Using Python to Conduct Emerging Hot Spot Analysis on Wildlife Road Mortalities in South Texas

Thomas J. Yamashita

7 May 2019

1 Abstract

Roads can have significant effects on wildlife including increased mortality from wildlife vehicle collisions. These effects can be reduced through the construction of mitigation structures such as exclusionary fencing and wildlife crossing structures. Gaps in the fenced area can funnel wildlife onto roads exposing them to the threat of vehicle collisions. In South Texas, the Texas Department of Transportation built 11.9 km of exclusionary fencing, broken by 18 private driveways and three major intersections, and five wildlife crossing structures between September 2016 and May 2018 to reduce vehicle mortalities of the endangered ocelot. Surveys of wildlife road mortalities were conducted from August 2015 to December 2018 and all mammalian and herpetofauna mortalities were recorded. Emerging Hot Spot Analysis in ArcGIS was used to assess how hot spots have changed over time at fine spatial scales. In setting up the data for analysis, ArcGIS assumes that all cells that contain no data are missing data and therefore NA. This included some cells within the survey transect known to be zero so a fix was needed to assign these cells the correct value. A function was developed that defined the study area of the analysis, changing all NA values within the area to zero. Analyses run on the updated file revealed new hot spots emphasizing the importance of accurately representing cells within a study area when conducting spatial analyses. Attempts to integrate this function into an ArcGIS workflow were unsuccessful due to incompatibility between ArcGIS and a required Python module. Developing the analysis entirely outside of ArcGIS was impractical due to poor documentation and lack of compatibility between Python modules.

2 Introduction

Roads can have wide-ranging impacts on wildlife, from increased mortality (6) to providing havens from predators (9). While roads can provide some benefits to wildlife, the majority of effects are negative with mortality caused by vehicle collisions being an important threat (10). To reduce the negative impacts of roads, mitigation structures are often built which are designed to keep wildlife off roads while maintaining habitat connectivity (3). This is often done through the construction of exclusionary fencing and wildlife crossing structures (WCSs). Fencing keeps wildlife from accessing the roadway and funnel them towards safe crossing points at WCSs (2). Unfortunately, on major roads, it is not always possible to have continuous fencing along the entire length of the road. Gaps often exist in the fence for private driveways and major intersections creating spaces where wildlife can access roadways (4). Fencing can actually funnel wildlife to these gaps and create hot spots of wildlife road mortality (4).

In Cameron County, Texas, ocelots (*Leopards pardalis*) are critically endangered with one of the major causes of known mortality being ocelot vehicle collisions (7). On State Highway 100 (SH100), a road with several known ocelot mortalities, the Texas Department of Transportation built 11.9 km of exclusionary fencing

and five wildlife crossing structures to reduce ocelot mortalities while still allowing them disperse across the highway. This construction occurred between September 2016 and May 2018. The fence is broken by three major intersections and 18 private driveways. At the intersections, wing walls (fencing that extends perpendicular from SH100) were constructed while at the driveways, wildlife guards (modified cattleguards) were built to reduce the chance that an animal will access the roads. Nevertheless, these locations represent gaps in the fence and therefore could become hot spots of road mortality.

Examining how the spatial pattern of road mortalities has changed over time with construction can be a useful tool for assessing how fencing and fence gaps have affected wildlife road mortalities. ArcGIS, a proprietary, python based spatial analysis program, provides several useful tools for analyzing spatial data (5). One of these tools, Emerging Hot Spot Analysis allows one to assess how hot spots change over time (8). The analysis breaks spatial data into equal space and time blocks and conducts Getis-Ord Gi hot spot analysis on each level of time then assess how hot spots change through time using the Mann-Kendall test (1). In order to run this analysis, the software creates a grid file in NetCDF format representing the full spatial extent of the input dataset and sets all cells that do not contain any data to NA. The analysis relies on information about relationships between nearby cells so having NA cells in areas known to be zero can be problematic. Because ArcGIS is Python based, Python provides a platform for modifying the input file to convert NA values within the survey transect to zero.

3 Methods

3.1 Creating a function to replace NA values in a study area

Wildlife road mortality surveys were conducted between September 2015 and December 2018 along a 15 km stretch of SH100 in Cameron County, Texas. Surveys were conducted by vehicle and all mammalian and herpetofauna mortalities were recorded using a GPS. It is assumed that all mortalities were recorded and the entire roadway was equally surveyed so each location had an equal probability of containing a mortality. Therefore, locations within the survey transect without any mortalities can be assumed to have zero mortalities. To assess how the spatial distribution of mortalities changed over time, I used Emerging Hot Spot Analysis in ArcGIS. Some locations within the survey transect did not have any mortalities throughout the full study period so I needed to correct those locations that were considered NA by the software but are actually zero. Using Python, I created a script that modifies the input NetCDF file to replace NA values within the study area to zero.

