Acoustic Primer Exercises

A Tutorial for Landscape Ecologists

Luis J. Villanueva-Rivera, Department of Forestry and Natural Resources, Purdue University

Bryan C. Pijanowski, Department of Forestry and Natural Resources, Purdue University

Jarrod Doucette, Department of Forestry and Natural Resources, Purdue University

Burak K. Pekin, Department of Forestry and Natural Resources, Purdue University

This tutorial is a supplement to the paper *A Primer on Acoustic Analysis for Landscape Ecologists* by Villanueva-Rivera et al featured in the Landscape Ecology special issue entitled "Soundscape Ecology" (vol. 26, pages 1233-1246, doi: 10.1007/s10980-011-9636-9). Accordingly, the exercises in the tutorial are meant to be undertaken while reading the article.

Data and links to tools used in this tutorial can be found at
http://ltm.agriculture.purdue.edu/soundscapes/primer

V 1.3 (25nov2013)

ACKNOWLEDGEMENTS

Data used in these exercises were collected as part of a grant from the Lilly Foundation to Purdue University in support of its Center for the Environment initiative. Tools that are used here were developed as part of an NSF funded project (Data to Knowledge III-XT) to Purdue University and the University of Texas-Austin. *Seewave* was developed by Jerome Sueur, Thierry Aubin and Caroline Simonis. Raven was developed and its copy written by Cornell University and Audacity is freeware distributed via the Sourceforge open source network. Ideas regarding data handling and analysis were improved through discussions with Stuart Gage, Brian Napoletano, Deepak Ray, Bernie Krause and Almo Farina. Some of the data used in this tutorial were collected by Jim Plourde. Thanks to Maryam Ghadiri for comments to improve this tutorial.

NOTES ABOUT THIS FILE

This tutorial was first published on February 6, 2011 as version 0.9. This is version 1.3 and was published on November 25, 2013 to use the package *soundecology* version 1.1 for R.

R code appears in this document in a different font, for example:

```
acoustic_diversity(sound1)
```

CONTENTS

Exercise 1 – Listening to and Visualizing the Soundscape

Inspecting data in its raw format before deciding on the type of analyses to conduct is an important part of effectively assessing and understanding the system which one is studying. Accordingly, this first exercise is designed to develop in the reader an intuitive sense of the sound data that is quantitatively analyzed in the following exercises. It also demonstrates the sounds produced by different objects (e.g. biophony, anthrophony) and their interactions within the soundscape. The reader should consult 'exercise 1'section of the manuscript by Villanueva-Rivera et al. (this special issue) for a description of the sounds they are hearing as well as the information conveyed by them.

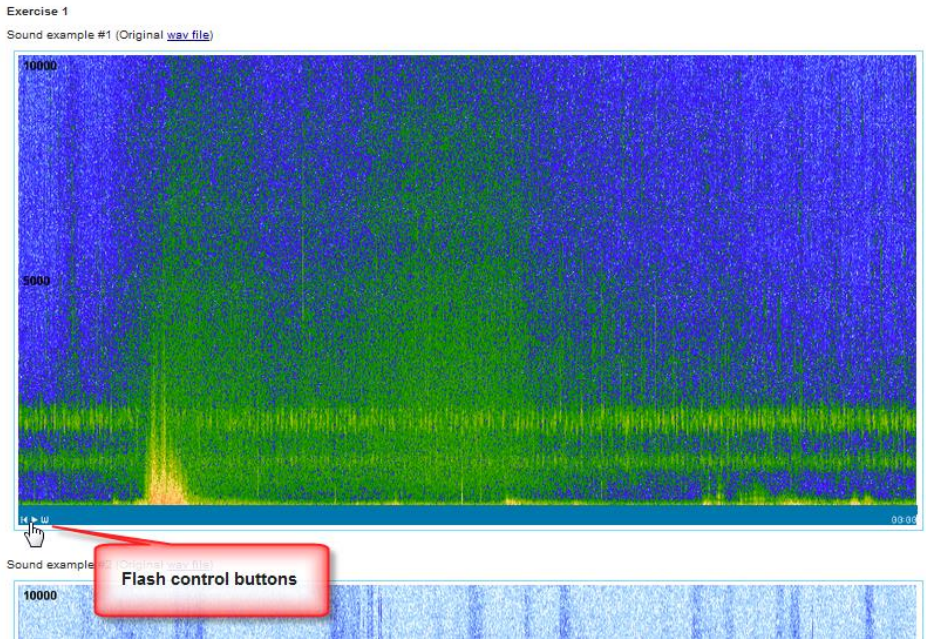
Part A. Listening

Getting Started

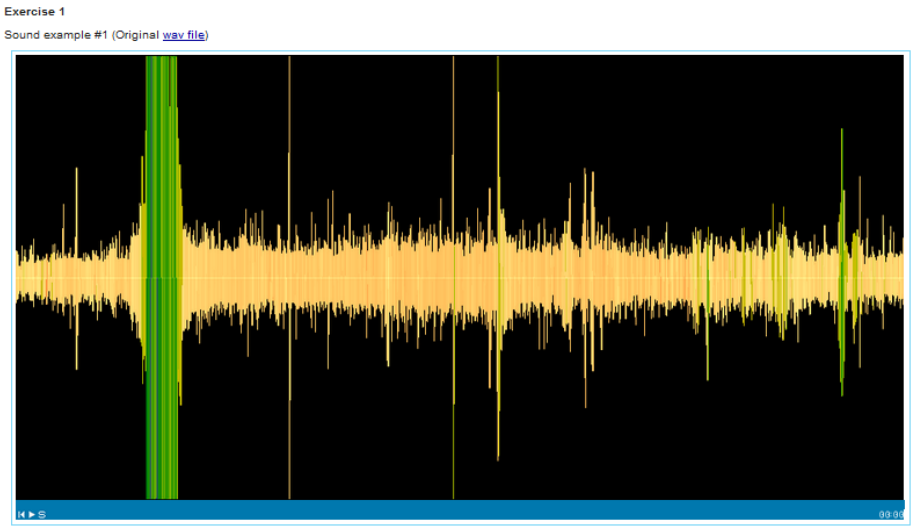
1. Visit the website ltm.agriculture.purdue.edu/soundscapes/primer to listen or download the three sound files in this exercise.

Listening to Soundscape Recordings

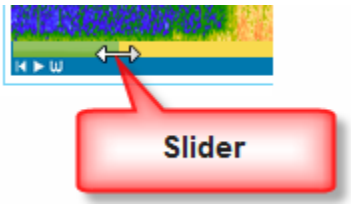
2. Click on exercise 1. Play sound example #1 by clicking play button on the left bottom corner. You will need a modern browser with the Adobe Flash plug-in installed. You will see this web page with



You can use the control buttons on the lower left of the window to toggle between spectrogram (labeled as S) and waveform (labeled as W). When you toggle to waveform and you would see this:



You can also select the arrow to play the recording. Once the file loads over the internet, you can also move along the sound file using the slider:



3. Repeat for sound example #2 and sound example #3. Watch the spectrogram as you listen, and note the region on the spectrogram that corresponds to the sounds you are hearing.

Part B Visualizing Sound

Getting Started

In this exercise, we will use Raven Lite (Charif et al. 2006) and Audacity to examine several soundscape recordings from the Tippecanoe Soundscape Study.

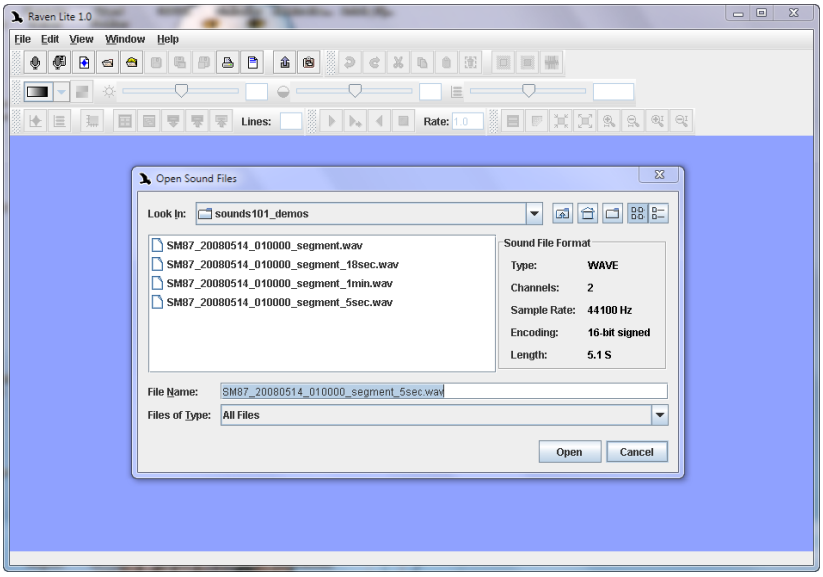
Raven Lite is the freeware version of Raven and can be downloaded from http://birds.cornell.edu/brp/raven/Raven.html, where versions for the Windows, Mac OS X, and Linux operating systems are available. After the software has been installed, you can go to the Help menu and then Get Free License. You will be directed to a web site and then requested to "purchase" the software for no charge. An email will be sent with your license code. Directions for providing the license number to the program will be provided in that file.

Audacity can be obtained from http://audacity.sourceforge.net, where versions for the Windows, Mac OS X, and Linux operating systems are available, as well as the source code.

