them in their raw form, but it is useful to know what these files look like.

Open up **reg78b2.asc** in a text editor (the top left portion of this file is shown below). This a space-delimited ascii file (i.e., there is a space between each cell entry) and is therefore interpretable. Note, this ascii file was created by converting the reg78b ArcGrid to an ascii file in ArcMap. Note the header information included in the first six lines of the file. This header information must be deleted before it can be analyzed in FRAGSTATS; however, the information on the grid dimensions (ncols=102 and nrows=102), cellsize (50), and nodata value (-9999) will be needed later when parameterizing the FRAGSTATS model. In particular, note the landscape border indicated by the negative class values in the first row and column.

```
ncols         102
nrows         102
xllcorner     137882.625
yllcorner     875599.5625
cellsize      50
NODATA_value  -9999
-500 -500 -500 -500 -700 -500 -500 -500 -100 -100 -100 -100 -100 -500 -500 -500 -500 -500 -500 -5
-500 500 500 500 500 100 100 100 100 100 100 100 500 500 500 500 500 500 500 500 500 500 500 300
-500 500 500 500 500 500 100 100 100 100 500 500 500 500 500 500 500 500 500 500 500 500 500 300
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 300 300
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
-500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500 500
```

3. Inspect Ascii grids in R

Viewing the ascii grids is a bit more difficult without importing them into your favorite GIS. However, if you are an R user, you can use the following script (or open the provided script, **tutorial_1.R**) to plot the grid in R. Note, there are several ways to plot the grids in R. If you are familiar with the Raster or Terra packages, you can import the

ascii grids and plot them quite easily, but specifying a color scheme for the legend and plotting a pretty legend is a bit tricky. The following script makes use of the base functions in the Graphics library:

First, set the working directory to wherever you have installed the tutorial; e.g.:

```
setwd('c:/work/fragstats/tutorial/tutorial_1')
```

Next, read in the ascii grid, as a matrix, into an object (m):

```
m<-as.matrix(read.table('reg78b.asc'))
```

Next, in order to assign colors to each landcover class, identify each unique class value:

```
uv<-sort(unique(as.vector(m)))
```

Next, create breaks for assigning colors to class values (breaks are at the minimum -1, midpoints and maximum +1). Note, this is necessary because the plot function (image) is designed for continuous variables not categorical variables as is the case with the landcover image:

```
my.breaks<- (c(min(uv)-2, uv) + c(uv, max(uv)+2 ))/2
```

Next, create a color legend for the plot:

```
my.colors<-c('gray','lightskyblue','lightgreen','lightpink','lightyellow','yellow',
    'purple','slateblue','green','skyblue','black')
```

Next, check to make sure you have a color for every unique class value:

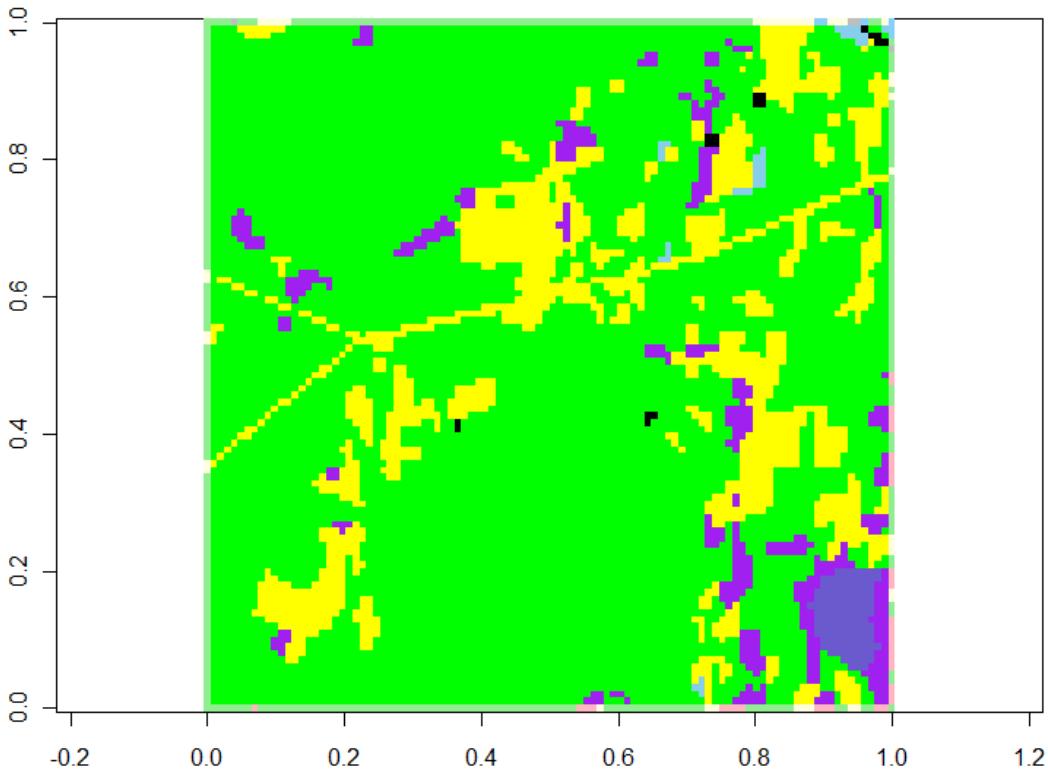
```
if(length(my.colors) != length(uv)) stop("You need a color for every unique
    value")
```

Next, print to the console the color associated with each class value to verify that you have what you want:

```
data.frame(code=uv, color=my.colors)
```

Finally, plot the image with the image() function in the graphic library. Note, because the image() function does a 90 degree counter-clockwise rotation of the image, a matrix transpose and some indexing is necessary to rotate the image back to its original orientation:

```
image(t(m)[,nrow(m):1],asp=1,breaks=my.breaks,col=my.colors
```



4. Inspect Geotiff grids in R

If you are an R user, you can inspect the Geotiff grids with R using the following script (or open the provided script, **tutorial_1.R**). Reading and plotting Geotiffs in R is quite a bit simpler than dealing with Ascii grids. Note, there are several ways to plot the grids in R. If you are familiar with the Terra package, you can import the Geotiff grids and plot them quite easily, as follows:

First, set the working directory to wherever you have installed the tutorial; e.g.:

```
setwd('c:/work/fragstats/tutorial/tutorial_1')
```

Next, load the Terra package in R:

```
library(terra)
```

Next, read in the Geotiff grid into a spatRaster object (g):

```
g<--rast('reg78b.tif'))
```

Next, in order to assign colors to each landcover class, identify each unique class value:

```
uv<-sort(unique(g[]))
```

Next, a color table by assigning colors to class values. Note, if you don't do this then R will simply choose a default color scheme:

```
my.colors<-c('gray','lightskyblue','lightgreen','lightpink','lightyellow','yellow',
'purple','slateblue','green','skyblue','black')
```

Next, check to make sure you have a color for every unique class value:

```
if(length(my.colors) != length(uv)) stop("You need a color for every unique
    value")
```

Next, print to the console the color associated with each class value to verify that you have what you want:

```
data.frame(code=uv, color=my.colors)
```

Next, assign the color table to the spatRaster:

```
coltab(g)<-data.frame(value=uv,col=my.colors)
```

Finally, plot the spatRaster with the plot() function:

```
plot(g)
```

The plot should look just like the one we produced above for the Ascii grid.