The first step was to create a polygon shapefile in ArcGIS representing the study area. This could be used to create the new mask for the NetCDF file. Using the Python modules Fiona and Shapely, I converted the shapefile to a Python polygon which could be used with other Python functions. The NetCDF file was imported using the Python module netCDF4. NetCDF4 was used over Xarray because it allows for reading from and writing back to a NetCDF file while Xarray only allows reading a file. A meshgrid was created from the x and y dimensions of the NetCDF file to represent the extent of the input dataset. A masked array was then created using the polygon.contains() function in Fiona. This function looks at individual cell coordinates of a meshgrid and checks if the coordinates of a polygon falls within the cell. Each cell that contains the polygon was given a value of one while cells not containing the polygon were given a value of zero. In order to check every cell in the meshgrid, a for loop was used to loop through every cell of the grid.

Once the new mask was created, it could be used to replace improper NAs in the data variable containing the number of mortalities. To do this, another for loop was created that looped through each level of in the data variable and changed NA values within the masked area to zero. NetCDF4 identifies NA values using a number (-9999) so the mask was added to the variable and all values that now equaled -9998 were set to 1. Then the mask was subtracted from the variable converting all values in the study area back to their true values. The modified values could not just be set to zero because the +1 from the mask was added to *all* values in the grid including those that were not NA converting some zeros to ones. Once this was done, the

old data variable and the old mask needed to be replaced by the new ones and the NetCDF file was saved so it could be used in ArcGIS for Emerging Hot Spot Analysis. This function was then generalized to make it more compatible with any NetCDF file.

3.2 Working with Anaconda and ArcGIS

Once this function was created, I attempted to turn it into a tool that could be used in ArcGIS by anyone. This required first creating an Anaconda environment that contained the same versions of Python, Numpy, and Matplotlib as ArcGIS Python. Then a .pth file containing the file path of the Anaconda environment needed to be added to the ArcGIS environment and vice versa so the environments could share packages. Lastly, the syntax had to be slightly modified to make the function an ArcGIS tool.

3.3 Recreating Emerging Hot Spot Analysis outside ArcGIS

Additionally, I attempted to replicate ArcGIS's version of Emerging Hot Spot Analysis within Python so adding compatibility and switching between programs was not required, therefore speeding up workflow for future usage of this function. Python has several modules that theoretically made this possible including Pysal and Mk_test. Pysal contains functions for conducting spatial analyses including hot spot analysis while Mk_test contains functions for conducting the Mann-Kendall test. Spatial data in excel format would be converted to a Numpy array based on spatial location and time. Then, hot spot analysis would be run on each level of time and the Mann-Kendall test run among time periods. Hot spot analysis also required creating a spatial weights matrix object which identifies the relationship between nearest neighbors so hot spot analysis knows how cells affect each other.

4 Results

4.1 Function to replace NA values in the study area

Generally, the developed Python function correctly adjusts the data within the NetCDF file, hereafter called NetCDF_StudyArea function and a rerun of the analysis revealed slightly different results. The new mask was successfully created using Fiona and accurately represented the full study area (Fig. 1). This mask was then effectively used to change cells within the analysis variable that should be zero to zero (Fig. 2).

A rerun of Emerging Hot Spot Analysis in ArcGIS revealed slightly different results with the reran analyses showing new hot spots near cells within the study area that had previously been NA (Fig. 3). The overall trends in hot spots showed decreasing trends in locations that had previously been NA (Fig. 4).

Because the function was successful, I generalized it to be useful for any NetCDF file that contained a data variable and a mask of that data variable. It takes an input NetCDF file, an input shapefile representing the study area, the input data variable name, and the input mask name and replaces the mask with the study area and NA values within the study area with zero.

4.2 Compatibility between Anaconda and ArcGIS

Integrating the NetCDF_StudyArea function into ArcGIS proved to be problematic due to compatibility issues between ArcGIS and required modules for the function. ArcGIS Desktop, which has been the standard

release until very recently, is based on a 32 bit version of Python 2.7 so it is important that any installed modules are also designed for this same environment.

Using Anaconda, I created a 32 bit, Python 2.7 environment with versions of Numpy and Matplotlib that were the same as those in the ArcGIS environment. By having this environment, I could then install other modules that theoretically would be compatible with ArcGIS as well. Unfortunately, the Python 2.7 version of Fiona is 64 bit only so it could not be loaded in the ArcGIS environment making use of the NetCDF_StudyArea function impossible within the current version of ArcGIS. Additionally, some ArcGIS tools required for this analysis do not run within Anaconda so the full ArcGIS version of the analysis cannot be run within Python either.