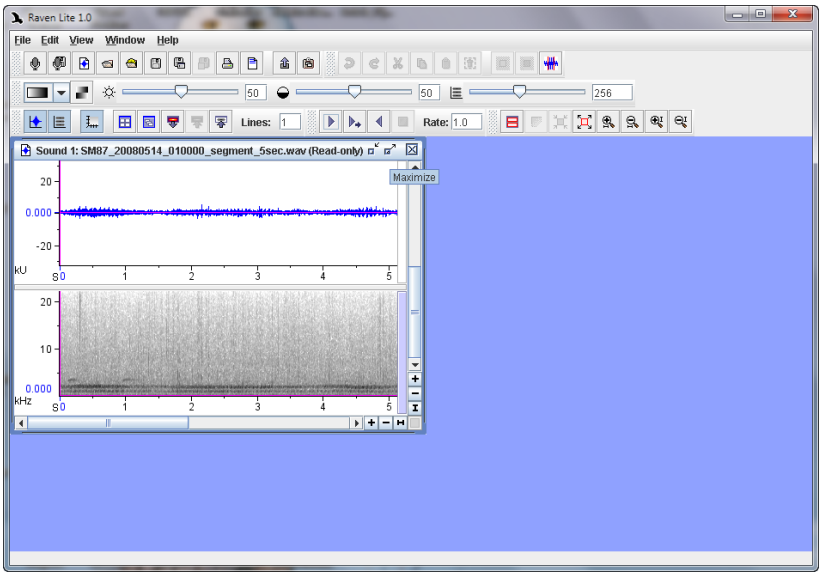
At our tutorial web site are several wav files that will be used. Download exercise2.zip and place the files (unzipped) into a folder called Sounds101_demo.

Opening and Viewing A Sound file In Raven Lite

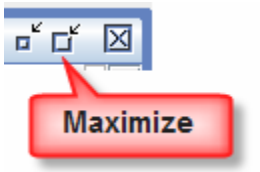
Let us open a small 5 sec sample of the May 14, 2008 Purdue Wildlife (wetland) recording from 01:00 (1:00am local time). This sample contains chorusing of Spring Peepers during a light rain shower. The interface to open a file will look like this:



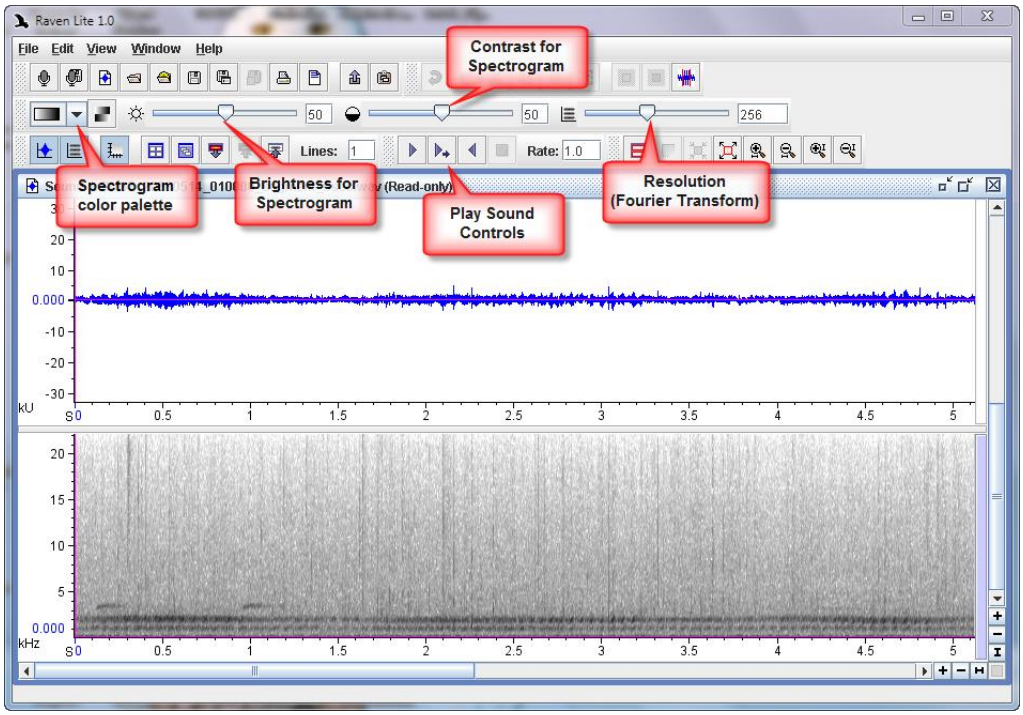
Once this file is selected, click the Open button and select Channel 1. The sound file will be loaded and a waveform and spectrogram shown like this:



You can select the Maximize button



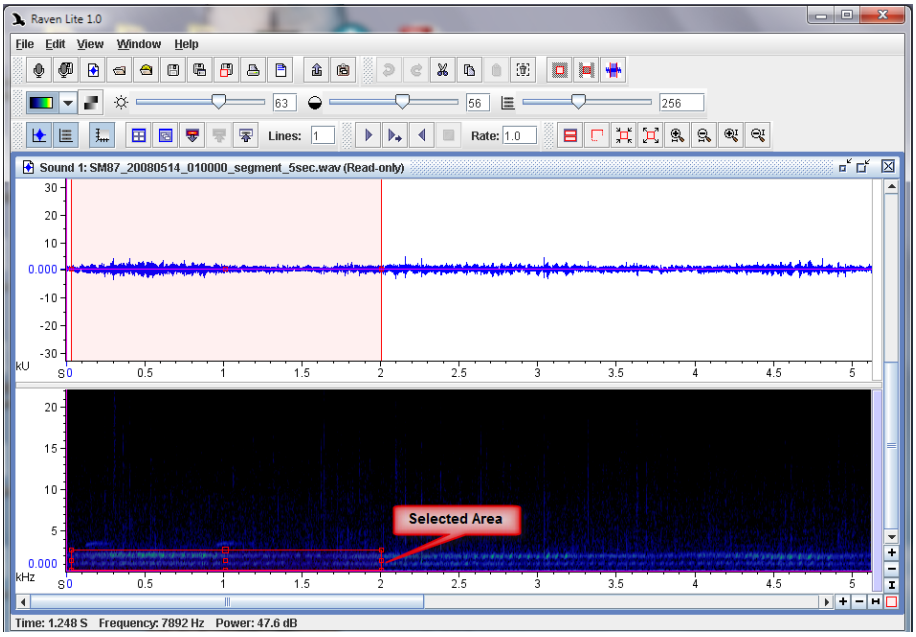
To create a large view of both the waveform and spectrogram:



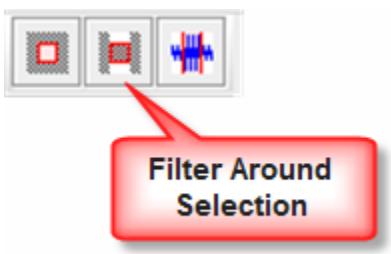
Note that there are several sets of buttons that control various functions related to visualizing, subsetting, file management, and zooming. The spectrogram color palette provides four color schemes for the spectrogram. The brightness, contrast and resolution have scroll bars controlling these three key features of the spectrogram. Change each to see how they modify your ability to see patterns within the spectrogram. Also note that you can also play the sound using the three play sound control buttons (play, play and scroll, and play backwards). After experimenting with the three main spectrogram visualization scroll bars, play the sound several times. You should hear the Spring Peepers chorusing during a light rainshower.

Subsetting Sounds in Raven Lite

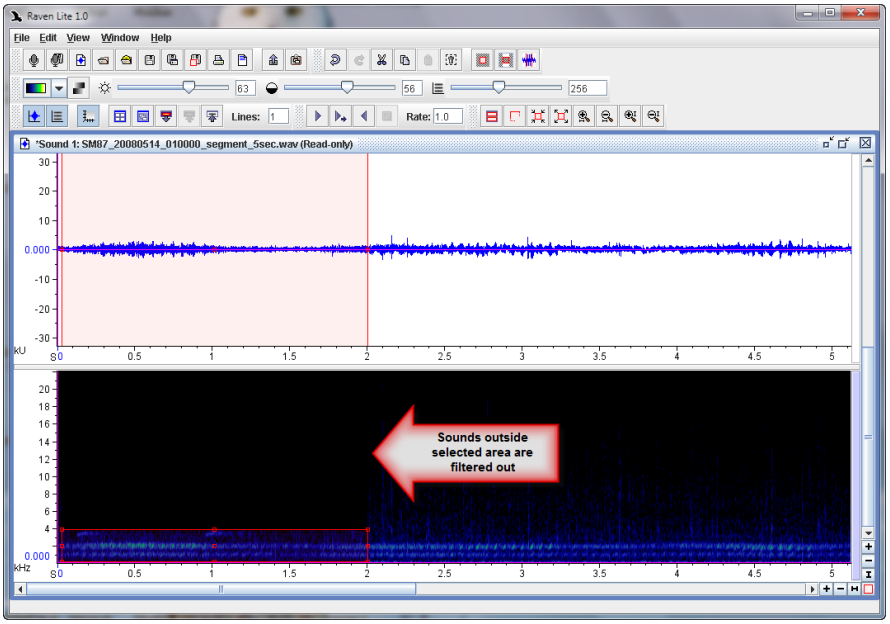
Raven Lite allows a user to select areas within the sound file and then remove other sounds so that you can hear specific portions of the spectrogram. Use your cursor to select sounds within the first 2 seconds that are less than 3 kHz. The selection should appear as a red box with portions highlighted with small squares:



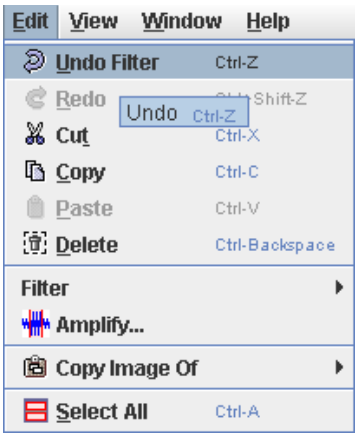
Using the Filter Around tool,



you will see the sounds outside the selected area set to 0 dB:



Play the sound now and compare this to the situation where the file does not have a filter applied. You can toggle between having the filter applied and not applied using the Edit menu (Undo Filter and Redo):

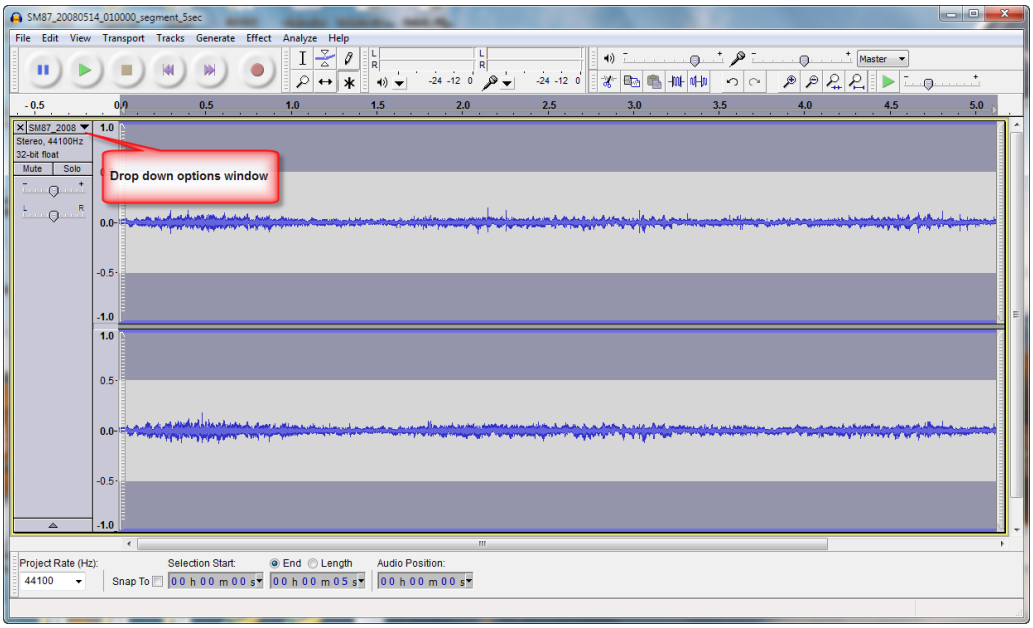


You may also amplify the sounds that are selected. Select Amplify from the Edit menu and then key in a value of 4 (values less than 1 reduce the sound volume) and then listen to the recording again.

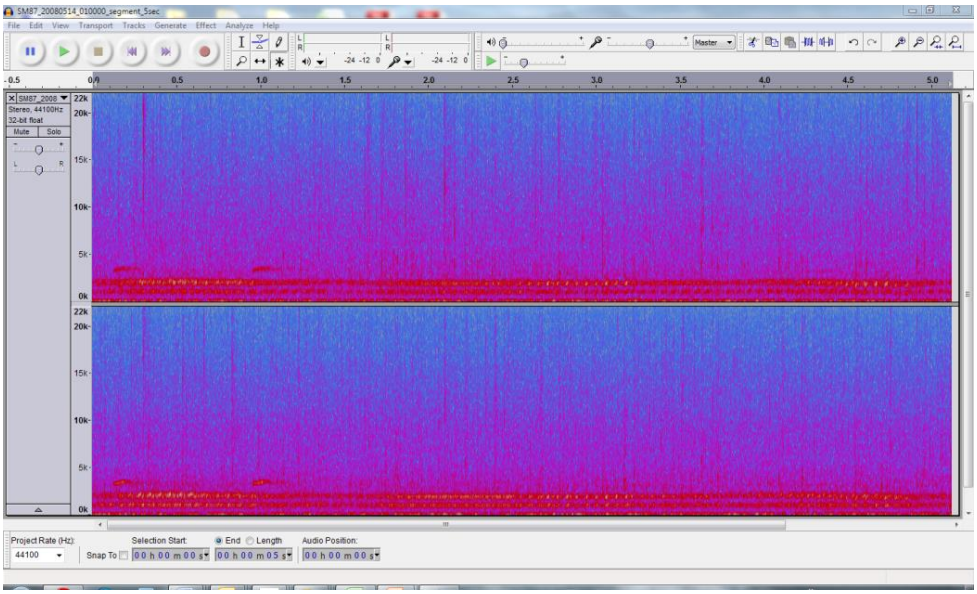
Readers interested in learning more about Raven Lite or the other Raven products can go to the Cornell Laboratory for Ornithology web site listed above. A Raven Lite tutorial is also available as well as access to thousands of files to listen to many individual vocalizations.

Visualizing Sound Files Using Audacity

Start Audacity and bring up the same 5 min recording used in the Raven Lite portion of this exercise. Doing so will provide you with this:

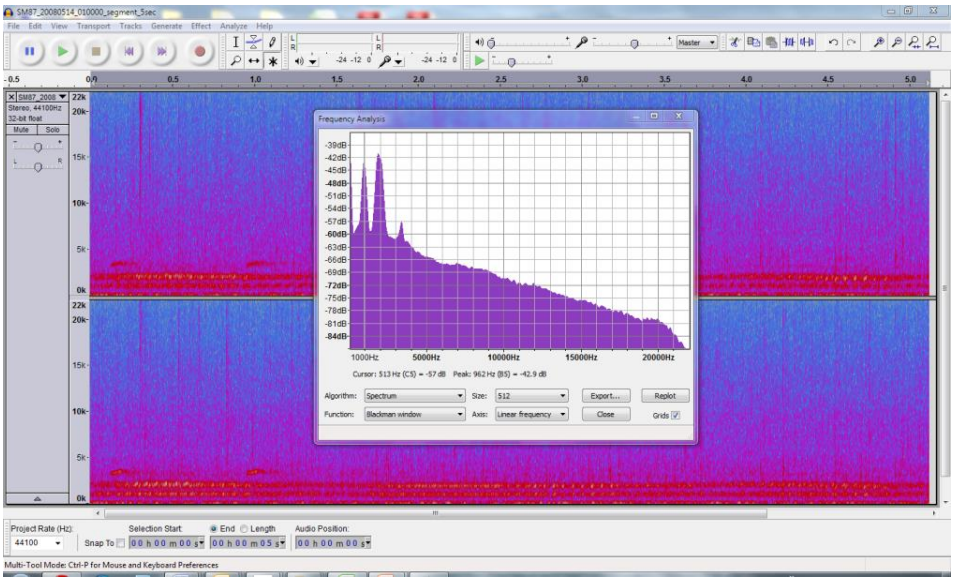


Because this is a stereo recording, you will see two waveforms, one for each channel. You can use the drop down options menu to toggle what you see between spectrograms and waveforms. A spectrogram view would show this:



You can control the playing of the sound using the large menu of buttons in the upper left.

You can create a frequency-amplitude plot of this 5-min recording using the Analyze and Plot Spectrum options:



Exercise 2 – Frequency Band Analysis

In this exercise, we will use the R package *soundecology* to analyze sound files.

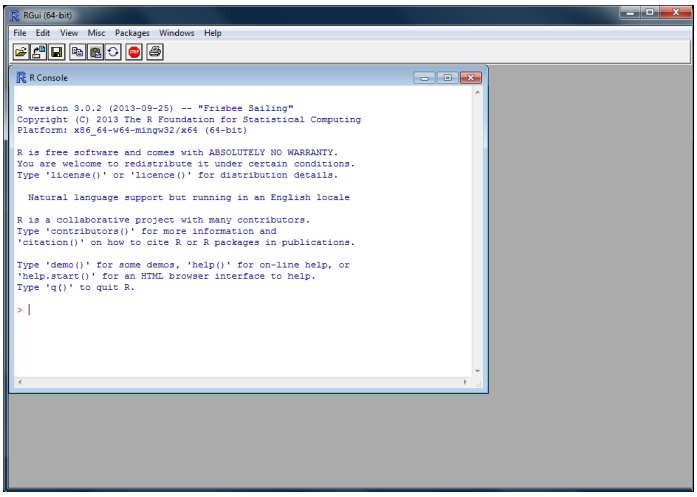
Getting Started

R is a free software environment for statistical computing and graphics that can be downloaded from The Comprehensive R Archive Network (CRAN) at http://cran.r-project.org. Select the operating system that you will be using (e.g., Windows) and then select the base binary option. You will need version 2.15.3 (published on March 2013) or higher of R for this tutorial. At the time of this publication, the current version of R is 3.0.2 (published on September 2013).

Installing and Loading Packages in R

While R comes with the most popular packages already installed, the functionality is extended by installing additional packages, which are collections of code submitted by users for the use of others. In this case, we will use the package *soundecology*, available from CRAN.

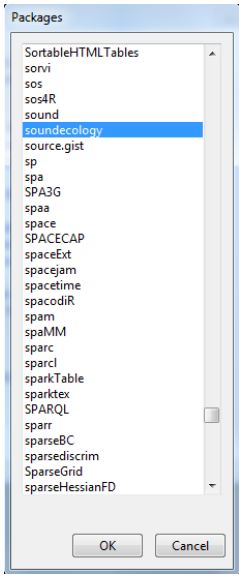
1. Once you have downloaded and installed R on your computer, open R. You should have a window that looks like this:



2. Click the ***Packages*** tab at the top, and then ***Install Packages***.

3. A separate window called “CRAN mirror” will pop up. These are servers that store the packages. They have the same contents, so select a nearby location and click ok.

4. The packages window will open. Scroll down until you find *soundecology* and click ok. You may be asked if you want to use a “personal library”, click Yes twice for R to install the packages in a directory inside your Documents directory. R will then install *soundecology* and all the required packages.
5. Now that the package is installed, it is necessary to load it. Click the “**Packages”** tab, click **“Load package…**”, select *soundecology* from the list and click OK. There are several required packages which will be loaded automatically. In particular, two packages are used for basic manipulation of the sound files: *tuneR* and *seewave*.



