GIS Analysis for Applied Economists [1]

Melissa Dell
Department of Economics, Massachusetts Institute of Technology

January, 2009.

[1]Prepared for 14.170: Programming for Economists. Suggestions for revisions very much welcome. Contact: mdell@mit.edu

Contents

Chapter 1

Getting started with Arc

A GIS is a computer system that can assemble, store, analyze, and display geographically referenced information. A preponderance of geospatial data has opened a number of new topics to investigation. Having a solid understanding of the GIS tools necessary to accurately and efficiently process geospatial data can be extremely useful.

The most common software used to analyze geospatial data is ArcGIS. These notes provide an introduction to data analysis in Arc, as well as briefly discussing spatial correlation and map making. The following are a few essential things about Arc, and will be elaborated on subsequently:

1. ArcGIS uses a programming language called Python, which is somewhat similar to C. Many Arc commands can also be run using the menus within Arc, but it is *essential* to use python files. Python scripting is ultimately much more efficient than using the menus, and also allows others to easily check and reproduce your results. I provide examples of Python code at the end of this document.

2. Arc has a tool called the **model builder**, which allows you to put in the commands using the menus, and Arc will then output the appropriate code. The code won't always run without some modifications, but the model builder can still be quite useful.

3. Arc is an incredibly **slow** and **buggy** program. Never do anything in Arc that you can do in Stata or Matlab (i.e. merging data files).

Here's a brief (and very incomplete!) list of things that Arc can do:

1. Perform most mathematical functions on geospatial data (i.e. recode elevation data to 0/1 for below/above a particular cutoff)

2. Calculate distance (between two points, choosing the least cost path according to an algorithm...)

3. Take averages within polygons

4. Calculate slope from elevation data

5. Make maps

And many other functions that we will briefly explore. The first chapter of these notes outlines the various details necessary to begin working with Arc. The second chapter discusses the GIS programming tools available in Arc and develops some applications. These applications are mostly from my own work, since I am most familiar with the details of these projects. The final chapter outlines additional topics, including spatial correlation, map making, and useful GIS resources.

This chapter outlines a number of essential topics that are important to understand before beginning analysis of geospatial data using the Arc utilities described in Chapter 2. Section 1.1 outlines how to access Arc on the MIT (or Harvard) networks, Section 1.2 introduces the types of data that can be analyzed in Arc, Section 1.3 discusses how to import data and navigate within Arc, Section 1.4 provides a discussion of coordinate systems, and Section 1.5 highlights some of Arc's most frustrating quirks and bugs.

1.1 Accessing ArcGIS

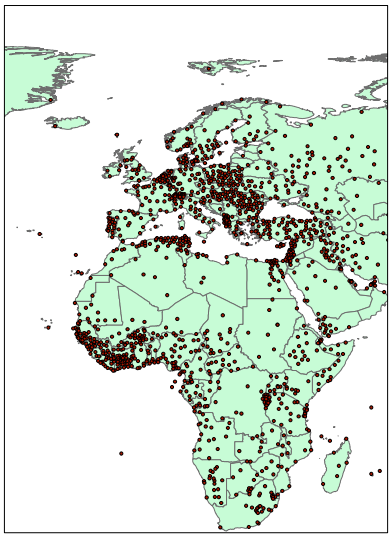
You can easily access Arc on your personal computer (if you have a PC). To request a copy from MIT, use a machine with your MIT personal certificate installed on it to navigate to the following link: https://web.mit.edu/ist/products/vsls/forms/esri.html. From this link, you can order Arc (free of charge) from MIT. Once your copy arrives, insert the DVD, and the installation instructions are self-explanatory. To access your copy of Arc, you must be on the MIT network. If you wish to access Arc off MIT's campus, you can log into the MIT VPN and then open Arc.

ArcGIS is not compatible with a MAC. You can access it via Citrix on the MIT network from the following link: https://citrix.mit.edu/Citrix-Staging/MetaFrameXP/login.jsp. You'll need to download the Citrix client for Macintosh. Then, you can log into Citrix, and use it to run ArcGIS remotely. However, when I tried the Citrix route about two years ago, and it did not work very well. Unless it has improved, MAC users are probably better off using a machine in the lab.

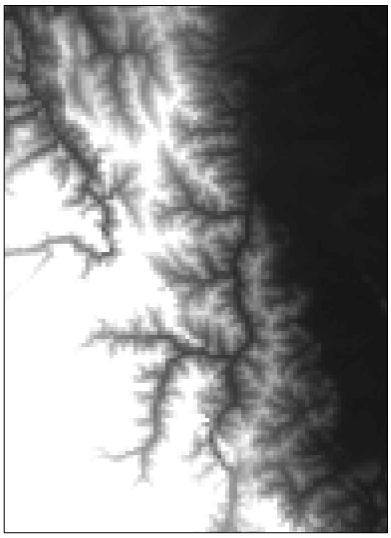
Instructions for Harvard students or associates wishing to access ArcGIS can be found at http://www.gis.harvard.edu/icb/icb.do?keywordk235&pageid=icb.page28957 (or by Googling "Harvard", "GIS Download").

1.2 Arc data types

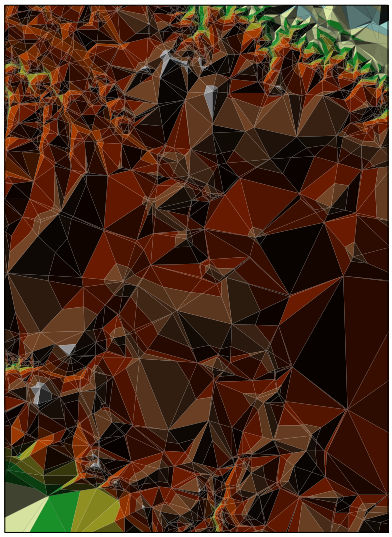
The three most common types of data used in Arc GIS are **features files** (ending in .shp), **raster files**, and **TIN files**. Features data are comprised of points or polygons with associated geospatial references. Raster data consists of pixels. The size of each pixel is given by the resolution of the raster. For example, 90 meter resolution data has a pixel size of 90X90 meters. Finally, TIN is three dimensional data. You may also come across a type of GIS data called a **coverage**, which is somewhat similar to the features class. Coverages have become rather antiquated and can be converted into features files using the conversion features in Arc Catalogue. (See the online help files for detailed instructions.) Later, I will discuss how to convert between the main data types within Arc. The following figure contains an illustration of each of the three main data types:



(a) Features Data



(b) Raster Data



(c) TIN Data

Figure 1.1: GIS Data Types

1.3 Importing data and navigating within Arc

To create a features file, save data files as .dbf files (which you can do, for example, from Excel). Then open the Arc catalogue and add the directory that your files are saved in (by clicking on the arrow button at the top). Next right click on the desired file and click "create features class from XY table" (setting the projection appropriately - see below). This will create a bunch of associated files (with extensions .shx, .shp, .dbf, .prj, etc). Leave these in the same directory. You're ready to either view the data in Arc or perform calculations. Don't open the .dbf file and edit it once the other features files have been created. The .dbf file is referenced to the other files created, so if you need to edit the data, it's best to delete all the GIS files besides the original .dbf, edit this, and then create the GIS files again.

If you wish to create a raster file, first convert your spreadsheet into a features dataset using the Arc catalogue. Then you can use the raster conversion tools (discussed below) to create a raster.

If you need to analyze NetCDF data in Arc, it is straightforward to first convert the data into text data by using Matlab NetCDF utilities (ncdump tends to be ridiculously slow). Then, it should be relatively straightforward to get the data into Arc. In the multidimension toolbox, Arc has some tools for working directly with NetCDF files that might be worth checking out if you are working with this type of data.

While data analysis should be performed using python files, it is important to be able to navigate in Arc. Among other things, this can be useful for viewing data, using the model builder, and creating maps.

You can add data layers by clicking on the plus button and navigating to the appropriate directory. Within Arc, there are two views, data view and map view. Data view is used for editing data, and map view is used for creating and editing maps. You can switch the view by clicking on view in the main menu and selecting either data view or map view.

Arc data capabilities are organized within toolboxes. Click the toolbox button in the main menu in order to view the toolboxes. We will come back to the tasks that these tools perform below. All of the tools in the toolboxes can be called from script files.

Finally, a very useful tool within Arc is the model builder, which allows you to input commands using menus and export them to python files. To use the model builder, right click on Arc Toolbox - at the top of the toolbox menu - and click "new toolbox." Then right click on the new toolbox and click "new model." A window will pop up. Drag and drop the tools you wish to use from the menus into the open window and fill in the appropriate fields. Then click model->export to script->python, to export the code to a python file.

1.4 Coordinate systems

Before analyzing data in Arc, it is **critical** to acquire a basic understanding of projections. The distinguishing feature of geospatial data is that each data point has a geographic identifier attached to it. To view and manipulate your data, these identifiers will need to be referenced to a coordinate system. You should think seriously about the coordinate system that is most appropriate for your application, as using the wrong coordinate system is a critical mistake that can result in your calculations being complete garbage. I provide a brief introduction to coordinate systems now. The following is a useful reference about effective map projections for global datasets: http://www.spatial.maine.edu/ onsrud/ucgis/testproc/mulcahy/mulcahy.html. A more technical paper can be found here: http://carto-research.er.usgs.gov/-projection/pdf/ofr01-181.pdf.

There are two types of coordinate systems:

1.4.1 Geographic coordinate systems

The unit of measure in geographic coordinate systems is degrees minutes seconds. Below is an example of a map that uses a geographic coordinate system:

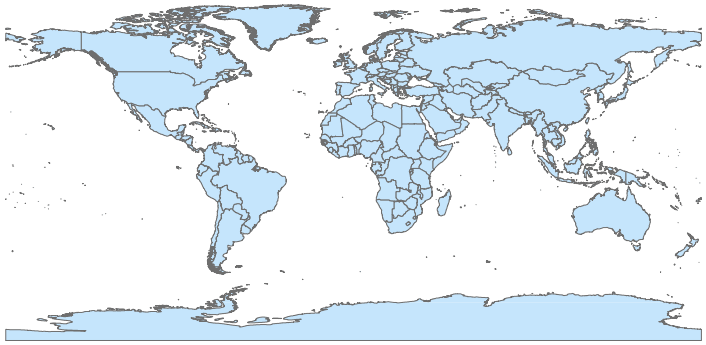


Figure 1.2: Geographic Coordinate System

The principal geographic coordinate system is WGS 1984, a coordinate system designed by the U.S. military. Other geographic coordinate systems exist, although I have not seen any of them used.