4.3 Streamlining the analysis

Running the analysis entirely within Python also had serious issues. Spatial analysis functions in Pysal had trouble reading and analyzing the input data in all attempted formats. Importing the data was the main issue with this analysis. Several methods for importing NetCDF files and shapefiles were tried including using GDAL, Geopandas, Fiona, Xarray, netCDF4, Pandas, and Numpy. In the end, Pysal would only accept data loaded through its own "read" functions limiting functionality and compatibility with other functions.

When Pysal did analyze data, it produced results of NAs and INFs corresponding with cells that contained data (INF) and those that were NA (NA). Poor documentation coupled with tools that were known to work in other programs meant that manipulating the input data and the analysis to successfully run outside ArcGIS would require more effort than it was worth.

5 Discussion

5.1 Effectiveness of the replacement function

The NetCDF_StudyArea function seemed to effectively replace NA values within a study area to zero and seems to increase the accuracy and validity of Emerging Hot Spot Analysis by allowing one to define a study area that can be used as the analysis mask. This makes it a useful tool for not only Emerging Hot Spot Analysis but for any analysis that uses a NetCDF file that may have NA values which should be zero. In this case, having zeros instead of NA values made all cells have the same number of nearest neighbors so a Gi score could be calculated from equally distant cells for all data points. The differences seen between the original analysis and the new analysis highlight the importance of having an accurate analysis mask when conducting spatial analyses.

The updated analysis also seems to improve confidence in the idea that wildlife crossings attract animals and fence gaps can serve as funnels for wildlife to access the roadway. Additional new hot spots in the area around WCS 1 and 2, an area containing a large number of private driveways and a major intersection, help show the importance of fence gaps in providing wildlife access to the roadway. It appears that areas with more fence gaps have more mortalities and these trends are only appearing after construction of mitigation structures.

5.2 Python and ArcGIS compatibility

Because the NetCDF_StudyArea function seems to be effective and an important step in running Emerging Hot Spot Analysis, it is important to make a tool for use in ArcGIS that anyone, even someone not familiar

with Python can use. Unfortunately, because of the lack of compatibility between the necessary module, Fiona, and ArcGIS Desktop, integrating the NetCDF_StudyArea seems unlikely to be achievable in the near future. The 32 bit architecture of ArcGIS limits its compatibility with Python, especially Python 2.7 because many of the modules in Python 2 were made for 64 bit unix systems so are not compatible with 32 bit windows programs. ArcGIS Pro is a 64 bit, Python 3 version of ArcGIS has recently been released although access to this version is still limited due to its proprietary nature. Improved Python coding functionality exists within ArcGIS Pro so, as this program becomes more mainstream, additional opportunities will arise to integrate new Python scripts into ArcGIS. However, for now, we are restricted to the inefficiency of using multiple programs for a single analysis.

Alternatives to Fiona were tried for converting the shapefile into a useable format including GDAL and Geopandas but I could find no way in either of these modules to convert the shapefile polygon into a useable format. This was partly due to the poor documentation that many Python modules have and the general lack of description about available functions in a module and what they do. While this is probably due the fact that Python is a very new coding language, it still makes it difficult for researchers to have confidence in functions to perform in the way they think they will.

Additionally, lack of good documentation makes finding functions that will perform the desired task difficult, limiting its usefulness outside the programming and computer science world. This was one of the main problems I had with trying to use Pysal to replicate hot spot analysis. As of the time of writing of this paper, the official Pysal documentation provides little information on what types of data are accepted and what was required to perform hot spot analysis. Pysal's incompatibility with data created by other modules also seriously limits the ability to use it in complex analyses dependent on a series of steps which are not all available in a single module.

Therefore, while Python is great for performing tasks that involve programming and developing custom tools, it does not seem suited for conducting complex statistical analyses. Other programs are better suited for this (ArcGIS for spatial analysis and R for other statistical analyses) and provide better documentation for analysis techniques so Python should be used for solving specific issues with a dataset or visualizing data then analyze the data in another program better suited for analysis.

