NCGIA

National Center for Geographic Information and Analysis

Spatial Data Analysis and GIS:

Interfacing GIS and Econometric Software

By

Luc Anselin, Sheri Hudak and Rustin Dodson
University of California, Santa Barbara

Technical Report 93-7
May 1993

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION
- BACKGROUND AND MOTIVATION
- ORGANIZATION OF THE REPORT
- ECONOMETRIC SOFTWARE
  - Versions
  - Hardware
  - Problem Size Limitations
  - Executing the Routines
    - Gauss
    - Limdep
    - Rats
    - Shazam
    - Splus
- REFERENCES

CHAPTER 2: EXTRACTING SPATIAL WEIGHTS MATRICES FROM A GIS
- INTRODUCTION
- OVERVIEW OF THE ROUTINES
  - General Characteristics
  - C Language Programs
  - Arc/Info AML Macros
- REFERENCES
- RASCONT.C - contiguity-based spatial weights matrices from raster polygons
- RASDIST.C - distance-based spatial weights matrices from raster polygons
- CONWEIGHT.AML - contiguity-based W-matrices from Arc/Info
- DISWEIGHT.AML - distance-based W-matrices from Arc/Info
- SP2FULL.C - converts sparse matrices to full format

CHAPTER 3: PRE-PROCESSING AND MANIPULATING SPATIAL WEIGHTS MATRICES
- INTRODUCTION
- OVERVIEW OF THE ROUTINES
- REFERENCES
- THE WRSTD.* AND EIGENW.* PROGRAMS
  - Table 3.1 - Gauss: WRSTD.GAS
  - Table 3.2 – Gauss: EIGENW.GAS
  - Table 3.3 - Limdep: WRSTD.LIM
  - Table 3.4 - Rats: WRSTD.RAT
  - Table 3.5 – Shazam: WRSTD.SHA
  - Table 3.6 - Splus: WRSTD.SPL
  - Table 3.7 - Splus: EIGENW.SPL

CHAPTER 4: REGRESSION DIAGNOSTICS
- INTRODUCTION
- METHODOLOGICAL BACKGROUND
- OVERVIEW OF THE ROUTINES
- REFERENCES
- THE REGDIAG.* PROGRAMS: REGRESSION DIAGNOSTICS

Table 4.1 – Gauss: REGDIAG.GAS
Table 4.2 – Limdcp: REGDIAG.LIM
Table 4.3 - Rats: REGDIAG.RAT
Table 4.4 – Shazam: REGDIAG.SHA
Table 4.5 - Splus: REGDIAG.SPL

CHAPTER 5: MAXIMUM LIKELIHOOD ESTIMATION OF SPATIAL AUTOREGRESSIVE MODELS
INTRODUCTION
METHODOLOGICAL BACKGROUND
OVERVIEW OF THE ROUTINES
REFERENCES
THE MLYX.* PROGRAMS: SPATIAL LAG MODEL
Table 5.1 - Gauss: MLYX.GAS
Table 5.2 - Limdep: MLYX.LIM
Table 5.3 - Rats: MLYX.RAT
Table 5.4 - Shazam: MLYX.SHA
Table 5.5 - Splus: MLYX.SPL

CHAPTER 6: MAXIMUM LIKELIHOOD ESTIMATION OF MODELS W! SPATIAL AUTOREGRESSIVE ERRORS
INTRODUCTION
METHODOLOGICAL BACKGROUND
OVERVIEW OF THE ROUTINES
REFERENCES
THE MLER.* PROGRAMS: SPATIAL ERROR MODEL
Table 6.1 - Gauss: MLER.GAS
Table 6.2 - Limdep: MLER.LIM
Table 6.3 - Rats: MLER.RAT
Table 6.4 – Shazam: MLER.SHA
Table 6.5 - Splus: MLER.SPL

APPENDICES
APPENDIX 1: CONTENTS OF THE INCLUDED DISKETTE
GIS: The GIS W-matrix extraction programs
ROUTINES: The statistical routines
COLUMBUS: The Columbus, Ohio dataset
APPENDIX 2: OVERVIEW OF STEPS FOR USING THIS REPORT
APPENDIX 3: SPARSE MATRIX FILE FORMATS
Sparse Binary Format
Sparse General Format
APPENDIX 4: SOURCE CODE FOR CHAPTER 2 ROUTINES
RASCONT.C
RASDIST.C
SP2FULL.C
INF2SMF.C
CONWEIGHT.AML
DISWEIGHT.AML
APPENDIX 5: MAP OF COLUMBUS, OHIO NEIGHBORHOODS

CHAPTER 1
INTRODUCTION

BACKGROUND AND MOTIVATION

It is often claimed that the implementation of spatial econometric methods in empirical work in mathematical and economic geography, regional science and urban economics has suffered from a lack of available and easy to use software. In this report, we present software code to alleviate this situation. Specifically, we outline routines to interface between a statistical or econometric package and two forms of geographic information systems: the Arc/Info system (on a Unix workstation platform) and a number of raster-based systems (Idrisi, OSU-Map-for-the-PC, and numerous other systems which support a common raster data structure). We also provide code to implement a number of spatial econometric techniques by means of the software packages Gauss (from Aptech Systems, Inc.), Splus (from Statistical Sciences, Inc.), Limdep (Greene, 1992), Shazam (White, 1988), and Rats (from VAR Econometrics, Inc.).

The present report complements two earlier publications. In Anselin and Hudak (1992), we reviewed some recent efforts to deal with the dissemination of software for spatial data analysis and outlined the practical issues encountered when spatial econometric techniques are implemented with existing commercial software. In Anselin, Dodson and Hudak (1993), we focused more specifically on the integration between GIS and spatial data analysis and how this may be achieved with existing software tools. In this report, the complete source code is provided that is needed to implement the techniques and interface outlined in the previous publications.

We chose to focus on three aspects of spatial econometric analysis: (1) the extraction of information on the spatial arrangement of the observations in the form of a spatial weights matrix, and the manipulation of this matrix; (2) the computation of diagnostics for spatial dependence in regression models; and (3) the estimation of spatial regression models, i.e., of specifications that incorporate a spatial autoregressive dependent variable or a spatial autoregressive structure for the error terms. These three aspects form the core of a spatial regression analysis to deal with a wide range of situations encountered in empirical work. The routines outlined in this report form a subset of the functionality included in the *SpaceStat* package (Anselin, 1992a). An earlier version of the Gauss routines appeared in Anselin (1989). We have made a conscious effort to try to keep the logic and organization of the routines similar between the various software packages. The starting point was the Gauss code of the *SpaceStat* package. The code for the other software packages was obtained by translating the Gauss instructions and making small adjustments where necessary. Further details on the code in Splus, Limdep, Shazam and Rats is given in Hudak (1992). The GIS interface code is based on the work outlined in Dodson (1993). It can run by itself, but has been adapted to facilitate interaction with the econometric software packages. All code in this report is provided as is and is not optimized for each individual software package.

The methods implemented in the routines are based on the spatial econometric techniques presented in Anselin (1988) and on the integration of GIS and spatial data analysis outlined in Anselin and Getis (1992). We do not include a comprehensive technical discussion in this report and refer to Anselin (1988) for a more extensive treatment. An introductory overview of many concepts discussed in this report is given in Anselin (1992b), to which we also refer for more extensive illustrations.

ORGANIZATION OF THE REPORT

The remainder of the report consists of five chapters. The first is devoted exclusively to the interface between a GIS and spatial data analysis. The others deal with each of the three aspects of spatial econometric analysis outlined above. The code for the GIS interface is listed in Appendix 4. The code for the various spatial econometric operations is listed in each chapter, for each of the five software packages. Due to space constraints, a complete illustration of each individual routine could not be provided. Instead, the results of a fully worked example are given in Appendix 1, where the output is listed from *SpaceStat* for the analysis of crime data for neighborhoods in Columbus, Ohio. This is the same data used to illustrate the spatial econometric techniques in Anselin (1988) and the GIS interface in Anselin, Dodson and Hudak (1993). This output may be used as a benchmark for comparison. The results for all routines should be the same as those given in Appendix 1, except for the nonlinear optimization in Chapters 5 and 6. The latter vary slightly by package, as discussed in more detail in Anselin and Hudak (1992).

In the remainder of this introductory chapter, we outline some summary characteristics of the econometric software packages that are considered. We do not provide extensive details, since we assume that the reader is familiar with this software.

Notational Conventions

In the following chapters, **boldface** print will denote variable and matrix names that can be found in the tables of code listed at the end of each chapter. Actual command names from the packages will be *italicized* in the report. The DOS wildcard character, *, will be used to facilitate the discussion of a set of routines. Program names have been standardized across the packages with the file name extension used to designate the package. For example, *wrstd.** will be used to denote the programs that standardize the rows of a spatial weights matrix, respectively *wrstd.gas* (for Gauss), *wrstd.lim* (for Limdep), *wrstd.rat* (for Rats), *wrstd.sha* (for Shazam), and *wrstd.spl* (for Splus).

In the tables of source code, <u>underlined</u> items are those which must be changed for each individual data set. The items listed in the tables pertain to the example data set with the Columbus crime data.

ECONOMETRIC SOFTWARE

Versions

The routines presented in this report were developed and tested for the following versions of the econometric software:

- Gauss386, version 2.2 and 3.0
- Limdep, version 6.0
- Rats, version 3.10
- Shazam, version 6.1
- Splus, version 3.0

Since the software is not optimized for each of the individual software packages, it may also run on later versions than the ones used for development. However, small adjustments may be needed in each individual case.

Hardware

All routines except those for Splus were developed and tested on an IBM PS/2 compatible machine, with a 386 CPU and mathematical co-processor and 4 megabytes of RAM. Some routines will run on 286 AT class machines as well, and some may run with small modifications on versions of the software for other platforms (e.g., Shazam on a mainframe, or Rats on a Macintosh). The routines in Splus were developed and tested on a Sun Sparc2 workstation running SunOS 4.1.1 and Openwindows 2.

Problem Size Limitations

Each software package differs in the maximum problem size that can be analyzed. These differences are due to both hardware and software requirements. We refer you to the respective manuals and reference guides for details. In this respect, it is important to keep in mind that the spatial weights matrix used in the various analyses has a dimension which corresponds to the square of the number of observations.

Executing the Routines

The routines for the extraction of a spatial weights matrix from a GIS (in Chapter 2) are self-contained. They consist of a combination of C programs and Arc/Info Macro Language (AML) commands. The latter must be executed from within Arc. The interface with the raster GIS runs as a program separately from the GIS.

The routines that implement the spatial data analysis must be executed from within the respective software. For Gauss and Splus, which contain extensive programming capabilities, the routines are designed as procedures (Gauss) or functions (Splus) that must be called from within a main program. The complete set of Gauss procedures or Splus functions could also be combined into a library, from which the routines could be called directly (refer to the respective manuals for technical details). The routines for Limdep, Rats, and Shazam are presented as batch files, since this is the most effective (and intuitive) way to carry out the analysis using these packages.

Some small variations between the execution in each of the software packages are listed below. For notational convenience, batch, function, and output filenames will be enclosed in angle brackets <> and referred to by the generic name *prog*. For Gauss and Splus, function or procedure arguments will be denoted as *in1*, *in2*, etc.

Gauss

Gauss procedures are called from within a main program. There are four ways to call a procedure, depending on the number of items that are returned:

```
<prog.gas>(in1,in2);
```

for a procedure with no returns.

```
call <prog.gas>(in1,in2);
```

to ignore the returns of a procedure (e.g., when the results are written to the screen or to a file in the procedure and need no further manipulation in the main program).

```
out1=<prog.gas>(in1,in2);
```

for a procedure with a single return **out1** (a scalar, vector or matrix).

```
{ out1,out2 }=<prog.gas>(in1,in2);
```

for a procedure with multiple returns.

The arguments in1, in2, etc. must be defined in the main program before they can be passed to the procedure.

The procedure must either be incorporated explicitly in the file that contains the main program (preceding the main program), or must be included by reference by means of the Gauss *include* command. This *include* command pertains to the file with the source code of the procedure, and must be listed as follows:

```
#include <prog.src>
```

When the source code of the procedure has been compiled (using the Gauss *compile* command) into a file <prog.gcc> or <prog.gci>, it can be incorporated by means of the *use* command:

```
use <prog.gcc>
```

Note that only one *use* command may be included and that it must be on the first line of the program file. The pre-compilation of the Gauss routines will speed up execution considerably.

The convention used in this report is that no listing of results is carried out in the Gauss procedures. All results are passed back to the main (or calling) program, where the listing must be implemented explicitly by means of the standard Gauss *print* command.

Limdep

To execute a batch routine in Limdep, the following must be typed at the DOS prompt:

```
limdep <prog.lim> <prog.out>
```

Using this method, output does not pause on the screen and require a carriage return to advance. A possible drawback of this approach is that upon completion of the routine, limdep is exited. However, in some instance this may be desirable in order to examine the output file.

To avoid exiting limdep after the batch file run, the following approach could be used instead:

```
open ; output = <prog.out> $
fast $   [to avoid carriage returns]
open ; input = <prog.lim> $
```

Rats

To execute a batch routine in Rats, the following must be typed at the DOS prompt:

```
rats386 <prog.rat> <prog.out>
```

Shazam

To execute a batch routine in Shazam, the following must be typed at the DOS prompt:

```
shazam <prog.sha> <prog.out>
```

Splus

In Splus all functions must be *source*d (compiled) before they can be executed. This is accomplished by typing the following at the ">" prompt:

```
source("<prog.spl>")
```

To execute a function that returns something, use the assignment operator "_" to assign the function output to an Splus object, e.g.:

```
<prog.out>_<prog.spl>(in1,in2,in3)
```

Splus functions that have no return value send all output to the screen. To trap screen output into a file, the *sink( )* function must be used, e.g., as in:

```
sink("prog.out")  [send output to the file "prog.out"]
<prog.spl>(in1,in2,in3)  [execute the function]
sink()  [turn off the sink function]
```

REFERENCES

Anselin, Luc, 1988. *Spatial econometrics: methods and models* (Kluwer Academic Publishers, Dordrecht).

Anselin, Luc, 1989. Spatial regression analysis on the PC: spatial econometrics using Gauss (Department of Geography, University of California, Santa Barbara, CA).

Anselin, Luc, 1992a. *SpaceStat: a program for the statistical analysis of spatial data* (National Center for Geographic Information and Analysis, University of California, Santa Barbara, CA).

Anselin, Luc, 1992b. Spatial data analysis with GIS: an introduction to application in the social sciences (National Center for Geographic Information and Analysis Technical Report 92-10, University of California, Santa Barbara, CA).

Anselin, Luc and Arthur Getis, 1992. Spatial statistical and geographic information systems. *The Annals of Regional Science* 26, 19-33.

Anselin, Luc and Sheri Hudak, 1992. Spatial econometrics in practice: a review of software options. *Regional Science and Urban Economics* 22, 509-36.

Anselin, Luc, Rustin Dodson and Sheri Hudak, 1993. Linking GIS and spatial data analysis in practice. *Geographical Systems* 1 (in press).

Dodson, Rustin, 1993. Integrating GIS and spatial analysis: an implementation and appraisal of two prototype software linkages (Unpublished Masters Thesis, Department of Geography, University of California, Santa Barbara, CA).

Greene, William H., 1990. *Limdep user's manual and reference guide, version 6.0* (Econometric Software, Inc., Bellport, NY).

Hudak, Sheri, 1992. Spatial econometrics in practice (Unpublished Masters Thesis, Department of Geography, University of California, Santa Barbara, CA).

White, K.J., 1988. *Shazam: a comprehensive computer program for regression models* (version 6.0). Computational Statistics and Data Analysis 7, 102-103.

CHAPTER 2
EXTRACTING SPATIAL WEIGHTS MATRICES FROM A GIS

INTRODUCTION

A fundamental distinguishing characteristic of spatial data analysis is that the spatial arrangement of the observations is made explicit. This is formalized in a contiguity or spatial weights matrix, often denoted by W (for an extensive discussion, see Cliff and Ord, 1981). The rows and columns of this matrix correspond to the observations. The non-zero elements for a row-column pair reflect contiguous locations (in a so-called binary contiguity matrix), or the strength of the potential interaction between two locations (in a so-called general spatial weights matrix). Two regions (or areal units of observation) are defined as being contiguous when they have a common border of non-zero length. Contiguity may also be based on a distance criterion, i.e., when the centroids (or other important points) of two regions are less than a critical distance apart. The elements of a general weights matrix could be any meaningful indication of potential interaction. Often, the share of the common border in the total border length is used. Clearly, there is a degree of arbitrariness associated with the specification of a spatial weights matrix for a set of observations, which has given rise to an extensive discussion in the literature (for a review, see Anselin, 1988, 1992).

