Arc_Mat, a Matlab toolbox for using ArcView Shape files for spatial econometrics and statistics

James P. LeSage[1]
University of Toledo
Department of Economics
Toledo, OH 43606
jlesage@spatial-econometrics.com
and
R. Kelley Pace
LREC Endowed Chair of Real Estate
Department of Finance
E.J. Ourso College of Business Administration
Louisiana State University
Baton Rouge, LA 70803-6308
OFF: (225)-388-6256, FAX: (225)-334-1227
kelley@pace.am, www.spatial-statistics.com

April 8, 2004

[1]The first author would like to acknowledge support from the National Science Foundation, BCS-0136229 and the second author acknowledges National Science Foundation support from BCS-0136193

Abstract

The design and implementation of software for extracting information from GIS files to a format appropriate for use in a spatial modeling software environment is described. This has resulted in publicly available c/c++ language programs for extracting polygons as well as database information from ArcView shape files into the Matlab software environment. In addition, a set of publicly available mapping functions that employ a graphical user interface (GUI) within Matlab are described. Particular attention is given to the interplay between spatial econometric/statistical modeling and the use of GIS information as well as mapping functions. In a recent survey of the interplay between GIS and regional modeling, Goodchild and Haining (2003) indicate the need for a convergence of these two dimensions of spatial modeling in regional science. Many of the design considerations discussed here would also apply to implementing similar functionality in other software environments for spatial statistical modeling such as R/Splus or Gauss.

Toolboxes are the name given by the *MathWorks* to related sets of Matlab functions aimed at solving a particular class of problems. Toolboxes of functions useful in signal processing, optimization, statistics, finance and a host of other areas are available from the *MathWorks* as add-ons to the standard Matlab software distribution. We label the set of functions described here for extracting GIS file information as well as the GUI mapping functions the *Arc_Mat Toolbox*.

1 Introduction

Recent developments in computer gaming have lead to powerful computer graphics hardware becoming part of basic laptop and desktop computers. This graphics hardware can draw thousands of polygons on computer screens in a fraction of a second, allowing software that reads thousands of map polygons from *ESRI/ArcView* map file format known as *Shape Files* into the Matlab software environment for graphical presentation. An underlying public domain library of c/c++ language Application Programming Interface (API) functions known as *shapelib* was used along with the Matlab c/c++ language interface labelled *C-MEX* by the MathWorks. The *shapelib* functions include one API for processing, manipulating and extracting information from ESRI/ArcView map format files known as *Shape Files*, and another API for processing, manipulating and extracting database information from associated files. The associated data information contains sample data observations for each region or polygon area on the map, that can be used in spatial econometric modeling.

We focus on design and implementation considerations that arose in producing *Arc_Mat* for the Matlab environment, but many of these issues would also apply to implementing similar functionality in other software environments for spatial statistical modeling such as R/Splus or Gauss. Most statistical software environments include functions for plotting polygons that would allow the underlying map polygons from ArcView mapping software to be presented within the statistical software environment, as we have done for the case of Matlab. This results in presentation quality within the statistical software environment that is nearly identical to that of the ArcView software mapping environment. An important advantage of operating in the statistical software environment is that powerful spatial econometric modeling tools such as those contained in LeSage's public domain *Spatial Econometrics Toolbox* or Pace's *Spatial Statistics Toolbox* in the case of Matlab, or Bivand's *spdep* for the R/Splus environment, and public domain matrix programming environments such as SciLab and Octave.

An alternative to the design approach taken here is the one utilized by

GeoDA (Anselin et. al, 2002), which is freestanding and does not require a specific GIS system. GeoDa runs under the Microsoft Windows operating system. GeoDa is an interactive environment that combines maps with statistical graphics, using dynamically linked windows. GeoDa adheres to ESRIs shape file as the standard for storing spatial information. It uses ESRIs MapObjects LT2 technology for spatial data access, mapping and querying and provides functionality through the use C++ classes with associated methods. Use of C++ programs for for spatial statistical modeling eliminates reliance on software such as Matlab or R/Splus, but requires that users learn a new statistical software interface. Reliance on software such as Matlab or R/Splus has the advantage that thousands of user-contributed functions for statistical and spatial statistical modeling already exist for this environment. In contrast, the approach taken by GeoDa requires that users rely entirely on the set of statistical functions selected by developers for inclusion in the new environment, or develop their own C++ modeling functions within the new environment. GeoDa currently does not yet contain specific techniques to analyze geostatistical or point pattern data. A technical review of the design and architecture of the software is detailed in Anselin et al. (2002).

Yet another approach would be to utilize operating-specific functionality for linking programs through DLL's in the case of the Windows operating environment, or other such functionality that would allow use of one program as a compute engine for statistical calculations and ESRI's ArcView software as a graphics engine for map presentation. The drawback to this approach would be that design and implementation would be specific to an operating system and perhaps a hardware platform. Another difference between this approach and the one described here is that development of compiled language code would represent the majority of the system implementing this type of approach. In contrast, the majority of the *Arc_Mat Toolbox* consists of Matlab programs that can be developed and debugged in an interactive programming environment, avoiding the edit-test-debug development loop common to compiled language program development.

Part of this paper focuses on a set of Matlab development tools and

support functions for *Graphical User Interface* (GUI) program development, that was used to produce an interface to the underlying Matlab mapping functions. While this is specific to the Matlab environment, similar GUI functionality exists in the other statistical software mentioned above. Our implementation calls a single function using the command-line which creates a GUI interface. This interface presents the map as well as pull-down menus and other GUI tools for changing map properties such as: the color scheme, legend, variables mapped, as well as zooming in and out on sub-regions of the map. Readers familiar with the ArcView program will recognize that this same type of functionality is available in ESRI's ArcView software environment.

The ability to provide mapping support for spatial econometric and spatial statistical analysis in a single software environment avoids the need to transfer files between different software programs. Practitioners involved in development of spatial econometric models can examine model results such as residuals, predicted values, observations identified as candidate outliers, etc., using a single function call that transfers matrices or vectors containing model results to the GUI mapping environment for visualization.

Another aspect of the tools described here is that they can be used to extract GIS information for use in other mapping software that presently exists for the Matlab environment. An extensive set of functions labelled *GeoXP* described in (Heba Malin, Thomas-Agnan, 2002) has been developed for both Matlab and R/Splus software environments. The polygon coordinates produced by the c/c++ language interface of Matlab described here can be used to provide support for this library of functions.

Section 2 of the paper describes software issues related to the underyling file-transfer mechanism used to extract both mapping and database information from the *ArcView* files. The Matlab mapping software functions and issues pertaining to design of the GUI interface are set forth in section 3. Section 4 discusses usage issues related to the mapping functions and spatial econometric analysis. Focus is on situations where information from the ArcView shape files is useful in spatial econometric and spatial statistical modeling.

2 Underlying algorithms and design for extracting and importing shape file information into Matlab

The Matlab programming environment consists primarily of a command-line interface, where commands input using the keyboard or files containing command instructions are processed by a just-in-time compiler. As the name suggests, the programming environment is based on matrix and vector constructs. User-written functions represent a key feature of the software environment, and these can be written in the Matlab programming language as well as external languages such as c, c++, or FORTRAN. Other statistical software environments such as R/Splus and Gauss operate in a similar fashion and provide an external language interface.

External language programs must be compiled and linked to the Matlab program execution environment using an interface gateway known as C-MEX or F-MEX, in the case of c/c++ or FORTRAN languages. C-MEX files are written in the c/c++ language, and in the case of our file-interface to the ArcView *Shape Files* they represent an intermediate file that translates input arising from a function call executed within Matlab into appropriate calls to a public domain library of functions known as **shapelib**. This library provides a series of functions for reading, writing and extracting information from ArcView shape files. The term shape file is typically used to refer to a number of files that are generated by ArcView mapping projects, leading us to use the plural term shape files. These files can contain information on sample data stored in a database (*dbf*) file format, map projections, map polygons, and a host of other things related to ArcView projects. The **shapelib** functions provide an API for manipulating, reading and writing both mapping and database information from the ArcView formatted files.

We rely on a single Matlab function written in the native programming language that is named **shape_read** and takes a single argument containing a string with the basename of the shape file to be imported into Matlab. As already noted, the term 'shape file' is a misnomer since this is a reference to at least three different files with the same basename with different filename

extensions, .shp, .shx and .dbf. There are other possible files such as those containing map projection information that have a .prj extension, that are not currently used in our implementation.

The **shape_read** function calls underlying C-MEX functions that call the **shapelib** API functions to: 1) read polygon information, and 2) extract database information. In addition to functionality for extracting information, the **shapelib** API also contains functions for performing calculations using information from the ArcView formatted files and writing new ahape files that contain transformed polygon and database information. We use API functionality to calculate polygon centroid coordinates, which represent a key variable used to construct spatial weight matrices in spatial modeling.