It is very important to note that geographic coordinate systems introduce large distortions along a number of dimensions. Imagine trying to wrap the map above

around an orange, and this point is immediately apparent. One degree equals around 110 km at the equator, but only half that at 60 degrees. Distance, area, and shape are all severely distorted as you move away from the equator. Therefore, you should not do distance or area calculations on data that is in a geographic coordinate system. (Below, I discuss how to project data into different coordinate systems using Arc). However, geographic coordinate systems are sometimes useful. For example, I used data in a geographic coordinate system to calculate population weighted temperature within each country. In this case, it was irrelevant whether the area of each cell was distorted, as all that mattered was how much population was referenced to each cell.

Finally, note that if you perform distance calculations in a geographic coordinate system, the output will be in degrees.

1.4.2 Projected coordinate systems

Projected coordinate systems project the round surface of the earth (in degrees) onto a flat surface. The unit of measure in a projected coordinate system is the meter. All projections introduce some distortions, and different projections minimize distortions along different dimensions. I briefly discuss the most common projected coordinate systems below.

UTM projections: The system of UTM projections divides the earth's surface into a number of regions, each of which has its own projection. Because there is little distortion introduced in projecting a small chunk of the earth's surface onto a flat surface, UTM projections have few distortions along any dimension. This type of projection is often useful for data that covers a small geographic area. For example, I've used UTM projections to examine data on a region in southern Peru and data on the West Bank. Both are quite small geographic areas, so the UTM projections worked well. Below is a map of the UTM zones, which can be used to find the appropriate UTM projection for the region that you are examining. UTM zones are identified by a number followed by N or S (for Northern or Southern hemisphere).

Goodes projection: The Goodes, on the other hand, is a global map with relatively little distortion along the area dimension. We used this map to calculate landmass weighted average temperature and precipitation using global climate data. There are other equal area global projections, but the Goodes is one of the most common ones. Note that Arc will crash if you try to project non-global data (i.e. for a single region) into the Goodes projection. Below is a map in the Goodes projection.

Equidistant cylindrical projection: This projection has minimal distortion along the distance dimension, and so is very useful for calculations of distance (i.e. distance between points or to the nearest point on a boundary).

Cylindrical Equal Area: This projection has minimal distortion along the area

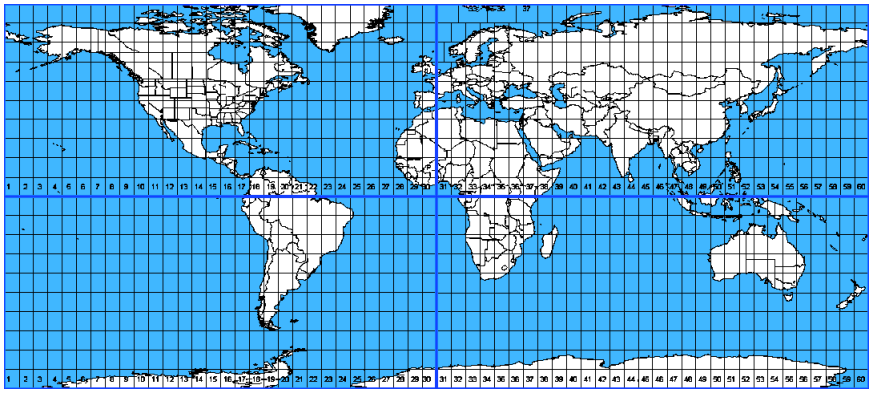


Figure 1.3: UTM Coordinate System

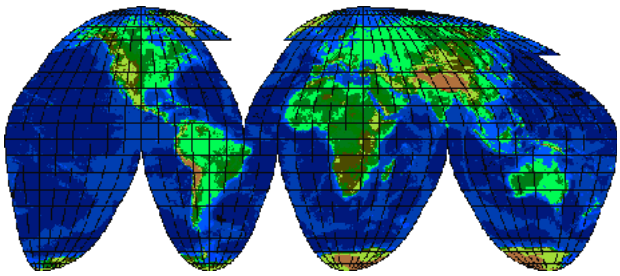


Figure 1.4: Goodes Global Projection

dimension, and hence is very useful for calculating, for example, averages within polygons. It is particularly helpful when working with medium sized geographic regions that are too large for the UTM projections to be useful.

There are a few things about projections that are essential to keep in mind when working in Arc. The first is that **what Arc displays is not necessarily your data.** The first data layer that you open in Arc will define the reference system for the visual output that Arc displays. When you add additional layers, they will be displayed according to the coordinate system of the first data layer that you imported. The underlying data files are not altered. Thus, the displayed version of you data may look very different from the underlying data. While it is important to look at your data - in order to check for mistakes - this should always be done with the above caveat in mind.

Moreover, it is **absolutely essential** to make sure all your layers (datasets) are in the same coordinate system before performing calculations that involve multiple layers (i.e. multiplying one layer by another layer). Often, Arc will more than happily perform calculations on two datasets with different identifiers (i.e. average temperature within country borders, when the temperature and country data are in different projections). However, the output will be total garbage.

Thus, it will likely be necessary to re-project some of your data to ensure consistent

calculations. Arc has three projection tools:

- Project. This command is used to re-project features datasets. For example, your data might come with a geographic coordinate system whereas you need it in an equidistant coordinate system to perform distance calculations.
- Project Raster. This tool is used to re-project a raster dataset.
- Define projection. Occasionally, geospatial data will not come with a projection file. (The projection file is the one with a .prj extension). In this case, you will have to set the appropriate projection before working with the data, and this can be done by using the define projection command. The only time that you need to define the projection is when your data do not come with a .prj file. *You should not define the projection for data that already has a projection defined. This can cause Arc to misbehave.* If you wish to change the projection for data that already have a projection defined, use the project command (shapefiles) or the project raster command (raster datasets).

The project and project raster tools create new data files, in the converted coordinate system. The define projection tool obviously does not.

Recall that projections introduce distortions. The first source of distortion is probably familiar - there is no perfect way to smash an orange peel onto a flat surface. The second type of distortion may be less familiar, and appears when projecting rasters. Each pixel of a raster is a little square, so when you project a raster, Arc will resample the raster so that the new raster is also made up of little squares. There are several options for resampling the raster. Nearest should be used for discrete data (i.e. a 0/1 raster indicating urban/rural). The bilinear option is usually used for continuous data (i.e. temperature). You can find out more about the details of resampling in the Arc help files.

See the attached code sheet for examples of python code for the project commands. These commands involve spatial references and tend to be very long and messy. Thus, getting code for projections is one place where the model builder can come in enormously useful, even for experienced python users.

1.5 ArcGIS quirks and bugs

Deleting Files: Arc creates A LOT of garbage intermediate files when performing calculations. These can fill up your computer's memory relatively quickly if you are performing a number of calculations. Worse, there is a ridiculous bug in Arc that causes the system to crash if you put a multiple of 1124 files in an Arc directory. This

has to do with the way that Arc stores data, and is a problem that can be avoided by getting rid of intermediate files as soon as you are done with them.

At the beginning of the loop, use the command *os.mkdir(directory name)* to create a temporary directory. Then use the command *gp.workspace = directory name* to set the directory to the temporary directory you just created. Store your final .dbf output files in a different directory. Finally, at the end of the loop, use the command *shutil.rmtree(directory name)* to delete the directory and all the intermediate files in it.

Clearing the Cache: I've also had this bizarre problem where the Arc cache gets full and every time you try to export a .dbf file, it will export the same .dbf file over and over. If this starts happening, first try clearing the cache from within Arc. Annoyingly, it is not possible to clear the cache from a script file (apparently this is being corrected for Arc 9.3), but you can clear it using the Arc menus. If all else fails, reinstall Arc.

Raster Resolution: When zonal statistics calculates the mean raster value within each polygon, it does this by turning the polygon temporarily into a raster, of the same resolution as the raster that you are averaging. Oftentimes, there may be small polygons in your dataset that are smaller than the resolution of your raster. For example, the resolution of our climate raster was 0.5 degrees, but countries such as the Bahamas, Trinidad, and Sao Tome are smaller than this so they got subsumed in the raster. In order to calculate mean temperature in these countries, we used the following steps. First, we clipped the raster down to the size of each respective country. Then, we used the resample command to change the raster resolution to 0.1 degrees. Then, we performed the calculations for each of the small countries separately. Note that increasing the resolution of the raster dramatically increases the memory necessary to store it as well as the time necessary for the calculations to run. Running climate calculations for just Trinidad with a 0.1 degree resolution raster takes about the same amount of time as running climate calculations for the whole world with a 0.5 degree resolution raster. Hence, it is important to clip the raster down to the minimum extent necessary before resampling and running calculations.

Identifiers: It is often necessary to add an identifier variable to a GIS data file. This is useful, for example, if you are exporting results of several calculations performed in Arc in different .dbf files to be merged together in Stata. Features files have an internal identifier, called the fid (features identifier) that is visible within Arc. As far as I've been able to figure out, the fid will not export to a .dbf file. There is also another annoying thing about the fid. When doing analysis with a grid you've created (i.e. calculating the average elevation within grid cells), it is necessary to use a unique features identifier to tell Arc that you wish to calculate averages within grid cells. However, the fid cannot be used for this purpose and creating a variable identical to the fid also will not work (presumably because the first fid is zero). The way to get around this is to create a variable equal to fid+10. Then Arc will recognize

this as a features identifier without problems, and it will also export to a .dbf file. You can create this variable by using the calculate field command.

Filenames: When exporting data from calculations performed within Arc to .dbf files, be sure to give the files short names (five or less characters). This is not an Arc quirk, but rather a Stata quirk. There is an enormously convenient command:

odbc load [extvarlist] [if] [in] , table("TableName")—exec("SqlStmt") [load_options connect_options]

that can be used to load .dbf files into Stata. This is especially convenient if you have a bunch of small .dbf files. odbc load will not run if the file name is longer than (I think) five characters, but rather just gives a useless error message.

Variable Formats: When you create a variable, you can choose from various data formats (short integer, long integer, float, double - which as in Stata is a float with twice the precision). Make sure to choose the correct type for your application (i.e. long integer for id numbers, float for continuous data, etc).

Chapter 2

ArcGIS Tools

This is by no means an exhaustive list of the tools available in Arc, but rather an introduction to tools likely to be useful for social scientists. Much more information can be found in the Arc help files. The sections are organized by broad classes of tools and further divided into sub-classes. Section 2.1 discusses the 3D Analysis tools, Section 2.2. the Analysis tools, Section 2.3 the Conversion tools, Section 2.4 the Data Management tools, Section 2.5 the Multidimensional tools, Section 2.6 the Spatial Analyst tools, Section 2.7 the Geoprocessing tools, Section 2.8 Hawth's Tools (an ArcGIS add-on), and Section 2.9 additional scripting utilities. Empirical examples related to the analysis of road networks and climate data are developed primarily in Section 2.6. (It will be useful to first peruse the more list-like descriptions of Arc capabilities in the earlier sections to become familiar with the tools discussed in the applications.) Python code for the functions discussed in this chapter can be found in the appendix.