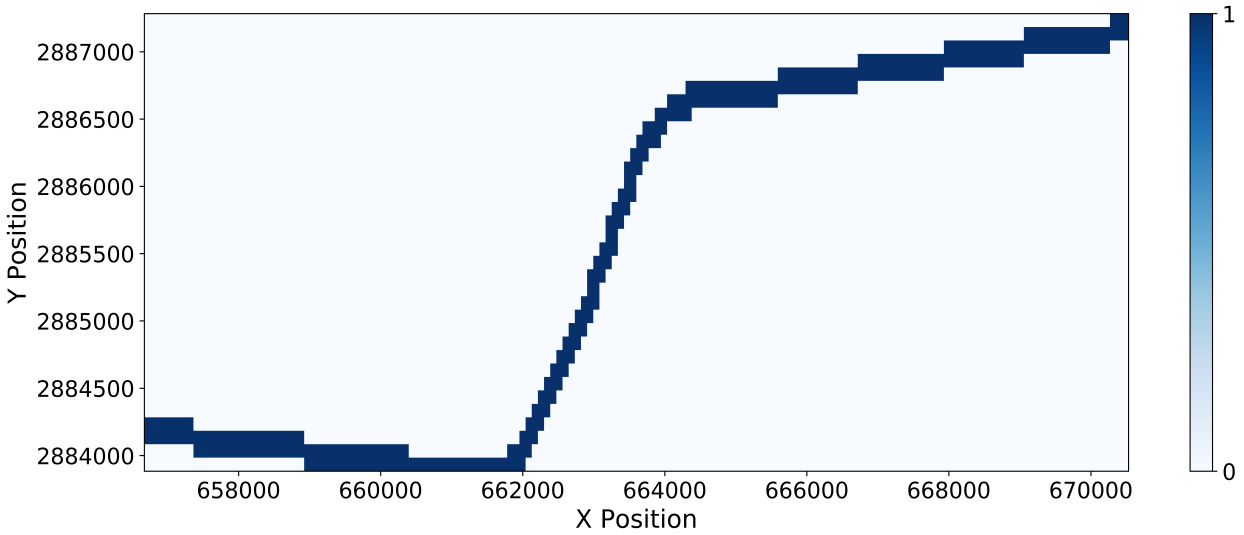

Figure 1: Meshgrid representing the study area of the mortality survey. Cells within the study area have a value of one while those outside the area have a value of zero. The x and y axes are coordinates in UTM (meters)

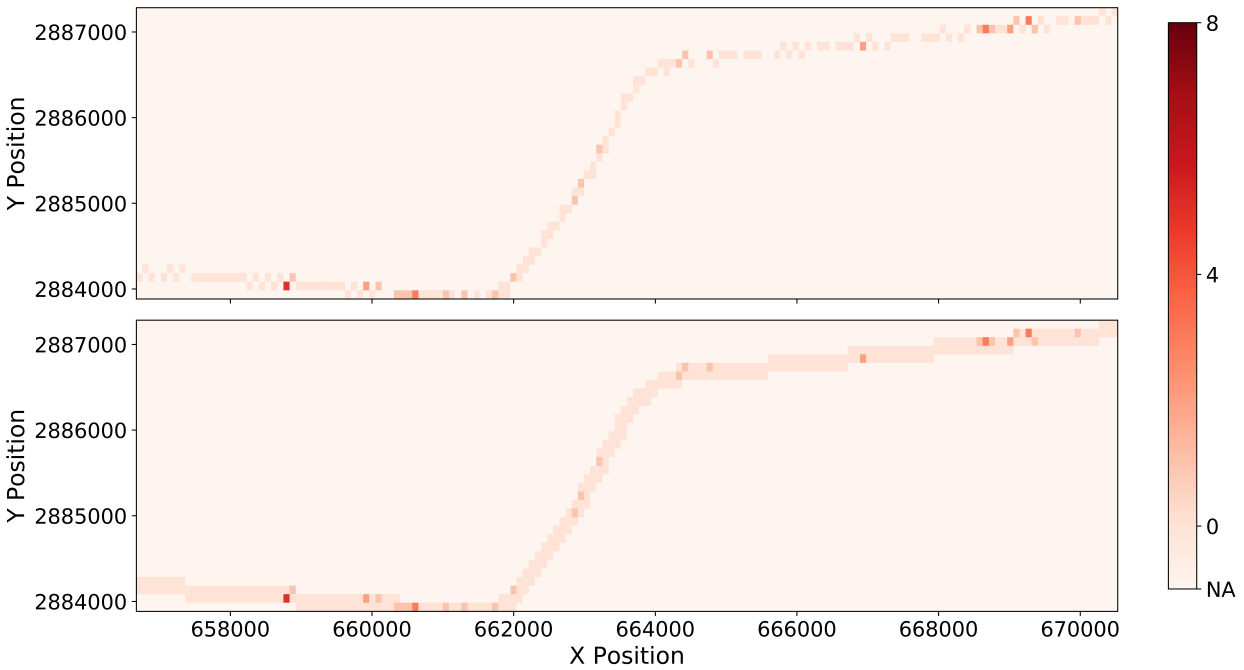

Figure 2: Meshgrids representing the number of mortalities over a single three month period (October 2015 – December 2015) before (top) and after (bottom) correcting the data file using the NetCDF_StudyArea function. After the correction, data within the study area that was NA is now zero. The x and y axes are coordinates in UTM (meters) format.