The single Matlab function carries out error checking regarding file availability and combines all information returned by calls to different complied API functions into a single Matlab 'structure' variable. Structure variables represent a container with 'fields' that can consist of strings, vectors, scalars and matrices of varying dimensions and variable types. This provides a convenient way to return large amounts of heterogenous information from function calls as well as between functions. Individual fields in the structure variable are referenced using a '.' and the field name. An example of using the **shape_read** function would be:

```
filename = 'africa';
results = shape\_read(filename);
npolygons = results.npoly;
```

The structure variable 'results' contains information that is available through '.' referencing the field names, as illustrated by transferring the scalar field '.npoly' indicating the number of polygons in the 'africa' shape file to another scalar variable named 'npolygons'. Documentation for the **shape_read** function provides complete information on fields contained in the 'results' structure variable. This can be viewed in the Matlab command window environment by typing: 'help shape_read', which returns the information shown below.

```
PURPOSE: reads an arcview shapfile and returns a structure
         that can be used by make_map(), arc_histmap(), moran_map()
         and other Arc_MAT mapping functions
---------------------------------------------------------------------------
USAGE: results = shape_read(filename)
where: filename = an arcview shape file name without the extension
---------------------------------------------------------------------------
Returns: a structure variable:
 results.npoly = a scalar with # of polygon regions (observations) on the map
 results.nvars = # of variables from the dbf file [should = the length(vnames)]
 results.nobs  = # of observations from the dbf file [should = npoly]
 results.xmin = an nobs-vector of minimum longitude for each region
 results.xmax = an nobs-vector of maximum longitude for each region
 results.ymin = an nobs-vector of minimum latitude for each region
 results.ymax = an nobs-vector of maximum latitude for each region
 results.nvertices = an nobs-vector of # of vertices for each region
 results.nparts = an nobs-vector of # of parts for each region
 results.xc = an nobs-vector with x-centroid for each polygon/region
 results.yc = an nobs-vector with y-centroid for each polygon/region
 results.data = (an nobs=npoly by nvars) matrix of sample data observations
                for each polygon/region
 results.vnames = variable names for data vectors from dbf file
 results.x = an (n by nvertices) vector of polygon points, with NaN separators
 results.y = an (n by nvertices) vector of polygon points, with NaN separators
```

Users need not be concerned with manipulating the information contained in the structure variable to produce map visualizations. The structure variable can simply be passed on to a mapping function, which will intelligently decipher information from the various fields required to produce the GUI interface containing the map and related map legends or statistical graphs. However, extracting the data information in the field '.data' as well as centroids of the map regions contained in the fields '.xc' and '.yc' is necessary to use this information in spatial econometric modeling.

A specific example of the simplest possible way to produce a map is shown below:

```
filename = '..\shape_files\china';

results = shape_read(filename);

arc_histmap(results.data,results);
```

This would produce a mapping GUI containing a map of China that presents the first variable vector in the matrix stored in results.data. The GUI provides a 'pull-down menu' that allows other variable vectors to be presented on the map. These variables are labelled with names constructed from the '.vnames' field obtained from the database (.dbf) component of the shape files. Before turning attention to usage of the mapping GUI functions, we discuss design considerations related to plotting and graphical handling of the map polygons embodied in a function named **make_map**. A description of the GUI mapping interface created by various mapping functions that make up the *Arc_Mat Toolbox* is in the next section.

2.1 Plotting map polygons

Recent developments in computer gaming have resulted in inexpensive laptop and desktop computers containing fairly robust graphics hardware that is capable of placing thousands of filled polygons on the display in a fraction of a second.

A general utility function was created to plot the polygon information contained in the 'results' structure and return a structure variable containing 'graphics handles' to each polygon and its possible parts. The term 'graphics handles' is used by the MathWorks to refer to an set of graphical objects that contains information about graphics elements including such things as figure window characteristics (e.g., window size, positioning, aspect ratios), polygons, axes, points, and text fonts. These graphical objects can be manipulated with messages to alter the characteristics of almost all aspects of a graphics figure window and its component parts.

The polygon structure used by ArcView can be represented as two vectors containing latitude and longitude coordinates for each map region,

where individual polygons are indicated by a separator symbol. However, a complication arises with respect to map features such as lakes and rivers, which is handled through the use of a sub-structure called a polygon part. Without getting into detail, parts can be recognized by a reversal of the coordinate directions. Knowledge of this organization scheme can be used to appropriately parse the vectors of polygon coordinates and plot these in a figure window. This work is done by a utility function named **make_map**, which takes the 'results' structure returned by the **shape_read** function and returns another structure. This contains a pointer to an invisible map figure along with a multi-dimensional graphics handle structure 'poly.handles' taking the form: **poly(i).handles(k)**, where i references each polygon associated with a map region, and k ranges over possible 'parts' associated with each region.

The function **make_map** produces an invisible graphics figure, which can be manipulated using the graphics handles. Individual polygons representing map regions can be made visible or invisible, the color scheme as well as polygon fill colors and numerous other characteristics of the polygons can be manipulated using Matlab functions that provide support for manipulation of a wide array of the graphic object properties contained in the handles. Note that R/Splus also utilizes an object-oriented approach to graphics, allowing a similar approach to that taken here in those environments. There are also numerous graphical software environments for languages such as c/c++ and FORTRAN for various operating systems such as Windows and Linux that implement object-oriented graphics developed using a similar philosophy.

The design intention is that the **make_map** function would be called once at the beginning of the actual GUI-based mapping functions. After this call, the component functions that make up the graphical user-interface operate on the graphics handles contained in the structure variable 'poly' that are passed to these sub-functions. Changing the properties and states of the polygons by manipulating the structure variable elements containing graphics handle references or pointers to each map polygon represent the method used to carry out mapping operations. For example, a pull-down

menu option might allow the user to change the color scheme of the map to reflect a gray-scale map rather than a color-coded map. This would be accomplished by the sub-function through operation on the graphics handle object properties contained in the multidimensional 'poly' structure variable.

The motivation for creating an 'invisible' graphics figures and map polygons at this point is to allow simultaneous presentation of the map figure as well as the associated map legend figure and various pull-down menus that constitute the mapping GUI.

Another important design consideration is that we plot the map polygons only once to produce speed in the graphical presentation. This is particularly important to facilitate functionality that allows the user to 'zoom' in and out on sub-regions of the map. During these operations, the same polygons originally placed in the graphic figure window are utilized by changing their state from visible to invisible when map zooming operations takes place. The state of map polygons outside the sub-region of interest is changed from visible to invisible, but these polygons are never destroyed. This eliminates the need to re-draw the polygons which can be computationally intensive for maps involving a large number of polygons. As an example of the speed, the time required to read the ArcView shape file information by the function **shape_read** for a file containing 3,111 US counties was 2.0 seconds on a Dell Pentium III-M (Centrino) 1.6 Ghz laptop. The time taken to plot the map polygons and form the graphics handle structure variable taken by the function **make_map** was 1.60 seconds, with 1.27 seconds of this time required to plot the polygons, and 0.33 seconds taken to form the polygon handles structure 'poly'. At present, the map zooming operation for this case of 3,111 polygons takes around 0.3 seconds, which suggests that re-drawing the map polygons during zooming operations would dramatically slow the interactive presentation requiring 1.27 seconds in place of the 0.3 seconds, or a factor of 4 slowdown.

One advantage of operating in a high-level statistical software programming environment already noted is the availability of econometric, statistical and spatial econometric modeling functions. Another advantage is portability across different operating systems and hardware environments such as

UNIX, MS-Windows and Apple OSX that most statistical software including Matlab supports. A third advantage is that the statistical software provides an intelligent interface to underlying hardware and software graphics functionality. For example, Matlab automatically handles numerous default decisions regarding hardware versions of OpenGL. Matlab detects the hardware version if it is available, and if a hardware version of OpenGL is not available, Matlab uses a software version of these low-level graphics functions. This default behavior can be overwritten by the programmer, but reliance on this allows underlying design decisions to be implemented across multiple hardware and operating system platforms in a consistent fashion. To illustrate the complexity of dealing with software/hardware graphics peculiarities, consider the following criterion used by Matlab when dealing with the decision to use OpenGL versus other underlying graphics hardware or software functions.[1] Matlab selects OpenGL if:

> The host computer has OpenGL installed and is in True Color mode (OpenGL does not fully support 8-bit color mode).
>
> The figure contains no logarithmic axes (logarithmic axes are not supported in OpenGL).
>
> Matlab would select zbuffer based on figure contents.
>
> Patch objects faces have no more than three vertices (some OpenGL implementations of patch tesselation are unstable).
>
> The figure contains less than 10 uicontrols (OpenGL clipping around uicontrols is slow).
>
> No line objects use markers (drawing markers is slow).
>
> Phong lighting is not specified (OpenGL does not support Phong lighting; if you specify Phong lighting, Matlab uses the ZBuffer renderer).

The point here is that developing mapping functionality in a native language environment such as c/c++ or FORTRAN would require a series of

[1] This information was taken from documentation in Matlab version 6.5

knowledgable decisions about the inherent strengths and weaknesses of underlying graphics hardware/software libraries. This would seem to be the situation confronted by the GeoDA project, which can be avoided when relying on high-level languages such as R/Splus or Matlab.

Another place where high-level language functionality comes into play for mapping is automatic control of modes related to data and plot box aspect ratios. The *Arc_Mat Toolbox* allows users to zoom in on sub-regions of the map figure, which can produce distorted images unless care is taken to preserve aspect ratios. Our current implementation utilizes the map projection present in the ArcView shape files. ESRI provides a number of tools to change map projections and newer versions the the software produce a component shape file with the '.prj' extension that contains projection information. There are a host of design issues related to implementing facilities for changing mapping projections. Starting with a global map, zooming in on a sub-region of the world requires that care be taken to change the map projection. A commercial *Mapping Toolbox* is available from the MathWorks that contains numerous functions for changing polygon coordinates to reflect alternative map projections. There is also a public domain Matlab mapping toolbox *M_map*.

We note that the objective of the *Arc_Mat Toolbox* is not cartographic reality, but rather comparative visualization of sample data relationships that may exist between regions. Visualizing a cluster of residuals that take on similar values to those from neighboring observations does not require high precision projections. Rather, emphasis is on preserving the aspect ratio of the initial projection so as not to disorient the user when zooming in or out on the map. In theory, map projection transformations of the polygon coordinates could be incorporated as an option in *Arc_Map*, but this is not currently implemented. We note that our earlier discussion of the time required to plot a large number of polygons suggests a trade-off between speed and cartographic reality. Changing from the initial map projection to a new one would require another call to **make_map** to draw a new set of polygons and form graphics handles based on these transformed polygons. As noted this would slow map zooming operations by a factor of 5

times when we take into account time needed to reform the graphic handles structure. Future advances in graphics hardware and software should change the nature of this trade-off, perhaps allowing a pull-down menu for changing map projections. Automatic changes in projection based on zooming in or out on map sub-regions would also be possible, but represent the most compute intensive approach.