2.1 3D Analysis Tools

Conversion

- *Conversion*: The 3D conversion toolbox contains various utilities to convert to and from TIN data to other data types (i.e. raster data, digital terrain data.)

Functional Surface

- *Surface Length*: Calculates the surface length of each line in a feature class over a three dimensional TIN surface or a raster surface.The surface length information is stored as an attribute of the input feature class. This is useful,

for example, for calculating the length of roads while accounting for changes in elevation.

- *Surface Volume*: This tool calculates the volume under three dimensional surfaces.

Raster Interpolation

This set of tools contains various utilities for creating interpolated surfaces from a series of points (i.e. points spaced every 1 km giving elevation). For details of the weighting schemes used, please see the Arc help files.

- *IDW*: Interpolates a surface from a series of points using an inverse distance weighting technique.
- *Kriging*: Interpolates a surface from points using kriging.
- *Natural Neighbor*: Interpolates a surface from points using a natural neighbor matching technique.
- *Spline*: Interpolates a surface from points using a minimum curvature spline technique.

Raster Math

- *Divide*: Divides the values of two rasters on a cell-by-cell basis.
- *Float*: Converts each cell value of a raster into a floating-point representation
- *Int*: Converts each cell value of a raster to an integer by truncation.
- *Minus*: Subtracts the value of the second input raster from the value of the first input raster on a cell-by-cell basis
- *Plus*: Like minus, but adds
- *Times*: Like divide, but multiplication

Raster Reclass

- *Reclassify*: Reclassifies (or changes) the values in a raster.
- This set of tools also includes utilities to reclassify the values of a raster using values stored in a .dbf file (*Reclass by Table*) or .ascii file (*Reclass by ASCIIFile*), rather than entering the values directly into the command. This is extremely useful when reclassifying more than three or four values.

2.2 Analysis Tools

Extract

- *Clip*: Extracts input features that overlay the clip features. You can think of the clip feature like a cookie cutter, selecting only the part of the data set to be clipped that are within its boundaries.
- *Select*: Extracts data based on attributes. For example, you might have a map of all countries in the world, that contains a field giving each country's continent. The select utility can be used to select only those countries with the continent field equal to "Asia."
- *Table Select*: Extracts selected table records or features from an input table or table view and stores them in a new output table.

Overlay

- *Erase*: Creates a feature class by overlaying the Input Features with the polygons of the Erase Feature. Only those portions of the Input Features falling outside the Erase Features outside boundaries are copied to the Output Feature Class. Erase is the opposite of *Clip*. It can be thought of as a cookie cutter that keeps only the portions of the input data set that are outside the Erase Feature's boundaries.
- *Intersect*: Computes a geometric intersection of the Input Features. Features or portions of features which overlap in all layers and/or feature classes will be written to the Output Feature Class.
- *Spatial Join*: Creates a table join in which fields from one layer's attribute table are appended to another layer's attribute table based on the relative locations of the features in the two layers. Note that this command can take a long time to run.
- *Union*: Computes a geometric intersection of the Input Features. All features will be written to the Output Feature Class with the attributes from the Input Features, which it overlaps.

Proximity

- *Buffer*: Creates buffer polygons to a specified distance around the Input Features. An optional dissolve can be performed to remove overlapping buffers. This is useful for making features more visible in maps, and also for some types

of analysis. For example, you may need to take some sort of intersection of two features data sets that do not quite precisely overlap due to slight imprecisions in the underlying data. I used this tool to make rivers slightly wider in order to determine which portions of a political boundary were formed by rivers.

- *Near*: Determines the Euclidean distance from each point in the input feature to the nearest point or polyline in the near feature, within the (optional) search radius. This is the easiest to use distance tool, useful when you need only the straight line distance from a point to another point or line. Arc has a number of more advanced distance utilities that are discussed in greater detail below.

2.3 Conversion Tools

- *Point to raster*: Converts point features to a raster dataset.
- *Feature to raster*: Converts features to a raster dataset.
- *ASCII to raster*: This converts an ASCII data file into a raster. This is extremely useful because some raster datasets are available only in ASCII form, and this tool must be used to open them in Arc.
- *DEM to Raster*: DEM stands for digital elevation model, which is a data type used by the U.S. Geological Service to store elevation data.
- There are also utilities to make all of these conversions in reverse, as well as to convert to and from lesser used data types.

2.4 Data Management Tools

Features

- *Add XY Coordinates*: Adds the fields POINT_X and POINT_Y to the point input features and calculates their values (i.e. adds latitude and longitude coordinates for all your data points). The units of the coordinates will depend ont the coordinate system in the input feature. If you want coordinates in degrees latitude and longitude, the input feature must be in a geographic coordinate system.
- *Copy Feature*: Makes a copy of the input features class. This is useful if you are editing a data set and wish to keep a copy of the original.

- *Feature to Point*: Creates a point Feature Class based on an input polygon, line, or multipoint feature class. The attributes of the input features are present in the resulting points. This can be useful, for example, in calculating the centroids of administrative units (i.e. counties or states).
- *Feature Vertices to Points*: Creates points at each of the vertices of the input features class.
- *Polygon to Line*: Converts a polygon feature into a line feature (i.e. only keeping the boundaries).
- *Split Line at Vertices*: Splits a line features data set into separate features data sets at its vertices.

Fields

- *Add Field*: Adds a field to the table of a feature class, feature layer and/or raster catalog.
- *Calculate Field*: Calculates the values of a field for a feature class, feature layer, or raster catalog. For example, you may have one field attached to a features data set that gives a state identifier and another that gives a municipality identifier. You can create a single identifier by first using the *Create Field* utility to create the field "geoid" and then using the calculate field utility to calculate the geoid as the desired combination of the state and municipality identifiers.
- *Delete Field*: Deletes one or more fields from a table of a feature class, feature layer, and/or raster catalog.

General

- *Append* Appends a features data set to an existing features data set.
- *Merge*: Merges two or more features data sets into a new features data set.

Generalization

- *Aggregate Polygons*: Combines polygons that are located within a pre-specified distance of each other. This is useful, for example, when you've taken the union of two polygon data sets and want to combine intersecting or nearby polygons into a single polygon. You can also use this tool to eliminate polygons that are smaller than a pre-specified area.

- *Dissolve*: Aggregates features based on specified attributes.

- *Simplify Line*: Simplifies a line by removing small fluctuations or extraneous bends from it while preserving the essential shape. This may be useful, for example, in making maps.

- *Simplify Polygon*: Same as the *Simplify Line* tool, except for polygon features classes.

Projections and Transformations

- *Project*: Changes the coordinate system of your Input Feature Class to a new Output Feature Class with the newly defined coordinate system, including the datum and spheroid.

- *Project Raster*: Transforms a raster dataset from one projection to another, creating a new raster layer.

- *Define Projection*: Records the coordinate system information for the specified input dataset or feature class including any associated projection parameters, datum and spheroid. It creates or modifies the feature class's projection parameters. This should only be used if a projection has not already been defined for the data.

Raster

- *Clip*: Creates a spatial subset of a raster dataset. The clipped area is specified by a rectangular envelope using minimum and maximum x and y coordinates.

- *Resample*: Alters the proportions of a raster dataset by changing the cell size. The cell size will be changed, but the extent of the raster dataset will remain the same.

- *Copy Raster*: Creates a copy of a raster data set, which may be useful if you are editing a raster and wish to preserve a copy of the original.

- *Mosaic to New Raster*: Merges two or more existing raster data sets into a new raster data set.

- *Mosaic*: Appends a raster or rasters to an existing raster data set.

2.5 Multidimensional Tools

This tool box contains various utilities for converting NetCDF files into data types used by Arc (i.e. rasters). NetCDF is a type of binary data storage that is sometimes used to store multidimensional data. For example, I have seen it used quite frequently by meteorologists to store large data sets containing satellite data or climate reanalysis data. Imagine that we have data on day, year, and location of temperature readings for a grid covering the earth's surface. A NetCDF file can be visualized as a data format that stores each observation of this data set in a cell of a cube. One dimension of the cube is geographic location, another dimension is the day of the year, and the third dimension is the year. To analyze this data in Arc, one needs to slice off one year at a time to produce a two dimensional table of countries and temperature observations for each day of a particular year.

This can be done in Arc using the multidimensional utilities. However, if you are working with a large NetCDF data set, it will be much quicker to convert NetCDF to ASCII in Matlab using Matlab utilities, and then read these ASCII files into Arc. For more information on Matlab's NetCDF utilities, see the following Web page: http://mexcdf.sourceforge.net/.

2.6 Spatial Analyst Tools

Density

- *Kernel Density*: This uses a kernel function to fit a smoothly tapered surface to each point or polygon in the underlying dataset. It can be useful for smoothing data in point format to present in maps. For a stunning example of the use of the density toolbox to clearly and concisely present economic data, I highly recommend the book *England on the eve of the Black Death: an atlas of lay lordship, land and wealth*, by Bruce Campbell.

Distance

I now describe Arc's distance capabilities in some detail, focusing on conceptual issues. To learn more about the specifics of scripting for these tools, I would recommend the Arc help files as an excellent reference. One example of the use of this set of tools in applied economics research is my paper with Daron Acemoglu on road networks in Latin America, which will (hopefully) be available on our Web pages soon.

In order to better understand the distance capabilities of Arc, we will start by discussing its simplest tools and progress to slightly more advanced ones. In the

simplest application, we may wish to calculate straight line distance between a city and nearby areas, not accounting for changes in elevation or allowing for differential costs along the route. This can be done using the *Euclidean Distance* tool. The input is a feature or raster source data set from which distance is calculated (i.e. Boston) and optionally, a maximal distance (i.e. 200 km). The output is a raster (recall that a raster is a grid containing equally-sized cells - you will need to specify the cell size), consisting of all cells located within the maximal distance to Boston. This raster data set gives the Euclidean distance of every cell in the raster to Boston. We could use this raster, for example, to read off the straight line distance from Boston to New York. We could also use it to take averages within polygons - for example, the average Euclidean distance to Boston across all cells in the state of Vermont.

Now suppose that we want to calculate the least cost routes for building roads from Boston to several nearby cities (i.e. Worcester, Amherst, and New York), making it more costly due to congestion to pass through densely populated areas. At this point, we still do not consider elevation in our calculations (later in these notes we will discuss how to account for changes in elevation and how to impose penalties for ascent and descent).