For applications with a small number of observations, contiguity can be derived from a visual inspection of boundaries on a map. However, for most medium and large size data sets, manual creation of a weights matrix is not practical. Instead, a geographic information system should be used, since the characteristics of the spatial arrangement of the observations are typically stored together with the data themselves. The way in which such information is contained in the GIS depends on the underlying data model (Goodchild, 1992). In a vector data model, adjacency information is typically stored explicitly and may be accessed by means of a simple query. On the other hand, in a raster data model, adjacencies are stored implicitly and cannot be retrieved directly. In order to obtain this information for so-called raster polygons (groups of like-valued grid cells), the grid cell values of neighboring cells must be compared explicitly by means of a rather slow search process (for technical details, see Dodson, 1993).

In this chapter, we outline routines for extracting spatial weights matrices from several familiar GIS software packages, including Arc/Info and a number of raster based packages. The spatial weights may be based on contiguity, on distance, or, for Arc/Info only, on the share of the common border between two areal units.

OVERVIEW OF THE ROUTINES

General Characteristics

This chapter outlines four routines which extract spatial weights matrices from a GIS. The complete listing of the source code is given in Appendix 4. The first two routines are the C language programs RASCONT.C and RASDIST.C. These programs extract weights matrices directly from raster data in both Idrisi and OSU Map-for-the-PC formats. In addition, these programs both have a "generic" option, which allows any raster data in uncompressed row scan order[1] to be read (e.g., raster data from the GRASS GIS package).

As input, the RASCONT and RASDIST programs both expect a raster of integer values. These may be individual pixels, or so-called "raster polygons", which are groups of like-valued cells. These groups need not be contiguous--the programs treat all cells with the same value as belonging to the same polygon. As output, RASCONT produces a contiguity-based spatial weights matrix and RASDIST produces a distance-based spatial weights matrix in sparse matrix format.

[1]These data may be in binary or ASCII format. The user must provide the following parameters: number of rows/columns, number of bytes-per-cell (for binary formats [1, 2, or 4]), and number of bytes of header data to skip (if applicable).

The final two routines in this chapter, CONWEIGHT.AML and DISWEIGHT.AML, extract spatial weights matrices from datasets in Arc/Info format. These routines are written in the Arc Macro Language (AML). However the CONWEIGHT macro requires access to an executable C program called INF2SMF, which is described in Appendix 4. As input, CONWEIGHT takes a polygon coverage[2], and it outputs one of three types of contiguity-based spatial weights matrices. The DISWEIGHT routine inputs either a point or polygon coverage, and it outputs a distance-based spatial weights matrix.

All four of the above routines offer a great degree of flexibility with respect to the choice of the set of spatial objects (points or polygons) which are included in the weights matrix. Each object is referenced by an integer value known as the polygon-ID[3]. The weights matrix extraction routines ignore polygon-IDs of zero. Therefore, in order to identify a subset of objects for analysis, one can use standard GIS procedures[4] to assign polygon-IDs of zero to objects which are not of interest. As output, all routines produce ASCII files in one of two sparse matrix formats. These formats are described in Appendix 3. The final routine in this chapter, SP2FULL.C, should be used to convert the sparse matrix format to a full format, which is expected by the routines in Chapters 3-6.

C Language Programs

Compilers

These programs are written using only ANSI C functions contained in Kernighan and Ritchie, (1988). All have been successfully compiled and run under the DOS, SunOS, and IBM AIX operating systems, with the exception of INF2SMF.C, which is not intended for a DOS environment and was not tested for that operating system.

Memory

The DOS-compatible programs were designed to be parsimonious with memory usage, however they will fail on extremely large files. A typical PC will fail on an 8,000 by 8,000 raster, for example. Nevertheless, the programs should work fine for most applications. Another consequence of designing software under a memory constraint is that speed is sacrificed. This will be apparent when processing very large files.

Return values to the invocation environment

All of the C-programs return the value 0 on successful completion and -1 on error.

Arc/Info AML Macros

The AML macros were developed under Unix Arc/Info version 5.0.1. They have been run successfully under version 6.0.1 as well. These macros are designed for Unix versions of Arc/Info only.

The following sections describe each W-matrix extraction routine in detail. The full source code for each routine is provided in Appendix 4.

REFERENCES

Anselin, Luc, 1988. Model validation in spatial econometrics: a review and evaluation of alternative approaches. *International Regional Science Review 11*, 279-316.

Anselin, Luc, 1992. Spatial data analysis with GIS: an introduction to application in the social sciences (National Center for Geographic Information and Analysis Technical Report 92-10, University of California, Santa Barbara, CA).

[2]*Coverage* is Arc/Info terminology for "data layer" or "theme".
[3]In some cases the polygon-ID may refer to point objects, however the techniques described in this report are mainly geared toward polygon-based data.
[4]For example, the RESELECT and CALCULATE functions in Arcplot, or the RECLASS function in Idrisi.

Cliff, A. and J.K. Ord, 1981. *Spatial processes: models and applications* (Pion, London).

Dodson, Rustin, 1993. Integrating GIS and spatial analysis: an implementation and appraisal of two prototype software linkages (Unpublished Masters Thesis, Department of Geography, University of California, Santa Barbara, CA).

Goodchild, M.F., 1992. Geographical data modeling. *Computers and Geosciences* (in press).

Kernighan, Brian W., and Ritchie, Dennis M., 1988, *The C Programming Language, Second Edition*, Prentice Hall, New Jersey.

RASCONT.C - CONTIGUITY-BASED SPATIAL WEIGHTS MATRICES FROM RASTER POLYGONS

Synopsis

This program reads a file of "raster polygons" (groups of like-valued cells) and produces a binary spatial weight matrix based on contiguity. The program accepts files of various formats, and it outputs the contiguity matrix as ASCII files in both full and sparse formats.

Usage

The program syntax is:

rascont <input file type>

where <input file type> refers to one of the letters "i", "o", or "g". These letters represent Idrisi, OSU-Map, and generic file formats, respectively. For the "g" option, the program will prompt for additional information, which is described in GENERIC FORMATS below.

Input File Formats

The program automatically reads files from certain GIS packages, and it provides a generic mode for reading many additional file formats. In general, the raster polygon file must contain integer cell values which represent polygon-ID numbers. These polygon-IDs need not be consecutive--they may be census tract numbers or arbitrarily-assigned county codes, for example. The program ignores polygon-IDs of zero. Thus, background cells or areas of "no data" should be coded as zero. Also note that polygons need not be contiguous. For example, the two peninsulas of the state of Michigan, USA, could be stored as two separate polygons with the same poly-ID, and RASCONT would correctly record all neighbors. The supported input file formats are:

IDRISI: This option works with IDRISI images in either ASCII or uncompressed binary format. Images must contain integer rather than real-number data. This option works with IDRISI's newest version 4.0 as well as older versions.

OSU MAP-for-the-PC: This option works with files from OSU MAP versions 3.0 and later.

GENERIC FORMATS: This option will read virtually any uncompressed raster which is in left-to-right row scan order and which contains integer polygon-IDs. For ASCII raster files, the integer cell values must be separated by white space only (space, tab, or end-of-line characters). For binary raster files, the program works with one, two, and four-byte integers, which may not be separated by any other characters or bytes. The generic format mode additionally prompts the user for the number of rows/columns, file format (ASCII or binary), number of bytes-per-cell (for binary file formats), and an optional number of header bytes which will be skipped over when the file is read.

Prompts

Upon execution, RASCONT prompts the user for the following:

Raster filename - the name of the file which contains the raster data. If this file is not in the current directory, include the full path specification. For Idrisi and OSU formats, do not include a filename extension. For generic format, include the entire filename.

Output Files

Program output consists of two ASCII files. Both files contain the same contiguity information, but they are in different formats.

sparse.asc - a file of contiguities in sparse binary matrix format.

full.asc - a complete NxN binary contiguity matrix. The first number in the file refers to the dimension of the following matrix. The next numbers are either zeros or ones (1 = contiguity) in left-to-right row scan order, separated by a space, with no end-of-line characters (the file is one long line). The full matrix is presented such that the polygon-IDs are sorted in increasing order. (NOTE: This one-line format may not be readable by some programs. If this is the case, use the program SP2FULL to convert the sparse.asc file to full matrix format.

Notes

How the program works

The program makes a first pass through the raster in order to create a sorted list of unique polygon-ID numbers. This list is then printed to the screen as an error check. A second pass through the raster examines cell values for contiguity, using a queen's case (8-neighbor) criterion. When scanning for contiguities, the program stores only three rows of the image in memory at any given time. This allows the program to run on hardware with limited memory.

Speed/Memory

Since the program was designed to work with limited memory, it is not fast on large rasters. On UNIX platforms, the program should work with rasters of virtually any size. On DOS platforms, the program will run out of memory on extremely large rasters (somewhere around 8,000x8,000 cells).

Maximum number of polygons

Currently, the program can handle a maximum of 1,000 polygons. To alter this limitation, the value of the symbolic constant "FULL" in the source code must be changed and the program must be re-compiled. The FULL constant must be equal to or larger than the number of polygons in an input raster.

Range limits for polygon-IDs

The program uses the *int* variable type to store polygon-IDs. On systems using two bytes for an *int* (most DOS systems), polygon-IDs are limited to the range 0 to 32,768. On four-byte systems (most UNIX systems), polygon-IDs may range from 0 to 2.14E+09.

Warnings

Polygon <n> has no neighbors - The program prints a warning to the screen if it detects a polygon with no neighbors (This would equate to a row of zeros in the full contiguity matrix).

RASDIST.C - DISTANCE-BASED SPATIAL WEIGHTS MATRICES FROM RASTER POLYGONS

Synopsis

This program reads a file of "raster polygons" (groups of like-valued cells) and produces a distance-based spatial weight matrix based on the locations of polygon centroids. The program accepts several input file formats and allows the user to specify a distance threshold and distance-decay power. The program ouput consists of two ASCII files: a sparse weight matrix and a list of polygon centroid coordinates.

Usage

The program syntax is:

rasdist <input file type>

where <input file type> refers to one of the letters "i", "o", or "g". These letters represent Idrisi, OSU-Map, and generic file formats, respectively. For the "g" option, the program will prompt for additional information, which is described in GENERIC FORMATS below.

Input File Formats

The program automatically reads files from certain GIS packages, and it provides a generic mode for reading many additional file formats. In general, the raster polygon file must contain integer cell values which represent polygon-ID numbers. These polygon-IDs need not be consecutive--they may be census tract numbers or arbitrarily-assigned county codes, for example. The program ignores polygon-IDs of zero. Thus, background cells or areas of "no data" should be coded as zero. The supported input file formats are:

IDRISI: This option works with IDRISI images in either ASCII or uncompressed binary format. Images must contain integer rather than real-number data. This option works with IDRISI's newest version 4.0 as well as older versions.

OSU MAP-for-the-PC: This option works with files from OSU MAP versions 3.0 and above.

GENERIC FORMATS: This option will read virtually any uncompressed raster which is in left-to-right row scan order and which contains integer polygon-IDs. For ASCII raster files, the integer cell values must be separated by white space only (space, tab, or end-of-line characters). For binary raster files, the program works with one, two, and four-byte integers, which may not be separated by any other characters or bytes. The generic format mode additionally prompts the user for the number of rows/columns, file format (ASCII or binary), number of bytes-per-cell (for binary file formats), and an optional number of header bytes which will be skipped over when the file is read.

Prompts

Upon execution, RASDIST prompts the user for the following:

Raster filename - the name of the file which contains the raster data. If this file is not in the current directory, include the full path specification. For Idrisi and OSU formats, do not include a filename extension. For generic format, include the entire filename.

Distance threshold - the cutoff distance beyond which the spatial weight will be zero. The units for distance threshold should be in cell widths. For example, if your image contains 100 by 100 meter cells and you want a distance threshold of 1000 meters, use a distance threshold of 10 cells.

Distance-decay power - the inverse distance power for the distance-decay weight function, entered as a positive number. For example, a distance-decay power of 2 yields a $1/d^2$ distance function.

Output Files

The program produces two or three ASCII files as output:

dmat.asc - the distance-based weight matrix in sparse general format. This matrix should be row standardized in order to remove scale effects due to the distance units.

centroid.asc - a file containing polygon centroid coordinates. These coordinates are presented assuming that the grid origin (0,0) occurs at the lower-left corner of the raster and that each cell is a one-unit square (i.e. the units are in terms of image column/row position, not in the coordinate system used in the GIS). This file is in a format which is directly readable by the spatial statistical package *SpaceStat* (Anselin, 1992):

```
<N> 3 ID X Y[5]
<polygon-ID> <X-coordinate> <Y-coordinate>
            .
            .
            .
            (etc.)
```

centroid.vec - This file is only created when RASDIST is used with the Idrisi option, "i". This is an ASCII file of centroid coordinates in Idrisi point vector format. To view this file in Idrisi, use the Idrisi DOCUMENT module to make a CENTROID.DVC documentation file. NOTE: the units for the centroids are in terms of image row and column values. To make the centroids display properly, you might have to change the image min. X/max. X to range from 0 to #columns and the image min. Y/max. Y to range from 0 to #rows.

Notes

How the program works

The program makes a first pass through the raster in order to create a sorted list of unique polygon-ID numbers. This list is then printed to the screen as an error check. A second pass through the raster calculates centroid locations for each polygon. Note: RASDIST computes Euclidian distances. For very large study areas, these distances can be significantly wrong.

Speed/memory

Since the program was designed to work with limited memory, it is not fast on large rasters. On UNIX platforms, the program should work with rasters of virtually any size. On DOS

[5]*<N>* refers to the number of observations. The remaining items (*3 ID X Y*) indicate that the file contains *3* variables, called *ID*, *X*, and *Y*.

platforms, the program will run out of memory on extremely large rasters (somewhere around 8,000x8,000).

Maximum number of polygons

Currently, the program can handle a maximum of 1,000 polygons. To alter this limitation, the value of the symbolic constant "FULL" in the source code must be changed and the program must be re-compiled. The FULL constant must be equal to or larger than the number of polygons in an input raster.

Range limits for polygon-IDs

The program uses the *int* variable type to store polygon-IDs. On systems using two bytes for an *int* (most DOS systems), polygon-IDs are limited to the range 0 to 32,768. On four-byte systems (most UNIX systems), polygon-IDs may range from 0 to 2.14E+09.

Polygon centroids

The algorithm used for calculating polygon centroids is to simply take the mean row and mean column values for all of a polygon's cells. For odd-shaped polygons, the centroid may lie outside the polygon's boundary. In the absence of additional information, however, the centroid is the most representative point with which to describe a polygon. The program calculates a single centroid for each polygon-ID. Thus polygons need not be contiguous. For example, if two islands were coded with the same polygon-ID, a single centroid would be computed rather than one centroid per island. This single centroid would be located at the center of gravity of the two islands, and therefore may lie in the water between the two islands.

Warnings

Distance less than 1.0 encountered - The distance between two polygon centroids is less than one unit; thus the weight matrix will contain weights greater than one.

Zero distance encountered--using 0.00000001 instead - In the rare case where two polygon centroids occupy exactly the same point, the program uses a distance value of 0.00000001 in order to avoid division by zero.

CONWEIGHT.AML - CONTIGUITY-BASED W-MATRICES FROM ARC/INFO

Synopsis

This macro inputs an Arc/Info polygon coverage and outputs a contiguity-based spatial weight matrix. Three contiguity options are available: binary queen's case, binary rook's case, and contiguity based on common boundary length (the length of the border shared by a polygon and its neighbor). A related program, INF2SMF, converts the output file from CONWEIGHT into sparse matrix format. The INF2SMF program is called by CONWEIGHT and thus its use should be transparent to the user. **NOTE: This routine requires access to the executable program "inf2smf".** Source code to the INF2SMF.C program is included in this report. The program must be compiled and the resulting executable must be called "inf2smf" and placed on the Unix search path.

Usage

CONWEIGHT runs at the ARC level within Arc/Info. The command syntax is

```
&r conweight <cover> <item>
```

A description of the arguments follows:

<cover> - An existing Arc/Info coverage with both polygon and line topology.

<item> - A valid item name from the <cover>.PAT file. This item will be the polygon-ID which is used to reference polygons in the output file. The macro ignores polygons with polygon-IDs equal to zero. Thus background and 'no data' polygons should be coded with an <item> value of zero. Polygon-IDs may take on integer or real number values.

Options

Upon execution, the user is presented with an option menu. The macro provides three options for the type of spatial weight matrix to be created. All are based on contiguity:

Binary queen's case contiguity - This option creates a binary spatial weight matrix based on polygons which share either a common arc or a common node. This is analogous to the "queen's case" on a regular grid, as it includes "node-adjacent" polygons which are contiguous at only one point (as in diagonal squares on a chessboard).