3 A GUI interface for mapping in Matlab

The Matlab programming environment provides a host of functions that can be used to produce a graphical user interface. Our discussion here focuses on the Matlab environment, but many of the design considerations would also apply to other statistical software environments. A program that utilizes the GUI functions can be constructed to allow a single function call from the Matlab command window, or a file containing Matlab commands to create the GUI interface. This *Arc_Map* design decision was motivated by the need to allow users to pass matrices or vectors returned by functions for spatial econometric analysis to produce a map. A number of different mapping GUI's exist as separate functions that produce different types of maps relevant for spatial statistical analysis.

Calling one of the mapping GUI functions turns control over to the mapping GUI, allowing various pull-down menus to be used to control characteristics of the map presentation. For example, pull-down menus allow the user to change: color schemes, variable selection from the input matrix, zooming of sub-regions on the map, choice of the number of legend categories, and labelling of the map polygons. On exit from the mapping GUI, control is returned to Matlab where further spatial econometric analysis or calls to another another type of GUI mapping function could be undertaken.

We provide an overview of the different types of GUI mapping functions currently available in *Arc_Map*, followed by sub-sections describing each of these in more detail.

The basic mapping function is named **arc_histmap**, which produces a map based on an n by k matrix input, where n denotes the rows in the

matrix which represent regions or observations, and k denotes the number of columns in the matrix which contain variable vectors. The mapping GUI produces a map of the first variable vector (column) of the input matrix in one figure window along with a histogram showing the distribution of the observation values in another figure window. Figure 1 provides an illustration of this type of map, with the associated legend shown in Figure 2. The map displays population growth of provinces in China over the 1980 to 1995 period, allowing easy identification of high versus low growth provinces.

A second type of mapping GUI is the Moran scatterplot that depicts a variable y on the horizontal axis with the average of "neighboring region" values for the variable y on the vertical axis, as a scatter of points. Neighboring regions averages are constructed using an n by n spatial weight matrix W, that contains non-zero entries in the i, jth position if observation j represents a "neighboring region", and zeros on the main diagonal. For the case of neighboring regions defined using a spatial contiguity, the matrix W could be constructed by placing values of unity in positions i, j, where j indicates regions that have borders touching region i. This matrix is then row-standardized to have row-sums of unity. The matrix product Wy then produces an average of the values from regions meeting this definition of neighbors. Various functions for producing spatial weight matrices based on contiguity, nearest neighbors, and distance are a part of the *Spatial Econometrics Toolbox* and the *Spatial Statstics Toolbox*. Since Matlab represents a programming environment, users are free to construct spatial weight matrices based on other criterion that appropriately define the structure of connectivity relationships between regions for their particular problem.

Figure 3 shows the Moran scatter plot where the population growth rates have been transformed to deviations from the mean form. This allows the population growth rates shown on the horizontal axis for each region to be expressed as positive if they above the overall average population growth rate, or negative if they are below the average. The vertical axis was constructed using a first-order contiguity spatial weight matrix that produces average growth rates of neighboring provinces. The four color-coded quadrants in the scatter plot depict: 1) regions where higher than average growth

rates are associated with an average of neighboring region growth rates that are also higher than average (the red points), 2) regions where lower than average growth rates are associated with an average of neighboring region growth rates that are also lower than average (the cyan points) 3) regions where lower than average growth is associated with neighboring regions having higher than average growth (the green points) and, 4) regions where higher than average growth is associated with lower than average growth in neighboring regions (the purple points).

The map figure depicts regions using the color coding from the scatter plot, and numerical labels can be provided as an option in both figures to allow identification of the growth rates associated with each region versus that of neighboring regions. This type of scatter plot is frequently used to assess the extent to which sample data observations exhibit spatial dependence. The presence of a large number of red points in the first quadrant are indicative of strong spatial association since provincial growth rates shown in the horizontal axis are positively associated with the average growth rate of neighboring provinces. Similarly, a large number of cyan points in the third quadrant point to positive spatial association since they indicate that the value of the variable on the horizontal axis is positively associated with the average value from neighboring regions. In contrast, points depicted as green and purple reflect weak spatial association, since these are observations that differ in value from those of neighboring observations (on average).

A third use for the mapping GUI functions is something labelled **sar_map**, where SAR stands for a spatial autoregressive econometric model. The spatial autoregressive model represents a variant of simple regression: $y = X\beta + \varepsilon$ which adds an additional explanatory variable to the model, Wy that reflects a spatial lag: $y = \rho Wy + X\beta + \varepsilon$.

This function allows the user to select sub-regions on the map for which spatial autoregressive model estimates of the parameters β and ρ as well as the noise variance estimate σ_ε^2, log-likelihood function value, R^2 measure of fit, and other information will be produced. The estimates for a sub-region are presented alongside those for the entire sample. This allows an assessment of local variability in the nature of the underlying linear spatial

autoregressive relationship of interest.

This type of GUI mapping interface illustrates a distinct advantage of Matlab over more conventional GIS mapping environments, in that spatial econometric methods can be used as part of the map interface. Solving for maximum likelihood estimates of basic spatial regression models provides some computational challenges when the problem involves large samples. This makes it difficult to provide estimation functionality in mapping software, which would need to provide sparse matrix functionality as well as algorithms for numerical linear algebra. In contrast, this functionality is already provided by the public domain *Spatial Econometrics* and *Spatial Statistics* toolbox functions for the Matlab environment. Similar spatial econometric modeling functionality exists in *spdep* for the R/Splus software environment.

The toolbox functions utilize sparse matrix algorithms available in Matlab as well as computationally efficient algorithms suggested in Pace and Barry (1999) for computing the log-determinant of an n by n matrix, and a vectorized approach to producing maximum likelihood estimates. Taken together, these allow maximum likelihood estimates to be produced in one or two seconds for problems involving over 3,100 US counties. This speed allows the initial global estimates for a model to be shown in a graphics window and rapid updating of estimates for sub-regions selected using the GUI interface. Single regions can be added or subtracted from the sub-region selection set to test for influential observations. Figure 5 shows the selection map, where a sub-sample of 1,257 observations (highlighted) in the center of the contiguous US states have been selected from the total of 3,111 county-level observations.

Figure 6 shows the graphics window that presents global estimates alongside those from the selected sub-region for comparison. The model used to produce this illustration represents a spatial autoregressive'growth regression', that relates employment growth rates over the 1980 to 1990 period to a host of county-level characteristics. Coefficients with a positive sign indicate that the associated characteristics promote higher employment growth rates, whereas negative signs suggest the converse.

As an example of how the comparative estimates and associated $t-$statistics might be useful, consider the coefficient on college, where this variable reflects the percent of population aged 25 and over holding college degrees as their highest level of educational attainment. For the global sample, the coefficient is not statistically significantly different from zero, whereas the sub-region in the center of the US produced a negative and significant estimate.

Having provided a brief introduction to the various types of GUI mapping functions, we provide a more detailed discussion regarding each of these in the following sub-sections.

3.1 The arc_histmap function

This function provides basic mapping of a matrix where rows of the matrix input are associated with the map regions contained in the ArcView shape file and each column represents a different variable. Two associated figure windows are displayed, one containing a map of the observation values for each region, and the other a histogram showing the distribution of these values. These figure windows can be resized and moved independently of each other, with the legend figure overlaid on top of the map figure.

The matrix supplied as an input argument could be that returned in the structure variable field, 'result.data' by the **shape_read** function. It can also be any user-supplied matrix, but this would need to be organized so the rows of the matrix are synchronized with the order of regions from the shape file. This can be accomplished by sorting a user-supplied matrix based on the geographic key, perhaps a county-code, zip code or census tract designator. This type of identifier is typically one column of the matrix 'result.data' read from the database file by the function **shape_read**.

Documentation for the function follows the consistent format used for functions in the *Spatial Econometrics Toolbox*, and can be viewed by typing 'help arc_histmap' in the Matlab command window. The documentation that would appear is shown below:

```
PURPOSE: produce a map with histogram legend using ArcView shape files
-------------------------------------------------------------------
USAGE: arc_histmap(variable,results,options)
where: variable = a variable vector (nobs x 1) or matrix (nobs x nvars)
       results  = a structure variable returned by shape_read()
       options  = a structure variable with function options
       options.vnames     = a string with the variable name or names
       e.g. vnames = strvcat('constant','pinstruction','pbuilding','padminist');
       options.cmap       = a colormap string, e.g., 'hsv','jet' (default 'hsv')
                            (see colormap menu in the gui for various options)
       options.nbc        = # of categories (default = 5)
                            (see #categories menu in the gui for various options)
       options.missing    = a vector of 0's and 1's for missing and non-missing observations
                            0 = missing, 1 = non-missing
                            (produces white map polygons for missing values)
       options.mapmenu    = 1, for a menubar, 0 = default, no menubar
                           (useful for printing, editing the map)
       options.legendmenu = 1, for a menubar, 0 = default, no menubar
                           (useful for printing, editing the map)
       options.labels     = 1, for numerical labels on the polygons,
                            0 = default, no labels
-------------------------------------------------------------------
RETURNS: a graphical user interface with the map and histogram legend
as well as menus for: colormap,zoom map,#categories,variable,quit
-------------------------------------------------------------------
see also: ahape_read(), make_map()
-------------------------------------------------------------------
```

The GUI for this function provides a series of pull-down menus along the bottom of the map figure that allow for user selection of:

1. The color scheme to be used on the map. Choices include all of the default colormaps from Matlab:

   hsv - Hue-saturation-value color map.

   hot - Black-red-yellow-white color map.

   cool - Shades of cyan and magenta color map.

   gray - Linear gray-scale color map.

   bone - Gray-scale with tinge of blue color map.

copper - Linear copper-tone color map.

jet - Variant of HSV.

pink - Pastel shades of pink color map.

white - All white color map.

autumn - Shades of red and yellow color map.

spring - Shades of magenta and yellow color map.

winter - Shades of blue and green color map.

summer - Shades of green and yellow color map.