The appropriate tool to use is the *Cost Distance* utility. In order to predict the least cost paths, we must first prepare the following inputs:

1. A features or raster data set that gives the source from which we wish to calculate the roads to other cities. For example, this might be a point that gives Boston's geographic location. Following the standard GIS terminology, we will refer to this input as the *source*.

2. A raster that gives the relative cost of passing through each cell in the raster. We will refer to this input as the *cost surface*. This raster should cover the entire area that we will allow potential roads to pass through. For example, this might be a raster that covers the states in New England. If densely populated cells are assigned a value in the cost raster of five and rural cells a value of one, the cost to pass through the densely populated cells will be five times as high as the cost to pass through the rural cells. To see this more concretely, imagine we have a raster consisting of 1 x 1 km cells. To pass diagonally through a cell assigned a value of one costs $\sqrt{2}$ (remember the Pythagorean theorem) and to pass directly across it costs 1. To pass diagonally through a cell assigned a value of 5 in the cost raster, in contrast, would cost $5\sqrt{2}$ and to pass directly through it would cost 5. If a cell is set to No Data, then the path is not allowed to pass through that cell. For example, one might wish to set water bodies to No Data.

   In an alternative example, we may not wish to calculate the least cost hypothetical path from Boston to New York, but rather calculate the least cost way to travel to New York on existing roads. The weak at heart could do this with

Google Maps, but you could also do it in Arc GIS by creating a cost surface raster equal to no data for all areas between Boston and New York not consisting of roads. In the simplest case, all roads would be symbolized by a chain of raster cells all equal to one. In a more complex example, the cost associated with each kilometer of roadway would depend on the traffic intensity.

The path distance tool will produce the following three outputs:

1. A raster (whose extent will be the the same as the cost surface raster described above) giving the cost from the nearest source to each cell in this output raster. We will call this the *accumulative cost raster*.

2. A raster that identifies which cell to move or flow into on the way back to the source, for every cell in the cost distance raster. That is, Arc will output a raster that identifies the next cell on the least accumulative cost path. We will call this the *output backlink raster*.

3. If the source data consists of more than one cell (i.e. if we are calculating the least cost path to Boston or Worcester, whichever is closer), Arc will also produce a raster that tells which source is the closest to each cell in the raster.

The following illustration provides a simple example of these inputs and outputs.

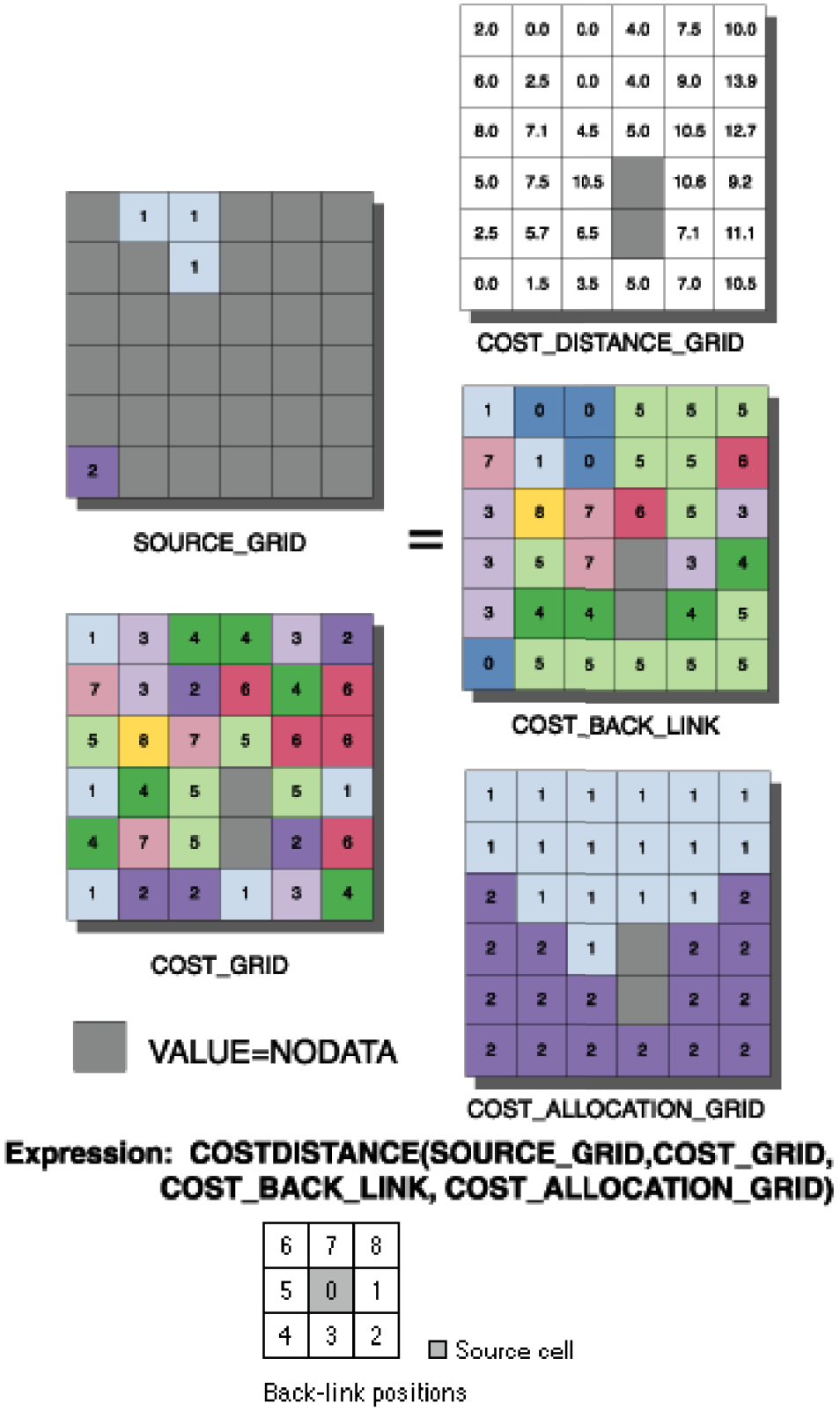


Figure 2.1: Cost Distance Calculations

Once the source dataset and cost surface raster have been defined, the accumulative path distance from the source to each cell in the raster is calculated iteratively. Imagine links between the centroid of every cell in the raster and its eight immediate neighbors. Every link has an impedance associated with it. The impedance is derived from the costs associated with the cells at each end of the link (which in this example is given by the cost surface raster) and from the direction of movement (i.e. resulting in a distance of 1.4142 or 1 depending on whether the movement is diagonal or directly through the cell). The steps are as follows:

1. In the first iteration, the source cell(s) are identified and assigned a value of zero since there is no accumulative cost to return to the city(s) that serves as the source.

2. Next, all the source cell's neighbors are activated, and a cost is assigned to the links between the source cell nodes and the neighboring cells' nodes.

3. The accumulative cost values for each of these cells are arranged in a list (called the active list) from the lowest accumulative cost to the highest, and the lowest cost cell is chosen from the active accumulative cost cell list.

4. The value for that cell location is assigned to the accumulative cost distance raster.

5. The list of active cells expands to include the neighbors of the chosen cell, because those cells now have a way to reach a source. Only those cells that can possibly reach a source can be active in the list. The cost to move into these cells is calculated.

6. Again, the active cell on the list with the lowest cost is chosen, its value is assigned to the cost distance raster, the neighborhood is expanded, the new costs are calculated, and the new cost cells are added to the active list. These costs are also calculated for the neighboring cells of the newly assigned output cell, even if the neighboring cells are on the active list through another cell. If the new accumulative cost for a location already on the active list is greater than the one the cell already had, the new value is ignored. If the accumulative cost is less, the old accumulative cost for the location is replaced on the active list with the new value.

7. When all cells have been chosen from the active list, the result is the *accumulative cost raster*.

Once the outputs from the above analysis have been produced, they can be used in combination with the *Cost Path* utility to create another raster that records the least cost path or paths from a selected destination (i.e. New York) to the closest source

cell defined within the accumulative cost surface (i.e. Boston). Then, for example, the least cost path can be displayed on a map or used in further calculations.

Another interesting possibility is to use the accumulative cost raster to calculate the average distance to the source within polygons. For example, perhaps the source is the coast, and the cost surface is a grid covering the landmass of the United States. We could use the steps above to calculate the distance for every cell in the United States to the nearest coast and then take the average of the accumulative cost raster within all states in the United States using the zonal statistics tools that will be described below.

Now let us consider a somewhat more complex example using Arc's distance capabilities, where we explicitly account for changes in elevation when calculating distance and also impose penalties for ascent and descent. We can do this using the *path distance* utility. The outputs are the same as those described above. Again, the source data set and cost surface form two of our inputs. In addition, we must also prepare the following inputs:

1. A raster giving the elevation of each cell, that has the same extent as the cost surface raster. We will refer to this as the *surface raster*.

2. A penalty for ascent or descent, which is described in more detail below.

3. A maximal incline of ascent or descent that the cost path can make. Depending on the application, an appropriate value for this parameter may be 10 or 15 degrees, or in some cases significantly less. If the ascent or descent into a neighboring cell has a greater incline than this threshold value, the path will not be allowed to pass into that cell.

Let us now discuss the penalties for ascent and/or descent. Arc allows you to choose one of their pre-set functional forms for the penalty, or you may set the schedule for the penalty yourself using an ASCII table. One example of the functional forms that the penalty can take is inverse linear. A graph of this penalty is shown below.

This graph allows the parameter to vary linearly from one (no penalty) to two as the incline moves from 0 to 90 degrees. More realistically, one may wish to set the maximum incline at a much lower value.