Downloading sound files and Setting the working directory

You will need to download the sound files, which are in .wav format, from the web and load them into R.

1. Go to http://ltm.agriculture.purdue.edu/soundscapes/primer and click on exercise 2.
2. Right click on wav file for sound example #1 and save the file SM87_20080420_000000.wav to a directory on your computer.
3. Right click on wav file for sound example #2 and save the file SM87_20080420_070000.wav to the same directory on your computer.

The next step is to select a directory where R will work from, called the working directory:

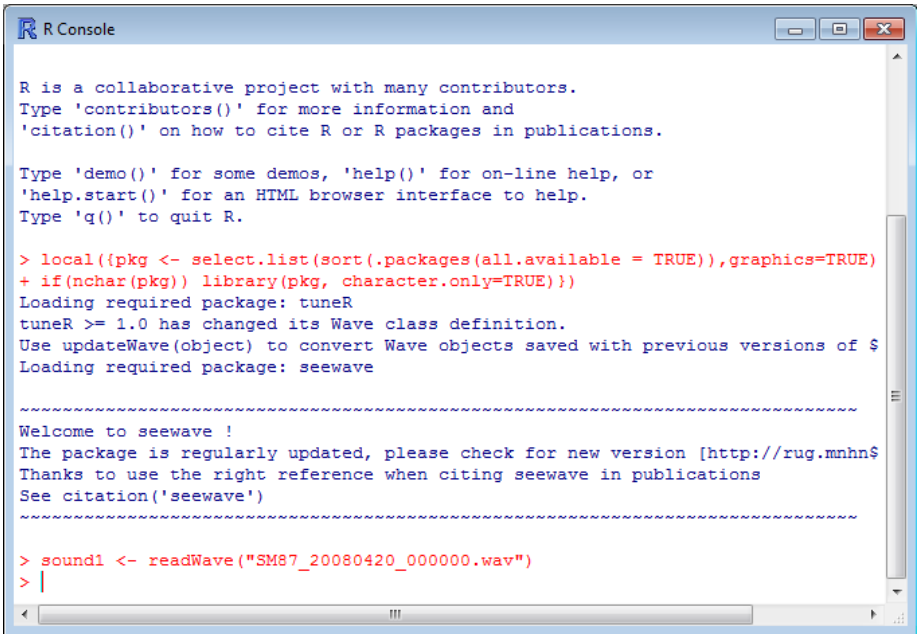
1. Click the ***File*** tab, and then ***Change dir***.
2. Select the directory where you saved the two sound files in. Click *OK*.

RUNNING R ANALYSIS PACKAGES

To load the file as an object[1], in this case called *sound1*, we will use the function *readWave()* from the package *tuneR*.

1. Copy and paste the following line into R console (note that R is case sensitive):

```
sound1 <- readWave("SM87_20080420_000000.wav")
```

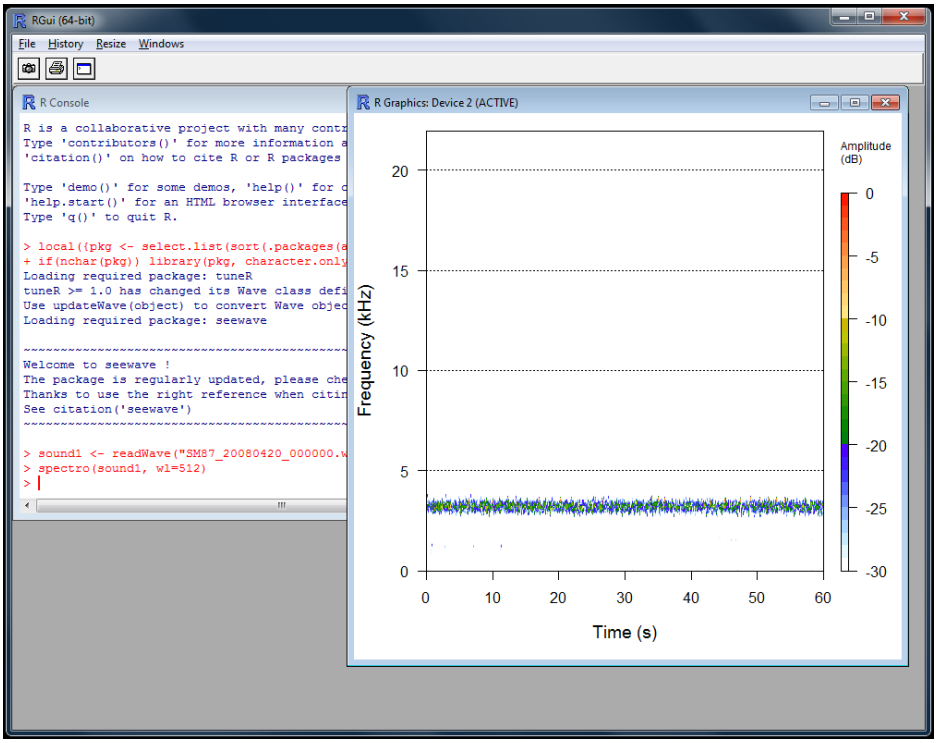


2. To see the spectrogram of the sound file, use the *seewave* function *spectro()*. In this case we will use an FFT window of 512. Make sure you maximize the R Window. Copy and paste following line:

```
spectro(sound1, wl=512)
```

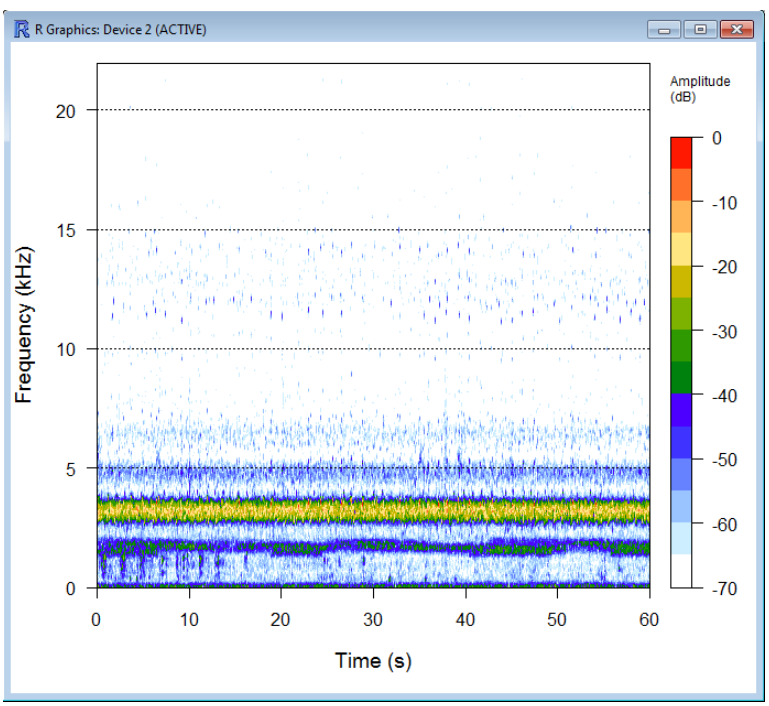
The following graphic is produced from the above command:

[1] An object is a computer programming term for an identity of a file or other "object" that can be manipulated by the program. Here, sound1 is an object associated with the wav file that is now loaded into the program R.



3. By default, the spectrogram does not show the fainter sounds. The option *collevels* lets you specify the levels at which to draw the sound intensity of the spectrogram. In this case we will specify a sequence of levels between -70 and 0, with steps between levels of 5, using the function *seq()*:

```
spectro(sound1, wl=512, collevels=seq(-70,0,5))
```



4. The function *spectro()* can also save the data of the spectrogram as a numeric matrix with amplitude values, with each column being a Fourier transform of the window length *wl*. In this case, we assign the result of the function to a new object, in this case *data*, and setting the value of *plot* to false:

```
data <- spectro(sound1, wl=512, plot = FALSE)
```

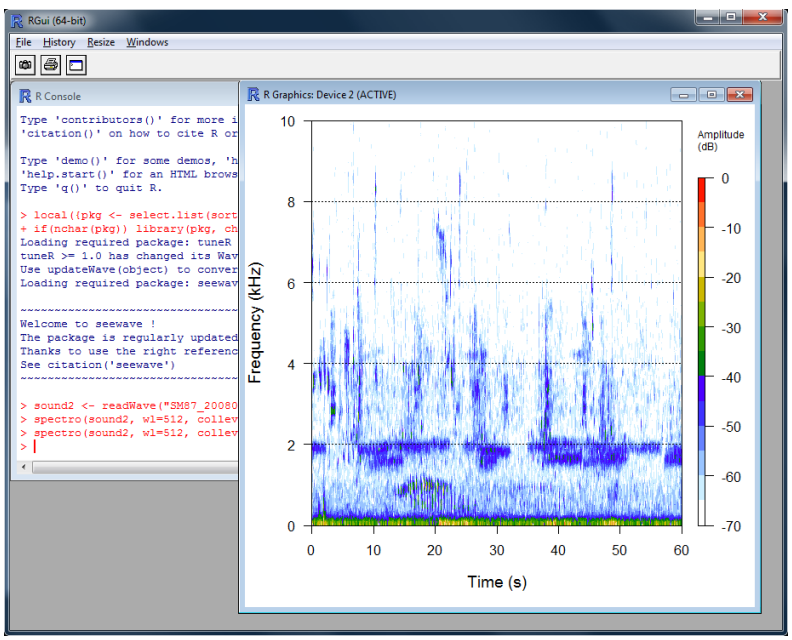
5. To see the spectrogram of the second file, use these commands:

```
sound2 <- readWave("SM87_20080420_070000.wav")

spectro(sound2, wl=512, collevels=seq(-70,0,5))
```

6. To plot the spectrogram up to 10 kHz, set the limits using the argument *flim*, so that it plots the image between 0 and 10 kHz:

```
spectro(sound2, wl=512, collevels=seq(-70,0,5), flim=c(0, 10))
```



Processing of Tippecanoe Soundscape Study Recordings

We use the matrix of values of the spectrogram to calculate the Acoustic Diversity Index (ADI; Pijanowski et al. 2011, Villanueva-Rivera et al. 2011). The matrix of values is divided in 10 bands of 1 kHz each, up to 10 kHz. A default cutoff value of -50 dBFS is specified to remove the faint sounds. The proportion of sounds that falls above that cutoff value is determined in each band. The ADI is the result of applying the Shannon Diversity Index to these proportions. We show how users can execute two scripts, one that processes a single file and another designed to batch many .wav files so that values can be plotted over time and at multiple sites.