2. A zoom menu that allows sub-regions on the map to be selected using a drag-rectangle. This can be repeated to provide a view of increasingly fine spatial scales, or un-zoomed to return to the original map focus. During zoom, the aspect ratio is preserved which results in some wasted screen space as the margins around the map may need to be enlarged to preserve the aspect ratio. As noted earlier, implementation of map projection transformations could be used to improve this aspect of the GUI interface.

3. A menu that allows the number of categories for the histogram to be selected. This allows the user to examine the distribution of a variable using a varying number of 'bins' or 'buckets' in the histogram.

4. A menu that allows each variable/column of the input matrix to be selected for viewing on the map and associated histogram.

5. A menu for exiting the mapping GUI and returning to the Matlab command line.

Use of the pull-down menus to select a new colormap, variable, zoom level or number of histogram categories results in returning focus to the legend figure window so that in cases where this has been overlaid on the map figure, both figures are visible. There are traditional minimize and maximize boxes on each figure window, so users can focus on a single figure

window by minimizing one of the two windows, or expand the view to full-screen.

Optional input arguments are provided in a structure variable as fields. These allow user input of:

options.vnames - a set of variable names that will appear on the pull-down menu for selecting variables to display on the map.

options.cmap - a color scheme choice to use in the initial map figure. This can be altered in the GUI interface using the pull-down menu.

options.nbc - the number of categories to be used in constructing the initial histogram and color scheme for the map. This can be altered in the GUI interface using the pull-down menu.

options.missing - a vector with values of zero and one to indicate map regions for which there is missing data information. Without this optional input vector, the histogram would be distorted by missing values. It may be the case that a user-supplied matrix does not contain sample data information for all regions on the map that was input using the **shape_read** function. This option allows these regions to appear on the map as 'white' regions, and these missing values will be ignored during construction of the associated histogram figure.

options.mapmenu, and options.legendmenu - a switch that adds a top menu to the map and legend figures respectively. This menu contains standard Matlab functions for operating on graphical figures. For example, a file menu item allows the figure window to be printed or exported in numerous graphic file formats, other menus allow the user to add lines and text annotations to the figure windows, and a host of other graphic functionality.

options.labels, which can be used to place numerical labels on the map polygons. This is useful for checking sample data observations from the input matrix that are associated with map regions. For example,

large residuals could be identified on the map and associated with the corresponding sample data observation.

The decision to make the top menu an option, rather than the default was made to increase the clarity of presentation, minimizing the amount of clutter in the GUI interface. Here we see another advantage to operating in a high-level language environment such as Matlab. The top menu for the map and legend figures represents a standard property of all Matlab figure windows. No programming was required to add these features to the mapping GUI, simply a change in the state of the figure window properties, achieved with a single call to the functions that operate on the graphics handles.

A screen capture of the mapping GUI associated with the **arc_histmap** function is shown in figure 7. A map of 1,008 Ohio zip code areas is shown, constructed using a matrix containing information on 4th grade student proficiency test scores for Ohio school buildings. Information from a sample of nearly 2,000 school building was averaged by zip code area to produce the input matrix. The missing values feature proved particularly important here. Some school buildings from the State of Ohio were indeed missing from the database provided by the Ohio Department of Education. In other cases, zip code areas had no school building with proficiency score information because the zip code area represented a major downtown business district or industrial/manufacturing region. These regions would not contain school buildings, resulting in missing student proficiency scores that are not missing due to reporting problems. Users could introduce a vector variable that contains variable values to reflect these alternative scenarios, which would appear color-coded on the map when this variable is selected in the GUI interface. This might be of interest if there were concerns about non-reporting school buildings being distributed in a spatially clustered fashion, so that one region of the state exhibited a large proportion of the reporting problems.

To illustrate the zoom feature, we selected a rectangular sub-region of the Ohio zip code areas shown in figure 8, that is centered on Toledo, Ohio. This was done after selecting the 'zoom' pull-down menu item labelled 'select

rectangle'. Both the map figure and associated histogram map legend are recalculated to reflect the sub-sample of observations selected.[2]

A clear pattern of lower scores exists for central city school buildings, which is more evident in the zoomed map than that depicting the statewide patterns. One point to note is that the histogram is based on a number of categories chosen using the GUI interface menu item. This histogram preserves the number of categories, but places blank bars for values of the variable that are not represented in the sub-sample. This decision was make for consistency with the design motivation to not create disorientation of the user when viewing sample data information from different perspectives.

3.2 The arc_moranplot function

Spatial autocorrelation can be viewed as a map pattern, as well as various other interpretations. This can be measured using an extension of Pearson's product moment correlation coefficient using a binary spatial weight matrix C, where elements $c_{ij} = 1$ to denote that observation j is relatively close to location i. This extended correlation coefficient known as the Moran Coefficient becomes:

$$MC = \frac{n}{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}} \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{1}$$

The most common interpretation of spatial autocorrelation is in terms of trends or patterns across a map. In this context we note that as in the case of a Pearson product moment correlation coefficient, the value of MC approaches unity when similar values of the observations tend to cluster on the map, referred to as positive spatial autocorrelation. Similarly, when the value of MC approaches -1, we see dissimilar values clustering on the map, known as negative spatial autocorrelation. A random pattern of values for a variable on the map results in the MC approaching $-(1/(n-1))$, which is asymptotically zero for large values of n.

[2]Note that the two independent figures showing the map and legend were merged into a single graphic figure for presentation purposes. Figure 8 does not depict the manner in which the map and legend actually appear in the GUI interface.

In contrast to a conventional correlation coefficient, MC is not restricted to the range $[-1, 1]$, but has a range determined by the minimum and maximum eigenvalues of the matrix C.

The **arc_moranplot** function attempts to illustrate the strength of spatial autocorrelation using a scatter plot of the relation between a variable vector $(y - \bar{y})$ measured in deviations from the mean form and the spatial lag of this variable, $W(y - \bar{y})$. The spatial weight matrix W differs from the binary matrix C in that it is row-standardized, so that row-sums are unity. One way to view this scatter plot would be to consider the first-order spatial autoregressive relationship:

$$(y - \bar{y}) = \rho W(y - \bar{y}) + \varepsilon \tag{2}$$

Where the disturbance term ε is distributed normally with mean zero and scalar noise variance $\sigma_{\varepsilon}^2 I_n$.

The slope of the Moran scatter plot would be represented by the scalar parameter ρ, so that values near unity indicate high levels of positive spatial autocorrelation, which would correspond to a Moran scatter plot having a large number of points in quadrant I, where high values of $(y - \bar{y})$ are associated with high values for neighbors $W(y - \bar{y})$, and quadrant III, where and low values are also associated with low values for the neighbors. A value of ρ near zero points to a random distribution of points across the four quadrants that make up the Moran scatter plot, pointing to a lack of spatial dependence between observations in the vector $(y - \bar{y})$ and the average of neighboring values $W(y - \bar{y})$. Negative values of ρ point to a large number of points in quadrant II, where where low values of $(y - \bar{y})$ are associated with high values for neighbors $W(y - \bar{y})$, and quadrant IV, where high values are associated with low values for the neighbors.

The ability of the spatial econometric routines to provide maximum likelihood estimates for the parameter ρ in the first-order spatial autoregressive model provides a nice complement to the Moran scatter plot mapping GUI. Another aspect of operating in an environment where spatial econometric routines are available is the ability to generate sets of different spatial weight matrices that can be used as an input argument to the Moran scatter plot

function. For example, one could examine the scatter plot and associated map using a spatial weight matrix constructed using first-order neighbors based on distance or contiguity, as well as higher-order neighbors based on contiguity or distance. Functions to construct a variety of spatial weight matrices represent a standard part of the *Spatial Econometrics* and *Spatial Statistics* toolboxes are well as the R/Splus spatial functions *spdep*.

Documentation for the function **arc_moranplot** is shown below:

```
PURPOSE: produce a map with moran scatterplot using ArcView shape files
---------------------------------------------------------------------
USAGE: arc_moranplot(variable,W,results,options)
where: variable = a variable vector (nobs x 1), which is transformed
                  to deviations from the mean form by the function.
             W = a spatial weight matrix
        results = a structure variable returned by shape_read()
       options  = a structure variable with function options
       options.vname      = a string with the variable name
       options.labels     = 1 for labels, 0 = default, no labels
       options.mapmenu    = 1, for a menubar, 0 = default, no menubar
                          (useful for printing, editing the map)
       options.legendmenu = 1, for a menubar, 0 = default, no menubar
                          (useful for printing, editing the map)
---------------------------------------------------------------------
RETURNS: a graphical user interface with the map and moran scatterplot
as well as menus for: zoom map,quit
---------------------------------------------------------------------
see also: shape_read()
---------------------------------------------------------------------
```

As in the case of the **arc_histmap** function, options are input using a structure variable, and the map and legend menu options are provided as well as the labelling option that produces numerical labels on the map as well as the scatter plot. This allows map regions to be identified with their associated points in the Moran scatter plot.

An illustration of the map produced by the function is provided in figure 9 and the associated scatter plot in figure 10. This was constructed using per capita income growth rates for a sample of 138 European Union regions. There is a great deal of literature that points to a core versus periphery pat-

tern of growth, with regions on the periphery exhibiting high growth while those at the core have low growth (Le Gallo, Ertur, and Baumont (2003) and Le Gallo Ertur (2003)). The color-coding of the map makes this pattern quite evidence. Cyan-colored regions occupy the core regions reflecting low growth for these regions and their neighbors. Red-colored regions represent those where the regions and their neighbors exhibit high growth rates, which are located on the periphery.