With these inputs, the accumulative path distance between the closest source cell a to a destination cell b, moving through a set of cells $\{c\}$ that form the path $\mathcal{P}$ is:

$$cost\ distance_{ab\mathcal{P}} = \sum_{c \in \mathcal{P}} cost\ surface_{abc} * \ surface\ distance_{abc} * vertical\ factor_{abc} \quad (2.1)$$

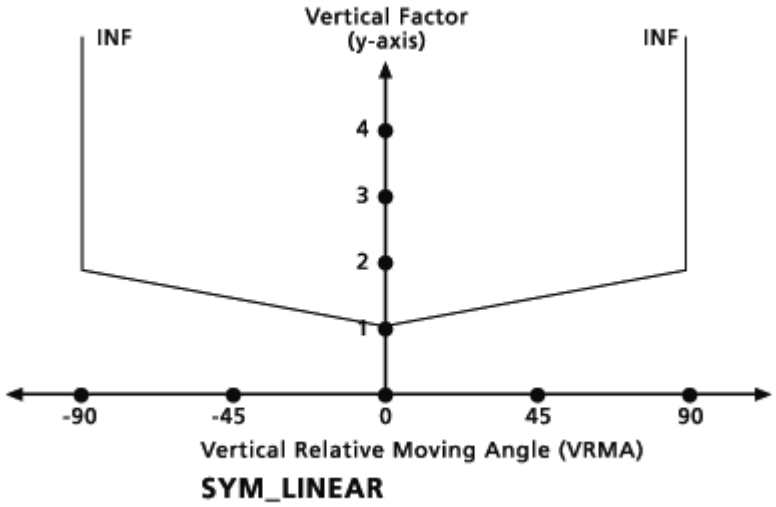


Figure 2.2: Example of Vertical Penalty

The variable *cost surface*$_{abc}$ gives the relative cost of moving across each cell in the set of cells $\{c\}$ between points a and b. In the example given for the *Cost Distance* tool above, this raster contained values of one for rural cells and values of five urban cells. If there are cells that the path should not be allowed to pass through (i.e. water bodies), they should be coded as no data. The variable *surface distance*$_{abc}$ gives the distance traveled to pass through each cell c in path $\mathcal{P}$, accounting for changes in elevation as one moves into the cell. Note that before - when we did not consider elevation - we were calculating distance as if all points on our map lay in a two-dimensional plane. Now we have a third dimension - given by the elevation raster - and distance is calculated in three-dimensional space. Finally, *vertical factor*$_{abc}$ imposes a penalty for ascents and descents, as discussed above.

The algorithm that Arc uses to calculate the accumulative cost raster is as above, except now the cost of passing from one cell to the next is given by (2.1): that is, it accounts for changes in elevation and imposes penalties for ascents and descents. An output raster giving the least cost path from Boston to New York can again be produced using the *Cost Path* utility.

Once can continue to add complexity to the distance calculations. These additional utilites are fairly easy to understand given an understanding of the techniques discussed above, and hence are mentioned here only briefly. More information is available in the Arc help files.

First, in addition to imposing penalties for vertical ascent and/or descent, it is possible to impose penalties for horizontal movement. For example, perhaps you are

trying to predict sailing routes and wish to make it more costly to sail against a northerly wind and less costly to sail in the same directions as the winds. You can impose horizontal penalties with the path distance tool using the same approach as was used above to impose vertical penalties.

Second, you may wish to calculate a path between points a and b that is required to pass through a third point c (i.e. you may wish to build a road from Denver to Dallas that is required to pass through Tulsa). This can be done using the *Corridor Utility* and similar steps to those described above.

Extraction

- *Extract by Mask*: Extracts the cells of a raster that correspond to the areas defined by a mask.
- *Extract by Attribute*: Extracts the cells of a raster based on a logical query.
- *Extract by Polygon*: Extracts the cells of a raster based on a polygon.
- *Extract by Rectangle*: Extracts the cells of a raster based on a rectangle.
- *Extract by Circle*: Extracts the cells of a raster based on a circle.
- *Extract Values to Points*: Extracts the cells of a raster to points. This is very useful if you need to read values off a raster. For example, you may have a point data set consisting of cities and wish to read their elevations off an elevation raster.

Generalization

- *Aggregate*: Generates a reduced resolution form of a raster. This tool is somewhat similar to resample, except that it adds cells together based on the factor you provide. If you provide a factor of three, the cell size in the output raster will be three times larger than in the input raster. The cell factor must be an integer greater than one (unlike with the resample tool, where you may resample the raster to any cell size you wish).
- *Boundary Clean*: Smooths the boundary between zones (different values of a raster) by expanding or shrinking it. This is helpful for making maps and smoothing small anomalies in data.
- *Expand*: Expands the specified zones of a raster by a specified number of cells. In other words, this is somewhat similar to the buffer tool used for features data.

- *Majority Filter*: Replaces the values of cells in a raster based on the values of the majority of their contiguous neighboring cells. This is again useful for smoothing out small inaccuracies, particularly likely to result if you have scanned in your own data.

- *Nibble*: Replaces cells of a raster corresponding to a mask with the values of their nearest neighbors. That is, this is similar to majority filter but you use a mask (another raster data set) to tell Arc which cells of the raster should have their values replaced.

- *Region Group*: Connected regions are groups of contiguous cells with the same values. The region group tool assigns a unique number to each region. For example, I used this tool to identify urban and rural areas. In the original data, urban areas were coded as one and rural areas as zero. The Region Group tool was used to give each urban area (connected series of cells with a value of 1) a unique identifier. I then used this identifier combined with a population raster and the zonal tools (described below) to calculate the total population of each urban area.

- *Shrink*: Shrinks the selected zones by a specified number of cells (i.e. the opposite of expand).

- *Thin*: Thins rasterized linear features by reducing the number of cells representing the width of the features. This is particularly useful when you've scanned in maps and the boundaries appear too thick.

Interpolation

The interpolation utilities in this toolbox are very similar to those in the 3D toolbox. A raster surface can be interpolated using the inverse distance weighting (IDW) technique, kriging, natural neighbors, or splines.

Math

- There are tools to perform a large number of mathematical operations on a raster (round up, round down, absolute value, log, negate, square root, trig functions, etc.)

- There are also useful logical operations that can be used to create a 0/1 raster: equal to, greater than, greater than or equal, less than, not equal, etc).

Overlay

- *Weighted Sum*: Overlays several rasters, multiplying each by their given weight and summing them together.

Raster Creation

- *Create Constant Raster*: Creates a constant raster with a specified extent. This can be useful as a base for creating your own raster data sets.
- *Create Normal Raster*: Creates a raster of random values with a normal distribution of cells.
- *Create Random Raster*: Creates a raster of random, floating point values between 0 and 1 on a cell-by-cell basis.

Neighborhood

- *Focal Statistics*: Calculates a statistic on a raster over a specified neighborhood (which can be a circle, rectangle, etc). The statistic can be the mean, min, max, standard deviation, sum, etc. This can be useful, for example, for creating a measure of high ground that gives how high one cell is in relation to the surrounding cells.

Reclass

- *Reclassify*: Reclassifies (or changes) the values in a raster.
- You can also reclassify data using an ASCII or .dbf table rather than entering the values directly into the command. This is particularly useful when you need to reclassify more than a few values.

Surface

- *Slope*: Identifies the rate of maximum change in z-value (height) from each cell.

A very nice application of the use of slope in applied economics research is Nathan Nunn and Diego Puga's paper "Ruggedness: The Blessing of Bad Geography in Africa," available at http://www.economics.harvard.edu/faculty/nunn/papers_nunn.

Zonal

- *Zonal Geometry as Table*: Calculates for each zone in a dataset the geometry measuresfor example, area, perimeter, thickness, and the characteristics of ellipseand reports the results as a table.
- *Zonal Statistics as Table*: Summarizes the values of a raster within the zones of another dataset and reports the results to a table.

- There are also tools called *Zonal Geometry* and *Zonal Statistics*, which perform the same calculations as the above two tools but report the results in raster format instead of in table format.

While the zonal statistics tools are straightforward to use, I will give a somewhat detailed example that illustrates a number of points touched on earlier in these notes.

The application involves calculating climatic averages within polygons, using various weighting schemes. (This application is based on methodology used in my papers on cross-country and sub-national climate with Ben Jones and Ben Olken, available at http://econ-www.mit.edu/grad/mdell/papers.) We will use three data sets in this example. The first is a climate raster data set (referred to here as the Matsuura and Willmot data, after the scientists who produced it). The Matsuura and Willmot data form a 0.5 degree x 0.5 degree grid. 0.5 degrees is approximately equal to 55 km at the equator, and half that at 60 degrees (to convert degrees to meters, multiply by the sine of the angle to the earth's center, which is the cosine of the latitude). That is, the Matsura Willmot data consists of a text file with a latitude coordinate, a longitude coordinate (both spaced at 0.5 degree intervals), and temperature and precipitation averages spaced at monthly intervals. Below is a picture of the Matsuura and Willmot data, overlayed on a first level administrative map of Guatemala. The points give the lat-lon coordinates of each grid cell centroid, and the grid gives the temperature value corresponding to each observation for May of 1955. The unit of measure is degrees.

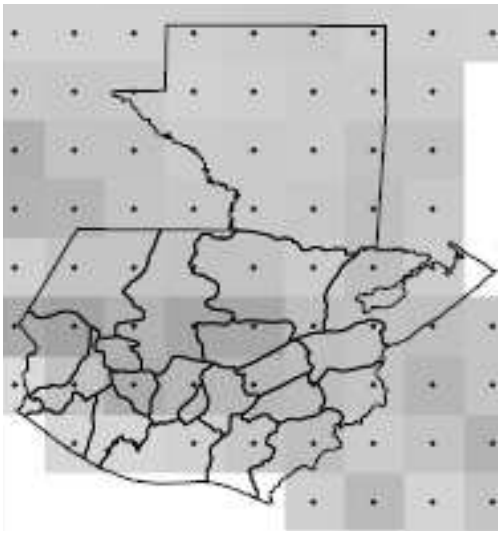


Figure 2.3: Climate Data

We will also use two thirty arc second (approx. 1 km) resolution population data sets, produced by the Center for International Earth Science Information. One gives the number of people living in each grid cell in 2000, and the other gives whether the grid cell is urban or rural.

First let us consider how to calculate the mean population weighted temperature within each of the "states" on the above map. There are two approaches that we can use. The first consists of the following steps:

1. Import the climate data into Arc using the Point to Raster tool.

2. Resample the population raster, originally at 30 arc second resolution, so that it has the same resolution as the climate raster (0.5 degrees).

3. Multiply the population raster by the climate raster using the Times function in the Spatial Analyst toolbox.

4. Use the Zonal Statistics as Table tool to calculate the sum of the population x climate raster and the sum of the population raster within each of the state polygons.

5. Use the Table Select tool to convert the Table Selects produced by Zonal Statistics to external .dbf files.

6. Use Stata's odbc capabilities to import the .dbf files into Stata and calculate the weighted temperature averages.

7. Repeat this process for every climate layer (i.e. you may have a layer for temperature in each month between 1950 and 2000, that is, 600 layers)

This process is very straightforward, but also will take a long time to run if you have a lot of climate data. If you have daily climate data for a large fraction of the world, for example, it may be completely computationally infeasible (i.e. it would take years to calculate everything). In this case, there is a much faster and still very straightforward alternative, which involves only creating one population weighting vector in Arc, rather than looping through the above process over and over for every climate layer. Here are the steps to follow for the second approach:

1. Do not import the climate data into Arc. Rather, write a dictionary to import it into Stata.

2. Create a features grid snapped to the same resolution as the climate data (I discuss how to do this below in the section on the Hawth's Tools add-on). That is, this features grid should have cells the same size as the climate data and should start at the same coordinates.