To demonstrate how this script works, we use the two sound files that were downloaded in the previous section. These files were recorded at the Purdue Wildlife Area using an automated digital recorder on 20 April 2008 (Figure 10 in Villanueva-Rivera et al. 2011). The first sound file was recorded at night, at 00:00, and the other one during the morning, at 07:00. The night sound is dominated by frogs in the 1.3-3.8 kHz range while the sounds in the morning are from birds that range from approximately 0.2 kHz to almost 8 kHz. Just by visually comparing the spectrograms of these sound files, there seems to be a larger diversity of sounds in the morning sound file that in the night sound file.

SINGLE FILE PROCESSING

For this example, we will use the file from the steps above, stored in the object *sound1*. The package *soundecology* includes a function that calculates the acoustic diversity index (ADI) of the file. This function, *acoustic_diversity*(), is run on the object that holds the data of the wav file.

1. To calculate the ADI, type this command that directs R to calculate the index on the object *sound1*, which holds the data of the file SM87_20080420_000000.wav:

   ```
   acoustic_diversity(sound1)
   ```

2. Then, to calculate the ADI of the morning file, type:

   ```
   acoustic_diversity(sound2)
   ```

In this example, the night recording has most of the signal in the bands corresponding to 1-4 kHz, while the signal for the morning has signals from the lower frequencies up to the 6 kHz bands. However, the night recording has a continuous chorus of frogs that occupies their frequency space during the whole file, while the morning chorus is comprised of many signals separated in time. These patterns result in a higher ADI value for the night sound file, with a value of 1.431, compared to the morning sound file, with a value of 1.338, both for the left channel.

Output from the function will appear on the screen like this:

```
Acoustic Diversity Index:

 Left channel: 1.431103

 Right channel: 1.508265
```

To calculate the evenness of these two files, using the Gini index, use the function *acoustic_evenness()*:

```
acoustic_evenness(sound1)

acoustic_evenness(sound2)
```

This will result in an AEI value of the left channel of 0.6758 for the night file and 0.7095 for the morning file. The strong signal component of the frogs in a small portion of the frequency space is driving the index to a lower, therefore less even, value.

PROCESSING MANY FILES[2]

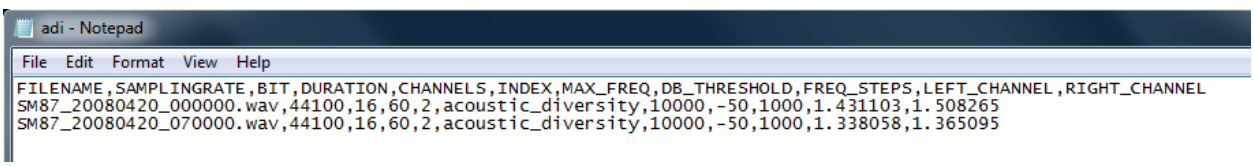
The function *multiple_sounds()* runs like a batch file and allows the user to easily analyze a set of files. All the files need to be in the same directory. The output of this function is saved to a comma-separated file, which contains the technical details of the file and the result of the index calculated.

To save the results of *acoustic_diversity*() of the two example files, type the following command in a single line:

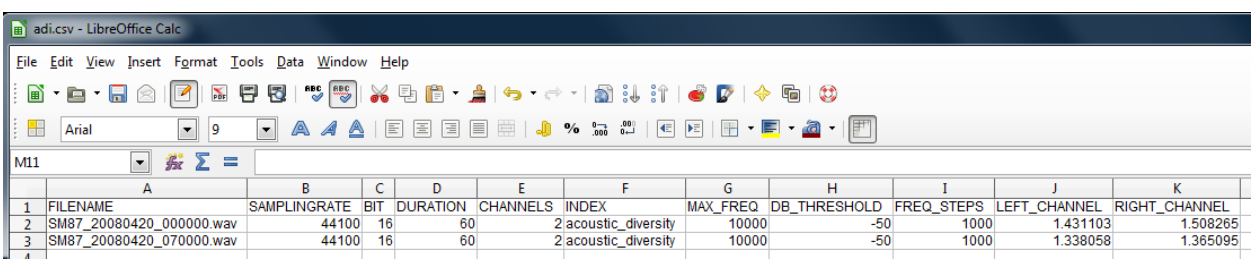
```
multiple_sounds(directory=".", resultfile="adi.csv",
     soundindex="acoustic_diversity")
```

The argument *directory* is set to "." (a period) to indicate R to use the current working directory.

This function saves a comma-separated file with the technical data of each file and the result of the same function we used above, *acoustic_diversity()*, the ADI of the left and right channels of the file:



This file can be opened in any spreadsheet software and can be imported to any analysis software, including R.



	A	B	C	D	E	F	G	H	I	J	K
1	FILENAME	SAMPLINGRATE	BIT	DURATION	CHANNELS	INDEX	MAX_FREQ	DB_THRESHOLD	FREQ_STEPS	LEFT_CHANNEL	RIGHT_CHANNEL
2	SM87_20080420_000000.wav	44100	16	60	2	acoustic_diversity	10000	-50	1000	1.431103	1.508265
3	SM87_20080420_070000.wav	44100	16	60	2	acoustic_diversity	10000	-50	1000	1.338058	1.365095

To see more details about the function, type this command in R to launch a web browser with the help page for the function:

```
?multiple_sounds
```

---

[2] Users should be acutely aware that processing several files will require (a) a lot of processing time, generally hours to days depending upon the number of files and the number of CPU cores used, and (b) a lot of disk space (100s of GB). Downloading all of the files available at the tutorial web site will require about a 1 TB of free disk space.

Exercise 3 – Spectrograms as Raster Files

In this exercise we will export a spectrogram as an ASCII raster file. This exercise will take advantage of tools traditionally used for Geographic Information Systems (GIS) to analyze the spectrograms. Note that ArcGIS 10 and Python 2.6 are required for this exercise.

A python script was used to link several tools together and iterate through the steps required to complete the analysis. The script creates polygons *(i.e.* patches) of sound based on a user specified dBFS level, and then generates statistics for these polygons that describe the sound within. It is designed to process multiple days' worth of data from multiple sites, but for the purposes of this exercise we will examine a single file.

Sound files as rasters

We can use common GIS functions to analyze the signals in a sound file in a similar way to patches in a landscape. To do this the spectrogram has to be saved in a format that the GIS software can understand. The function *sound_raster()* will output a raster version of a spectrogram to raster files in ASCII format:

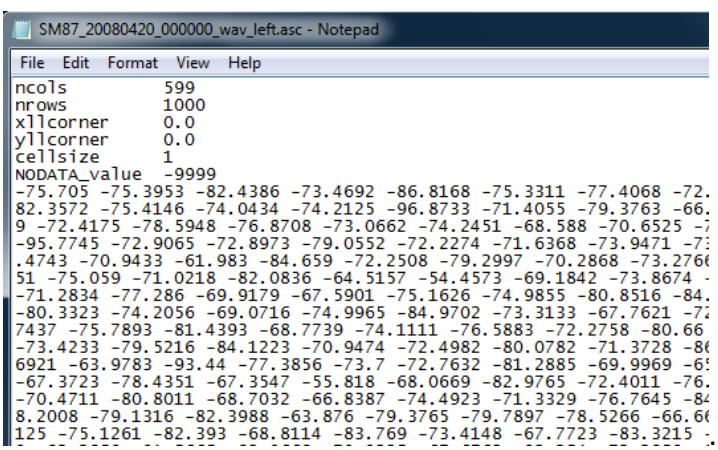
```
sound_raster(wavfile = "SM87_20080420_000000.wav")
```

```
> sound_raster(wavfile = "SM87_20080420_000000.wav")

 This is a stereo file. A raster will be written for each channel. Please wait...

  ASCII raster files:
    SM87_20080420_000000_wav_left.asc
    SM87_20080420_000000_wav_right.asc
```

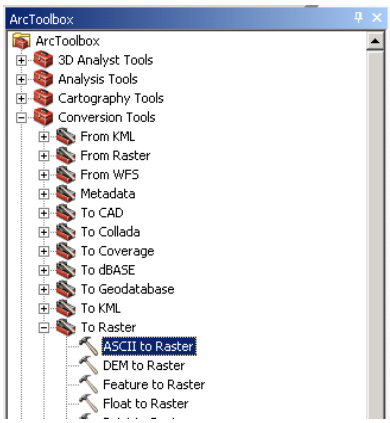
Since the file is in stereo, the function saves two files, one per channel. The file SM87_20080420_000000_wav_left.asc can be opened by any text editor:



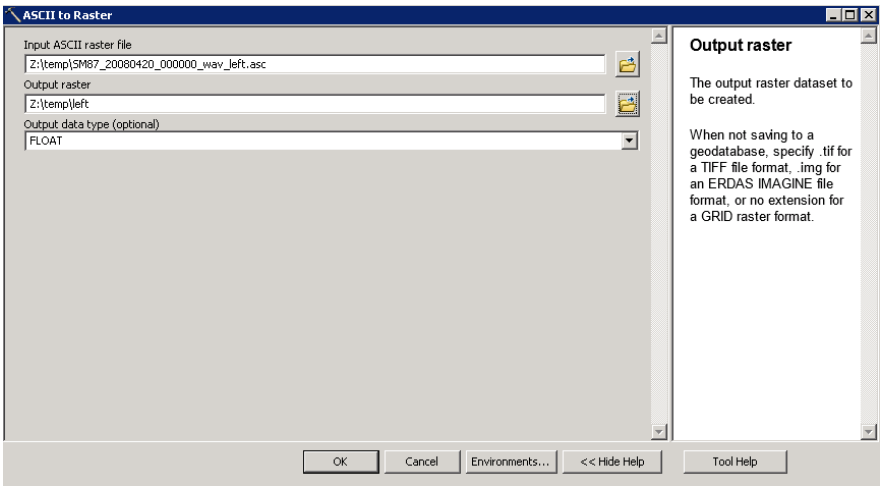
```
SM87_20080420_000000_wav_left.asc - Notepad
File  Edit  Format  View  Help
ncols          599
nrows          1000
xllcorner      0.0
yllcorner      0.0
cellsize       1
NODATA_value   -9999
-75.705 -75.3953 -82.4386 -73.4692 -86.8168 -75.3311 -77.4068 -72.
82.3572 -75.4146 -74.0434 -74.2125 -96.8733 -71.4055 -79.3763 -66.
9 -72.4175 -78.5948 -76.8708 -73.0662 -74.2451 -68.588 -70.6525 -
-95.7745 -72.9065 -72.8973 -79.0552 -72.2274 -71.6368 -73.9471 -
.4743 -70.9433 -61.983 -84.659 -72.2508 -79.2997 -70.2868 -73.276
51 -75.059 -71.0218 -82.0836 -64.5157 -54.4573 -69.1842 -73.8674 -
-71.2834 -77.286 -69.9179 -67.5901 -75.1626 -74.9855 -80.8516 -84.
-80.3323 -74.2056 -69.0716 -74.9965 -84.9702 -73.3133 -67.7621 -
7437 -75.7893 -81.4393 -68.7739 -74.1111 -76.5883 -72.2758 -80.66
-73.4233 -79.5216 -84.1223 -70.9474 -72.4982 -80.0782 -71.3728 -
6921 -63.9783 -93.44 -77.3856 -73.7 -72.7632 -81.2885 -69.9969 -
-67.3723 -78.4351 -67.3547 -55.818 -68.0669 -82.9765 -72.4011 -76.
-70.4711 -80.8011 -68.7032 -66.8387 -74.4923 -71.3329 -76.7645 -
8.2008 -79.1316 -82.3988 -63.876 -79.3765 -79.7897 -78.5266 -66.
125 -75.1261 -82.393 -68.8114 -83.769 -73.4148 -67.7723 -83.3215 -
```

This file is formatted as an ASCII raster file. The first six lines contain data necessary to determine the location of the signals. The x axis is time while the y axis is frequency. The rest of the file contains the data for each pixel in the raster. Importing the ASCII file into ArcGIS 10 is

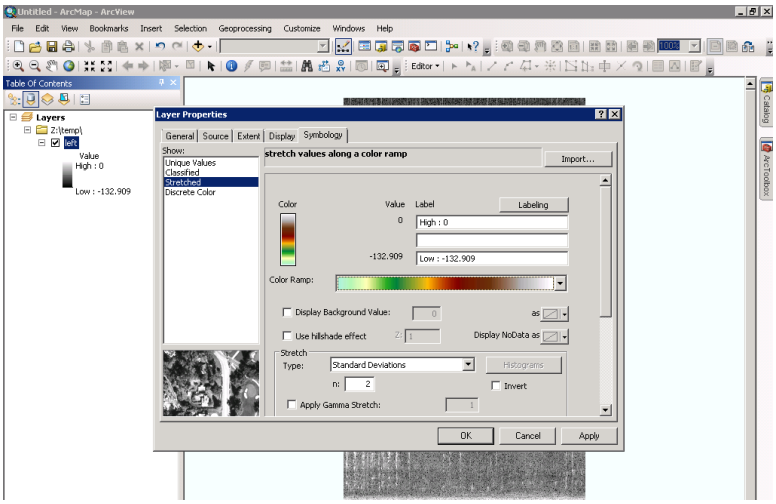
straightforward for those users familiar with this GIS software package. Use the tool ASCII to Raster which is inside the Conversion Tools category in ArcToolBox:

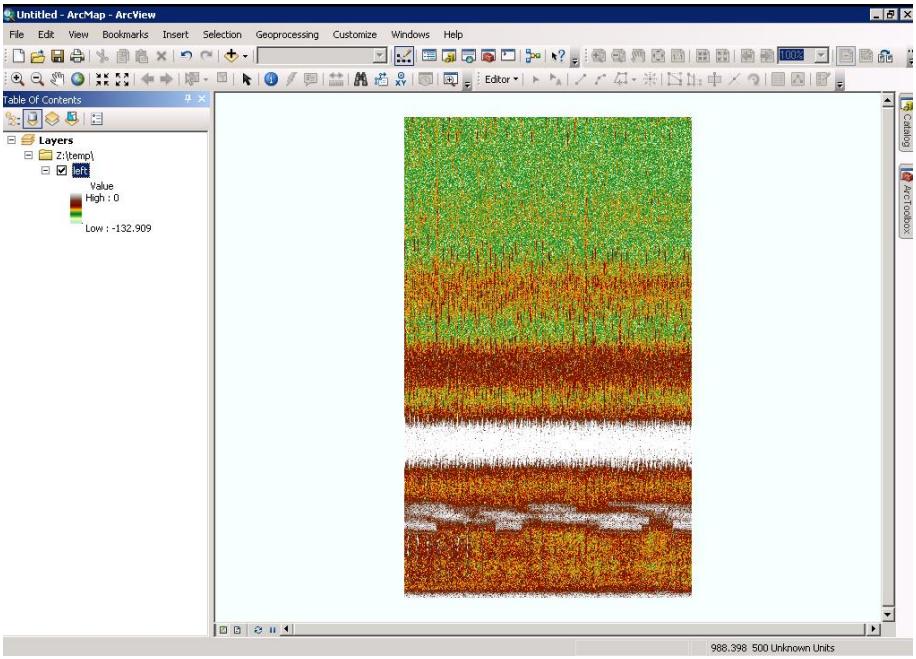


Since dBFS values are floating point, using the FLOAT as the Output data type:

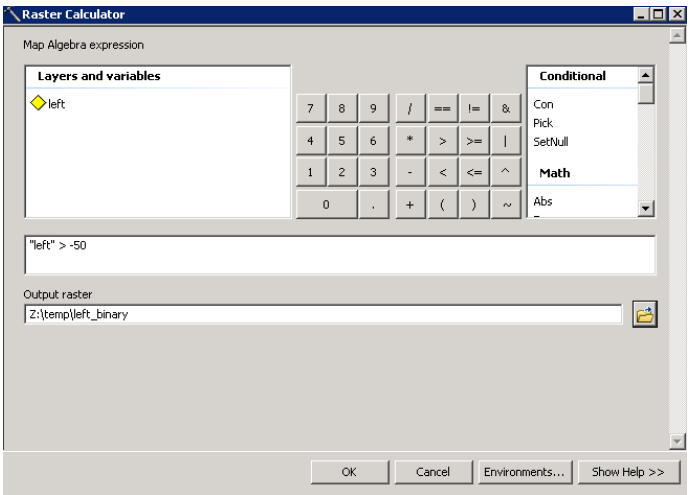


After the ASCII file is imported and the color ramp changed from the default grey scale, the SM87_20080420_000000.wav file for the left channel should look like a raster map:

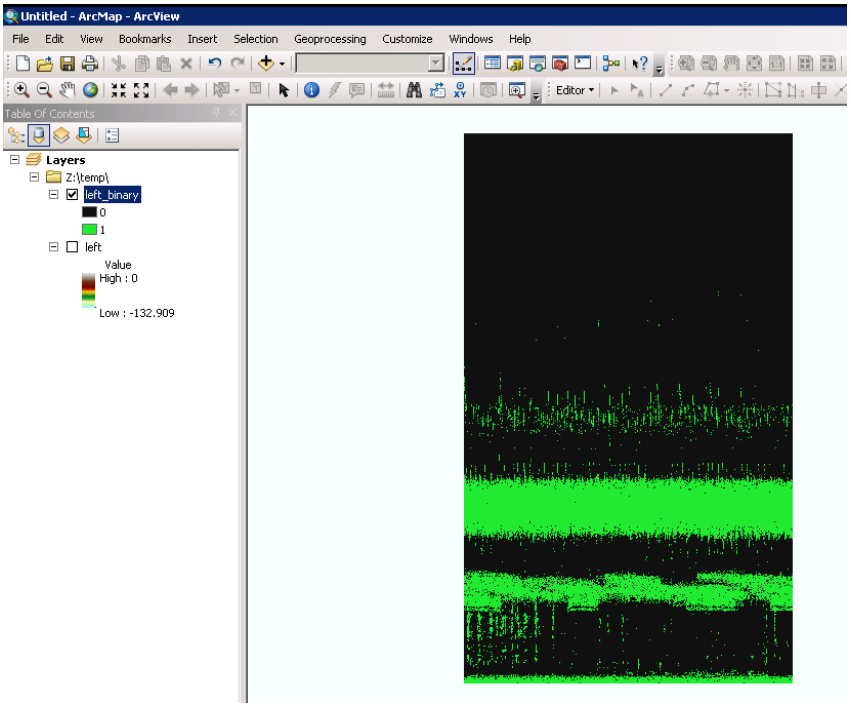




Using Raster Calculator in ArcGIS 10, those sounds that have intensity values greater than, viz. -50 dBFS, can be saved as a binary map. The equation in Raster Calculator is “file1” > -50