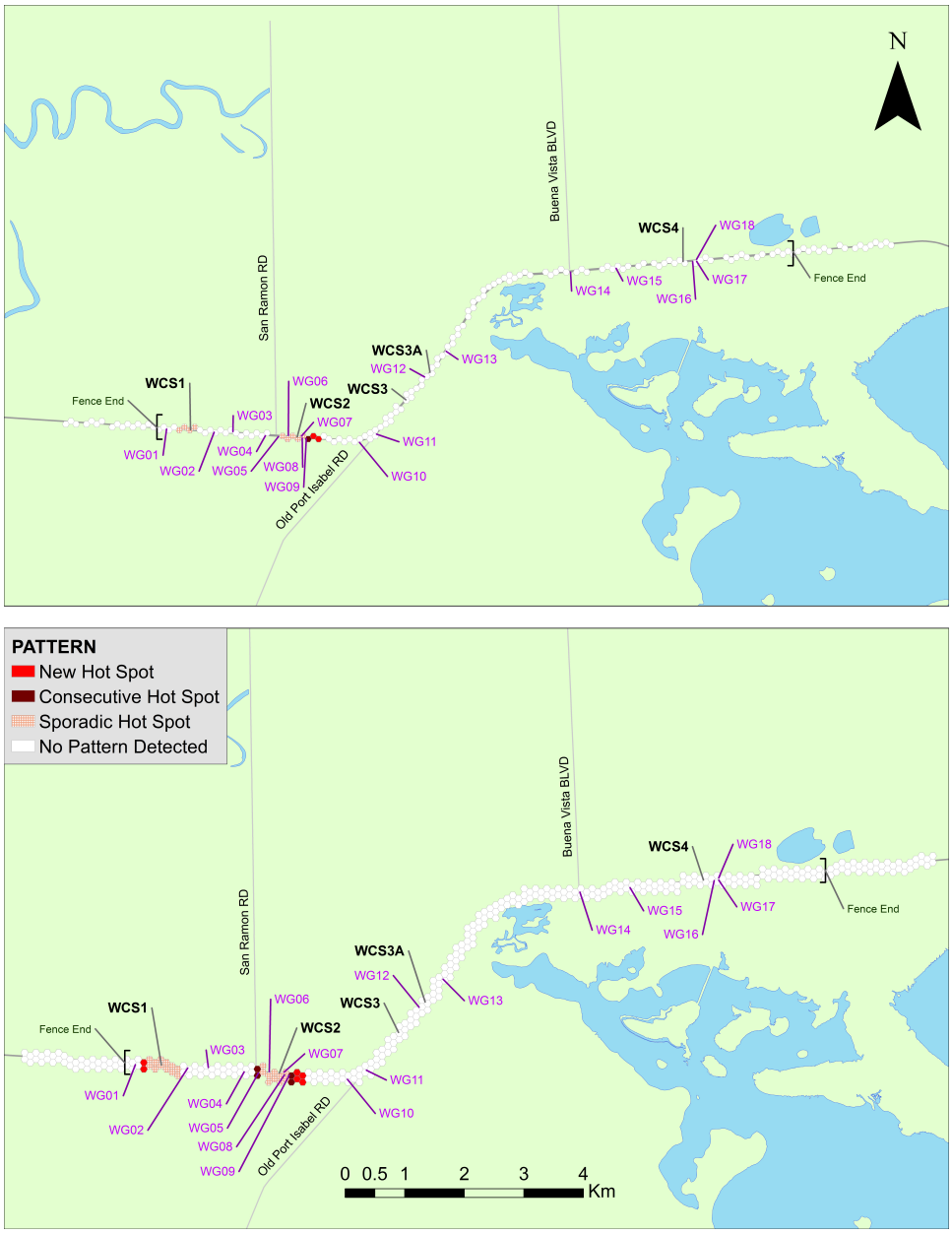

Figure 3: Map of the Emerging Hot Spot Analysis run in ArcGIS before (top) and after (bottom) the correction was made on the data. A new hot spot is one that only emerged since October 2018, a consecutive hot spot is one that has only appeared around July 2018 and a sporadic hot spot is one that switches from hot to not significant over several time periods.