3. Intersect the polygon grid and the administrative boundaries using the Intersect tool.

4. Use zonal statistics to calculate the total population in each grid cell x state polygon (denoted pop_{gd}), export this data to a .dbf file, and then import it into Stata. Each temperature observation in Stata will be associated with a grid cell index g and each population sum from Arc will be associated with a grid cell x admin. district index $\{g, d\}$. The data can be merged in Stata on the index g. Then, the administrative district's population weighted average temperature can be calculated as:

$$ptem_d = \frac{\sum_g tem_g * pop_{gd}}{\sum_g pop_{gd}} \tag{2.2}$$

where tem_g is the temperature corresponding to each grid cell in our climate raw data.

Next, I discuss urban population weights. The construction of rural population weights is analogous. To construct these measures, we can use population extents data, in addition to the population and climate data sources described above. The extents data are a 30 arc second x 30 arc second raster grid where each cell is coded as urban or rural. To construct an urban raster, we utilize the following steps:

1. Code urban cells as 1 and rural cells as zero using the Reclassify tool.
2. Multiply the extents raster by the population raster using the Raster Times tool - this produces an urban population raster at 30 arc second resolution.
3. Then follow the same steps as above, whether using the first or second approach.

That is, if $upop_g$ is the aggregate urban population in each grid cell and uwt_{gd} is the percentage of each grid cell's urban population that pertains to administrative district d, the urban weighted temperature for each district is:

$$utem_d = \frac{\sum_g tem_g * upop_g * uwt_{gd}}{\sum_g upop_g * uwt_{gd}} \tag{2.3}$$

Finally, I discuss weighting by landmass area. Note that it makes no sense to think about area weighted measures in the above system of measurement, where the unit is degrees. Landmass area is a concept measured in square kilometers, and kilometers are not constant as you move across the surface of a map denominated in degrees.

Thus, the first step is to project the administrative boundaries and the climate data. This introduces two forms of distortion - the first is well-known, the surface of a sphere cannot be perfectly squashed onto a flat surface. The second form of distortion results because the raster grid must be resampled to form a grid in meters (as opposed

to a grid in degrees). The first type of distortion can be minimized by choosing the appropriate projection for the task at hand. As discussed above, some projections lead to minimal distortion in the area dimension (the relevant type for area-weighted calculations), others minimize distortion in the distance dimension, others in the angular dimension, and yet others in the shape dimension. UTM projections can be used when looking at a small slice of the earth's surface - such as Guatemala - to minimize distortion along all dimensions.

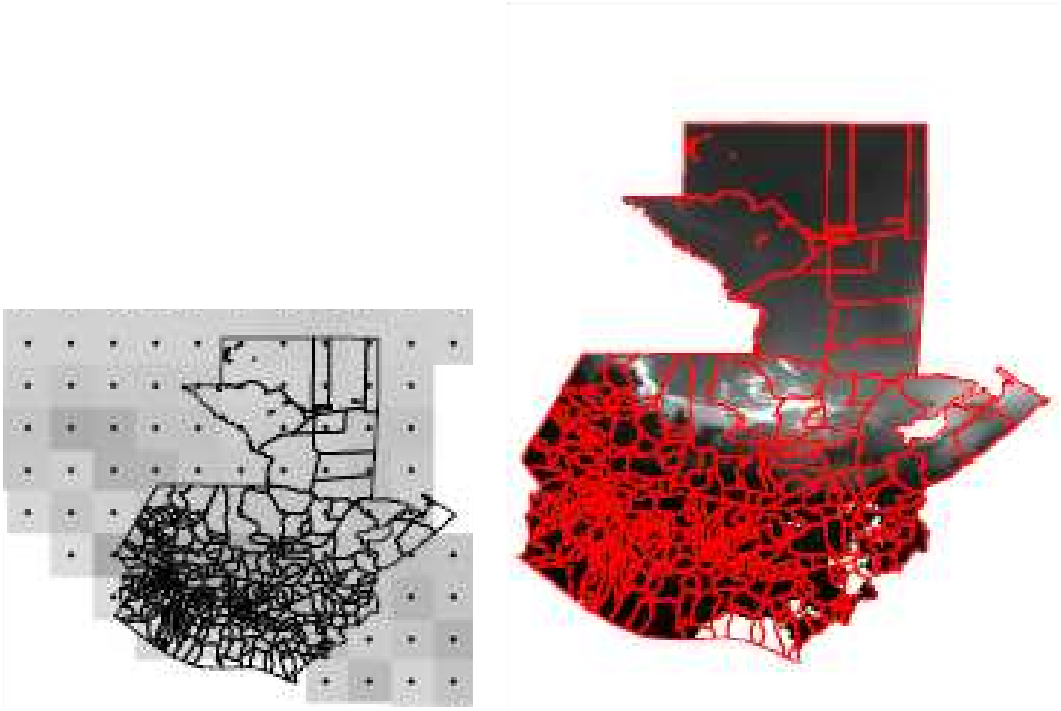
There are a couple of details that must be decided when projecting the rasters. First, when examining ordinal data such as the urban extents data discussed above, it's important to use the count interpolation option when projecting the data. If you use the bilinear interpolation option, which treats the data as continuous, the projected output raster will no longer be 0/1. On the other hand, when projecting the climate data (which is continuous), the bilinear interpolation option is most appropriate. Second, you must choose a cell size (in meters) for the projected rasters. Note that our original climate and population data, which come in a geographic coordinate system, are not spaced evenly across the earth's surface: at the Equator, points in our 0.5 degree resolution climate data are spaced at approximately 55 kilometers whereas at Montreal, at 45 degrees, they are spaced at around 39 kilometers ($55 * \cos 45$). Thus, it is not obvious what to set the cell size at in the projected data - choosing a resolution that is too low throws away information, and choosing one that is too high unnecessarily increases processing time. I choose the grid cells in the projected climate raster to be 35 kilometers x 35 kilometers, since this is about the resolution of the data in Northern Europe, where there is a high density of climate stations.

Once we have projected the boundaries data and the climate data to a projected coordinate system, we can produce a map that looks very similar to the one above, except that the unit of measure is in meters. Again, either of the two approaches above can be used to calculate the landmass weighted temperature. Note, however, that there is no way to bypass importing the climate data into Arc since it has to be projected to a system of measure where the units are in meters.

Now, I touch briefly to an additional useful observation. The Matsuura and Willmot data may work okay for state or country level means. However, suppose we wish to look at municipality means, as for example in Dell, Jones, and Olken (forthcoming), where we look at sub-national climate. The figure below shows that the resolution of the Matsuura and Willmot data is not appropriate for this purpose.

Consider instead the WorldClim Version 4 high resolution climate data, developed by climatologists at Berkeley, which consists of a 30 arc second grid (giving averages for the entire 1950-2000 period, rather than monthly observations as in Matsura-Wilmott). As can be seen below, the WorldClim resolution is much more appropriate.

Finally, note that the above style of analysis can be useful in a number of economic applications that do not directly deal with geographic phenomenon. For example,



(a) Low Resolution (b) High Resolution

Figure 2.4: The Importance of Resolution

Daron Acemoglu and I (forthcoming) used GIS population and boundaries data to calculate population weights for all municipalities in the Americas, which was much more feasible than obtaining this data from various non-GIS census data sets.

2.7 Geocoding Tools

Geocoding is the process of assigning a location, usually in the form of coordinate values, to an address. Addresses come in many forms, ranging from the common address format of house number followed by the street name, etc. to location descriptions such as postal zones or census tracts. Many common tasks are related to the geocoding process, such as the creation, maintenance, and deletion of address databases, as well as the actual matching of addresses to geographic features classes. Below I give a very brief description of Arc's address tools, as a detailed description is beyond the scope of these notes. Much more information can be found in the Arc help files, by typing in the search phrase "geocoding."

- *Create Address Locator*: One of the first processes in geocoding is the creation and tailoring of an address locator. The address locator is the main tool for

geocoding in ArcGIS. It is a dataset that contains information including address attributes, indexes, and queries for geocoding. This process always begins in ArcCatalog. Address locators can be created in any workspace such as a geodatabases or file folder. Right-click a selected workspace, click New, then click Address Locator. An address locator contains a combination of location-specific attributes and certain style-specific guidelines based on the address locator style selected. The address locator style defines what reference data can be used in creating an address locator. For instance, you may use the US Streets with Zone address locator style for street centerline data containing left and right address ranges, directionals, street names, street types, and left and right ZIP Codes. If you want to geocode U.S. ZIP Code data, you may use the 5-Digit ZIP address locator style for creating a ZIP Code address locator based on a ZIP Code point reference feature class. (See the Arc help files for many more details.) An address locator also contains information of how an address is standardized, searching methods for possible matches, and what output information of a match would be returned in a search.

Reference data can have many different characteristics. Each address locator requires at least a primary reference dataset (i.e. coordinates, U.S. street addresses or zip codes, etc.). In addition to the primary reference data, ancillary data can also be attached, again using the Arc Catalogue interface. Tables containing place names or aliases of particular features can be added as reference data. This allows you to search for locations based on the name of the location rather than the street address.

After an address locator is built, addresses can be matched to geospatial references using the address locator. The reference data will be disconnected from the address locator. If the reference data is modified, you will need to rebuild the address locator to refresh the information (more on this below).

- *Geocdoe Addresses*: This tool uses an input table of addresses in the address locator to create a shapefile or geodatabase feature class. For example, the input table may consist of a list of university addresses, and this tool can be used to create a map of the universities' geographic locations.

- *Rebuild Address Locator*: When you modify your address database (i.e. by adding, deleting, or changing entries in the original .dbf files used to create the address locator), the address locators will not be automatically updated with the changes. An address locator is a snapshot of the reference data, and it becomes independent from the reference data once it is created. That is, updating your reference data will not automatically update the address locator, which is the tool used to merge the address data with geospatial data. If you want to refresh the locator with the changes you make in the reference data, you need to create a new locator or rebuild the existing one. To rebuild an

address locator after updating the underlying reference data, follow these three easy steps:

1. Select the address locator you want to rebuild in ArcCatalog.
2. Right-click the locator and click Rebuild.
3. Click OK on the Rebuild Address Locator dialog box.

- *Rematch Addresses*: When addresses from a table have been geocoded into a point feature class there are sometimes addresses left over that could not automatically be matched. You can revise and then rematch addresses that could not be automatically geocoded. See the Arc help files or online tutorials for details.