Binary rook's case contiguity - This option creates a binary matrix as above, but only considers polygons which share a common arc. This is analogous to the "rook's case" on a regular grid, where diagonal squares are not included.

General rook's case based on common boundary length - This option creates a general spatial weight matrix (a matrix whose elements are not simply zero or one). The weight for each polygon/neighbor contiguity is calculated as the percentage of the common boundary length with respect to the perimeter of the polygon (not the neighbor polygon). Calculation of the weights in this manner causes the resulting matrix to by asymmetric. This option uses rook's case contiguity, as node-adjacent polygons would have a common boundary length of zero.

Output file

CONWEIGHT outputs data to the default INFO print file, which is called **arcnsp** and resides in the INFO directory of the current active workspace. This file is then read by the *inf2smf* program, which creates the final output file called **con_mat.asc** in the current workspace directory. The output file is in sparse matrix format, binary or general, depending on the option chosen.

Notes

- The macro adds an alternate item name, "NEIGHBOR#", to the <cover># item in the <cover>.PAT file. The macro will not work if an item name "NEIGHBOR#" exists anywhere else in the <cover>.PAT file.

- The macro works fine for "island" polygons within other polygons (e.g. the countries of Lesotho and South Africa). However, true islands (polygons isolated within the background polygon) will not show up in the output file, since they have no contiguous neighbors.

Files created and left in the user's workspace

The macro adds an alternate item name called "NEIGHBOR#" to the <cover>.PAT file. In addition, one or more INFO files are created. If these files already exist, they will be deleted. The macro does not delete them at the end of its execution because they may be of interest to the user. The following is a description of these INFO files:

<cover>.PNL: Polygon neighbor list - each record in this file contains three items: the <cover># of a polygon and a neighbor (from the .PAT), and a weight based on common boundary length.

The following files are only created under the queen's case option:

<cover>.NAL: Node arc list - each record contains two items: a node and an arc identifier from the .AAT.

<cover>.VAL: Valency list - two items: a node identifier (from the .AAT) and the valency of that node.

<cover>.HVN: High valency node list - this file contains information on all nodes with a valency greater than three. These high valency nodes are the locations where polygons are contiguous at a node but have no common arc. Each record in the file has three items: a node identifier from the .AAT, and the <cover># of a polygon and a neighbor (from the .PAT). The node identifiers have been multiplied by -1 in order to flag the high-valency nodes in the temporary output file **arcnsp**.

DISWEIGHT.AML - DISTANCE-BASED W-MATRICES FROM ARC/INFO

Synopsis

DISWEIGHT produces a distance-based spatial weight matrix from a polygon (or point) coverage, based on a user-specified distance threshold and distance-decay power. For polygon coverages, distances are measured between polygon label points. The Arc command CENTROIDLABELS can be used to first move label points to true geographic centroid locations. Output is a spatial weight matrix in sparse general matrix format. DISWEIGHT uses the Euclidean distance metric.

Usage

The command syntax is:

```
&r disweight <cover> <item> <distance> {decay}
```

A description of the arguments follows:

<cover> - An existing Arc/Info coverage with a .PAT file.

<item> - A valid item name from the <cover>.PAT file. This item will be the polygon-ID which is used to reference polygons in the output file. The macro ignores polygons with polygon-IDs equal to zero. Thus background and "no data" polygons should be coded with an <item> value of zero.

<distance> - The distance threshold in coverage units. This is the maximum distance at which two points will be assigned a non-zero weight value.

{decay} - An optional positive integer representing the value of the distance-decay power. The default value is 1, which represents simple inverse distance ($1/d^1$).

Output file

dis_mat.asc - A distance-based spatial weight matrix in sparse general matrix format. This ASCII file prints the weight values with 15 decimal places. For very steep distance-decay functions (large values for the {decay} parameter), some weight values may be smaller than the precision and thus show up as zero.

Notes

- DISWEIGHT is sensitive to polygon label errors in the input coverage. To check for label errors, run the Arc command "LABELERRORS <cover>". The only output from the LABELERRORS function should be "Polygon 1 has 0 label points". If additional label errors exist, they must be fixed[6] before DISWEIGHT is run.

- The macro adds an alternate item name, "NEIGHBOR#", to the <cover># item in the <cover>.PAT file. The macro will not work if an item name "NEIGHBOR#" exists anywhere else in the <cover>.PAT file.

- The geographic centroid of a polygon will not necessarily fall within the boundaries of its polygon (e.g. the centroid of the country of Somalia is actually in Ethiopia), but in the absence of additional information, the centroid is arguably the most meaningful way to represent a polygon's location with a single point. NOTE: polygon label points created by Arc/Info are not necessarily true geographic centroids, as they are constrained to fall within their polygons' boundaries. The Arc command CENTROIDLABELS moves label points to geographic centroids.

SP2FULL.C - CONVERTS SPARSE MATRICES TO FULL FORMAT

Synopsis

This program works with two types of sparse spatial weight matrices: binary format and general format. Input is a sparse-format matrix, and output is an ASCII file containing a full NxN matrix.

Usage

The program syntax is:

```
sp2full <infile> <outfile>
```

Where <infile> is an existing sparse-format matrix file, and <outfile> is the name of the full matrix file to be created. If it exists, <outfile> will be overwritten.

Input file format

The program reads ASCII files in sparse binary or general matrix format. Polygon-ID and weight values may be integer or real numbers. In the case of real-number polygon-IDs, SP2FULL expects polygon-IDs to have the same number of precision digits (digits to the right of the decimal point). For real-number weight values, SP2FULL preserves the first seven significant digits. (Note that polygon-IDs do not need to have the same precision as the weight values.)

Output file format

SP2FULL outputs an ASCII file which contains the full NxN weight matrix in left-to-right scan order. This file contains end-of-line characters at the end of every line of screen display (every 80 characters). For sparse binary input files, the elements of the output matrix consist of the character "0" or "1". For sparse general input files, the output matrix elements are presented in scientific notation. For example, the weight value 1.234567

[6]Often a single run of the Arc command CREATELABELS will take care of all label errors.

would be presented as 1.234567E+00. The rows of the output matrix represent the N spatial observations (sorted in ascending order of polygon-ID), and the columns represent the neighbors of these observations. Thus, non-zero elements of the matrix represent some degree of spatial interaction between two observations. The diagonals of the matrix are zeros by convention.

Unconnected cells

If the full matrix output file is to contain unconnected cells (observations with no neighbors), the sparse matrix input file must follow these conventions (these are the same conventions used in the *SpaceStat* software):

Binary format

The value for <number of neighbors> must be zero, and the following line, where the neighbor-IDs would go, must be blank. For example, if polygon #32 had no neighbors, the following lines should occur in the input file:

```
32  0
<blank line>
```

NOTE: The above format convention is followed by the program RASCONT.C.

General format

A line in the input file must show the unconnected cell to be a neighbor of itself with a weight of zero. For example, if polygon #32 had no neighbors, the following line should occur in the input file:

```
32  32  0.0
```

NOTE: The above format convention is followed by the program RASDIST.C.

Limitations

- For large numbers of observations, the resulting full matrix file can be quite large, as it will contain N^2 elements.

CHAPTER 3
PRE-PROCESSING AND MANIPULATING SPATIAL WEIGHTS MATRICES

INTRODUCTION

Once the information for a spatial weights matrix has been obtained, either from an existing file, or from a file constructed by means of the routines outlined in Chapter 2, it is used in spatial data analysis in three respects. First, a weights matrix is needed to create spatially lagged variables, i.e., a product of the weights matrix W with a vector of observations on the dependent variable (e.g., y to yield Wy), the explanatory variables (X, to yield WX), or the residuals (e, to yield We). All five econometric software packages considered here have facilities for matrix algebra, so that the creation of a spatially lagged variable is easily implemented. In practice, the elements of the weights matrix are often used in row-standardized form to compute the spatial lag. In order to achieve this, each element of the matrix (w_{ij}) must be divided by the row sum ($w_{i\bullet} = S_j\, w_{ij}$) so that the row elements of the standardized matrix sum to 1. As a result, the spatial lag operation becomes essentially a weighted average of observations at neighboring locations (for an extensive illustration, see Anselin, 1992). The row-standardization can be carried out in each package, albeit with slight variations in the degree of complexity. This is discussed in more detail below.

A second instance where the spatial weights matrix is needed is to compute the various standardization coefficients used in the diagnostics for spatial autocorrelation, outlined in Chapter 4.

A third and most important role for the weights matrix is in the calculation of the Jacobian determinant $|I-\rho W|$ (with I as an identity matrix and ρ as the autoregressive coefficient) in the likelihood of spatial autoregressive processes (see Chapters 5 and 6). In the iterations of the nonlinear optimization used to obtain a maximum likelihood estimator, the magnitude of this Jacobian and/or its partial derivative needs to be computed for each value of the autoregressive parameter ρ. Clearly, this will be numerically demanding for even medium sized data sets, since the matrix $(I-\rho W)$ has a dimension of N by N, with N as the number of observations. However, as shown by Ord (1975), the determinant can be expressed in function of the eigenvalues of the weights matrix W, w_i:

$$|I-\rho W| = \Pi_i (1-\rho w_i)$$

Moreover, even though a row-standardized weights matrix is asymmetric, its eigenvalues are all real, and they are the same as those of a symmetric matrix W*, obtained as

$$W^* = D^{1/2} W D^{1/2},$$

where D is a diagonal matrix with elements composed of the inverse of the row sums $w_{i\bullet}$ of the matrix W (for details, see Ord, 1975). This has great practical importance, since neither Splus nor Rats have procedures for the computation of eigenvalues of a general real matrix, but only for a real symmetric matrix (the function *eigen* in both packages). In Shazam, there is also only one command for the eigenvalue calculation (*eigval*), but it handles the asymmetric weights matrix without problems. In Limdep, both symmetric (command *root*) and asymmetric matrices (command *cxrt*) can be handled, while in Gauss, a total of nine different eigenvalue commands are available, for real as well as complex matrices (eigenvalues of a general real matrix are obtained by means of the procedure *eigrg*).

In this chapter, we outline procedures to row-standardize a spatial weights matrix and to compute its eigenvalues. For use in later routines, whenever possible both the row-standardized weights and the eigenvalues are stored in files in the binary format used by the software package.

OVERVIEW OF THE ROUTINES

User Input: Loading the Spatial Weights Matrix

The routines in this chapter require the input of a square weights matrix whose row/column dimension is equal to the number of observations. This weights matrix is read from a separate file and must be constructed before any of the spatial analysis operations can be carried out. Typically, the spatial weights matrix will first be stored in sparse format and must therefore be converted to a full matrix. This may be done by means of the program *sp2full* (discussed in Chapter 2). The resulting ASCII matrix file is then used as input into each statistical package as a conventional matrix, using the appropriate commands for that package. These commands differ slightly between the packages and are summarized below. They are not listed separately, but are included in the user input part of the program listings in Tables 3.1 through 3.7. In the remaining chapters, we assume that the row-standardized weights matrix and the vector with its eigenvalues have been created in the format appropriate for each software package.

Gauss

The Gauss programming system contains a special data type for matrices, which allows them to be stored in double precision floating point format. An ASCII data set is converted to the binary format used by Gauss by means of the *load* and *save* commands. The easiest way to read an ASCII file is to consider it as a long vector, with the elements stacked row by row, and to load it into memory, as in:

load **wmat[] = file.asc**;

This vector can subsequently be converted to a matrix of the proper dimensions by means of the *reshape* command. For example, if the dimension of the matrix wmat is n by n, the n^2 by 1 vector is converted into a square matrix by the command:

wmat = *reshape*(**wmat,n,n**);

This matrix is saved to a file in the Gauss matrix format (characterized by the file extension .FMT) by a save command:

save **file = wmat**;

Once a matrix is saved in the Gauss matrix format, it may be loaded without the need to specify its dimensions, as:

load **wmat = file**;

For the procedure *wrstd* listed in Table 3.1, it is assumed that the matrix **wmat** has been loaded into memory in the main program.

Limdep

The spatial weights matrix is read as an ASCII file for input into Limdep (using the *read* command). Limdep also reads binary, DIF, spreadsheet (.WKS), or Fortran formatted files. Such types need a proper *;format = type* statement in the *read* command.

In order to facilitate the reading of a large matrix stored as a file, its columns are read in as variables, specified by means of the *namelist* command. In the routine listed in Table 3.3, these have been designated as **w1***, which takes advantage of the Limdep wildcard character (*) in the *namelist* command. The *namelist* command is used to give a collection of variables (**w10** to **w58** for the 49 observations of the example Columbus Ohio dataset) a common name to use in following commands. However, each *namelist* can only contain 40 names, so that some adjustments must be made to deal with weights matrices of a higher dimension. In the routines listed here, the example matrix of 49 observations was partitioned into two *namelist* commands, one for **wt1** (the first 40 columns) and the other for **wtr** (the next 9 columns). Other dimensions may need similar adjustments. The resulting two (or more) namelists (**wtl, wtr**) are copied to the matrix work area via the *copy* command (as in line 19 of Table 3.3) and the newly created matrices (**mwtl, mwtr**, etc.) are treated as submatrices to a larger matrix **wmat**. The submatrices are joined together with **mwtl** on the left and **mwtr** on the right (as in line 21 of Table 3.3).

Rats

A square spatial weights matrix is read into the Rats code by means of the *read* command, after the matrix has been declared (*declare rectangular* on line 13 of Table 3.4). It is important that the length of each input line in the ASCII file does not exceed eighty columns. However, a row of the matrix may be contained on more than one line[1]. If the lines are too long, an "End of File Encountered" and "Execution Terminated" error message will be generated by Rats.

Shazam

A square spatial weights matrix is read into Shazam by means of the / *rows = dim cols = dim* option to the *read* command, where *dim* is replaced by the proper dimension (e.g., 49 in Table 3.5). The = and ? symbols preceding the commands serve to suppress the command being echoed in the output file and to suppress output messages.

Splus

As in Gauss, the row standardization routine in Splus comes in the form of a function (*wrstd*) which takes as its only argument a weights matrix in Splus matrix format. Such a matrix is created from an ASCII input file by means of the *scan* and *matrix* commands, e.g.:

```
wmat_matrix(scan("file.asc"),nrow=n,ncol=n,byrow=T)
```

where "file.asc" is an ASCII file containing a full n x n matrix.

Row-standardizing a Spatial Weights Matrix and Computing its Eigenvalues

The routines outlined in this chapter include procedures to row-standardize a spatial weights matrix and to compute its eigenvalues. If only the row-standardization is needed, the parts pertaining to the eigenvalue calculation can easily be commented out. For Limdep, Shazam and Rats the combined row-standardization and computation of the eigenvalues is contained in the routines labeled *wrstd.**. In Gauss and Splus, the row-standardization and eigenvalue calculation are contained in separate routines, respectively labeled *wrstd.** and *eigenw.**.

For Limdep, Rats and Shazam, the input into these routines consists of a file name for the spatial weights matrix and its dimensions, together with the respective matrix declarations needed in each software package. Both the row-standardized matrix and the eigenvalues are saved as a file in the most efficient format for each package. In Gauss and Splus, the weight matrix is passed as an argument to the procedure and its dimension is a local variable, computed internally. The row-standardized matrix (in *wrstd.**) or a vector with the eigenvalues (in *eigenw.**) is returned to the calling program.

The code described in this chapter is organized to run with the Columbus, Ohio dataset included with this report and described in Appendix 1. The source code can be adjusted to any other data set by replacing the underlined items.

Weights Matrix Standardization

The first step in performing the row standardization of a spatial weights matrix is the calculation of its row sums. Splus and Gauss allow direct operations on a row or column of a matrix. In the other three packages, the row sums are obtained by post-multiplying the matrix (**wmat**) by a column vector of ones (**onevec**).

[1]This format is followed by the *sp2full* program (Described in Chapter 2).

The second step in the standardization is the division of each element in **wmat** by its corresponding row sum. This involves the division of elements in a matrix by elements in a vector. In Splus and Gauss, the usual division operator (respectively / and ./), can be used to perform this operation. Rats provides the *ewise* (element wise) command so that each element in **wmat** can be divided by its corresponding row sum. Shazam requires a match in dimensions of **wmat** and the row sums in order to perform Hadamard division. Therefore, the column vector containing the row sums is blown up to a full matrix with dimensions n by n by post-multiplying the row sum vector by a row vector of ones. In effect, this serves to repeat each element in the row sum vector across a row of the new full sized matrix. Similarly, in Limdep, a column vector of row sums is transformed by taking the reciprocal of each element using the *diri* (direct inverse) command and is then blown up to a full matrix. Each element in **wmat** is multiplied by its corresponding element in **sumat** via the Limdep command *dirp* (direct product). The *wrstd.** routines assume that no row sum equals zero, i.e., that all observations in the data set have at least one contiguity. If this is not the case (i.e., unconnected observations), the code must be altered to take this into account, since a division by zero error will result otherwise.