3.3 The arc_sarmap function

This function illustrates the benefit from having access to mapping functions in a spatial econometric model software environment. Fast and efficient spatial econometric estimation functions exist for estimating a class of spatial regression models described in Anselin (1988), shown in (3).

$$\begin{aligned} y &= \rho W y + X\beta + u \\ u &= \lambda D u + \varepsilon, \quad \varepsilon \sim N(0, \sigma_\varepsilon^2 I_n) \end{aligned} \tag{3}$$

The terms W and D represent n by n non-negative spatial weight matrices with zeros on the diagonal, where n represents the number of sample observations. The spatial weight matrices W and D are specified by elements $(i, j) > 0$ for observations $j = 1, \ldots, n$. The n by k matrix X contains k explanatory variables, and the dependent variable is y. The parameters to be estimated are: β, ρ, λ, and σ_ε^2. Anselin (1988) notes that a family of models can be derived by: setting the parameter ρ on the spatially lagged dependent variable Wy to zero producing a model we label the spatial error model (SEM); setting the parameter λ on the spatially lagged disturbance term to zero produces a model we label the spatial autoregressive model (SAR); and a third model can be based on both ρ and λ non-zero, which we label SAC. Yet another model arises if we introduce spatial lags of the explanatory variables taking the form WX, along with an associated parameter vector, known as a spatial Durbin model (SDM).

The same functionality of the **arc_sarmap** function that allows the user

to explore spatial stability of the SAR model could be provided for these other models. This could be achieved easily by relying on the fast maximum likelihood estimation functions from the spatial econometrics toolbox.

In addition to maximum likelihood estimation procedures, algorithms for robust Bayesian estimate for this family of models that allow for non-constant variance across the observations. Using Markov Chain Monte Carlo (MCMC) estimation, estimates of variance scalars are produced for each observation using methods described by Geweke (1993) for the case of linear least-squares and LeSage (1997) for this family of spatial regression models. The ability to plot the variance estimates for each region/observation on a map would help identify regions where the model may be inadequate. For example, if urban regions exhibit higher noise variances, this may be an indication of an inadequately modeled regression relationship.

Other approaches to exploring spatial stability that have been proposed are geographically weighted regression (Fotheringham et. al) and Bayesian variants (LeSage, 2004). These models produce locally linear non-parametric estimates for each observation, based on a sub-sample of observations nearby. Mapping these parameter estimates can be useful in exploring systematic patterns between the dependent and explanatory variables over space. Pace and LeSage (2004) suggest a spatial autoregressive locally linear estimation procedure that they label SALE, that relies on a recursive estimation scheme to produce estimates for the SAR model for every observation. We use parameter estimates from this model constructed using a function from the *Spatial Statistics Toolbox* to demonstrate the value of mapping locally linear parameter estimates.

A model from Pace and Barry (1997) of the relationship between voter participation in the 1980 presidential election and variables measuring population eligible to vote, home ownership, educational attainment and per capita income was estimated using 3,107 observations using the SALE methodology. This produces 3,107 parameter estimates for each explanatory variable in the model. Figure 11 shows the map and legend figures of the estimates for the 'home ownership' variable in the model. The histogram legend shows that this variable exerts a predominately positive impact on

voter participation rates. From the map we see that home ownership exerts a larger impact on voter participation in the south and midwest counties, and a smaller impact for counties on the east and west coasts.

4 Using the *Arc_Mat Toolbox* functions

In this section we discuss issues that arise in using the mapping functions. One is the need to sort spatial econometric data sets to match the order of the polygon and data information read from the shape files. Another issue that arises is the need to construct spatial weight matrices, which require polygon centroids, distances or other features related to the polygon relationships. These two issues are discussed in the next subsections.

4.1 Merging data for mapping in Matlab

One approach to using the **ARC_MAT** toolbox would be to rely on ArcView shape files for the mapping polygons and perhaps centroids of the polygons, while relying on a separate data file for spatial econometric analysis. This requires that a geography key be included in the econometric analysis data set. It is likely such a code would exist in any spatial econometric sample data. For example, consider a sample of 1,008 Ohio zip code areas and a set of shape files that contain the polygon information as well as the latitude and longitude centroids for the zip code areas.

An example of the Matlab code needed to sort the spatial econometric data sample into an order consistent with the polygon ordering in the shape file is provided below. In this example the shape files are named 'ohio.dbf' and 'ohio.shp', while the spatial econometric data file is named 'odata.txt'. We assume that the zip code identifier is contained in the first column of the results.data structure variable matrix obtained from the database file. We also assume that a zip code identifier exists in the first column of the data file 'odata.txt'.

```
filename = 'ohio';
results = shape\_read(filename);
```

```
nobs = results.npoly; % should equal 1,008 the # of observations
map_zips = results.data(:,1);
load odata.txt;
dat_zips = odata(:,1);
% loop over the map_zips and find an index corresponding to
% each of these zip codes in the odata matrix
out = zeros(size(dat_zips)); % allocate a matrix for the results
missing = zeros(nobs,1);     % keep track of missing values
for i=1:nobs
zipi = map_zips(i,1);
ind = find(zipi == dat_zips);
   if length(ind) > 0
    out(i,:) = odata(ind,:);
   else
    missing(i,1) = 1;
   end;
end;
options.missing = missing;
arc_histmap(out,results,options);
```

The code loops over all zip code areas using the order from the shape files. Each zip code area is extracted into the scalar variable 'zipi', and this is used in the Matlab **find** function that searches the vector 'dat_zips' for a matching zip code. If a match is found, an index into the vector 'dat_zips' is returned and this index is used to extract a row from the matrix 'odata' and place it in the matrix 'out'. If no match is found, we record this in a vector 'missing' that could be used as an input argument to the mapping GUI function. Note that in this case the matrix 'out' would contain a row of zero values. This would adversely affect the histogram legend in the **arc_histmap** function which would interpret these as actual values without the optional 'options.missing' vector used as an input to the function.

We note that any data information contained in the structure variable 'results.data' could be combined with the sorted version of the spatial econometric sample data matrix. Because these are now sorted by zip code area,

Matlab would combine these two matrices with the sample command:

```
combined = [results.data out];
```

Since regression estimates are unaffected by the ordering of the observations, it would seem that sorting the spatial econometric data to match the order of the shape file information should occur at the outset. This would allow users to map residuals, predicted values and other information using results structure information returned by the estimation functions in the *Spatial Econometrics Toolbox*.

4.2 Polygons and spatial weight matrices

Current functions for constructing spatial weight matrices rely on two vectors of polygon centroid coordinates. Functions to produce weight matrices based on contiguity, nearest neighbors and distances using the centroid coordinates are part of the *Spatial Econometrics* and *Spatial Statistics Toolboxes*.

Centroids of the polygons are returned in the 'results' structure of the **shape_read** function. These are constructed using a c-/c++ language API function that is part of the public domain *shapelib*. The function returns the following information regarding latitude and longitude coordinates associated with the polygons as well as the vectors required to draw the polygons.

results.xmin = an nobs-vector of minimum longitude for each region

results.xmax = an nobs-vector of maximum longitude for each region

results.ymin = an nobs-vector of minimum latitude for each region

results.ymax = an nobs-vector of maximum latitude for each region

results.xc = an nobs-vector with x-centroid for each polygon/region

results.yc = an nobs-vector with y-centroid for each polygon/region

The toolbox functions rely on a Delaunay triangularization scheme to find neighboring observations, and a Matlab function exists to calculate

Delaunay triangles. To illustrate this approach, figure 12 shows a Delaunay triangularization centered on an observation located at point A. The space is partitioned into triangles such that there are no points in the interior of the circumscribed circle of any triangle. Neighbors could be specified using Delaunay contiguity defined as two points being a vertex of the same triangle. The neighboring observations to point A that could be used to construct a spatial weight matrix are B, C, E, F.

One way to specify a spatial weight matrix W would be to set column elements associated with neighboring observations B, C, E, F equal to 1 in row A. This would reflect that these observations are neighbors to observation A. Typically, the weight matrix is standardized so that row sums equal unity, producing a row-stochastic weight matrix. Row-stochastic spatial weight matrices, or multidimensional linear filters, have a long history of application in spatial statistics (e.g., Ord, 1975).

An alternative approach would be to rely on neighbors ranked by distance from observation A. We can simply compute the distance from A to all other observations and rank these by size. In the case of figure 1 we would have: a nearest neighbor E, the nearest 2 neighbors E, C, nearest 3 neighbors E, C, D, and so on. Again, we could set elements $W_{Aj} = 1$ for observations j in row A to reflect any number of nearest neighbors to observation A, and transform to row-stochastic form. This approach might involve including a determination of the appropriate number of neighbors as part of the model estimation problem.

Weight matrices can be constructed using functions: **yx2cont** and **make_nnw**, which produce contiguity and nearest neighbor weight matrices respectively. Their usage takes the form:

```
[W1,W2,W3] = xy2cont(latitude,longitude);
Wm = make_nnw(latitude,longitude,m);
```

The function **xy2cont** returns three sparse weight matrices:

W1 = W*S*W, a symmetric spatial weight matrix (max(eig)=1)

W2 = W*W*S, a row-stochastic spatial weight matrix, where S represents the adjacency matrix from Delaunay triangles (Voronoi tesselation).

W3 = diagonal matrix with i,i equal to 1/sqrt(sum of ith row)

while the function **make_nnw** returns a row-stochastic sparse weight matrix based on the m nearest neighbors.

The polygon coordinates produced by extracting information from the ArcView shape files should allow additional weight matrix functions. For example, weight matrices that rely on the relative length of common borders between two regions could be constructed. Shape files containing information concerning highway linkages and other geographical features from the map of regions could in theory also be used to provide information for use in constructing weight matrices for use in spatial analysis. This remains a topic for future exploration.

5 Conclusions

The ability to the use statistical functionality for spatial modeling and analysis in conjunction with a mapping interface in the same environment has received a great deal of attention in the spatial analysis literature dating at least to Anselin (1994). We demonstrate the feasibility of extracting map polygon and database information from ArcView shape files for use in statistical software environments. The motivation for extracting database information for use in a statistical programming environment seems quite evident. We demonstrate that information containing map polygons can also be used in these environments to produce high quality mapping functionality. Improvements in recent computer graphics hardware and software have arisen from interest in computer gaming. This allows rapid rendering of map polygons on almost all recent desktop and laptop computers using basic plotting functionality that is part of statistical software environments. A byproduct of this is that mapping functionality based on the high quality ArcView map polygons can be created in a statistical software environment

in support of spatial econometric and statistical analysis. Although our focus was on a particular implementation of these ideas in the Matlab software environment, a similar approach would be feasible in a number of other statistical software environments such as R/Splus, Octave and SciLab.