Hitting “OK” will produce the following binary map:

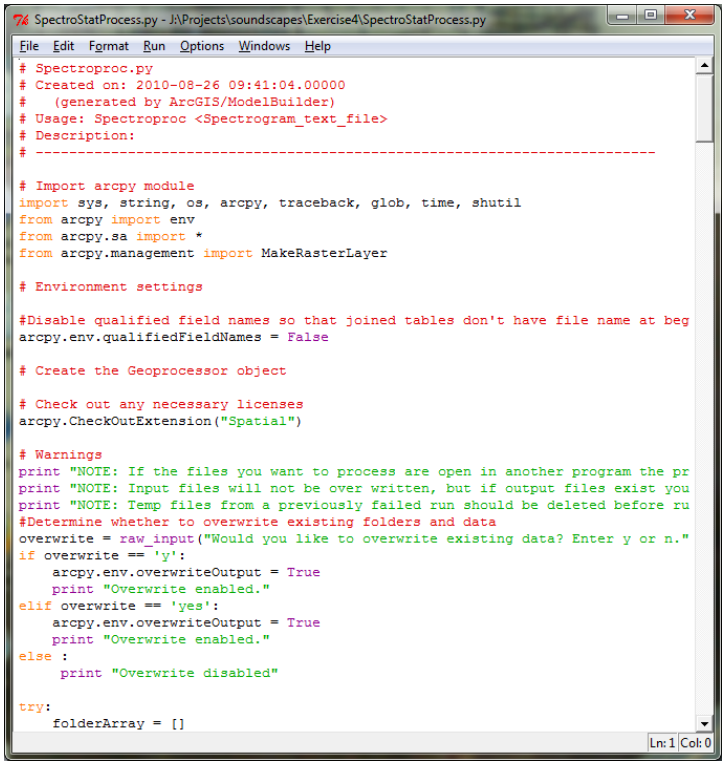


The python script that is covered in the next exercise performs this raster calculator step automatically. The python script does require the user to convert the ASCII files to an ArcGIS raster map.

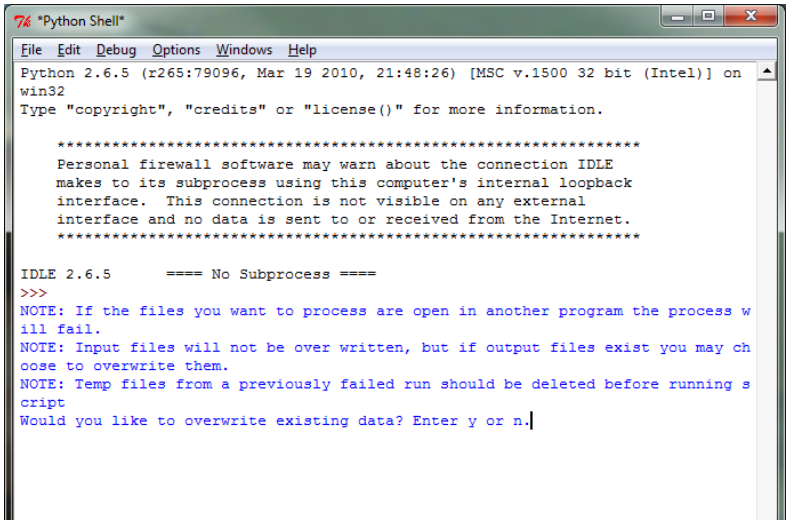
ANALYZING THE FILE USING PYTHON

The examples in this exercise will be completed using the integrated development environment (IDE) software installed by default with Python called IDLE. For different IDE options and to learn more about Python see the Python Wiki (wiki.python.org).

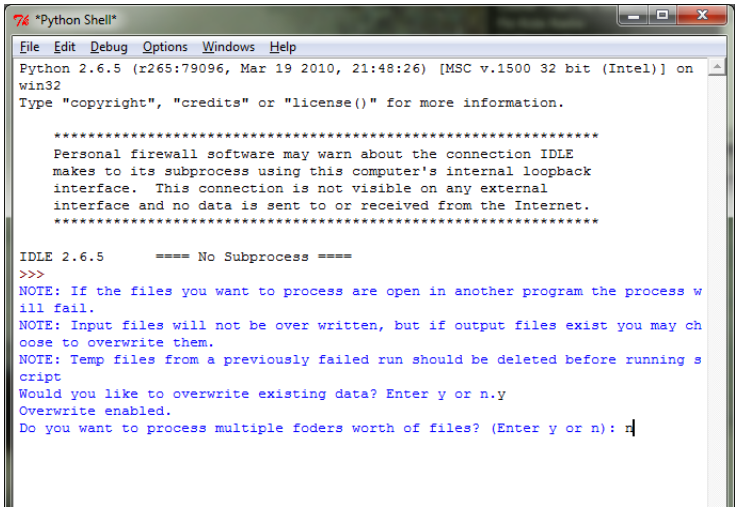
1. Begin by opening the script in IDLE. Browse to the folder location where the exercise data was saved, right click the SpectroStatProcess.py file, and choose “edit with IDLE”.
2. This will open two windows. Leave both windows open as one is used to display the code while the other displays the process results.
3. In the window labeled SpectroStatProcess.py click the Run drop down menu and choose Run Module.



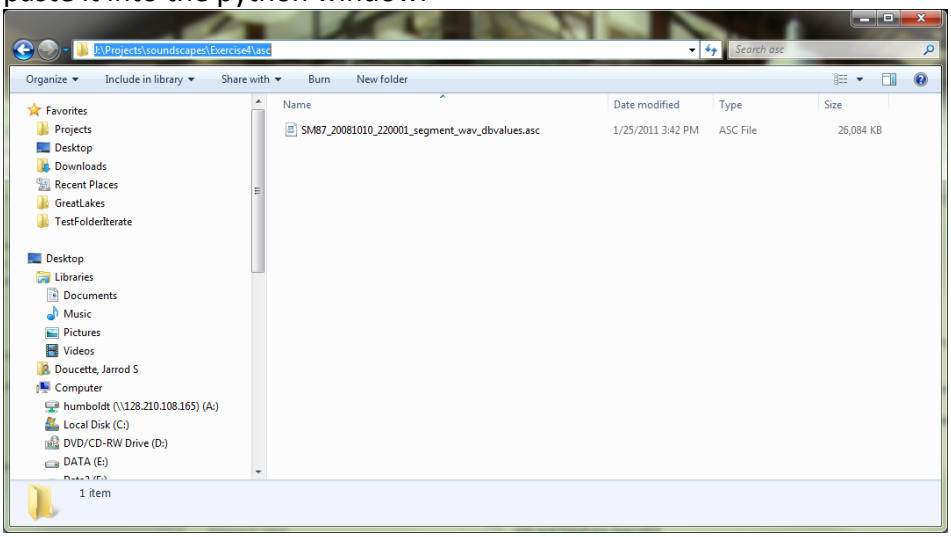
4. The program will run in the Python Shell window. It begins by writing several notes for processing issues encountered during more advanced runs., and then asks whether to overwrite existting data. Input data is never overwritten, and only output from previous runs will be overwritten by enter y. Enter the letter “y” and press the enter key.



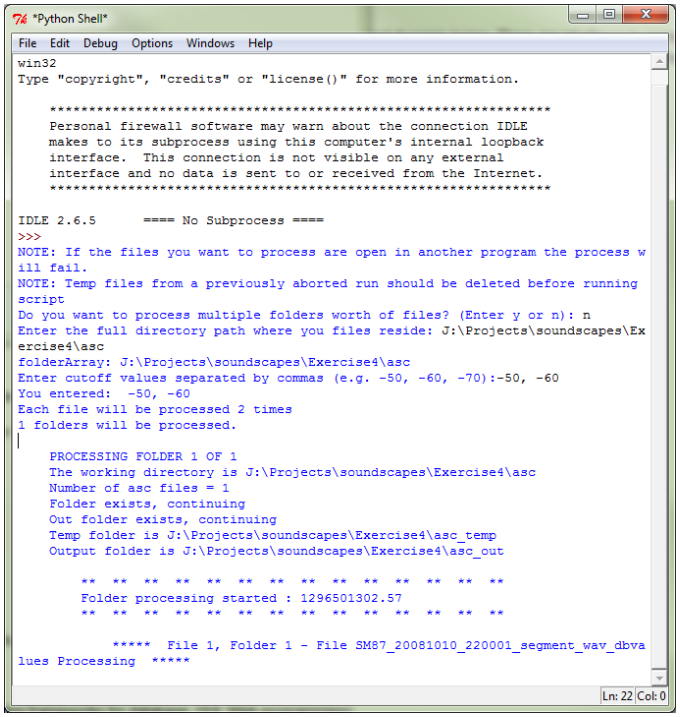
5. The script is designed to process either one folder of data, or a folder containing several folders worth of data (e.g. each site in a different folder). For this exercise there is only one folder so choose “n”.



6. Next enter the full directory path for the data. The easiest way to enter the path is to browse to the folder in windows explorer and copy the path from the address bar and paste it into the python window.