The final step is to save the standardized matrix in an output file. This task is handled in the standard manner for each of the programs.

Calculation of the Eigenvalues of the Spatial Weights Matrix

As mentioned in the introduction, the computation of the eigenvalues in Rats and Splus must be based on the Ord (1975) procedure. In the other packages, the eigenvalues of a general real matrix are calculated directly.

REFERENCES

Anselin, Luc, 1992. Spatial data analysis with GIS: an introduction to application in the social sciences (National Center for Geographic Information and Analysis Technical Report 92-10, University of California, Santa Barbara, CA).

Ord, J.K., 1975. Estimation methods for models of spatial interaction. *Journal of the American Statistical Association* 70, 120-126.

THE WRSTD.* AND EIGENW.* PROGRAMS

Table 3.1 - Gauss: WRSTD.GAS

```
/*
**  /* and */ enclose comments
**
**  Input:  wmat   weights matrix in full format (n by n)
**
**  Return: wrstd  row-standardized spatial weights matrix
**
*/
proc(1)=wrstd(wmat);
local wrstd;
wrstd = wmat ./sumc(wmat');  /* sumc is column sum of transpose, i.e. row sum */
retp(wrstd);
endp;
```

Table 3.2 - Gauss: EIGENW.GAS

```
/*
**  /* and */ enclose comments
**
**  Input:  wmat or wrstd    spatial weights matrix (unstandardized or row-standardized
**          stanflag   flag on row-standardization
**                     1  weights are rowstandardized (use wrstd)
**                     0  weights are unstandardized (use wmat)
**
**  Return: wroot   vector with eigenvalues (no imaginary parts)
**
*/

proc(1)=eigenw(wmat,stanflag);    /* input could be wmat or wrstd */
local wroot,pp;
/* eigenvalue computation: wroot is real part, pp is imaginary part (ignored) */
if stanflag;   /* matrix is asymmetric */
   { wroot,pp }=eigrg(wmat);  /* use eigrg to compute eigenvalues */
else;    /* matrix is symmetric */
   wroot=eigrs(wmat);   /* use eigrs to compute eigenvalues */
endif;
retp(wroot);
endp;
```

Table 3.3 - Limdep: WRSTD.LIM

```
?
?[notes are contained in brackets [], ? denotes comments, $ denotes end of input, ; is used
to separate commands]
?
????????????????????????????????????USER INPUT?????????????????????????????????
?(CHANGE ONLY UNDERLINED ITEMS)
?
READ; NOBS = 49                                                    [number of observations]
    ; NVAR = 49                                 [number of columns=number of observations]
    ; NAMES = 3                         [how many lines the variable names take up in ASCII file]
    ; FILE = COLWMAT.LIM $                                        [weights matrix file name]
NAMELIST ; WTL = W1*,W2*,W3*,W4*                                  [first 40 columns if nvar >40]
         ; WTR = W5* $                                      [columns beyond 40 to at most 80]
?
?????????????????????????????????????????????????????????????????????????????
?
MATRIX ; NOLIST ; MWTL = COPY(WTL); MWTR = COPY(WTR) $                          [copy 40 column
                                                                   partions to matrix work area]
MATRIX ; NOLIST ; WMAT = MWTL,MWTR $                       [join partions (separate by commas)]
MATRIX ; NOLIST ; ONEVEC = INIT(N,1,1) $                                       [vector of ones]
MATRIX ; NOLIST ; SUMAT = WMAT | ONEVEC                                     [calculate row sums]
       ; SUMAT = DIRI(SUMAT)                                       [take inverses of row sums]
       ; SUMAT = SUMAT | ONEVEC' $                         [blow up row sums to full n by n matrix]
MATRIX ; NOLIST ; WMAT = DIRP(SUMAT,WMAT) $                            [multiply element by element]
DELETE ; WTL, WTR $                                    [delete namelists since matrix now joined]
?
?????????????????????????????????????????????????????????????????????????????
SAVE ; FILE = COLWS.LIM $                               [name output file for standardized matrix]
?????????????????????????????????????????????????????????????????????????????
?
MATRIX ; NOLIST ; MEIGERS = CXRT(WMAT)                                [calculate eigenvalues of an
                                                                                asymmetric matrix]
       ; MEIGERS = PART(MEIGERS,1,N,1,1) $                      [take only 1st column, eigenvalues
                                                                                        are real]
CREATE ; M0 = 0 $
NAMELIST ; WROOT = M0 $
CREATE ; WROOT = DTA(MEIGERS) $                             [move eigenvalues to a variable in data work
                                                                                           area]
?
?????????????????????????????????????????????????????????????????????????????
WRITE ; WROOT ; FORMAT = BINARY ; FILE = COLWS.EIG $                           [provide output file
                                                                          name for eigenvalues]
?????????????????????????????????????????????????????????????????????????????
?
STOP
```

Table 3.4 - Rats: WRSTD.RAT

```
*
*[notes are contained in brackets]
*[*denotes comments, $ denotes that command continues on next line]
*
OUTPUT NOECHO                                   [no echoing of commands to output file]
*
*******************************USER INPUT SECTION****************************
*[CHANGE ONLY UNDERLINED ITEMS]
*
IEVAL NOBS = 49                                      [enter number of observations]
DECLARE RECTANGULAR WMAT(NOBS,NOBS)
OPEN DATA COLWMAT.ASC                               [enter weights matrix file name]
READ(FORMAT=FREE) WMAT
*
*********************************END USER INPUT******************************
*
DECLARE VECTOR ONEVEC(NOBS) SUMROW
MATRIX ONEVEC = CONST(1.0)                                         [vector of ones]
*
DECLARE RECTANGULAR WRSTD(NOBS,NOBS)
MATRIX SUMROW = WMAT*ONEVEC                                  [calculate row sums]
EWISE WRSTD(I,J) = WMAT(I,J)/SUMROW(I)              [divide each element in W by its
                                                             corresponding row sum]

*****************************************************************************
OPEN COPY COLWS.BIN                      [enter output file name for standardized weights
                                                                            matrix]
*****************************************************************************
*
WRITE(UNIT=COPY,FORMAT=BINARY) WRSTD                            [write matrix to file]
*
CLOSE COPY                                                        [close copy unit]
*
DECLARE RECTANGULAR SUMROWI(NOBS,1) SQSUM(NOBS,1) DSUMROW(NOBS,NOBS)
EWISE SUMROWI(I,J) = 1.0/SUMROW(I)                             [inverse of row sums]
EWISE SQSUM(I,J) = SQRT(SUMROWI(I,J))                     [square root of 1/row sums]
MATRIX DSUMROW = DIAG(SQSUM)               [put transformed row sums on diagonal of new
                                                                            matrix]
DECLARE SYMMETRIC WMATS(NOBS,NOBS)
MATRIX WMATS = DSUMROW * WMAT * DSUMROW                [transform weights matrix to a
                                                                  symmetric matrix]
EIGEN WMATS WROOT                         [calculate eigenvalues of the symmetric matrix]
*
*****************************************************************************
OPEN COPY COLWS.EIG                        [enter output file name for eigenvalues of W]
*****************************************************************************
*
WRITE(UNIT=COPY,FORMAT=BINARY) WROOT                       [write eigenvalues to file]
*
END
```

Table 3.5 - Shazam: WRSTD.SHA

```
=*[notes are contained in brackets]
=*[* denotes comments, = means no echo of command line in output file, and ? suppresses
output from these lines]
*
=*********************************USER INPUT*********************************
=*(CHANGE ONLY UNDERLINED ITEMS)
=?FILE 12 COLWMAT.ASC                                         [enter weights matrix file name]
=?READ(12) WMAT / ROWS=49 COLS=49                            [number of observations for row &
                                                                            column dimensions]
=?FILE 13. COLWS.BIN                           [output file name for standardized weights matrix]
=?FILE 14. COLWS.EIG                                     [enter output file name for eigenvalues]
=***************************END USER INPUT*****************************
=SET NOCC                                                                   [no carriage control]
=SET NOWARN                                                               [no messages in output]
=*
=GENR ONEVEC = 1.0                                                              [vector of ones]
=MATRIX SROW = WMAT*ONEVEC                                                   [calculate row sums]
=MATRIX SROWFULL = SROW * ONEVEC'                        [blow up row sums to a full n by n matrix]
=MATRIX WRSTD = WMAT / SROWFULL                          [divide each element of W by corresponding
                                                                                         row sum]
=WRITE(13) WRSTD / BINARY                                [write standardized matrix to output file]
=*
=MATRIX WROOT = EIGVAL(WRSTD)                                        [calculate eigenvalues of W]
=WRITE(14) WROOT / BINARY                                       [write eigenvalues to output file]
*
STOP
```

Table 3.6 - Splus: WRSTD.SPL

```
#
# denote comments
#
# Input:  wmat    weights matrix in full format (n by n)
#
# Return: rstdw   row standardized weights matrix
#
wrstd_function(wmat) {
   sumwj_apply(wmat,1,sum)                                                   #calculate row sums
   rstdw_wmat/sumwj                         #divide each element of wmat by corresponding row sum
   rstdw                                                                         #value returned
}
```

Table 3.7 - Splus: EIGENW.SPL

```
#
# denote comments
#
# Purpose:  Transforms an asymmetric matrix to a symmetric one
# in order to use the Splus eigen function
#
# Input:   wmat     weights matrix in full format (n by n)
#
# Return:  wroot    vector of eigenvalues
#
eigenw_function(wmat) {
   sumrow_apply(wmat,1,sum)                                                      #row sums of W
   sqrs_sqrt(1/sumrow)                                              #square root of 1/row sums
   dsumrow_diag(sqrs)
#put transformed row sums on diagonal of new matrix
   wmats_dsumrow%*%wmat%*%dsumrow                           #transform W to a symmetric matrix
   wroot_eigen(wmats)$values                                            #list of eigenvalues
   wroot                                                                       #return value
}
```

CHAPTER 4
REGRESSION DIAGNOSTICS

INTRODUCTION

In this chapter, we present routines to carry out regression diagnostics for spatial autocorrelation. Such diagnostics should become a standard part of any regression analysis for cross-sectional observations that are spatially adjacent. The code is listed in Tables 4.1 to 4.5. When the null hypothesis of spatial independence cannot be rejected, there is no problem and all standard regression methods may be applied in the usual way. However, when the null hypothesis is rejected in favor of the alternative of spatial error autocorrelation or spatial lag correlation, specialized estimation techniques are needed, typically based on the Maximum Likelihood principle. These techniques are covered in Chapters 5 and 6.

METHODOLOGICAL BACKGROUND

Null and alternative hypotheses

The standard linear regression model is:

$$y = X\beta + \varepsilon$$

where y is a N by 1 vector of observations on the dependent variable, X is a N by K matrix of observations on a number of non-stochastic (i.e. fixed) explanatory variables, β is a K by 1 vector of population regression coefficients and ε is a N by 1 vector of error terms. In the standard model, the errors are assumed to be independent and identically distributed normal variates (or, less strictly, uncorrelated and homoskedastic):

$$E[\varepsilon] = 0$$

$$E[\varepsilon\varepsilon'] = \sigma^2 I$$

Spatial dependence is the situation where the observations or error terms at one location are correlated with observations or error terms at neighboring locations. In its most general form, spatial dependence can be expressed as:

$$E[y_i y_j] \neq 0, \text{ or } E[\varepsilon_i \varepsilon_j] \neq 0$$

Unfortunately, this general expression has too many unknown parameters to be estimable. Therefore, the form of the spatial dependence is given structure by the use of a spatial weights matrix (W - see Chapters 2 and 3), which reduces the number of unknown parameters to one, i.e., the coefficient of spatial association.

Diagnostics for the presence of spatial dependence in a linear regression model are developed by testing the null hypothesis of independence against a specific alternative that embodies spatial dependence. There are two types of alternative hypotheses: one pertains to dependence in the form of a spatially lagged variable; the other to dependence in the error term.

For spatial dependence in the dependent variable, the alternative model contains a so-called spatial lag, which is actually a weighted sum of the observations on the dependent variable at neighboring locations. The relevant "neighbors" are specified in the weights matrix W, which is typically standardized such that the row elements sum to one (see Chapter 3). Consequently, a regression model with a spatial lag is of the form:

$$y = \rho Wy + X\beta + \varepsilon$$

where Wy is the spatially lagged dependent variable, ρ is its associated coefficient, and the other notation is as before. Such a model is referred to in the literature as a mixed regressive (the $X\beta$ term) spatial autoregressive (the ρWy term) model.

A consequence of the presence of the spatial lag in the right hand side of the regression equation is that OLS no longer gives a consistent estimate for this model. Therefore, specialized estimation methods need to be developed, most of which are based on the Maximum Likelihood principle. In order to test for the presence of spatial dependence in the form of a spatial lag, it makes sense to try to avoid having to carry out the nonlinear ML estimation, and to use the results of the OLS estimation for the model without spatial dependence instead. As it turns out, the null hypothesis of no spatial dependence (H_0: ρ = 0) corresponds exactly to the standard regression model:

$$y = X\beta + \varepsilon$$

The Lagrange Multiplier test for a spatial lag discussed below is based on OLS estimation in the standard regression model.

In applied empirical work, the other instance of spatial dependence, namely where the dependence is for the error terms, is often more relevant. Whereas the model with a spatial lag expresses a particular spatial process, dependence in the error terms is largely a nuisance, which needs to be corrected. It is important to note that dependence in the errors does not affect the unbiasedness of OLS, but it does make it a less efficient estimator. If the spatial dependence is ignored, the inference in the regression model and associated measures of fit will be based on the wrong estimate of variance.

There are two general forms in which the spatial dependence in the error term can be modeled: as a spatial autoregressive process, or as a spatial moving average process. It turns out that most tests for spatial error dependence (including the ones discussed here) have power against both specifications. In other words, if the only concern is with a misspecification diagnostic, it does not matter whether the errors follow a spatial autoregressive or a spatial moving average process. Of course, when the attention focuses on estimation, it is important that the correct specification be implemented.

Formally, a spatial autoregressive error term is specified as:

$$\varepsilon = \lambda W\varepsilon + \mu$$

where λ is the autoregressive parameter and μ is a well behaved (i.i.d) error term with variance σ^2. This can also be expressed as:

$$\varepsilon = (I - \lambda W)^{-1}\mu$$

so that the error variance becomes:

$$E[\varepsilon\varepsilon'] = E\ \{\ (I - \lambda W)^{-1}\mu\mu'(I - \lambda W')^{-1}\ \}$$

or

$$E[\varepsilon\varepsilon'] = \sigma^2\ \{\ (I - \lambda W)'(I - \lambda W)\ \}^{-1}$$

A spatial moving average error term has the following specification:

$$\varepsilon = \lambda W\mu + \mu$$

in the same notation as above. The resulting error variance for the regression model is different from that in an autoregressive specification:

$$E[\varepsilon\varepsilon'] = E\ \{\ (I + \lambda W)\mu\mu'(I + \lambda W)\ \}$$

$$E[\varepsilon\varepsilon'] = \sigma^2(I + \lambda W)(I + \lambda W)'$$

There are two main approaches to test for spatial error dependence based on the results of OLS estimation. In both, the null hypothesis is expressed as H_0: $\lambda = 0$. One approach is based on the extension of Moran's I test for spatial autocorrelation, the other on the Lagrange Multiplier principle. In contrast, only the Lagrange Multiplier principle can be used to develop a diagnostic for the presence of a spatial lag.

Moran's I test for residual spatial autocorrelation

The extension of Moran's I statistic to the case of least squares residuals is the most familiar diagnostic for spatial autocorrelation in regression analysis. It was originally proposed by Cliff and Ord (1972). Its properties are discussed at length in Cliff and Ord (1981) and Upton and Fingleton (1985). This test is even acknowledged in the standard econometric literature (King, 1987), although its application is still rather rare in empirical work.