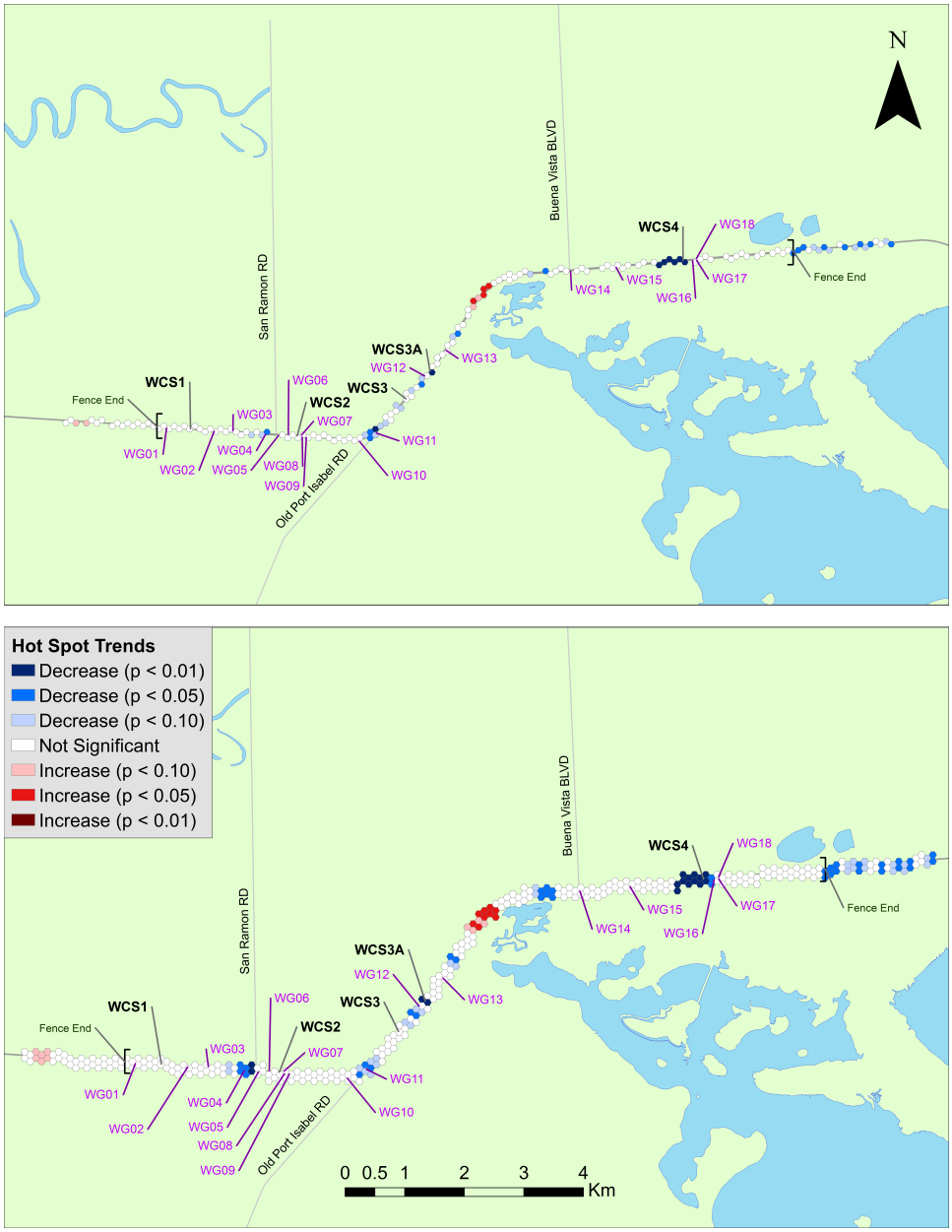

Figure 4: Map of hot spot trends run in ArcGIS before (top) and after (bottom) correction was made. The trend type and its associated p value are represented. Decreasing trends are those that have fewer mortalities over time while increasing trends are those with more mortalities over time.

References

[1] Bass, C. A. *Emerging Hotspot Analysis of Florida Manatee (Trichechus manatus latirostris) Mortality (1974-2012)*. Master's thesis, 2017.

[2] Clevenger, A., Chruszcz, B., and Gunson, K. Highway mitigation fencing reduces wildlife-vehicle collisions. *Wildlife Society Bulletin 29*, 2 (2001), 646–653.

[3] Clevenger, A. P. Conservation value of wildlife crossings: Measures of performance and research directions. *Gaia-Ecological Perspectives for Science and Society 14*, 2 (2005), 124–129.

[4] Cserkész, T., Ottlecz, B., Cserkész-Nagy, , and Farkas, J. Interchange as the main factor determining wildlife–vehicle collision hotspots on the fenced highways: spatial analysis and applications. *European Journal of Wildlife Research 59*, 4 (2013), 587–597.

[5] ESRI. *ArcGIS Desktop: Release 10*. Environmental Systems Research Institute, Redlands, CA, 2017.

[6] Forman, Richard, T. T., Sperling, D., Bissonette, J. A., Clevenger, A. P., Cutshall, C. D., Dale, V. H., Fahrig, L., France, R., Goldman, C. R., Heanue, K., Jones, J. A., Swanson, F. J., Turrentine, T., and Winter, T. C. *Road Ecology: science and solutions*. Island Press, Washington D.C., 2003.

[7] Haines, A. M., Tewes, M. E., and Laack, L. L. Survival and sources of mortality in ocelots. *The Journal of Wildlife Management 69*, 1 (2005), 255–263.

[8] Harris, N. L., Goldman, E., Gabris, C., Nordling, J., Minnemeyer, S., Ansari, S., Lippmann, M., Bennett, L., Raad, M., Hansen, M., and Potapov, P. Using spatial statistics to identify emerging hot spots of forest loss. *Environmental Research Letters 12*, 2 (2017), 024012.