- *Standardize Addresses*: When you create an address locator, the address fields in the reference data that you use for the locator should be standardized according to the geocoding rules used by the address locator. Doing so will increase the geocoding match rate that you can achieve with the address locator. This is done using the geocoding rule base. The geocoding rule base is a collection of files used to standardize the address data into the desired format in order to prepare the address for matching. For each address locator style, there are specific files in the rule base used in the process of standardization and matching. Further details are beyond the scope of these notes. For those seriously interested in working with street addresses in Arc, I recommend the following book on geocoding rule base development, available online: http://webhelp.esri.com/arcgisdesktop/9.2/pdf/Geocoding_Rule_Base_Developer_Guide.pdf

2.8 Hawth's Tools

This is a particularly useful ArcGIS add-on, and can be downloaded (free of charge) at http://www.spatialecology.com/htools/. You can read about Hawth's tools in the documentation available at this site. The main reason why I've found this package to be essential is the *Create Vector Grid* tool, which allows you to create features grids (as mentioned in the climate data application above). It also contains a number of other useful applications.

2.9 Additional Scripting Utilities

I will be brief here. More useful information can be found in the Arc help files.

First, you may not want your script to crash when it encounters an error. This can be achieved by embedding your script in the following code:

```
try:
       insert commands here
except:
       # If an error occurred while running a tool, then print the messages.
       print gp.GetMessages()
```

Second, you may wish to use values in a loop that are listed in a separate .dbf file. For example, in research on road networks in Latin America, I needed to connect each city in a long list of cities to the ten closest cities, using the least cost path tool. I had previously calculated which cities were the ten closest for every city in my database and stored the appropriate nearby city id numbers for each source city in a .dbf file. These id numbers matched the id numbers in a series of city shapefiles that were to serve as the destinations in the least cost path calculations. I obviously did not want to enter similar code thousands of times for each predicted path, but rather needed to use a loop that read the appropriate cities to build roads to for each source from a separate .dbf file.

The utility used to do this is the *Create search cursor* tool. A search cursor is a utility called from a script file that will extract values from a .dbf file row by row or column by column, and in turn use them as inputs in the python code file. A detailed description of the search cursor is beyond the scope of these notes, but more information can be found by entering the term "search cursor" in the Arc help files search menu.

Finally, when working with various data layers that have different extents, you may wish to set the extent of the output data to be the intersection or union of the inputs. To set the extent to be the union, place the following command at the beginning of your python code file:

```
# Set Extent
gp.Extent = "MAXOF"
```

Replace "MAXOF" by "MINOF" in the above command to set the extent of the output data to be the intersection of the inputs.

Chapter 3

Additional Topics

This chapter covers several additional important topics. Section 3.1 discusses spatial correlation, Section 3.2 outlines Arc's map-making capabilities, and Section 3.3 contains a brief list of additional GIS resources.

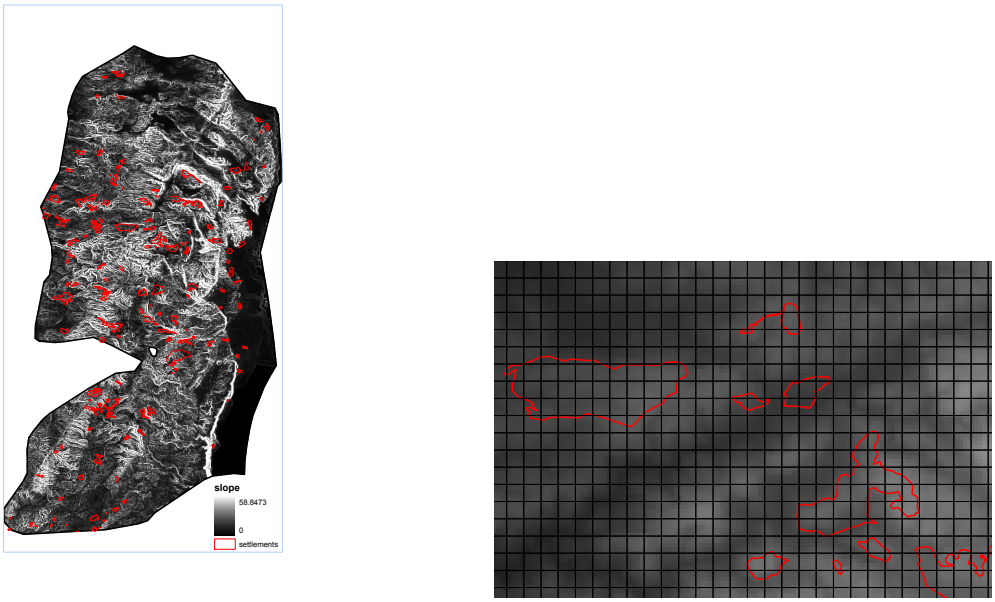
3.1 Spatial correlation in GIS analysis

Disclaimer: This section is not meant to instruct you on which estimators you should use to correct for spatial correlation. Rather, the aim is to emphasize that spatial correlation is an essential feature of geospatial data that should not be ignored.

Below I show two maps. The red boundaries encircle settled areas. Suppose that I was interested in comparing slope (calculated using the Slope utility) in grid cells outside settlements to slope in grid cells inside settlements. Let's consider starting with a large grid cell (say 10 square kilometers), and then re-snapping the grid, so that the cells become smaller and smaller ... Hence the number of observations becomes larger and larger... You can guess what is going to happen if we use normal robust standard errors that treat each grid cell as an independent observation.

While this might seem like an artificial example, it is representative of a general feature of spatial data. It makes no sense to think of elevation along one side of a boundary versus the other as iid. There is significant spatial correlation between the observations in this example, and the standard errors should account for it.

The best resource that I've found for starting to learn about spatial correlation is Tim Conley's (Chicago GSB) website: http://-faculty.chicagogsb.edu/timothy.conley/research/. A number of recent papers have used standard errors developed by Conley (1999) to correct for spatial correlation (i.e. Michael Greenstone and Olivier Deschenes' recent AER paper



(a) Slopes (b) Grid Cells

Figure 3.1: Spatial Correlation in Geographic Data

"The Economic Impacts of Climate Change: Evidence from Agricultural Output and Random Fluctuations in Weather"). Conley (1999) errors allow for spatial dependence of an unknown form. Specifically, the Conley covariance matrix is a weighted average of spatial autocovariances, where the weights are the product of Bartlett kernels in two dimensions (North/South and East/West). They start at one and decline linearly to zero when a pre-specified cut point is reached. Stata code for calculating the Conley standard errors can be downloaded from Tim Conley's website. The reference is:

Conley, Timothy. 1999. "GMM Estimation with Cross Sectional Dependence." Journal of Econometrics, 92(1): 1-45.

There have been a number of other estimators suggested to correct for spatial correlation, and this is a frontier area of econometrics.

3.2 Making maps in ArcGIS

Making a map in Arc is as much art as science, and my treatment here is brief. The best way to make attractive looking maps is to play with Arc yourself. Remember that many people may print your paper in black and white, so the map should be legible in black and white as well as in color.

It is often necessary when making maps to switch back and forth between data and layout view (using the view drop down menu). You can edit data from data view and see what your map will look like from layout view.

From edit view, right click anywhere on your map and click data frame properties. From the Data Frame tab, you can fix the extent of your map. The Coordinate System tab allows you to choose the most appropriate coordinate system. The Frame tab can be used to put a border around your map. You can also set the background color for your map using this tab.

The leftmost window in Arc displays your data layers. You can double click on the layer names to give them the names that you wish to appear in the map legend. You can right click on the layer names and then click properties to change the layer colors, symbols, etc as desired.

A frequent application is to merge data contained in a Stata or Excel file with polygons in a map (i.e. you may have a map of U.S. counties and a data file listing each county's population from the 2000 Census and wish to show the population data visually on the map). This is straightforward, and requires the following steps:

1. Create a single identifier that will be used to match the GIS boundaries with the separate data file.

2. Convert your data file to .dbf format (this can be done in Excel, using Stata's odbc capabilities, or in Stat Transfer).

3. Load the shapefile and the .dbf file that you want to merge it with into Arc.

4. Right click on the shapefile layer in the leftmost window, and then click "Joins and Relates" -> "Join". Choose the field in the shapefile and .dbf data file that was created for merging the data and click OK.

5. Right click on properties, and click on the tab labeled "Symbology." This will allow you to choose a classification and color scheme for displaying the merged data.

If you do not like Arc's preloaded color schemes, you can mix your own color palettes by clicking "Tools" -> "Styles" from the main menu.

Once the map itself has been created, you can use the "Insert" tab in the main menu to insert a map title, a textbox, and a legend.

Finally, you can export your map to a number of file types by clicking "file"->"export map". .eps is a useful format if you plan to insert the file into a word or latex document. .pdf is useful for working with pdf latex.

3.3 Useful GIS data resources

MIT geospatial library: http://libraries.mit.edu/gis/

MIT GIS coordinator: Daniel Sheehan dsheehan@mit.edu

Harvard geospatial library: http://peters.hul.harvard.edu:8080/HGL/jsp/HGL.jsp

Wiki Guide to Python: http://en.wikibooks.org/wiki/Non-Programmer's-_Tutorial_for_Python/Contents

Lecture slides on using ArcGIS with python: http://nrm.salrm.uaf.edu/ dverbyla/nrm638/lectures/

Harvard University's introduction to GIS tutorials: http://hcl.harvard.edu-/libraries/maps/gis/tutorials.html

90 meter resolution elevation data: http://srtm.csi.cgiar.org/

Examples of Python Codes for GIS Processing

Typical Header Material for a Python Code File

```
# Import system modules (This imports the modules used later in the python file. To import arcgisscripting, you need to have Arc installed on your machine.)
import sys, string, os, arcgisscripting, shutil
print "system modules imported"

# Create the Geoprocessor object
gp = arcgisscripting.create()
print "created geoprocessor object"

# Set the necessary product code
gp.SetProduct("ArcInfo")
print "set product code"

# Check out any necessary licenses (This checks out whether a license is installed for Spatial Analyst and 3D Analyst, two common Arc add-on modules that MIT has licenses for.)
gp.CheckOutExtension("spatial")
gp.CheckOutExtension("3D")
print "checked extensions"

# Load required toolboxes (This loads the toolboxes used in the following file – spatial analyst, data management, 3D analyst, and analysis).
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Spatial Analyst Tools.tbx")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/3D Analyst Tools.tbx")
gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Analysis Tools.tbx")
print "loaded required toolboxes"

#Turn overwrite on
gp.overwriteoutput = 1
print "overwrite on"

#Set directory path
gp.workspace = "C:\\mdell\\mita\\data"
print "directroy set"
```