Formally, Moran's I is defined as:

$$I = (N/S_0)\ e'We\ /\ e'e$$

where e is a N by 1 vector of least squares residuals, and S_0 is a standardizing factor, i.e., the sum of all elements of the spatial weights matrix:

$$S_0 = \Sigma_i\ \Sigma_j\ w_{ij}$$

For a row-standardized spatial weights matrix, S_0 equals N, so that Moran's I simplifies to:

$$I = e'We\ /\ e'e$$

This statistic shows a striking similarity to the Durbin Watson (DW) test for time wise autocorrelation (see King, 1987, for a more extensive discussion). As for the DW, its distribution will depend on the values of the explanatory variables included in the regression (the X). Moreover, the distribution of Moran's I will also depend on the structure of the spatial weights matrix W. For practical inference, Moran's I is transformed to a standard variate which has been shown to follow a normal distribution. The standardized z-value is obtained in the usual fashion, as:

$$I_z = \{\ I - E[I]\ \}\ /\ \{\ V[I]\ \}^{1/2}$$

Under the assumption of a normal error, the expected value and variance for Moran's I are (Cliff and Ord, 1981, pp. 202-203):

$$E[I] = -\ N.tr(A)\ /\ [(N-K)S_0\ ]$$

with tr as the trace operator, S_0 as above and

$$A = (X'X)^{-1}X'WX$$

The variance is:

$$V[I] = \{N^2/[S_0^2(N-K)(N-K+2)]\}\{S_1+2tr(A^2)-tr(B)-2[tr(A)]^2/(N-K)\}$$

with

$$S_1 = 1/2\ \Sigma_i\ \Sigma_j\ (w_{ij} + w_{ji})^2$$

and

$$B = (X'X)^{-1}X'(W+W')^2X$$

Lagrange Multiplier test for spatial error dependence

The approach towards testing for spatial error dependence that is based on the Lagrange Multiplier principle is outlined in Burridge (1980) and Anselin (1988). In formal terms, the statistic is very similar to the Moran, except for the use of a different scaling constant. Its properties are asymptotic. The statistic is:

$$LM_{err} = \{ e'We / s^2) \}^2 / tr[W'W + W^2]$$

with $s^2 = e'e/N$.

For a row standardized weights matrix, this corresponds to:

$$LM_{err} = (N.I)^2 / tr[W'W + W^2]$$

where I is Moran's I. This LM_{err} statistic is the same for an alternative hypothesis of spatial autoregressive and that of spatial moving average errors. It is distributed as a χ^2 variate with one degree of freedom. A high value of the statistic (and a low value of the probability) implies rejection of the null hypothesis of no spatial association. However, it does not indicate which precise form the spatial association will take, since the test has power for many different choices of weights matrices and for both an autoregressive and a moving average process.

Lagrange Multiplier test for spatial lag dependence

A Lagrange Multiplier test for a spatially lagged dependent variable was suggested as a special case of a test for an omitted variable in Anselin (1988). Its properties are also asymptotic. The statistic is:

$$LM_{lag} = \{ e'Wy / s^2 \}^2 / \{ (WXb)'MWXb / s^2 + tr[W'W + W^2] \}$$

where $M = I - X(X'X)^{-1}X'$ and the rest of the notation is as above. This statistic is also distributed as a χ^2 variate with one degree of freedom.

Interpretation

The interpretation of the results for the tests on spatial effects has to be done with caution. As shown in a wide range of simulation experiments in Anselin and Rey (1991a, 1991b), all three tests depend in a crucial manner on the choice of the spatial weights matrix. Also, the two tests for spatial error dependence turn out to be sensitive to the assumption of a normal error distribution (there is some evidence that they under-reject the null hypothesis for lognormal and exponential errors). The Moran test in particular is also found to be susceptible to the presence of heteroskedasticity, leading to an over-rejection of the null hypothesis. In comparison, the LM_{lag} test is found to be more robust and less susceptible to the presence of heteroskedasticity. The asymptotic distribution for the test statistics (normal for Moran's I, $\chi^2(1)$ for LM_{err} and LM_{lag}) turns out to be a fairly good guide in small samples, except for the LM_{err} test, which tends to under-reject the null hypothesis in small samples. This is a typical characteristic of LM tests. It is in agreement with the general result about the relative magnitudes in finite samples of asymptotically equivalent tests based on Maximum Likelihood estimation. In finite samples, the Lagrange Multiplier test (LM), the Likelihood Ratio test (LR) and the Wald statistic (W) tend to follow the ranking:

$$W \geq LR \geq LM$$

In other words, one is always less likely to reject a null hypothesis based on a LM test than based on its asymptotic equivalents. Given this characteristic, even weak evidence of spatial dependence that follows from an LM test should be an indication to pursue a modeling strategy that incorporates spatial effects explicitly.

In terms of providing guidance in the choice between spatial error autocorrelation and a spatial lag, Anselin and Rey (1991a) find the LM tests to be the most appropriate. Moran's I has power against both alternatives and thus cannot be used to discriminate between the two. In contrast, the LM tests are found to have the highest power in the case for which they are designed. In other words, if both the LM_{err} and the LM_{lag} have high values (and thus low probabilities that the null hypothesis is satisfied), the one with the highest value points to the most likely alternative hypothesis (i.e., error dependence or spatial lag).

OVERVIEW OF THE ROUTINES

User Input Section

The user input section is the part of the code which differs the most among the various software packages. In this section, the data set, spatial weights matrix and the variables used in the regression analysis must be entered. For Limdep, Shazam and Rats, the code has been organized so that no adjustments are needed in the other sections, once the proper data sets etc. have been entered in the user input section. For the sake of clarity, the items that need to be changed for each particular analysis are underlined in the annotated source code. However, the underlines are NOT in the source code on the diskette, and should NEVER be included in actual code. In Gauss and Splus, the diagnostics are called as a procedure/function and the arguments are passed as generic vectors and matrices. These must be defined or computed in the main program (or calling program).

As before, the routines are written with the input sections appropriate for the example using the Columbus data.

Gauss

The main or calling program must contain a regression analysis (e.g., using Gauss procedures *ols* or *olsqr2*) with the regression coefficients as **b** and the residuals specified as vector **resvct**. In addition, a spatial weights matrix must be loaded as matrix **wmat** (see Chapter 3). The matrix with observations on the explanatory variables is specified as **xmat**, with its first column equal to ones (for the constant term), while the vector of observations on the dependent variables is specified as **depvar**. Both of these must be in memory when the diagnostics are called. Gauss contains many different ways in which data can be loaded into a program, but for the sake of consistency we chose to only treat the matrix format (data can easily be converted from a Gauss data format to a matrix format).

The explanatory variables, dependent variable, spatial weights matrix and vector of residuals are passed as arguments to the *regdiag.gas* procedure, as in:

{ **moran,lm** }=*regdiag*(**xmat,depvar,wmat,resvct,b**);

Two items are returned by the procedure:

- **moran**: a 1 by 3 row vector with the Moran's I statistic, its corresponding standardized z-value and the associated probability (following a normal distribution);
- **lm**: a 2 by 2 matrix with in the first row the LMerr statistic and its associated probability, and in the second row the LMlag statistic and its associated probability.

Limdep

The user input section for Limdep requires the editing of three command lines for each analysis: *load*, *read*, and *namelist*. No other items need to be changed.

The first command (*load*) restores all the data and results obtained in the row-standardization procedure (Chapter 3). The *load* command ensures that the weights matrix (**wmat**) is restored to the matrix work area and is ready to be used in the computation of the diagnostics.

The second command (*read*) deals with the data on the dependent and explanatory variables. Limdep has great flexibility in entering data, but for the sake of clarity we have only included one form in the code below. In our format, there are five required items for the *read* command. The first is the number of observations (**nobs**). The second item is the number of variables in the entire dataset (**nvar**). This number is typically larger than the number of variables used in the analysis since all variables in the data set must be specified. Next, the name for the file that contains the dataset must be specified (**file**). Note that a full path name is needed if the data set is not in the current directory. The fourth item is a list of all the variable names in the dataset (*names*). Note that this may be contained in the first row of the input file. In those instances, the names must not be specified explicitly and the *names* subcommand should be used as *;names = #rows*, where *#rows* refers to the number of rows in the data file occupied by the variable names (see the Limdep reference manual for details). The final item to be entered is the format for the data set (*format*). This must only be adjusted if the data set is not in the binary format used by Limdep (the default in the code below is *format = binary*).

The third command line that must be edited is the *namelist* command. It provides an easy way to assign a name to a group of variables for later reference in other commands. **Depvar** is used to refer to the dependent variable; **expvar** refers to the explanatory variable(s).

Rats

The user input section for Rats is slightly more complex than in the other programs. In addition to the file name for the data and weights matrix, and the names for the variables in the regression, the dimensions of the problem must be specified as well. Also, the format of the data must be specified explicitly. Since Rats is primarily designed for time series analysis, its preferred format is that of a "series". However, in the analysis of spatial effects, most computations are carried out on matrices (arrays), which need to be declared explicitly in Rats.

The first entry in the user input section gives the number of observations (the maximum length of the data series), with the *allocate* command. Next follow five command lines that deal with the actual data input: *open data*, *data*, *declare rectangular*, *open data*, and *read*. The first two pertain to the variables used in the analysis. Similar to Limdep and Shazam, Rats allows various formats for the data set. For the sake of clarity, we have assumed a Rats binary format in the code that follows (for details on other formats and on format conversion, see the Rats reference manual). Data input is achieved by the two Rats commands *open data* and *data*. The first is followed by the name of the data set. The *data* command lists all variable names as well as the format for the data set (*format=*). The *data* command brings the data set into memory, ready for manipulation by Rats expressions. The three other command lines are needed to set up the spatial weights matrix. In the first (*declare rectangular*), the dimensions of the matrix need to be declared. In the code that follows, the matrix is referred to as **wmat**. The second command (*open data*) opens the data file with the weight matrix data. The matrix is moved into memory by means of the *read* command.

In contrast to the other programs, in Rats the list of dependent and explanatory variables must be edited explicitly in the regression command (*linreg*). This is due to the distinction between series and array types. The regression is carried out using variable names that refer to series, while the other computations use arrays. Before the dependent and explanatory variables can be converted into the generic arrays **ymat** and **xmat** used in later calculations, the *linreg* command must be executed. This command includes the variable **crime** as the dependent variable and the variables **income** and **hoval** as explanatory variables. These must be adjusted for each particular application. Also, the variable name **crime** must be edited in the *make ymat* command (lines 19 and 20 in Table 4.3). Note that the format for the regression output is set such that the probabilities will fit in 80 column output (the t-statistics are suppressed by the *notstat* option in line 16).

Shazam

The user input section of Shazam consists of three sets of commands: *file*, *read* and *copy*. Following the file commands, the filenames for the data set and the spatial weights matrix need to be entered. In the example, the second file command (*file 12*) is followed by a dot, to characterize the spatial weights matrix as a binary file. This will be the case if the code in Chapter 3 was used to carry out row-standardization. The *read* commands read in the actual data. All the variable names in the data file must follow the *read(11)* command, as in Limdep. The dimension of the spatial weights matrix (**wmat**) must be listed following the *read(12)* command. The *copy* commands move the variable names for the explanatory variables and the dependent variable into the generic names **expvar** and **depvar**, for use in later computations.

Splus

Similar to the approach taken for the Gauss procedure, the *regdiag* function in Splus expects five arguments: *regdiag*(**xmat,depvar,wmat,resids,bhat**). **xmat** is a matrix of explanatory variables, **depvar** is a vector containing the dependent variable, **wmat** is the spatial weights matrix, **resids** is a vector of residuals, and **bhat** is a vector of regression coefficients. These last two arguments, **resids** and **bhat**, are returned by the *lsfit* command as **$residual** and **$coef**, respectively.

The function returns a 3 by 3 output matrix. The first row of this matrix contains the Moran's I statistic, its z-value, and its significance probability. The second and third rows of the matrix contain the value and significance probability for the LM error and LM lag statistics, respectively. The second column of rows 2 and 3 contains NAs since the LM tests follow a chi square distribution. For ease of interpretation, the rows and columns of this output matrix are given descriptive names with the Splus *dimnames( )* function.

In some instances, particularly when a problem contains very large numbers, the computation of the inverse matrix $X'X^{-1}$ may give an error because of large number overflow. If this happens, the variables should be scaled down (e.g. by dividing by a constant).

Least Squares Regression

Before the diagnostics for spatial dependence can be computed, an ordinary least squares regression must be carried out and its residuals retained. This is accomplished slightly differently in each package. In all programs except Rats, the regression is carried out using generic names for the dependent and explanatory variables. In Rats, the specific variable names must be entered explicitly.

There are also slight differences with respect to the treatment of a constant term. In Shazam and Splus, the default is to include a constant term. In Gauss, the constant must be made explicity by joining a vector of ones with the matrix containing the other explanatory variables. In Limdep and Rats, the constant term must be included as a variable, either as **one** in the *namelist* command (Limdep), or as **constant** in the list of explanatory variables (Rats).

The residuals are retained in slightly different ways. In Gauss, the residuals are returned as one of the items in the *ols* and *olsqr2* regression procedures. In Splus, they are contained in the variable **$residuals** which is generated by the *lsfit* regression command. In the three other programs, residuals are obtained by specifying an option in the regression command: option *res=* in Limdep and *resid=* in Shazam, where the equal sign is followed by a variable name for the residuals; in Rats the variable name for residuals is included on the *linreg* command line. Other items that make the computation of the diagnostics more efficient and that are stored internally by some of the packages are the ML estimate of error variance (e'e/N), or the residual sum of squares (e'e), and the cross product matrix (X'X) or its inverse. In Gauss and Splus, the residual sum of squares must be computed explicitly using matrix algebra, whereas in Limdep, Rats and Shazam it is stored automatically as an internal variable after each regression (respectively **sumsqdev** in Limdep, rss in Rats, and **$sse** in Shazam). In Limdep and Rats, the residuals and other variables in the model (i.e., y and X) must be copied explicitly from a "variable" type to a "matrix" type. The diagnostics themselves are computed by means of the matrix algebra facilities in each package.

Moran's I Test For Regression Residuals

The computation of Moran's I is fairly straightforward using matrix expressions. The statistic consists of a product of a scalar and a matrix. In Limdep and Rats, this requires special notation: in Limdep the scalar has to be put into braces { } in order to invoke the calculator; similarly, in Rats the scalar expression is enclosed by round brackets ( ) and preceded by the *scale* function.

There are small differences between the packages in the calculation of matrix traces and in the way the factors S_0 and S_1 are computed. Limdep, Rats and Shazam have built-in trace command. In contrast, in both Gauss and Splus, the trace has to be computed explicitly as the sum of the diagonal elements (using the *sumc* command in Gauss and the *sum* command in Splus). The sum of all matrix elements (to obtain S_0) is a built-in function in both Splus and Rats. In Gauss, this is obtained by computing the column sum twice, e.g., as *sumc(sumc(wmat))*. In both Limdep and Shazam, this has to be carried out by means of explicit matrix multiplication commands: the matrix is first post-multiplied by a vector of ones, which is then pre-multiplied by a row of ones (or, its transpose is post-multiplied by another column of ones). In order to compute S_1, the squares of the sum of the elements of the weights matrix and its transpose are required. In Limdep and Rats, the squared elements are computed by means of an element by element or Hadamard product (respectively, the *dirp* command in Limdep and the *ewise* command in Rats). In the other routines, an element by element second power is used instead (note that both Gauss and Splus also allow for element by element operators).

Inference for Moran's I is based on the standard normal distribution. The significance level is computed from a call to the respective distribution function in each package. In Rats, the output of the *cdf normal* procedure yields a value for a two tailed test. In order to obtain this in Gauss, the result of the *cdfnc* command must be multiplied by two. In Limdep (*phi* command) and Splus (*pnorm* command), the result must first be subtracted from 1 and then multiplied by 2. The Shazam *distrib (type=normal)* command does not allow for this flexibility and the user must manually multiply the figures in the output column (*1 - CDF*) in order to have results compatible with those of the other programs.

Lagrange Multiplier Test For Spatial Error Dependence

In order to compute the LM error statistic, only one trace is needed in addition to Moran's I. The differences between the software packages in this respect are the same as outlined in the previous section. The significance level for the statistic is derived by means of the built-in functions for the chi-squared distribution. In Gauss (*cdfchic*), Rats (*cdf chisquare*) and Shazam (*distrib (type=chi)*) this is the desired value. In Splus (*pchisq*) and Limdep (*chi*) the results must first be subtracted from 1 in order to conform.

Lagrange Multiplier Test For A Spatial Lag

For the LM lag statistics, there are two main issues that are treated differently between the software packages. The first is the manipulation of the vector of observations on the dependent variable **y**, which is needed to compute the cross product **e'Wy** (**e'W** is saved from the Moran's I section). As pointed out above, in Rats and Limdep this requires an explicit copying of the dependent variable into the matrix work area. The second issue pertains to the computation of the term in the denominator of the test statistic. Upon close examination, it can be seen that this is a residual sum of squares in an auxiliary regression with **WXb** as the dependent variable and the **X** as explanatory variables. In Gauss and Splus it is easiest to compute this expression explicitly, using matrix notation. In the other programs, the built-in residual sum of squares function associated with a regression command can be used instead. In Limdep, after the **WXb** vector is constructed in the matrix work area, it needs to be moved back to the data work area for input into the *ess* function (it is computed explicitly, rather than as a byproduct of a regression). In Rats and Shazam, the auxiliary regression must be called and the error sum of squares is saved internally as a scalar value by the program (as **RSS** in Rats and **$SSE** in Shazam).