[9] Planillo, A., and Malo, J. E. Motorway verges: Paradise for prey species? a case study with the european rabbit. *Mammalian Biology 78*, 3 (2013), 187–192.

[10] van der Ree, R., Smith, D. J., and Grilo, C. *Handbook of Road Ecology*. John Wiley Sons, Ltd., Chichester, Sussex, UK, 2015.

```
# -*- coding: utf-8 -*-
"""
Created on 7 May 2019 19:49:00

@author: Tom Yamashita
"""

#%% Non-function version for bug testing and explanation of what is happening

# Import Packages required for the function
import fiona
from shapely.geometry import Polygon, Point
import numpy as np
import netCDF4

# Import packages required for visualization but not for running the function
import matplotlib.pyplot as plt

# Define the input file paths
netcdf_file = "Cube.nc"                                 # This is the netcdf created in ArcGIS that needed to be modified
StudyArea = "buffer_shapefiles\\SH100_Buffer_100m.shp"     # This is the area that serves as the new mask for the data

# These 2 defined variables allow the function to be more generalized
mod_field = "OCCURRENCE_SUM_ZEROS"                      # Define the field to modify
mask_field = "OCCURRENCE_SUM_ZEROS_MASK"                # Define the field containing the original mask


# Load the netcdf file as a netCDF4 object
cube = netCDF4.Dataset(netcdf_file, 'r+')               # Need to open with 'r+' so that the file can be edited


# Load the analysis field in the netCDF file as an object
modify = np.array(cube[mod_field])                      # Define the field that needs to be modified


# Create copy of modify field for comparison purposes
mod_original = np.copy(modify)                          # This is for comparison and checking if it works. Not necessary in
function
mask_original = np.copy(np.array(cube[mask_field]))       # This is for comparison and checking if it works. Not
necessary in function


# Load the x and y coordinates from the netcdf file
x = np.array(cube["x"])                                 # Converts the x coordinates (longitude) into a numpy array
y = np.array(cube["y"])                                 # Converts the y coordiantes (latitude) into a numpy array


# Load the study area shapefile and modify it for use in the meshgrid
file = fiona.open(StudyArea)                            # Converts the shapefile into a collection datatype
pol = list(file)                                        # Converts the shapefile collection into a list containing a dictionary datatype
poly_data = pol[0]["geometry"]["coordinates"][0]          # File with the geometry of the shapefile
poly = Polygon(poly_data)                               # Creates a new polygon from the shapefile geometry
```

```
# Create a meshgrid from the x/y coordinates arrays
X, Y = np.meshgrid(x, y, indexing = 'xy')              # Create a meshgrid containing the coordinates of the original data
coords = np.dstack((X, Y))                             # Combine the X and Y meshgrids into a single stacked grid


# Create array with the correct shape but with all zeros in it
sh = (len(y), len(x))                                  # This is the shape of the data needing modification
mask = np.zeros(sh)                                    # Create a 2D array of 0s of the shape of the data


# Create loop that checks if the points in the grid are within the study area polygon
# It fills the mask with 1s where the study area polygon is and leaves the 0s where it isn't
# For every value in the object mask, check if the polygon is overlapping with it. If yes, fill with a 1
for a in range(0, coords.shape[0]):
    for b in range(0, coords.shape[1]):
        mask[a,b] = np.array([poly.contains(Point(coords[a,b,:]))])


# For loop that replaces all NaN values within the masked area to 0
# NaN values in netcdf4 are represented as -9999
# For every level of the object modify, add the mask, set values in mask = 1, subtract the mask
for i in range(0, modify.shape[0]):
    mod_temp = modify[i] + mask
    mod_temp[mod_temp==-9998] = 1
    mod_temp = mod_temp - mask
    modify[i] = mod_temp


# The below is necessary in the function but not in testing

# Save the created objects in the original netCDF file
#cube[mask_field][:] = mask                            # Saves the mask into the original netcdf file
#cube[mod_field][:] = modify                           # Saves the updated data into the original netcdf file

#cube.close()                                          # Updates the original netcdf file with the new variables


#%% Plots for checking function

axis_size = 16
label_size = 18
title_size = 20
suptitle_size = 24

# Plot the new mask
fig, ax = plt.subplots(figsize = (16,6))
m = ax.pcolormesh(x, y, mask, cmap = "Blues")
ax.tick_params(labelsize = axis_size)
ax.set_title("Study area", fontsize = suptitle_size)