Polygon information extracted from ArcView shape files can also be used in conjunction with the *GeoXP Toolbox* of public domain programs for linking spatial statistics in both Matlab and R/Splus software environments. These functions have been constructed to take advantage of polygon coordinates, but no facility for importing this information into Matlab or R/Splus environments is provided in the toolbox.

There is also the potential to use polygon boundaries along with the database information to identify map zones that correspond to abrupt change in variables of interest, a process known as *wombling*, after a seminal paper on this topic by Womble (1951). This is a popular technique used by among others, geneticists, demographers, and environmental scientists in spatial statistical work. The goal of these methods is to identify important regional differences across shared borders of the regions of analysis that might relate to "barriers" or "edge detection". For example, wombling analysis was used by Bocquet-Appel and Jakobi (1996) to identify barriers in population diffusion that impacted the demographic transition in Europe. Recent wombling work for point level data rather than areal data (where information is aggregated or averaged over geopolitical regions) by Banerjee, Gelfand, and Sirmans (2004) relies on Bayesian hierarchical spatial statistical models to determine boundaries by locating points associated with abrupt gradient changes measured on a fitted spatial surface. The ability to extract map polygon information into a spatial statistical software environment should advance research on wombling. For this type of modeling, the need for interactive visualization of polygon boundaries based on statistical classification calculations remains a key to this type of analysis. Existing software such as *BoundaySeer* Maruca, and Jacquez (2002) provides some functionality along these lines but lacks the capability to implement Bayesian hierarchical models and Markov Chain Monte Carlo estimation methods.

6 References

Anselin L. (1988), *Spatial Econometrics: Methods and Models*, Kluwer Academic Publishers, Dordrecht.

Anselin, L. (1994), "Exploratory Spatial Data Analysis and Geographic Information Systems," in M. Painho (ed.), *New Tools for Spatial Analysis,* Eurostat, Luxembourg, 1994, pp. 4554.

Luc Anselin, Ibnu Syabri and Oleg Smirnov (2002), "Visualizing Multivariate Spatial Correlation with Dynamically Linked Windows," In L. Anselin and S. Rey (Eds.), *Proceedings, CSISS Workshop on New Tools for Spatial Data Analysis,* Santa Barbara, CA, May 10-11, 2002. Center for Spatially Integrated Social Science, CD-ROM (pdf file, 20pp, 517K).

Banerjee, S., A.E. Gelfand and C.F. Sirmans (2004) "Directional rates of change under spatial process models," *Journal of the American Statistical Association*, Vol. 98, pp. 946-954.

Anselin, Luc and Shuming Bao. 1997. "Exploratory Spatial Data Analysis Linking SpaceStat and ArcView," In: Manfred Fischer and Arthur Getis (Eds.), Recent Developments in Spatial Analysis, Berlin: Springer-Verlag, pp35-59.

Barry R., Pace R.K. (1999), "A Monte Carlo Estimator of the Log Determinant of Large Sparse Matrices", *Linear Algebra and its Applications*, Vol. 289, pp. 41-54.

Bivand, Roger S. (2002) "Spatial econometrics functions in R: Classes and methods", *Journal of Geographical Systems*, Vol. 4, pp. 405-421.

Bocquet-Appel, J.P., and L. Jakobi. (1996) "Barriers for the spatial diffusion for the demographic transition in Europe," in *Spatial Analysis of Biodemographic Data,* J.P. Bocquet-Appel, D. Courgeau and D. Pumain (eds.) Eurotext, John Libbey, London and Paris.

Brunsdon, C., A. S. Fotheringham, and M.E. Charlton (1996), "Geographically weighted regression: A method for exploring spatial nonstationarity," *Geographical Analysis*, Vol. 28, pp. 281-298.

Geweke J. (1993), "Bayesian Treatment of the Independent Student t Linear Model," *Journal of Applied Econometrics*, Vol. 8, pp. 19-40.

Goodchild, Michael F. and Robert Haining (2003), "GIS and spatial data analysis: Converging perspectives," *Papers in Regional Science*, Vol. 83, pp. 363-385.

Heba, Ines, Eric Malin and Christine Thomas-Agnan, (2002) "Exploratory spatial data analysis with GEOXP," *European Regional Science Association conference papers*.

Le Gallo, J., C. Ertur C., Baumont C. (2003) "A Spatial Econometric Analysis of Convergence across European Regions, 1980-1995," in *European Regional Growth*, B. Fingleton, (ed.), Springer-Verlag, Advances in Spatial Science, pp. 99-130.

Le Gallo J., and C. Ertur (2003) "Exploratory spatial data analysis of the distribution of regional per capita GDP in Europe, 1980-1995," *Papers in Regional Science*, Vol. 82 no. 2, pp. 175-201.

LeSage, James P. (1997) "Bayesian Estimation of Spatial Autoregressive Models," *International Regional Science Review*, Vol. 20, nos. 1&2, pp. 113-129.

LeSage, J.P. (2004) "A Family of Geographically Weighted Regression Models," forthcoming in *Advances in Spatial Econometrics*, Luc Anselin, J.G.M. Florax and S.J. Rey (eds.) Springer-Verlag.

Maruca, S.L., and G.M. Jacquez. (2002) "Area-based tests for association between spatial patterns," *Journal of Geographic Systems* Vol. 4 no. 1, pp. 69-83.

Ord, J.K. (1975), "Estimation Methods for Models of Spatial Interaction," *Journal of the American Statistical Association*, Vol. 70, pp. 120-126.

Pace, R. Kelley, and Ronald Barry, "Quick Computation of Regressions with a Spatially Autoregressive Dependent Variable," *Geographical Analysis*, 29, 3, 1997, 232-247.

Pace, R.K, LeSage. James P., (2004) "Spatial Autoregressive Local Estimation," in *Recent Advances in Spatial Econometrics*, Jesus Mur, Henri Zoller and Arthur Getis (eds.), Palgrave Publishers, pp. 31-51.

Womble, W. (1951) "Differential systematics," *Science*, Vol. 114, pp. 315-322.