The significance of the LM lag statistic is computed by calling the chi-square distribution function, in the same way as for LM error.

REFERENCES

Anselin, L. 1988. "Lagrange Multiplier test diagnostics for spatial dependence and spatial heterogeneity," *Geographical Analysis*, 20, 1-17.

Anselin, L. and S. Rey. 1991a. "Properties of tests for spatial dependence in linear regression models," *Geographical Analysis*, 23, 112-131.

Anselin, L. and S. Rey. 1991b. "The performance of tests for spatial dependence in a linear regression." Technical Paper 91-13, National Center for Geographic Information and Analysis, University of California, Santa Barbara.

Burridge, P. 1980. "On the Cliff-Ord test for spatial autocorrelation," *Journal of the Royal Statistical Society B*, 42, 107-108.

Cliff A.D. and K. Ord. 1972. "Testing for spatial autocorrelation among regression residuals," *Geographical Analysis*, 4, 267-284.

Cliff A.D. and K. Ord. 1981. *Spatial Processes, Models and Applications* (London: Pion).

King, M. 1987. "Testing for autocorrelation in linear regression models: a survey," in *Specification Analysis in the Linear Model*, edited by M. King and D. Giles, pp. 19-73 (London: Routledge and Kegan Paul).

Upton, G. and B. Fingleton. 1985. *Spatial Data Analysis by Example* (New York: Wiley).

THE REGDIAG.* PROGRAMS: REGRESSION DIAGNOSTICS

Table 4.1 - Gauss: regdiag.gas

```
/*
**   /* and */ enclose comments
**
**   Input:  xmat    matrix with explanatory variables (ones as first column)
**           depvar  vector with dependent variable
**           wmat    spatial weights matrix
**           resvct  vector of OLS residuals
**           b       vector of OLS estimates
**   Return: moran   1 by 3 vector with Moran's I, z-value and probability
**           lm      2 by 2 matrix with LMerr and probability in first row
**                   LMlag and probability in second row
*/

proc(2)=regdiag(xmat,depvar,wmat,resvct,b);
local moran,lm,nobs,df,rw,rss,sigml,S0,d,
      morani,WX,A,trA,EI,trA2,U,UX,C2,trB,S1,VI,ZMI,pZMI,
      trWtW,trWW,tr11,LMERR,pLMERR,
      eWy,WXb,LMLAG,pLMLAG;

/* auxiliary computations */
nobs=rows(xmat);             /* number of observations */
df=nobs-cols(xmat);          /* degrees of freedom */
rW=resvct'wmat;              /* spatial lag of residuals - row vector */
rss=resvct'resvct;           /* residual sum of squares */
sigml=rss./nobs;             /* ML estimate of residual variance */
S0=sumc(sumc(wmat));         /* sum of elements of weights matrix */
d=invpd(xmat'xmat);          /* inverse of cross products of X */

/* Moran's I */
/* Compute statistic */
morani=(nobs./S0).*((rW*resvct)./rss);     /* Moran's I */
/* Mean of Moran's I */
WX=wmat*xmat;                /* spatial lag of X */
A=d*(xmat'WX);               /* A=(X'X)-1X'WX */
trA=sumc(diag(A));           /* trace of A */
EI=-nobs*trA./(df*S0);       /* mean of Moran's I */
/* Variance of Moran's I */
trA2=sumc(sumc(A.*A'));      /* trace of A squared */
U=wmat + wmat';              /* sum of weights matrix and its transpose */
UX=U*xmat;                   /* auxiliary matrix (W + W')X */
C2=UX'UX;                    /* X'(W+W')2X */
trB=sumc(sumc(d.*C2));       /* trace of B */
S1=sumc(sumc(U.^2))./2.0;    /* S1 */
VI= ((nobs.^2)./(S0*S0*df*(df+2.0)));
VI= VI .* (S1+trA2+trA2-trB-((2.0./df)*trA*trA));
                             /* variance of Moran's I */
/* Standardized z-value for Moran's I */
ZMI = (morani-EI)./sqrt(VI);   /* z-value */
pZMI=2.0.*cdfnc(abs(ZMI)); /* probability */
moran=morani~ZMI~pZMI;       /* store results in return vector */

/* Lagrange Multiplier test for spatial error autocorrelation */
trWtW = sumc(sumc(wmat.^2));      /* trace of W'W */
trWW = sumc(sumc(wmat.*wmat'));   /* trace of W2 */
tr11=trWtW + trWW;                /* trace of W'W + W2 */
LMERR=((nobs*morani).^2)./tr11;   /* statistic */
pLMERR = cdfchic(LMERR,1);        /* probability */
lm=LMERR~pLMERR;                  /* store results in return matrix */

/* Lagrange Multiplier test for spatial lag */
eWy=rW*depvar;                   /* e'Wy */
WXb= WX*b;                       /* WXb */
LMLAG= ((eWy./sigml).^2) ./ ((WXb'(WXb - xmat*d*xmat'WXb))./sigml + tr11);
                                 /* statistic */
pLMLAG = cdfchic(LMLAG,1);       /* probability */
lm=lm|(LMLAG~pLMLAG);            /* store results in return matrix */

retp(moran,lm);
endp;
```

Table 4.2 - Limdep: regdiag.lim

```
?
?[notes are contained in brackets [], ? denotes comments, $ denotes end of input, ; is used
to separate commands]
?
?USER INPUT SECTION
[CHANGE ONLY UNDERSCORED ITEMS]
?
LOAD; FILE = COLWS.LIM $                    [enter output file from weights preprocessing routine]
READ ; NOBS = 49                                                        [#of observations]
     ; NVAR = 3                                                 [# of variables in data file]
     ; FILE = COLREG.BIN                                                  [data file name]
     ; NAMES = CRIME,INCOME,HOVAL                            [variable names in file above]
     ; FORMAT = BINARY $                                                [file format type]
NAMELIST ; DEPVAR = CRIME                                        [dependent variable name]
         ; EXPVAR = INCOME,HOVAL $                            [explanatory variable names]

? END OF USER INPUT SECTION
?

NAMELIST ; XMAT = ONE,EXPVAR $               [add column of ones to explanatory variables]
REGRESS ; LHS = DEPVAR ; RHS = XMAT; RES = RESIDS $                [preliminary regression]
?
? OFTEN USED MATRICES
?
                                                                         [vector of ones]
MATRIX ; NOLIST; VECTONE = INIT(N,1,1) $          [move residuals into matrix work area]
MATRIX ; NOLIST; RESVCT = COPY(RESIDS) $           [move X matrix into matrix work area]
MATRIX ; NOLIST; MXMAT = COPY(XMAT) $                                       [(X'*X)^-1]
MATRIX ; NOLIST; MXPXINV = XPXI(XMAT) $
MATRIX ; NOLIST; MXPXIXP = MXPXINV | MXMAT' $ [(X'*X)^-1*X', | denotes matrix multipication,
                                                                  ' for transposition]
CALC ; SIGSQML = SUMSQDEV / NREG $                 [ML estimate of the error variance]
?
TYPE ; $
TYPE ;  SPATIAL DEPENDENCE       $
?
TYPE ;  Moran's I for Spatial Error Dependence $
?                                                                               [e'*W]
MATRIX ; NOLIST; RW = RESVCT' | WMAT $
MATRIX ; NOLIST; S0 = WMAT | VECTONE                         [sum of all elements in W]
       ; S0 = S0' | VECTONE $                                [Moran's I unstandardized]
MATRIX ; MORANI = {N/(S0*SUMSQDEV)} * RW | RESVCT $
?                                                                                [W*X]
MATRIX ; NOLIST; WX = WMAT | MXMAT $                                  [(X'*X)^-1*X'*W*X)]
MATRIX ; NOLIST; AMAT = MXPXIXP | WMAT | MXMAT $    [trace (sum of diagonal elements)]
MATRIX ; NOLIST; TRAMAT = TRCE(AMAT) $                              [mean of Moran's I]
CALC ; NOLIST; EMI = -(N*TRAMAT)/(DEGFRDM*S0) $
?
MATRIX ; NOLIST; AMAT2 = AMAT | AMAT $
MATRIX ; NOLIST; TRAMAT2 = TRCE(AMAT2) $
MATRIX ; NOLIST; WW = WMAT & WMAT'
       ; WW2 = WW | WW $                                          [(W + W')*(W + W')]
MATRIX ; NOLIST; BMAT = MXPXIXP | WW2 | MXMAT $       [(X'*X)^-1*X'*(W + W')*(W + W')*X]
MATRIX ; NOLIST; TRBMAT = TRCE(BMAT) $
MATRIX ; NOLIST; MWW2 = DIRP(WW,WW) $          [DIRP is element by element multiplication]
MATRIX ; NOLIST; S1 = MWW2 | VECTONE
       ; S1 = 0.5 * S1' | VECTONE $                    [1/2 * sum of all elements of MWW2]
CALC ; NOLIST; VMI1 = N^2/(S0^2*DEGFRDM*(DEGFRDM+2.0)) $       [variance of MI, step 1]
CALC ; NOLIST; VMI = VMI1*(S1+2*TRAMAT2-TRBMAT-((2.0/DEGFRDM)*TRAMAT^2)) $     [var MI]
CALC ; NOLIST; STDMI = SQR(VMI) $                           [standard deviation of MI]
CALC ; MORANIN = (MORANI - EMI) / STDMI $                                  [Z-value: MI]
CALC ; PROB = 2.0*(1.0 - PHI(MORANIN)) $                                    [inference]
?
TYPE ; $
TYPE ;  Lagrange Multiplier (Error) $
?
MATRIX ; NOLIST; WW4 = WMAT' | WMAT
       ; WW4 = WW4 & WMAT | WMAT $                                  [(W'*W) + (W + W)]
MATRIX ; NOLIST; TRWW4 = TRCE(WW4) $
CALC ; LMERR = (N*MORANI)^2/TRWW4 $                      [Lagrange Multiplier (error)]
CALC ; PROB = 1.0 - CHI(LMERR,1) $           [inference: Chi-Squared, 1 degree of freedom]
?
TYPE ; $
TYPE ;  Lagrange Multiplier (Lag) $
?
MATRIX ; NOLIST; DEPVARVT = COPY(DEPVAR) $       [move dependent var. to matrix work area]
MATRIX ; NOLIST; RWY = RW | DEPVARVT $                                          [e'*W*y]
MATRIX ; NOLIST; ZMAT = WX | B $                                             [W*X*Betas]
```

```
CREATE ; NOLIST; Z1=0 $                                  [create series of 0's to fill below]
NAMELIST ; ZDATA = Z1 $                                                             [rename]
CREATE; NOLIST; ZDATA = DTA(ZMAT) $                    [move zmat from matrix to data work area
                                                                          for input to ESS]
CALC; LMLAG = (RWY/SIGSQML)^2/((ESS(XMAT,ZDATA))/SIGSQML + TRWW4) $                 [LM lag]
CALC ; PROB = 1.0 - CHI(LMLAG,1) $                [inference: Chi-Squared, 1 degree of freedom]
?
?
STOP
```

Table 4.3 - Rats: regdiag.rat

```
OUTPUT NOECHO                                             [no command echo in output file]
*
*[notes are contained in brackets]
*[* denotes comments, $ denotes that command continues on next line]
*
*   USER INPUT SECTION
[CHANGE ONLY UNDERLINED ITEMS]
*
ALLOCATE 0 49                                                 [enter # of observations]
OPEN DATA COLREG.RAT                                            [enter data file name]
DATA(FORMAT=RATS) / CRIME INCOME HOVAL          [enter file format, all variable names in file]
DECLARE RECTANGULAR WMATT(49,49)       [enter dimensions of weights matrix, should be square]
OPEN DATA COLWS.BIN                                          [weights matrix file name]
READ(FORMAT=BINARY) WMAT                                                 [file format]
OUTPUT NOTSTAT                                [preliminary regression, no t-stat printing]
LINREG(DEFINE=1) CRIME / RESIDU BETAEQ1                   [enter dependent variable name]
#CONSTANT INCOME HOVAL                                       [explanatory variable names]
MAKE YMAT /
#CRIME                                        [dependent variable series becomes an array]
*
*   END OF USER INPUT
*
MAKE(LASTREG) XMAT /                 [explanatory variables from last regression become array]
MAKE(LASTREG,TRANSPOSE) TRXMAT /                                                    [X']
*

EVAL RSS1 = RSS                                  [save residual sum of squares w/new name]
EVAL SIGSQML = RSS1/NOBS                                  [ML estimate of error variance]
*
*OFTEN USED MATRICES
*
DECLARE RECTANGULAR XPX XPXINV XPXINVXP IDENMAT
DIMENSION IDENMAT(NOBS,NOBS)
*
MATRIX XPX = TRXMAT * XMAT                                                        [X'*X]
MATRIX XPXINV = XX                                                           [(X'*X)^-1]
MATRIX XPXINVXP = XPXINV*TRXMAT                                            [(X'*X)^-1*X']
MATRIX IDENMAT = IDEN(1.0)                                        [n by n identity matrix]
*
DECLARE VECTOR ONEVEC
DIMENSION ONEVEC(NOBS)
*
MATRIX ONEVEC = CONST(1.0)                                                [vector of ones]
MAKE RESVECT /                                      [series of residuals becomes an array]
#RESIDU
*
*
*********************************************SPATIAL
DEPENDENCE*********************************
*
DECLARE RECTANGULAR WRSTD AMAT AMAT2 WW WW2 BMAT RW WX WWSTAR2 WTT
DIM WWSTAR2(NOBS,NOBS)
*
****MORAN'S I FOR SPATIAL ERROR DEPENDENCE
*
MATRIX RW = (TR(RESVECT))*WMAT                                                     [e'*W]
MATRIX S0 = SUM(WMAT)                       [sum of matrix elements, parameter for Moran's I]
MATRIX MORANRES = SCALE(NOBS/(S0*RSS))*RW*RESVECT            [Moran's I - unstandardized]
*
MATRIX WX = WMATT * XMAT                                                             [W*X]
MATRIX WTT = TR(WMAT)                                                                 [W']
MATRIX AMAT = XPXINVXP*WX                                                [(X'*X)^-1*X'*W*X]
MATRIX TRAMAT = TRACE(AMAT)                 [trace (sum of diagonal elements) of previous line]
EVAL EMI = -NOBS*TRAMAT/(NDF*S0)                                         [mean of Moran's I]
*
MATRIX AMAT2 = AMAT * AMAT
MATRIX TRAMAT2 = TRACE(AMAT2)
MATRIX WW = WMAT + WTT                                                           [W + W']
MATRIX WW2 = WW*WW                                                   [(W + W')*(W + W')]
MATRIX BMAT = XPXINVXP * WW2 * XMAT                        [(X'*X)^-1*X'*(W + W')*(W + W')*X]
MATRIX TRBMAT = TRACE(BMAT)
EWISE WWSTAR2(I,J) = WW(I,J)*WW(I,J)                  [(W + W')*(W + W'), element by element]
MATRIX S1 = SCALE(1.0/2.0)*SUM(WWSTAR2)               [1/2 * sum of all squared elements of WW]
EVAL VMI1 = NOBS**2.0/((S0**2)*NDF*(NDF+2))                         [variance of MI, step 1]
EVAL VMI = VMI1*(S1 + 2.0*TRAMAT2 - TRBMAT - 2.0*(TRAMAT**2.0)/NDF)                [variance: MI]
EVAL MORANIN = (MORANRES - EMI)/SQRT(VMI)                                       [Z-value: MI]
CDF(NOPRINT) NORMAL MORANIN                                                       [inference]
```

```
FETCH PMORRES = SIGNIF
DISPLAY ' '                                                              [print empty line]
DISPLAY 'MORAN*S I FOR'                                                  [output headings]
DISPLAY 'SPATIAL ERROR'
DISPLAY 'DEPENDENCE'  '       Z-VALUE'   '         PROB'
DISPLAY  MORANRES MORANIN PMORRES
*
****LM ERROR
*
DECLARE RECTANGULAR WW4
*
MATRIX WW4 = (WTT*WMAT) + (WMAT*WMAT)                                    [(W'*W) + (W + W)]
MATRIX TRWW4 = TRACE(WW4)
EVAL LMERR = (NOBS*MORANRES)**2/TRWW4                      [Lagrange Multiplier (error)]
CDF(NOPRINT) CHISQUARE LMERR 1                [inference: Chi-Squared, 1 degree of freedom]
FETCH LMERRX2 = SIGNIF
DISPLAY ' '
DISPLAY 'LM ERROR' '          PROB'
DISPLAY LMERR LMERRX2
*
****LM LAG
*
DECLARE SERIES ZSER
*
MATRIX RWY = RW*YMAT                                                     [e'*W*y]
MATRIX ZSER = WX*BETAEQ1                                                 [W*X*Betas]
LINREG(LASTREG,NOPRINT) ZSER /          [auxiliary regression-need residual sum of squares]
EVAL LMLAG = ((RWY/SIGSQML)**2)/(RSS/SIGSQML+TRWW4)          [Lagrange Multiplier (lag)]
CDF(NOPRINT) CHISQUARE LMLAG 1                [inference: Chi-Squared, 1 degree of freedom]
FETCH LMLAGX2 = SIGNIF
DISPLAY ' '
DISPLAY 'LM LAG' '           PROB'
DISPLAY LMLAG LMLAGX2
*
END
```