Projection Commands

```
# Process: Project ... (Distritos – This command projects a features layer, from WGS 1984 to UTM Zone 18 South, and makes it evident why I recommend using the model builder to get these commands!)
distritos_shp="distritos.shp"
distritos_eq="distritos_eq.shp"
gp.Project_management(distritos_shp, distritos_eq,
"PROJCS['mita_equi',GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],PROJECTION['Equidistant_Cylindrical'],PARAMETER['False_Easting',0.0],PARAMETER['False_Northing',0.0],PARAMETER['Central_Meridian',-73.0],PARAMETER['Standard_Parallel_1',-13.0],UNIT['Meter',1.0]]", "",
"PROJCS['WGS_1984_UTM_Zone_18S',GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],PROJECTION['Transverse_Mercator'],PARAMETER['False_Easting',500000.0],PARAMETER['False_Northing',10000000.0],PARAMETER['Central_Meridian',-75.0],PARAMETER['Scale_Factor',0.9996],PARAMETER['Latitude_Of_Origin',0.0],UNIT['Meter',1.0]]")
```

```
# Process: Project ... (Elevation – this command projects a raster)
elev_eq = "elev_eq"
so_am_land = "so_am_land"
gp.ProjectRaster_management(so_am_land, elev_eq,
"PROJCS['mita_equi',GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,
298.257223563]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],PROJECTION['Equidistant_C
ylindrical'],PARAMETER['False_Easting',0.0],PARAMETER['False_Northing',0.0],PARAMETER['Central_Merid
ian',-73.0],PARAMETER['Standard_Parallel_1',-13.0],UNIT['Meter',1.0]];-19524100 -10018900
230668744.135222;#;#;0.001;#;#;IsHighPrecision", "BILINEAR", "915.800179", "", "",
"GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],PR
IMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]];-400 -400
11258999068426.2;#;#;8.98315284119521E-09;#;#;IsHighPrecision")
print "projected elevation raster to UTM Zone 18S"
```

3D analysis tools

Functional Surface

```
# Process: Surface Length (This calculates length correcting for changes in elevation. The first is the input file, and
the second is the features file, which contains roads data.)
elev_eq = "elev_eq"
gp.SurfaceLength_3d(elev_eq, vec_intg_shp, "SLength", "", "1")
print "calculated length of local roads"
```

Raster Math

```
# Process: Float  (first filename is the input file and the second is the output file)
elev_fl = "elev_fl"
gp.Float_3d(elev_msk, elev_fl)
print "converted elevation data from integer to float"

# Process: Divide (if you wish to calculate x divided by y equals z, the first file is x, the second y, and the third z)
elev_sh = "elev_sh"
gp.Divide_3d(elev_fl, constant_value, elev_sh)
print "divided elevation data by 1000"

# Process: Times (the first two files are input files, the last is the output file)
popelev = "popelev"
gp.Times_3d(pop_sh, elev_sh, popelev)
print "multiplied elevation and population rasters"
```

Raster Reclass

```
# Process: Reclassify... (i.e. convert area above 4800m from 0 to system missing; first file name is the input file, the
second is the output file)
inhab_rc = "inhab_rc"
gp.Reclassify_3d(inhab, "VALUE", "0 NODATA;1 1", inhab_rc, "DATA")
print "areas>4800 now system missing"
```

Analysis tools

Extract

# Process: Table Select (Exports data from an internal tableview to an external .dbf file. The first input is the name of the tableview and the second is the name of the .dbf file.)
popelev_dbf = "popelev.dbf"
gp.TableSelect_analysis(sumpopelev, popelev_dbf, "")
print "exported to .dbf file"

# Process: Select (The first file is the input file, the second is the output file, and the third piece of code selects portions of the input filing using a query.)
roads_eq_shp = "E:\\Melissa\\mdell\\gisdata\\roads_eq.shp"
proad_shp = "proad.shp"
gp.Select_analysis(roads_eq_shp, proad_shp, "\"SUPERFICIE\" =1")

# Process: Clip (Clips features files. The first file in the command is the input file, the second file is the one used as a “cookie-cutter”, and the third file is the clipped output file)
streg_poly = "sreg_poly.shp"
centeq_cl = "centeq_cl.shp"
gp.Clip_analysis(cent_sr, streg_poly, centeq_cl, "")
print "clipped centroids to study region extent"

Overlay

# Process: Intersect (Intersects two features files. The first two filenames are the features to be intersected (you can intersect more than two features) and the last filename is the name of the output file.)
inmita_poly_shp = "inmita_poly.shp"
inmita_centroids = "cent_inmita.shp"
gp.Intersect_analysis("centeq_cl.shp #;inmita_poly.shp #", inmita_centroids, "ALL", "", "INPUT")
print "intersected centroids with mita extent data"

# Process: Erase (Note that you should erase water bodies before calculating slopes. The first file is the input file, the second is the features class that you are using as an eraser, and the third is the output file.)
pnwater_shp = "pnw.shp"
peru_eq_shp = "peru_eq.shp"
water_eq_shp = "water_eq.shp"
gp.Erase_analysis(peru_eq_shp, water_eq_shp, pnwater_shp, "")
print "created a mask with water bodies as no data"

Proximity

# Process: Buffer (The first is the input file, the second is the output file, then I specify the length of the buffer, the shape of the end of the buffer, and that I wish to buffer the entire feature.)
streams_buf_shp = "streams_buf.shp"
gp.Buffer_analysis(streams_cl_shp, streams_buf_shp, "1500 Meters", "FULL", "ROUND", "ALL", "")
print "create 2 km buffer around streams"

# Process: Near (Calculates the Euclidean distance between points and a feature (in this example, a boundary). The first input lists the points that we are calculating distance from, the second gives the feature that we are calculating distance to, the third tells Arc that I want to know the coordinates of the closest point on the feature, and the fourth tells Arc that I don’t want to know the angle to the nearest point on the feature.)
ph_bnd = "ph_bnd.shp"

```
gp.Near_analysis(centeq_cl, ph_bnd, "100000 Meters", "LOCATION", "NO_ANGLE")
print "calculated distance to mita boundary"
```

Data management tools

Fields

```
# Process: Delete Field (When you perform calculations within Arc, sometimes it may produce a lot of fields that you don't want. You can delete them from the tableview – before exporting the table to a .dbf file – using this command. The capital letters specify the fields to be deleted.)
gp.DeleteField_management(sumpopelev, "MEAN;COUNT;AREA;MIN;MAX;RANGE;STD")
print "unneeded fields deleted"
```

Features

```
# Process: Feature To Point (This is useful for creating centroids. The first file is a grid file, and the second is the output file, which will contain the centroids of the grid. The Inside option tells Arc that I want the centroids to fall inside the grid cells. This is obviously not an issue with a grid, but for some features, the centroids may fall outside the feature. This option can limit them to fall inside.)
cent_sr_eq_shp = "cent_sr_eq.shp"
gp.FeatureToPoint_management(grid_equi_shp, cent_sr_eq_shp, "INSIDE")
print "created grid centroids"
```

Generalization

```
# Process: Dissolve (I use this command to turn a map with boundaries for all the districts in Peru into a map with only the outer (international) Peruvian boundaries – i.e. I dissolve all the other boundaries. The first file is the input file and the second is the output file.)
Peru_shp="temp.shp"
gp.Dissolve_management(distritos_eq, Peru_shp, "", "", "MULTI_PART")
print "created a Peru polygon"
```

Raster

```
# Process: Resample (This command changes the resolution of the raster to 0.5, using the bilinear option to resample the raster. The first is the input file and the second is the output file.)
rpoprs = "rpoprs"
gp.Resample_management(ruralXpop, rpoprs, "0.5", "BILINEAR")
print "resampled"
```

Conversion tools

```
# Process: Point to Raster (Converts a point dataset to a raster dataset. The first file name is the input file, the second input defines the field that will provide the raster values, the third is the raster output file, 0.50 is the size of the raster.)
clim_mask="climate_mask"
climdata = "XY070728pre50s.shp"
value_field = "V150"
gp.PointToRaster_conversion(climdata, value_field, clim_mask, "MEAN", "NONE", "0.50")
```

Spatial analyst tools

Extraction

```
# Process: Extract by Mask (The first file is the input file, the second is the file to be used as a cookie-cutter, and te third is the output file.)
elev_eq = "elev_eq"
elev_msk = "elev_msk"
gp.ExtractByMask_sa(elev_eq, Peru_shp, elev_msk)
print "selected elevation data only for Peru"
```

Zonal

```
# Process: Zonal Statistics as Table (Calculates averages within polygons (zones). The first file is the polygon file, the GRID_ID option specifies to calculate averages within that field, the next file is the raster, and the final file is the output tableview name.)
grid_equi_shp = "grid_equi.shp"
sumpopelev = "sumpopelev"
gp.ZonalStatisticsAsTable_sa(grid_equi_shp, "GRID_ID", popelev, sumpopelev, "DATA")
print "calculated sum of popXelev in each grid cell"
```

Math

```
# Process: Equal To (create raster where urban==0, rural==1, before it was rural==1 urban==2 no_data (i.e. oceans)==0)...
extents_eq="extents_eq"
rural_eq="rural_eq"
constant_value_1 = "1"
gp.EqualTo_sa(extents_eq, constant_value_1, rural_eq)
print "0/1 rural aster created"

# Process: Less Than (first file is the input file, second name defines the constant value which we are comparing the input raster to, and the third value is the output file.)
inhab = "inhab"
constant_value_2 = "4.8"
gp.LessThan_sa(elev_sh, constant_value_2, inhab)
print "created raster==1 if area<4800 msnm"
```

Distance

```
# Process: Euclidean Distance (Creates a raster giving Euclidean distance to the feature.)
dproad = "dproad"
pop_eq = "E:\\Melissa\\mdell\\gisdata\\pop_eq"
Output_direction_raster = ""
gp.EucDistance_sa(proad_shp, dproad, "", "pop_eq", Output_direction_raster)
```

Surface

```
# Process: Slope (Calculates slope in degrees. 1 specifies that we don’t need to convert the Z units since they are already given in meters. The first filename is the input file, and the second filename is the output file. Don’t forget to erase water bodies from your data before calculating slope.)
slope = "slope_nw"
```

```
gp.Slope_sa(elev_nolake, slope, "DEGREE", "1")
print "calculated slope"
```

Useful Stata Command

The following is an example of how the odbc load command can be used to read a bunch of small .dbf files created in Arc into stata.

```
*Import data from dbf files - 1950s
foreach M of num 1/12 {
        foreach Y of num 50/59 {
                odbc load FIPS_CNTRY SUM, table("mt`M'`Y'.dbf") dsn("dBASE Files")
                        lowercase
                sort fips_cntry
                save C:\Melissa\t_weighted\t`M'`Y', replace
                clear
        }
}
```