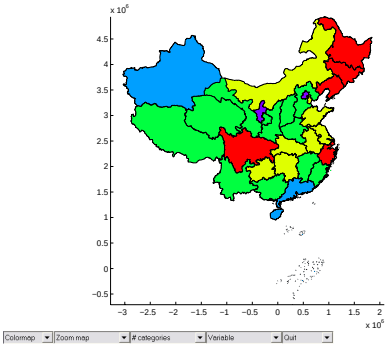

Figure 1: China map of population growth rates 1980-1995

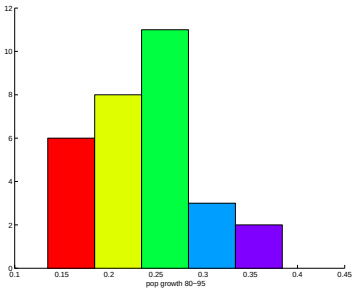

Figure 2: Legend for China map of population growth rates

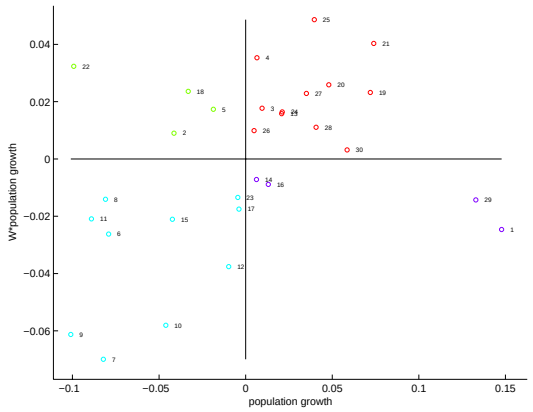

Figure 3: Moran scatter plot for Chinese provincial population growth rates

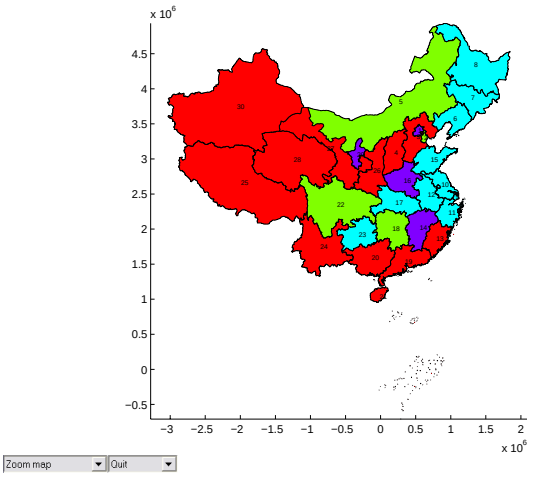

Figure 4: China map linked to the moran scatter plot

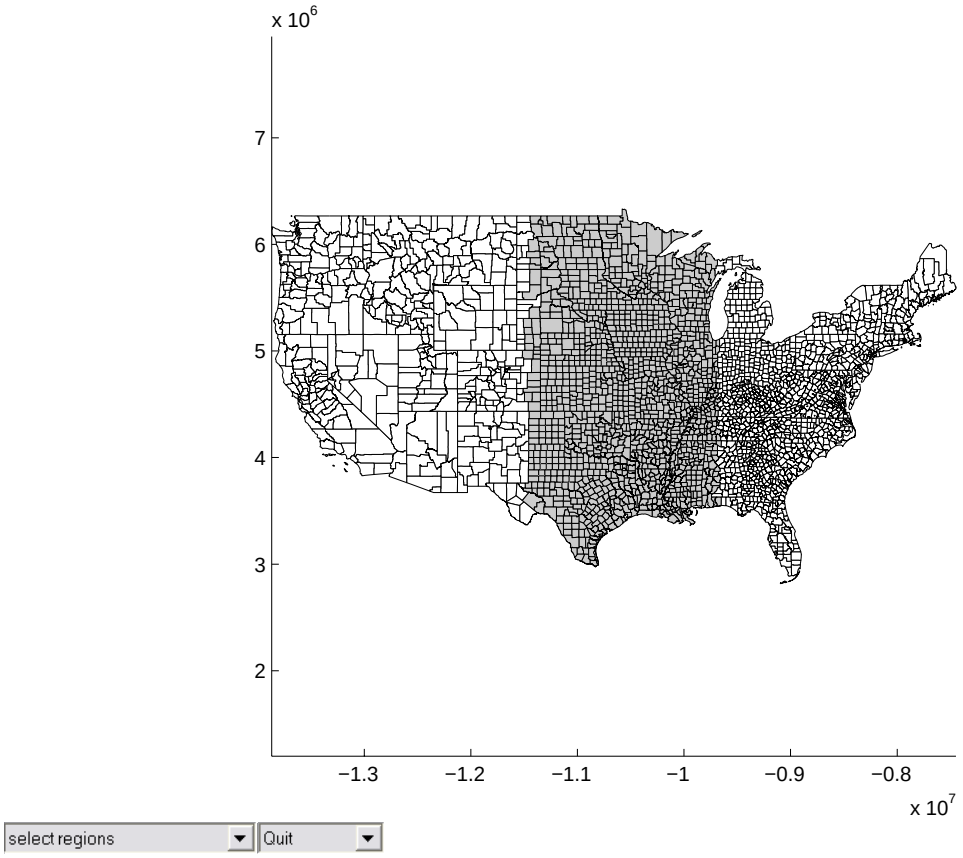

Figure 5: SAR model selection map

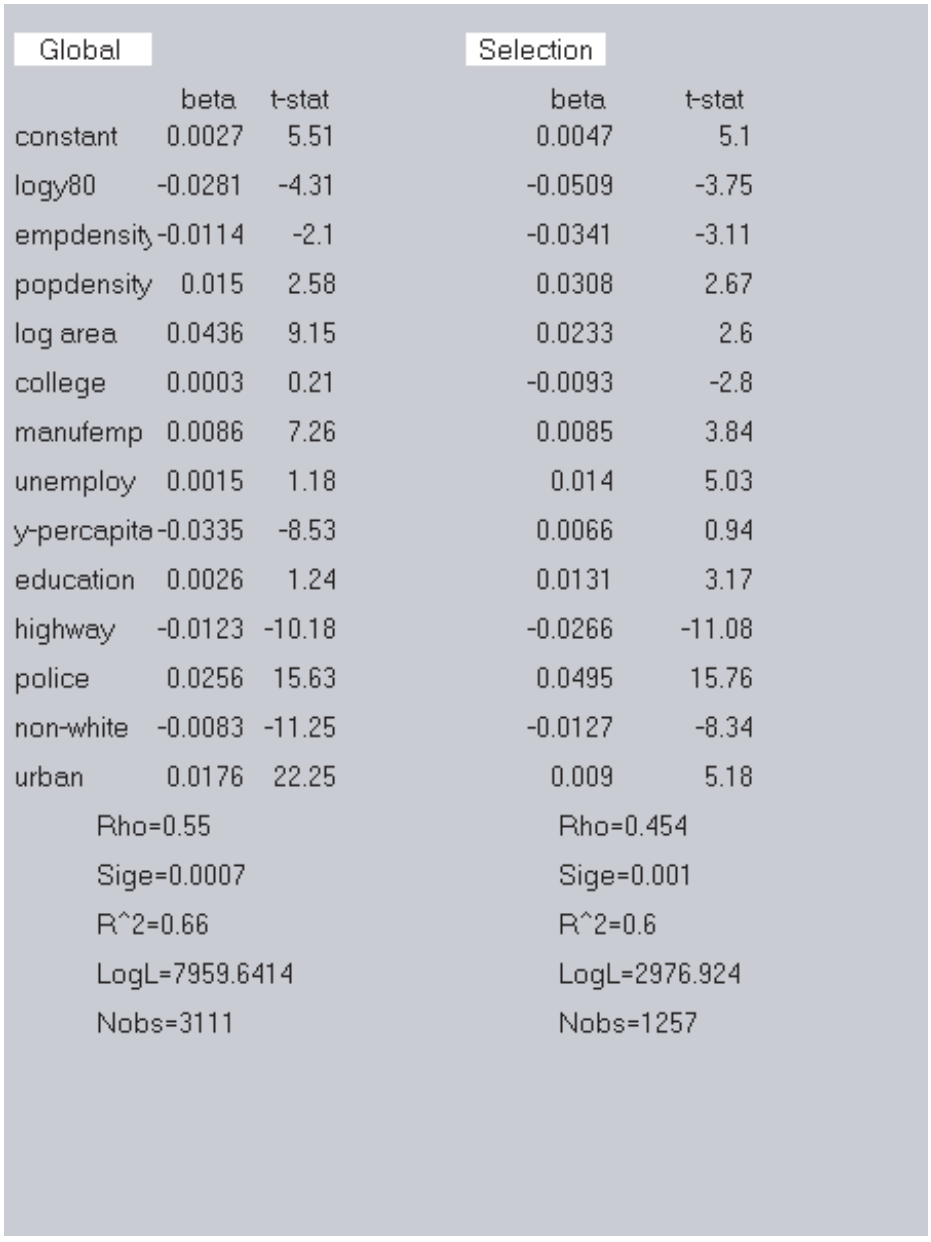

Figure 6: SAR model comparison of global and sub-sample estimates