ax.set_xlabel("X Position (in UTMs)", fontsize = label_size)
ax.set_ylabel("Y Position (in UTMs)", fontsize = label_size)
bar1 = plt.colorbar(m, ticks = [0, 1])
bar1.ax.set_yticklabels(['0','1'], fontsize = axis_size)
```

```
plt.savefig("Mask2.jpg", dpi = 1000, bbox_inches = "tight", format = "jpg")
plt.show()

# Plot the original and new grids
colormap = "Reds"
t = 0
fig, ax = plt.subplots(nrows = 2, figsize = (16,8), sharey=True, sharex=True)
m1 = ax[0].pcolormesh(x, y, mod_original[t], cmap = colormap, vmin = -1, vmax = 8)
ax[0].set_title("Original Mortality Occurrence File", fontsize = title_size)
#ax[0].set_xlabel("X Position (In UTM Coordinates)", fontsize = 12)
ax[0].set_ylabel("Y Position (In UTMs)", fontsize = label_size)
m2 = ax[1].pcolormesh(x, y, modify[t], cmap = colormap, vmin = -1, vmax = 8)
ax[1].set_title("Updated Mortality Occurrence File", fontsize = title_size)
ax[1].set_xlabel("X Position (In UTMs)", fontsize = label_size)
ax[1].set_ylabel("Y Position (In UTMs)", fontsize = label_size)
bar2 = fig.colorbar(m2, ax = ax, pad = 0.05, orientation = "vertical", ticks = [-1, 0, 4, 8])
bar2.ax.set_yticklabels(['NA','0','4','8'], fontsize = axis_size)
fig.suptitle("Comparison original mortality file and updated mortality file", fontsize = 20, x = 0.425)
fig.tight_layout()
fig.subplots_adjust(top = 0.9, right = 0.75)  # This must come after tight_layout()
plt.savefig("Occurrence_Compare2.jpg", dpi = 1000, bbox_inches = "tight")
plt.show()

#%% Create a function

def netcdf_sa(netcdf_file, StudyArea, mod_field, mask_field):
    """
    This function takes a netcdf file created in ArcGIS for Emerging Hot Spot Analysis and changes the mask.
    The original purpose was to force ArcGIS to include all parts of the study transect in the analysis.
    The ArcGIS function excludes all locations where no mortalities were found, however this does not mean there is
missing data, just that there were 0 mortalities.

    Required Packages:
    fiona,
    shapely,
    numpy,
    netCDF4

    Inputs:

    netcdf_file = input netcdf file which needs to be changed

    StudyArea = input shapefile which will be used as the study area representing the area where data will be replaced

    mod_field = the input field/column/variable that will be modified

    mask_field = the input field/column/variable containing the original mask that will be replaced with the study area

    Output:
    This function will modify the input netcdf file. No new files will be created
    """

    # Import Packages required for the function
    import fiona
```

```
    from shapely.geometry import Polygon, Point
    import numpy as np
    import netCDF4

    # Load the netcdf file as a netCDF4 object
    cube = netCDF4.Dataset(netcdf_file, 'r+')

    # Load the analysis field in the netCDF file as an object
    modify = np.array(cube[mod_field])

    # Load the x and y coordinates from the netcdf file
    x = np.array(cube["x"])
    y = np.array(cube["y"])

    # Load the study area shapefile and modify it for use in the meshgrid
    file = fiona.open(StudyArea)
    pol = list(file)
    poly_data = pol[0]["geometry"]["coordinates"][0]
    poly = Polygon(poly_data)

    # Create a meshgrid from the x/y coordinates arrays
    X, Y = np.meshgrid(x, y, indexing = 'xy')
    coords = np.dstack((X, Y))

    # Create array with the correct shape but with all zeros in it
    sh = (len(y), len(x))
    mask = np.zeros(sh)

    # Create loop that checks if the points in the grid are within the SH100 polygon
    for x in range(0, coords.shape[0]):
        for y in range(0, coords.shape[1]):
            mask[x,y] = np.array([poly.contains(Point(coords[x,y,:]))])

    # For loop that modifies NaN values within the mask to 0 for each level of the netcdf file
    for i in range(0, modify.shape[0]):
        occur_temp = modify[i] + mask
        occur_temp[occur_temp==-9998] = 1
        occur_temp = occur_temp - mask
        modify[i] = occur_temp

    # Save the newly created values in the original netCDF file
    cube[mask_field][:] = mask
    cube[mod_field][:] = modify
    cube.close()


#%% Testing the function

# Import required modules for testing
import netCDF4 as nc
import numpy as np

# Define variables for function
cube_file = "Cube.nc"
```

```
Studyarea = "buffer_shapefiles\\SH100_Buffer_100m.shp"
mod_field = "OCCURRENCE_SUM_ZEROS"
mask_field = "OCCURRENCE_SUM_ZEROS_MASK"

# Pre function values
print(nc.Dataset(cube_file))
occ_pre = np.array(nc.Dataset(cube_file)[mod_field])
mask_pre = np.array(nc.Dataset(cube_file)[mask_field])

# Run the function
netcdf_sa(cube_file, Studyarea, mod_field, mask_field)

# Post function values (for comparison)
occ_post = np.array(nc.Dataset(cube_file)[mod_field])
mask_post = np.array(nc.Dataset(cube_file)[mask_field])
```