Table 4.4 - Shazam: regdiag.sha

```
=*
=*[notes are contained in brackets]
=*[* denotes comments, = means no echo of command line in output file, and ? suppresses
output from these lines]
=* .
=*   USER INPUT SECTION
[CHANGE ONLY UNDERSCORED ITEMS]
=*
=?FILE 11 COLREG.ASC                                                   [enter data file name]
=?FILE 12. COLWS.BIN               [weights matrix file name, . after the 12 if in binary format]
=?READ(11) CRIME INCOME HOVAL /CLOSE              [enter all variable names for data in above file]
=?READ(12) WMAT / BINARY ROWS=49 COLS=49 CLOSE              [binary if needed, # rows, columns]
=?COPY INCOME HOVAL EXPVAR                                          [explanatory variable names]
=?COPY CRIME DEPVAR                                                    [dependent variable name]
=*
=*   END OF USER INPUT SECTION
=*
=SET NOCC                                                             [turn off carriage control]
=SET NOWARN                                                          [turns off messages in output]
=*
=OLS DEPVAR EXPVAR /RESID=RESIDS COEF=BETAS                                  [preliminary regression]
=*
=*OFTEN USED EXPRESSIONS & MATRICES
=*
=GEN1 NOBS = $N                                                          [number of observations]
=GEN1 NXVAR = $K                                  [number of estimated parameters in regression]
=GEN1 SSE1 = $SSE                                                      [sum of squared residuals]
=GEN1 SIGSQML = SSE1/NOBS                                           [ML estimate of error variance]
=GENR ONEVEC = 1.0                                                               [vector of ones]
=MATRIX XMAT = EXPVAR|ONEVEC                [add vector-ones to matrix of explanatory variables]
=MATRIX XP = XMAT'                                                     ['=matrix transposition]
=MATRIX XPX = XP*XMAT                                                                    [X'*X]
=MATRIX XPXINV = INV(XPX)                                                   [inv=matrix inverse]
=MATRIX XPXIXP = XPXINV*XP                                                           [(X'*X)-1*X']
*
=*
*                                                                                            .
******************************************SPATIAL
DEPENDENCE***********************************
*
******MORAN'S I FOR SPATIAL ERROR DEPENDENCE
*
=MATRIX RW = RESIDS'*WMAT                                                                 [e'*W]
=MATRIX S0 = (WMAT*ONEVEC)'*ONEVEC             [sum of matrix elements, parameter for Moran's I]
=MATRIX MORANI = (NOBS/S0)*(RW*RESIDS/SSE1)                        [Moran's I - unstandardized]
=*
=GEN1 NDF = $DF                                                              [degrees of freedom]
=MATRIX WX = WMAT*XMAT                                                                     [W*X]
=MATRIX A = XPXIXP*WX                                                          [(X'*X)-1*X'*W*X]
=MATRIX TRAMAT = TRACE(A)                  [trace (sum of diagonal elements) of previous line]
=GEN1 EMI = -(NOBS*TRAMAT)/(NDF*S0)                                            [mean of Moran's I]
=*
=MATRIX A2 = A*A
=MATRIX TRAMAT2 = TRACE(A2)
=MATRIX WTT = WMAT'                                                                         [W']
=MATRIX WW = WMAT + WTT                                                                 [W + W']
=MATRIX WW2 = WW*WW                                                         [(W + W')*(W + W')]
=MATRIX BMAT = XPXIXP*WW2*XMAT                                 [(X'*X)-1*X'*(W + W')*(W + W')*X]
=MATRIX TRBMAT = TRACE(BMAT)
=MATRIX S1 = (1.0/2.0)*((WW**2)*ONEVEC)'*ONEVEC      [1/2 * sum of all the squared elements of WW]
=GEN1 VMI1 = NOBS**2.0/((S0**2.0)*NDF*(NDF+2.0))                          [variance of MI, step 1]
=GEN1 VMI = VMI1*(S1 + 2.0*TRAMAT2 - TRBMAT - ((2.0/NDF)*(TRAMAT**2)))            [variance: MI]
=DISTRIB MORANI / TYPE=NORMAL MEAN=EMI VAR=VMI                                         [inference]
=*
*
******LM ERROR
*
=*
=MATRIX WW4 = (WTT*WMAT) + (WMAT*WMAT)                                       [(W'*W) + (W + W)]
=MATRIX TRWW4 = TRACE(WW4)
=GEN1 LMERR = (NOBS*MORANI)**2/TRWW4                              [Lagrange Multiplier (error)]
=DISTRIB LMERR / TYPE=CHI DF=1                     [inference: Chi-Squared, 1 degree of freedom]
=*
*
******LM LAG
*
=MATRIX RWY = RW*DEPVAR                                                                  [e'*W*y]
```

```
=MATRIX ZMAT = WX*BETAS                                                            [W*X*Betas]
=?OLS ZMAT XMAT / NOCONSTANT                  [auxiliary regression-need sum of squared errors]
=MATRIX LMLAG = (RWY/SIGSQML)**2/(($SSE)/SIGSQML + TRWW4)           [Lagrange Multiplier (lag)]
=DISTRIB LMLAG / TYPE=CHI DF=1                 [inference: Chi-Squared, 1 degree of freedom]
*
*
STOP
```

Table 4.5 - Splus: regdiag.spl

```
# FUNCTION REGDIAG - regression with spatial diagnostics
#
# Purpose: Diagnose OLS regression model for spatial dependence
# Inputs:  xmat    matrix of explanatory variables (NO column of ones)
#          depvar  dependent variable
#          wmat    spatial weights matrix
#          resids  vector of residuals from an OLS regression ($residuals)
#          bhat    vector of OLS regression coefficients ($coef)
#
# Return:  3x3 matrix of statistics, z-values, and probabilities.
#
regdiag_function(xmat,depvar,wmat,resids,bhat) {
#
#often used expressions & matrices
#
outmat_matrix(c(NA),nrow=3,ncol=3)     #matrix for return values
dimnames(outmat)_list(c('Moran','LMerr','LMlag'),c('Stat','Z-val','Prob'))
nobs_nrow(xmat)                        #number of observations
k_ncol(xmat) + 1
df_nobs - k                            #degrees of freedom
### add column of ones, since later stuff assumes it
xmat_cbind(1,xmat)
epe_t(resids)%*%resids                 #[e'*e]
sigsqml_epe/nobs                       #ML estimate of error variance
tx_t(xmat)                             #X'
xpx_tx%*%xmat                          #X'*X
xpxi_solve(xpx)                        #inverse of X'* X
#
###########################Spatial Dependence##########################
#
#     Moran's I for Spatial Error Autocorrelation
#
rw_t(resids)%*%wmat                    #[e'*W]
s0_sum(wmat)                           #sum of all elements in W
mres_(nobs/s0)*(rw%*%resids/epe)       #Moran's I
outmat[1,1]_mres[1,]
#
wx_wmat%*%xmat                         #[W*X]
amat_xpxi%*%tx%*%wx                    #[inv(X'*X)*X'*W*X]
tramat_sum(diag(amat))                 #trace (sum of diagonal elements)
emres_-(nobs*tramat)/(df*s0)           #mean of Moran's I
#
a2_amat%*%amat
tra2_sum(diag(a2))
tw_t(wmat)                             #[W']
wpw2_(wmat+tw)%*%(wmat+tw)             #[(W+W')*(W+W')
b_xpxi%*%tx%*%wpw2%*%xmat
#[inv(X'*X)*X'*(W+W')*(W+W')*X]
trb_sum(diag(b))
vmres1_(nobs^2/(s0^2*df*(df+2)))       #intermediate step: Var of MI
s1_0.5*(sum((tw+wmat)^2))
vmres_vmres1*(s1+2*tra2-trb-((2/df)*tramat^2))    #variance of Moran's I
sdmres_sqrt(vmres)
mresz_(mres-emres)/sdmres              #Z value of Moran's I
outmat[1,2]_mresz[1,]
prob_pnorm(mres,mean=emres,sd=sdmres) #inference
probml_2*(1-prob)
outmat[1,3]_probml
#
#     Lagrange Multiplier (Error)
#
wpww2_(tw%*%wmat)+(wmat%*%wmat)          #[(W'*W)+(W*W)]
trwpww2_sum(diag(wpww2))
lmerr_((nobs*mres)^2)/trwpww2            #Lagrange Multiplier (error)
outmat[2,1]_lmerr[1,]
prob_pchisq(lmerr,df=1)
probml_1-prob
outmat[2,3]_probml
#
#     Lagrange Multiplier -- Lag
#
rwy_rw%*%depvar                          #[e'*W*y]
z_wx%*%bhat                              #[W*X*betas]
m_(diag(nobs))-xmat%*%xpxi%*%tx          #[I-X*inv(X'*X)*X']
lmlag_((rwy/sigsqml)^2)/((t(z)%*%m%*%z)/sigsqml+trwpww2) #LM Lag
outmat[3,1]_lmlag[1,]
prob_pchisq(lmlag,df=1)
probm1_1-prob
outmat[3,3]_probm1
outmat
}
```

CHAPTER 5
MAXIMUM LIKELIHOOD ESTIMATION OF SPATIAL AUTOREGRESSIVE MODELS

INTRODUCTION

The routines included in this chapter deal with the maximum likelihood estimation of models that incorporate spatial dependence in the form of a spatially lagged dependent variable Wy. The presence of this term on the right hand side of the regression equation makes ordinary least squares estimation inconsistent (see Anselin, 1988, Chapter 6, for an extensive discussion). This is similar to the situation for endogenous variables in systems of simultaneous equations. This interpretation of the spatial autoregressive model is therefore often referred to as a simultaneous spatial autoregressive model.

Estimation of this model may be done by means of instrumental variables or two-stage least squares methods (Anselin, 1990), although by far the most commonly suggested approach is to use maximum likelihood (ML). Since instrumental variables estimation can be implemented straightforwardly in all software packages considered here, we will only deal with ML estimation in this chapter.

METHODOLOGICAL BACKGROUND

Model Specification

As mentioned, the distinguishing characteristic of a spatial autoregressive model, or more generally, of a mixed regressive spatial autoregressive model, is the inclusion of a spatially lagged dependent variable Wy as one of the explanatory variables. Formally, this is expressed as:

$$y = \rho Wy + X\beta + \varepsilon$$

where y is a N by 1 vector of observations on the dependent variable, Wy is the spatial lag, X is a N by K matrix of observations on the explanatory variables, ε is a N by 1 vector of error terms, ρ is the autoregressive coefficient and β is a K by 1 vector of regression coefficients. The error term ε is typically assumed to follow a normal distribution with mean zero and variance σ^2.

The presence of the spatial lag Wy among the right hand side variables of the model makes ordinary least squares an inconsistent estimator for the coefficients. Indeed, while the expected value of $X'\varepsilon$ will typically be zero (under the standard assumptions), this is not the case for the expected value of $(Wy)'\varepsilon$, irrespective of the dependence or independence of the errors. This invalidates the unbiasedness and consistency properties of OLS, in contrast with the more familiar situation in time series analysis, where OLS is still consistent (but no longer unbiased) as long as the error terms are uncorrelated. The main difference between spatial dependence and serial dependence in the time domain is that the former implies a simultaneity between Wy and ε, whereas the latter does not (for technical details, see Chapter 6 of Anselin, 1988).

Likelihood and Concentrated Likelihood Function

The starting point for the derivation of a maximum likelihood estimate in a spatial autoregressive model is the joint density function for a multivariate normal distribution. The general form for this density is:

$$f(y) = (2\pi)^{-N/2}.|\Sigma|^{-1/2}.\exp[-1/2(y-\psi)'\Sigma^{-1}(y-\psi)]$$

where y is a N by 1 vector of random variates with mean ψ and covariance matrix Σ, and $| \ |$ stands for the determinant of a matrix. When the ψ and Σ consist of unknown parameters that need to be estimated from observations on y, the joint distribution is referred to as a likelihood function. The principle of maximum likelihood estimation consists of finding those values for the coefficients in the model for which the joint density (or likelihood) achieves the highest probability (the maximum likelihood). Once the likelihood is formulated in terms of observable magnitudes (e.g., observations on y), ML estimation becomes an application of optimization: expressions of the parameter estimates in function of observations on the variables in the model are derived from the first and second order conditions for the maximum of a function. In practice, these expressions are found by solving a system of simultaneous equations that results by setting the first partial derivatives of the likelihood equal to zero (the usual first order condition for a maximum or minimum of a function).

The problem with estimating the parameters of a spatial autoregressive model is that we have the joint distribution for the error terms (joint independent normal by assumption), but these are not observable. They are only abstract concepts that allow us to formally express the probabilistic content of the model. However, since these unobservable error terms are functionally related to an observable vector of values for the dependent variable y, we can find the joint distribution for the latter as a function of the former. Moreover, since the functional relation is linear, we know from probability theory that the joint distribution of y will be multivariate normal as well. Formally, the relation between the error term and the other variables is nothing but:

$$\varepsilon = y - \rho Wy - X\beta$$

The joint distribution of the errors is:

$$f(\varepsilon) = (2\pi)^{-N/2}.|\sigma^2 I|^{-1/2}.\exp(-\varepsilon'\varepsilon/2\sigma^2),$$

or

$$f(\varepsilon) = (2\pi)^{-N/2}.(\sigma^2)^{-N/2}.\exp(-\varepsilon'\varepsilon/2\sigma^2),$$

since $\Sigma=\sigma^2 I$ by assumption and the mean of ε is zero. After substituting the expression of ε in terms of the y and X, this becomes:

$$f(\varepsilon) = (2\pi)^{-N/2}.(\sigma^2)^{-N/2}.\exp[-(y-\rho Wy-X\beta)'(y-\rho Wy-X\beta)/2\sigma^2]$$

Since X is assumed to be fixed and non-stochastic, we only need to concern ourselves with the relation between ε and y. The concept used to move from the joint distribution of ε to the joint distribution of y is the Jacobian J of the transformation. It is a well known result from probability theory that:

$$f(y) = |J|.f(\varepsilon),$$

or, the density for y is obtained as the product of the density of ε with the determinant of the Jacobian. Hence, we only need to specify the Jacobian term to move from the density of ε to the density of y. This Jacobian guarantees that the transformed variable has a proper density function. It is defined as the partial derivative of ε with respect to y (note that this is a matrix of partial derivatives of each element ε_i of ε with respect to each element y_j of y):

$$J = \partial\varepsilon \ / \ \partial y$$

From the expression above, it follows that the Jacobian for the spatial autoregressive model is:

$$J = \partial(y-\rho Wy-X\beta) \ / \ \partial y$$

or,

$$J = I - \rho W$$

where I is an identity matrix of dimension N by N.

Consequently, the joint distribution for the vector of observations on y is:

$$f(y) = |I-\rho W|.f(\varepsilon)$$

or,

$$f(y) = |I-\rho W|.(2\pi)^{-N/2}.(\sigma^2)^{-N/2}.\exp[-(y-\rho Wy-X\beta)'(y-\rho Wy-X\beta)/2\sigma^2]$$

By definition, the likelihood for the model takes on the same form. In practice, it is often simpler to manipulate the logarithmic likelihood instead of the likelihood itself. Since the logarithm is a monotone transformation, the coefficients that maximize the log likelihood will be the same as those that maximize the likelihood. The log likelihood that needs to be maximized is:

$$L = \ln |I-\rho W| - N/2 \ln(2\pi) - N/2 \ln(\sigma^2) - (y-\rho Wy-X\beta)'(y-\rho Wy-X\beta)/2\sigma^2$$

which is a function of the parameters ρ, σ^2 and β. Clearly, for the purposes of maximizing the log likelihood, the constant term $- N/2 \ln(2\pi)$ can be ignored. From a careful inspection of L we also see why least squares estimation is not appropriate. Indeed, LS only minimizes the term $(y-\rho Wy-X\beta)'(y-\rho Wy-X\beta)$ and thus ignores the Jacobian expression $\ln|I-\rho W|$. The latter is only zero when $\rho=0$ (since $\ln|I|=0$), or, in other words, when there is no spatial autoregressive term in the model.