7. Now enter the values, in dBFS, to use as the cutoff for defining a patch of sound. In this case enter -50 and -60. The numbers entered are confirmed on the screen and the number of times each file will process.
8. The script is now running. There are many screen outputs designed to provide information about the current file being processed, and past processes that have run. When processing multiple files and folders worth of data it shows how many files and folders have been processed and the total number to be done. Time information such as when file processing started, how long each file took to process, and how much longer it is expected to run is also available. On a moderately fast computer, with a 3.1GHz processor and 4GB of memory, it typically takes 4-8 minutes per file depending on the complexity of the soundscape (i.e. number of polygons).



```
*Python Shell*
File  Edit  Debug  Options  Windows  Help
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 2.6.5      ==== No Subprocess ====
>>>
NOTE: If the files you want to process are open in another program the process w
ill fail.
NOTE: Temp files from a previously aborted run should be deleted before running
script
Do you want to process multiple folders worth of files? (Enter y or n): n
Enter the full directory path where you files reside: J:\Projects\soundscapes\Ex
ercise4\asc
folderArray: J:\Projects\soundscapes\Exercise4\asc
Enter cutoff values separated by commas (e.g. -50, -60, -70):-50, -60
You entered:  -50, -60
Each file will be processed 2 times
1 folders will be processed.

    PROCESSING FOLDER 1 OF 1
    The working directory is J:\Projects\soundscapes\Exercise4\asc
    Number of asc files = 1
    Folder exists, continuing
    Out folder exists, continuing
    Temp folder is J:\Projects\soundscapes\Exercise4\asc_temp
    Output folder is J:\Projects\soundscapes\Exercise4\asc_out

        **  **  **  **  **  **  **  **  **  **  **  **  **  **
        Folder processing started : 1296501302.57
        **  **  **  **  **  **  **  **  **  **  **  **  **  **

            *****  File 1, Folder 1 - File SM87_20081010_220001_segment_wav_dbva
lues Processing  *****
Ln: 22 Col: 0
```

WHAT IS HAPPENING

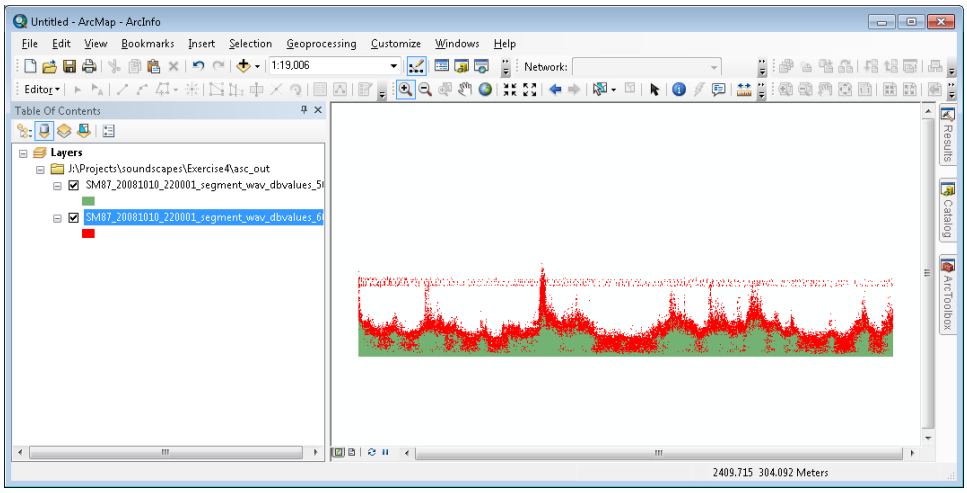
1. The python script takes the ASCII files and creates a binary image based on the cutoff values from step 7 above.
2. Next the binary image is smoothed by running an 8 X 8 majority filter three times in order to reduce the number of polygons to <100,000 on average.
3. The raster file is then converted to a polygon shape file where each polygon represents a patch of sound.
4. Fields are added to the shapefile table to hold information about the file being processed and statistics describing the different sound patches.
5. Date and sensor information about the recording are extracted from the file name and added to fields (sensor, year, month, day, hour, row count, and column count)
6. Next information about the sound patches (area, perimeter, x, y coordinate of centroid) is calculated for each polygon.
7. Finally zonal statistics are run on the spectrogram for each polygon (min, max, range, mean, standard deviation, sum, variety, majority, minority, and median).
8. The resulting shapefile can be viewed in ArcGIS or the dbf portion of the file can accessed using spreadsheet software such as Excel.

ANALYZING THE OUTPUT

We can now look at the polygon file of sound patches and the associated statistics that were generated by the script. For this exercise use ArcGIS to visually examine the sound patches and look at their associated attributes.

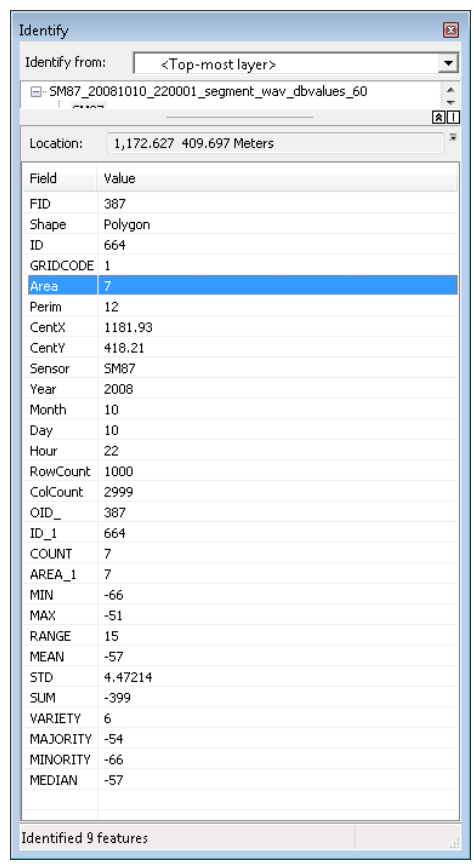
1. Open ArcGIS and add the shapefile from the newly created asc_out folder.

2. The lower cutoff used for the -60 dBFS file will capture a larger amount of sound patches so it's best to place it beneath the -60 dBFS file for visualization.



Notice that the -60 dBFS file (red) extends above the -50 dBFS file (green), and that it also picked up a series of repeating sounds in the upper frequencies. This pattern is the repeating calls of the frogs heard in the first exercise. The red peaks represent cars driving past the recorder.

3. The statistics produced by the script are useful for dissecting sounds within a spectrogram. Use the ID tool to click on one of the sound patches in the upper frequency range. An example is shown on the figure on the right.
4. Begin by looking at the area field. This field represents the size of the sound patch measured in spectrogram pixel units. Pixels are one time unit (i.e. 1/10 of a second) wide and one frequency unit (*i.e.* 10 HZ) tall. In this case a small patch of 7 pixels has been identified.
5. Next the CentX field provides information on the time the sound occurred. Each pixel from the spectrogram represents 1/10 of a second. In this case the sound occurred at 1,181.93 tenths of a second, or 1.97 minutes into the recording.
6. The CentY field provides information on the frequency of the sound patch. Each pixel represents 10 Hz, which means the center of this sound patch occurred at approximately 4.2 khz.
7. The Sensor, Year, Month, Day, Hour fields were extracted from the file name by the script. These fields are most useful when comparing multiple recordings.



Field	Value
FID	387
Shape	Polygon
ID	664
GRIDCODE	1
Area	7
Perim	12
CentX	1181.93
CentY	418.21
Sensor	SM87
Year	2008
Month	10
Day	10
Hour	22
RowCount	1000
ColCount	2999
OID_	387
ID_1	664
COUNT	7
AREA_1	7
MIN	-66
MAX	-51
RANGE	15
MEAN	-57
STD	4.47214
SUM	-399
VARIETY	6
MAJORITY	-54
MINORITY	-66
MEDIAN	-57

8. RowCount and ColCount are derived from the spectrogram raster during processing and are most useful for ensuring the frequency range and recording time are consistent when comparing multiple files.
9. The statistical measures beginning with MIN and ending with MEDIAN were computed based on the spectrogram pixels for each sound patch polygon.
    a. Notice the mean value of -57 dBFS is just above the cutoff value of -60 dBFS. Choosing a lower cutoff of would create large sound patch, but choosing too large of a cutoff value can cause patches to expand into one another and cause loss of detail.
    b. Also notice that the minimum dBFS value in the patch is -66 dBFS, which is below the cutoff of -60 dBFS. This is a result of the smoothing process (i.e. majority filter tool) used to reduce noise within the spectrogram.

CONCLUSION

The script analyzes spectrogram files in units of sound patches. The attributes generated provide information about the time and frequency for each sound patch as well as statistics describing the intensity of the sound. Varying the cutoff frequencies results in outputs that capture different phenomenon and the script is capable of generating multiple output files at once making it easier to find a suitable sound patch file.

You should now be able to process spectrograms to examine sound patches. You can download more recordings from www.purdue.edu/soundscapes and process several sites or days of data.

REFERENCES

Charif RA, Ponirakis DW, Krein TP. 2006. Raven Lite 1.0 User's Guide. Cornell Laboratory of Ornithology, Ithaca, NY.

Ligges U. 2009. tuneR: Analysis of music. http://r-forge.r-project.org/projects/tuner/.

Pijanowski, B.C., A. Farina, S Dumyahn, B. Krause, and S. Gage. 2011. What is soundscape ecology? Landscape Ecology 26: 1213-1232. doi: 10.1007/s10980-011-9600-8

Sueur J, Aubin T, Simonis C. 2008. Seewave: a free modular tool for sound analysis and synthesis. Bioacoustics 18: 213-226.

Villanueva-Rivera L, B.C. Pijanowski, J. Doucette, and B. Pekin. 2011. A Primer of Acoustic Analysis for Landscape Ecologists. Landscape Ecology 26: 1233-1246. doi: 10.1007/s10980-011-9636-9