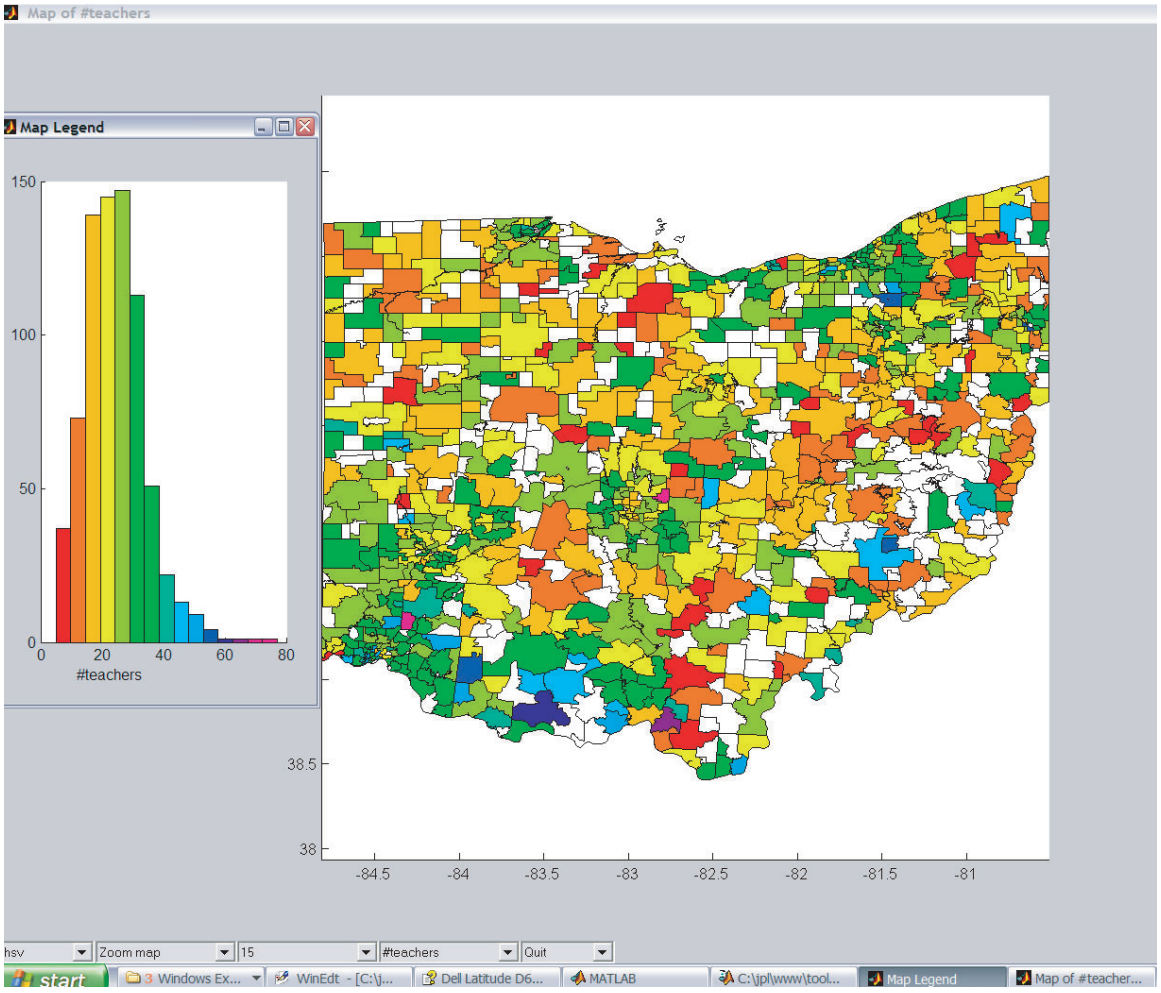

Figure 7: A screen showing the **arc_histmap** GUI

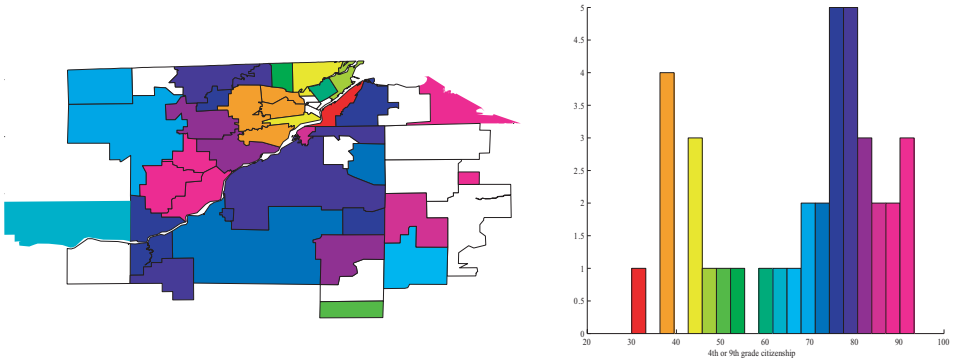

Figure 8: Histogram with linked map zoomed in on Toledo, Ohio

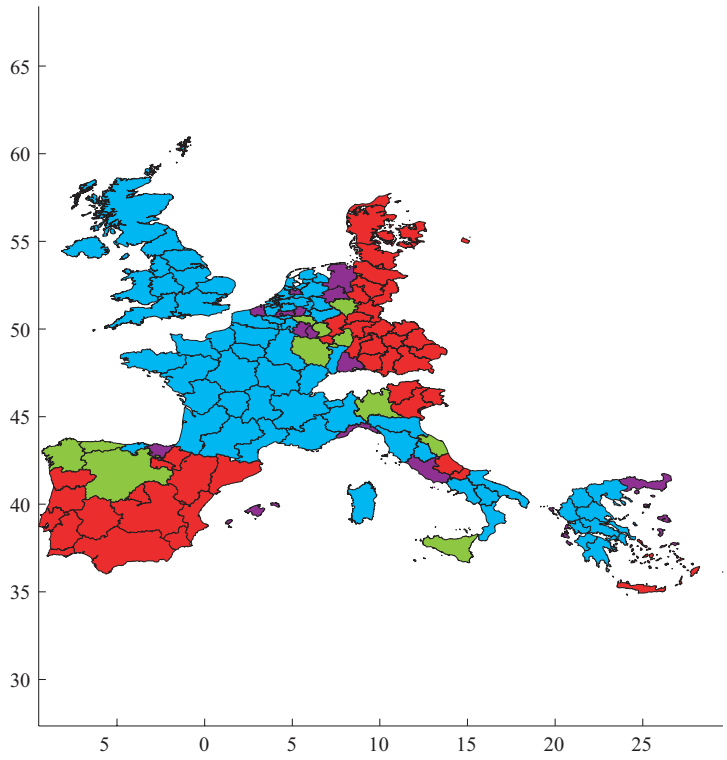

Figure 9: Moran scatter plot map for 138 European regions

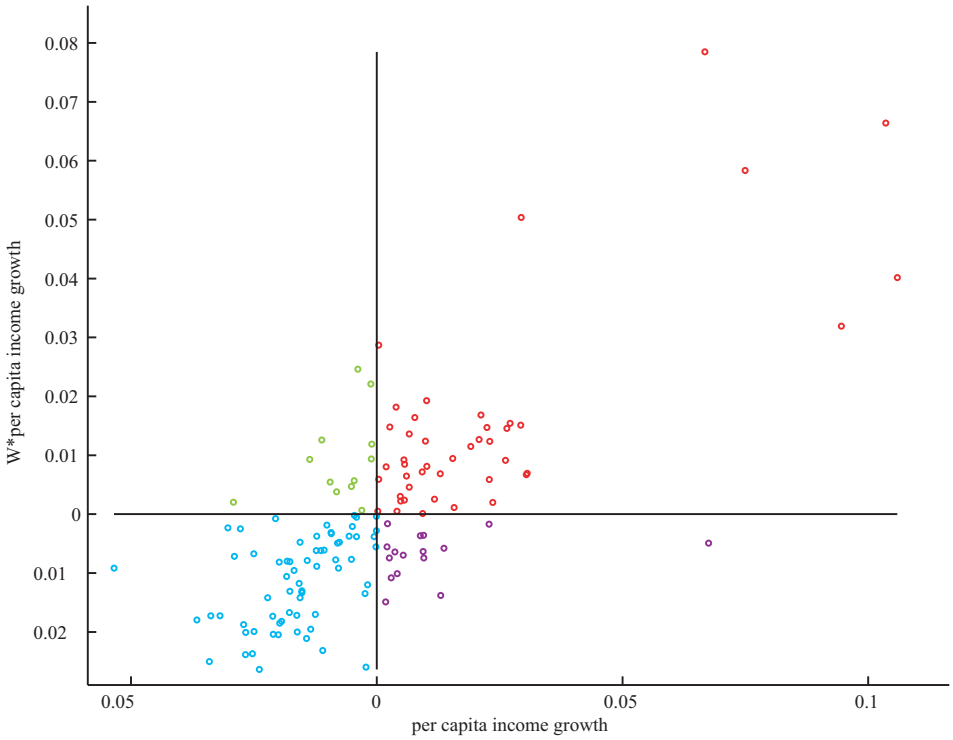

Figure 10: Moran scatter plot for European Union regions

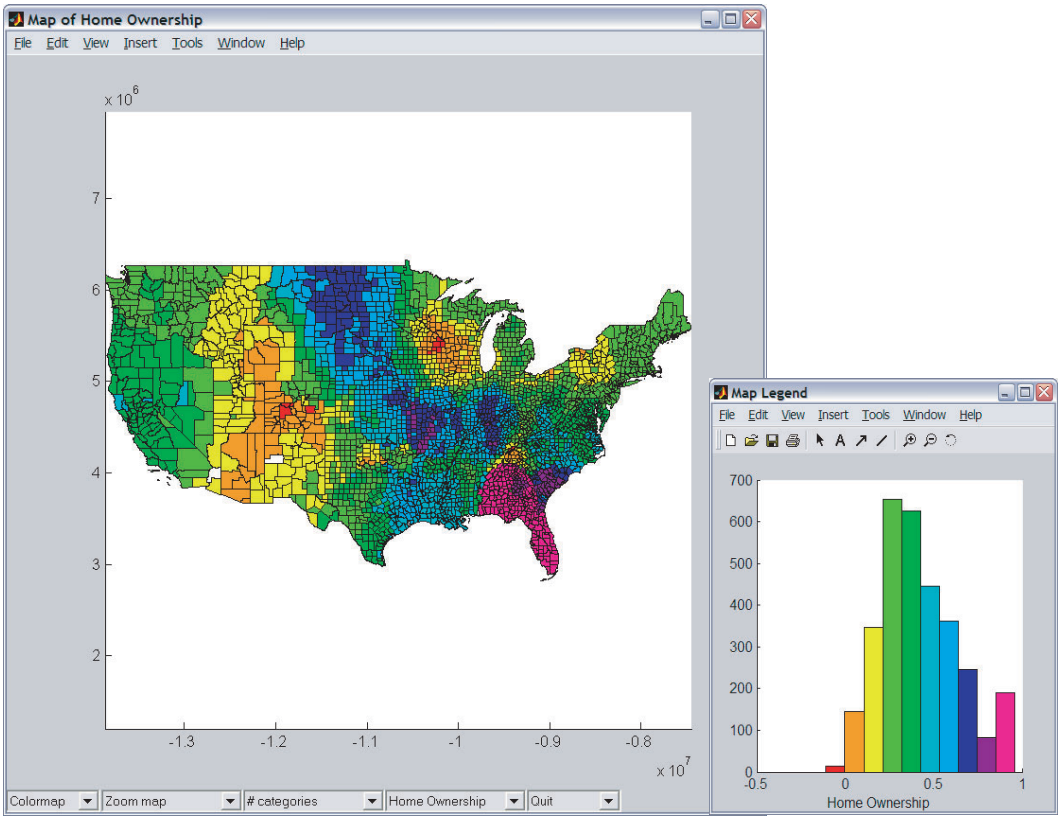

Figure 11: SALE parameter estimates for the home ownership variable

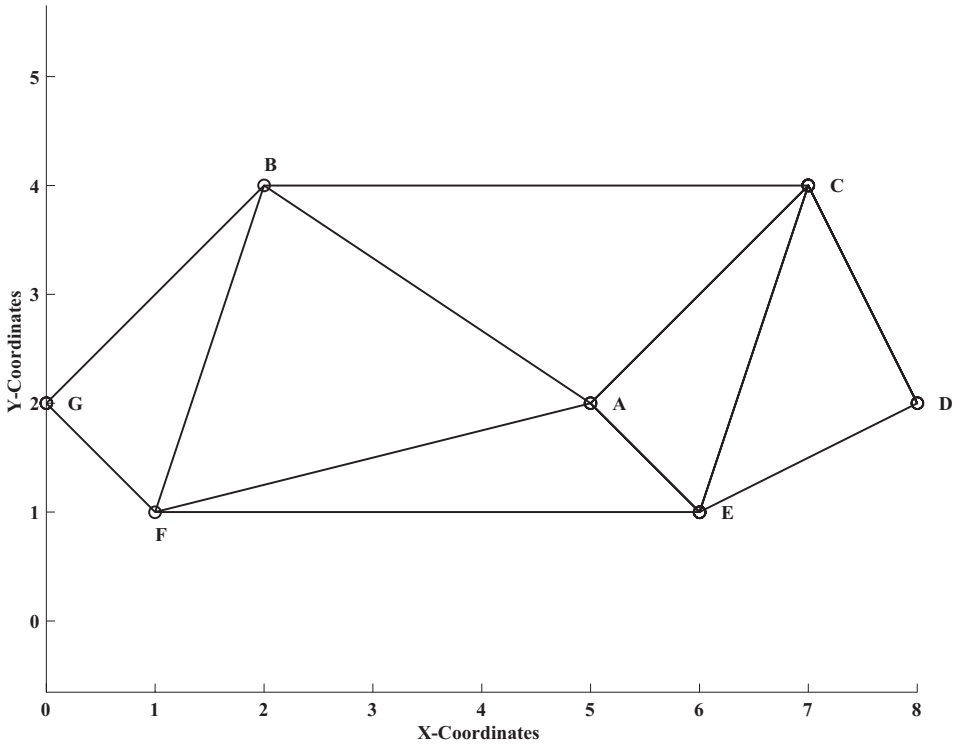

Figure 12: Delaunay triangularization