The maximum likelihood estimates for the parameters are obtained by solving the following first order conditions:

$$\partial L/\partial\rho = 0$$

$$\partial L/\partial\beta = 0$$

$$\partial L/\partial\sigma^2 = 0$$

Due to the presence of the Jacobian expression, this system does not have an analytical solution. A detailed treatment of the derivation of the ML estimates is beyond the scope of this report (see Chapter 6 in Anselin, 1988 for technical details). However, it is useful to point out that the estimates for β and σ^2 can be expressed as simple functions of the autoregressive parameter ρ. In other words, once we have an estimate for ρ, we can find the estimates for β and σ^2 in a straightforward way.

The ML estimate for β is found to be:

$$b = (X'X)^{-1}X'(I-\rho W)y$$

or, in other words, b is the vector of regression coefficients in a regression of X on $(y-\rho Wy)$ as the dependent variable, conditional upon the value for ρ. It can be easily seen that b can be expressed as:

$$b = (X'X)^{-1}X'y - \rho(X'X)^{-1}X'Wy$$

or

$$b = b_0 - \rho b_L$$

where b_0 and b_L are the standard regression coefficients in a regression of X on y and Wy respectively. Clearly, the coefficient vectors b_0 and b_L are independent of the other parameters in the model, but b is conditional upon the estimate for ρ. A similar result can be found for the ML estimate for the error variance σ^2, which can be expressed as a function of the usual residual sum of squares:

$$\sigma^2 = (y-\rho Wy-Xb)'(y-\rho W-Xb)/N$$

or

$$\sigma^2 = (e_0 - \rho e_L)'(e_0 - \rho e_L)/N$$

where e_0 and e_L are the residual vectors in the regressions for b_0 and b_L:

$$e_0 = y - Xb_0$$

$$e_L = Wy - Xb_L$$

Substitution of these expressions into the log likelihood function yields a so-called concentrated log likelihood function, which is only a function of the autoregressive parameter ρ:

$$L_C = - (N/2)\ln[(e_0-\rho e_L)'(e_0-\rho e_L)/N] + \ln |I-\rho W|$$

The concentrated log likelihood function consists of a sum of squared residuals corrected by the Jacobian term $|I-\rho W|$. It can be maximized (or, alternatively, its negative can be minimized) by means of numerical optimization methods. However, since the Jacobian term changes for each different value of ρ, it will have to be evaluated at each iteration. This is the main practical problem in the application of numerical optimization routines to this concentrated likelihood.

In a by now classic paper, Ord (1975) points out some strategies that simplify this problem considerably. Most importantly, it turns out that the determinant $|I-\rho W|$ can be expressed as a product of a simple function of the eigenvalues of the spatial weights matrix W:

$$|I-\rho W| = \prod_i (1 - \rho\omega_i)$$

or,

$$\ln|I-\rho W| = \sum_i \ln(1 - \rho\omega_i)$$

where ω_i are the eigenvalues. The main implication of this simplification is that there is no longer a need to explicitly evaluate the determinant of a N by N matrix at each iteration. Instead, each $(1-\rho\omega_i)$ can be considered as a scaling factor that can be applied to each observation individually.

Furthermore, the requirements for stationarity of the spatial autoregressive process (i.e., to ensure that the variance of the process is finite, or non-explosive) can be expressed as a simple constraint on the parameter ρ, in function of the largest and smallest eigenvalue of W. As shown in more detail in Anselin (1982), this constraint boils down to:

$$- 1/\omega_{min} \leq \rho \leq 1/\omega_{max}$$

where ω_{min} and ω_{max} are respectively the absolute values for the smallest (i.e., the most negative) and largest root of the spatial weights matrix. For any row-standardized spatial weights matrix, the largest eigenvalue equals 1. However, the smallest root is typically not equal to -1.

Asymptotic Variance Matrix

The variance for the ML estimates of the parameters in a spatial autoregressive model is obtained from the second order conditions for maximizing the log likelihood. This variance matrix has asymptotic validity, but may not be very reliable as an indicator of the precision of estimates in finite samples. If the usual regularity conditions are satisfied, the ML estimates for ρ, β and σ^2 will be asymptotically efficient and thus achieve the so-called Cramer-Rao variance bound. This corresponds to the inverse of the information matrix:

$$[I(\theta)]^{-1} = -E[\partial^2 L/\partial\theta\partial\theta']^{-1}$$

where θ is a K+2 by 1 vector of the model parameters, $[\rho\ \beta'\ \sigma^2]'$. The elements of this inverse cannot be obtained analytically. However, an expression for the information matrix can be derived from the second partial derivatives of the log likelihood function. As a result, the asymptotic variance matrix can be calculated by substituting the values for the ML estimates of the coefficients into the elements of the information matrix and taking the inverse. The upper triangular elements of the information matrix are (for the parameters organized as ρ, β, σ^2):

$$\begin{array}{lll} tr(W_A)^2 + tr(W_A'W_A) + (W_AX\beta)'(W_AX\beta)/\sigma^2 & X'W_AX\beta/\sigma^2 & tr(W_A)/\sigma^2 \\ & X'X/\sigma^2 & 0 \\ & & N/2\sigma^4 \end{array}$$

where $W_A = W(I-\rho W)^{-1}$ to simplify notation, and tr stands for the trace of a matrix. As shown in Anselin (1980, pp. 81-83), the traces of W_A and $(W_A)^2$ simplify to $\Sigma_i\ \omega_i/(1-\rho\omega_i)$ and $\Sigma_i\ [\omega_i/(1-\rho\omega_i)]^2$ respectively.

The asymptotic variance and standard error for the ML estimates are the diagonal elements of the inverse of the information matrix.

Estimation Strategies

There are two common strategies to obtain ML estimates for the spatial autoregressive model: a search strategy and the explicit nonlinear optimization of the log likelihood function. Each will be briefly discussed next.

Search Strategies

As shown above, the concentrated likelihood that needs to be maximized,

$$L_C = -(N/2)\ln[(e_0-\rho e_L)'(e_0-\rho e_L)/N] + \Sigma_i \ln(1-\rho\omega_i)$$

is a function of a single parameter, the autoregressive coefficient ρ. We also know that the range of acceptable values for ρ (for a row-standardized spatial weights matrix) is limited to

$$-1/\omega_{min} \le \rho \le 1/\omega_{max}$$

It is thus fairly straightforward to search this parameter space for the value of ρ that yields the largest L_C. Essentially, one would start with computing L_C for roughly 20 values of ρ, ranging from close to -1 (or close to $-1/\omega_{min}$) to close to +1 (or $1/\omega_{max}$) at intervals of .1. Note that the values of $-1/\omega_{min}$ and $+1/\omega_{max}$ are themselves unacceptable, since they will result in explosive spatial processes. Once the value of ρ that yields the largest L_C is identified, say ρ_1, one starts the process anew, but now going from ρ_1-0.1 to ρ_1+0.1, taking values at intervals of 0.01. This process can be repeated until the desired precision is obtained. Typically, this will result in a very large number of iterations. For example, to obtain a precision of 6 decimals, one would need to compute roughly 120 concentrated likelihoods. However, if one has the time and the patience, this form of "manual iteration" is by far the simplest way to obtain the ML estimates (see Griffith, 1988b, and Griffith et al. 1990, for examples of the implementation of this approach in Minitab and SAS for the model discussed in Chapter 6).

The brute force search approach can be considerably improved upon by exploiting some knowledge about the shape of the concentrated likelihood function. As Ripley (1988, Chapter 2) and Cressie (1991, Chapter 7) illustrate, the log likelihood in a spatial autoregressive model (and thus also L_C) changes with values of ρ in a very smooth fashion, following a reverse U-shape. Hence, as argued in Anselin (1980), this problem lends itself extremely well to the application of a more structured search, such as a bisection search. The main goal of a bisection search is to include the optimal value for the parameter to the desired degree of accuracy between a lower (L) and an upper bound (U), for which:

$$\partial L_C/\partial \rho(\rho_L) > 0$$

$$\partial L_C/\partial \rho(\rho_U) < 0$$

where L_C is the concentrated log likelihood and ρ is the spatial autoregressive parameter. In other words, since the slope of L_C is positive for ρ_L and negative for ρ_U, it must reach zero somewhere in between them. Consequently, the log likelihood must reach a maximum for a value of ρ between ρ_L and ρ_U.

The logic of a bisection search is very simple. At each iteration, the partial derivative of L_C is evaluated for ρ at the midpoint between its lower and upper bound. If this derivative is positive, the midpoint becomes the new lower bound. Conversely, if the derivative is negative, the midpoint becomes the new upper bound. The next iteration proceeds by evaluating the partial derivative for the midpoint between the new lower and upper bounds. The algorithm is guaranteed to converge in a small number of steps, depending on the desired precision (the difference between ρ in two iterations is always equal to half the length of the interval). For example, to obtain a precision of 6 decimals, the bisection search converges in less than 25 iterations (compared to roughly 120 for the brute force search).

The selection of starting values should be such that the lower bound, i.e., the smaller one (in real terms), has a positive partial derivative and the upper bound (the larger value in real terms) a negative partial derivative. The partial derivative is of the form:

$$\partial L_C/\partial \rho = N.(e_0'e_L - \rho.e_L'e_L)/(e_0'e_0 - 2\rho.e_0'e_L + \rho^2.e_L'e_L) - \Sigma_i\, \omega_i/(1 - \rho\omega_i)$$

in the same notation as above. Note that the cross products between the residuals in the two auxiliary regressions (the e_0 and e_L) only need to be computed once, since they do not depend on ρ.

For ρ=0, the partial derivative of LC equals:

$$\partial L_C/\partial \rho(\rho=0) = N.\ e_0'e_L\ /\ e_0'e_0.$$

or, N times the coefficient in a regression of e_L on e_0. If this value is positive, ρ=0 should be taken as ρ_L. The upper bound could be taken as 1-δ, where δ is a small value in order to avoid singularity (for ρ=1, an explosive spatial process would result). For non-standardized spatial weights matrices, the value of +1 should be replaced by $1/\omega_{max}$. Alternatively, the OLS estimate for ρ could be chosen as the upper bound, provided that it is positive and less than 1 (or $1/\omega_{max}$), and that its partial derivative is negative. If the partial derivative for ρ=0 is negative, ρ=0 should be taken as ρ_U. The lower bound could then be taken as $-1/\omega_{min} + \delta$, where δ is a small value in order to avoid singularity. Again, the OLS estimate for ρ could be used in this case as a lower bound, provided that it is negative, larger than $-1/\omega_{min}$ and that its partial derivative is positive. Alternative starting strategies can be constructed using the OLS estimate and the bounds $-1/\omega_{min}$ and +1 (or $1/\omega_{max}$), or any other pair of values (e.g., randomly generated) that satisfies the logic of the bisection search within the allowed parameter space.

Full Nonlinear Optimization of the Log Likelihood Function

A more straightforward way to obtain maximum likelihood estimates for the parameters in the spatial autoregressive model is to carry out an explicit nonlinear optimization of the log likelihood function. Many econometric software packages contain routines that allow the user to specify the form for the likelihood. The optimization is then based on numeric approximations of the partial derivatives or on the explicit computation of their values based on expressions provided by the user. Note that for some software packages a minimization is carried out, rather than a maximization. Clearly, the correct results will be obtained if all the signs in the log likelihood function are reversed.

In most nonlinear optimization routines, the objective function (likelihood function) is supposed to be formulated as a sum of terms that correspond to individual observations. At first sight, the log likelihood function for the spatial autoregressive model does not seem to fit into this format, due to the presence of the Jacobian, i.e., the first term in the following expression:

$$L = \ln |I-\rho W| - N/2 \ln(2\pi) - N/2 \ln(\sigma^2) - (y-\rho Wy-X\beta)'(y-\rho Wy-X\beta)/2\sigma^2$$

However, if the Jacobian is expressed as a function of the eigenvalues of the weights matrix, we can re-write the log likelihood as:

$$L = \Sigma_i \{\ln(1 - \rho\omega_i) - 0.5\ln(2\pi) - 0.5\ln(\sigma^2) - 0.5(y_i-\rho Wy_i-x_i\beta)^2/\sigma^2 \}$$

where the subscript i refers to each observation, x_i is a row vector with the observations on the explanatory variables for i, and the other notation is as before.

This expression conforms to the usual format, provided that we create a pseudo-variable that contains the eigenvalues (see Bivand, 1991, for an example of this approach using Systat).

The partial derivatives of the log likelihood with respect to the parameters are (for detailed derivation, see Anselin, 1980):

$$\partial L/\partial\rho = \Sigma_i \{ Wy_i.e_i/\sigma^2 - \omega_i/(1-\rho\omega_i) \}$$

$$\partial L/\partial\beta_k = \Sigma_i \{ x_{ki}.e_i/\sigma^2 \}$$

$$\partial L/\partial\sigma^2 = 0.5 \Sigma_i \{ -1/\sigma^2 + e_i^2/\sigma^4 \}$$

where e_i is the residual for observation i, $y_i-\rho Wy_i-x_i\beta$, x_{ki} is the k-th term of row vector x_i, and β_k is the k-th regression coefficient.

Two types of problems may occur when using a "canned" nonlinear optimization routine. First, most routines do not prevent the spatial autoregressive parameter from taking on values outside the acceptable range. This will often be a problem when the starting values are poor, or when the true parameter value is close to the bounds of the acceptable range. In those instances, it will be necessary to manually "tune" the optimization algorithm in order to constrain the maximum change allowed in a coeffient between iterations. The extent to which this can be done varies considerably between software packages. A second problem pertains to the estimate of the asymptotic covariance matrix that is provided by the nonlinear optimization routine. Typically, this is not the inverse of the information matrix as outlined above, but is computed by means of numerical approximations. Consequently, inference about the significance of the estimated coefficients that is based on this numerical variance matrix may differ slightly from that derived from the "correct" analytical asymptotic variance matrix.

Interpretation

The estimates of the coefficients in a mixed regressive spatial autoregressive model can be interpreted in two different ways. In one, the main interest is in measuring the strength of the spatial dependence, i.e., in modelling a spatial autoregressive process. From this perspective, the inclusion of the other explanatory variables in the model (X) in addition to the spatial lag (Wy) yields a measure of spatial dependence that controls for the effect of those exogenous variables. In other words, it indicates the extent of spatial autocorrelation, after the other variables have been controlled for. This is an important alternative to the usual descriptive measures of spatial autocorrelation (such as Moran's I and Geary's c). For the latter, an misleading impression of spatial autocorrelation can be given when the spatial clustering in the values for a variable y are really due to spatial clustering in variables X with which y is highly correlated.

In another perspective, the inclusion of a spatial lag in the model allows one to assess the significance of the other explanatory variables, while controlling for spatial dependence. Formally, this can also be expressed as a spatial filter:

$$y - \rho Wy = X\beta + \varepsilon$$

or,

$$(I - \rho W)y = X\beta + \varepsilon$$

This is a regression of a spatially filtered dependent variable on the explanatory variables. In order to carry out an optimal filtering, the estimate for ρ should be obtained from a ML estimation of a mixed regressive spatial autoregressive model (see also Getis, 1990, for an alternative approach to spatial filtering in regression).

OVERVIEW OF THE ROUTINES

User Input Section

The nonlinear optimization facilities implemented in the five software packages considered here differ slightly. In Gauss (*maxlik*), Rats (*nonlin*) and Shazam (*nl*), these procedures find a maximum of the likelihood function, whereas in Splus (*ms*) and Limdep (*minimize*), the negative of the likelihood is minimized. Optimization is based on the same methodology in all packages, i.e., versions of the so-called variable metric or quasi-Newton methods, the well-known DFP (Davidson-Fletcher-Powell), BHHH (Berndt-Hall-Hall-Hausman), and BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithms (for technical issues, see the review in Press et al., 1988). Several packages provide a choice between these algorithms. In the routines presented here, we used the DFP algorithm, except for Rats (BFGS).

The routines described in this chapter need to have starting values for the parameters passed to them. This is illustrated here with values appropriate for the analysis of the Columbus, Ohio dataset included with this report. As in the previous chapters, the routines are in the form of functions for Gauss and Splus, and in the form of batch programs for the other software packages. Prior to the nonlinear optimization, the values for the variables in the model, the spatial weights matrix, and the eigenvalues of the spatial weights matrix must be computed or loaded into the calling program. In each routine, the form for the log-likelihood for each observation must be spelled out, with the specific variables in place. The degree of generality by which this may be carried out varies between the packages, as detailed below. In Gauss, the partial derivatives of the likelihood are computed explicitly (though this is not necessary), for the other packages the computation of the partial derivatives is done numerically. All routines except Splus return an estimate of the asymptotic variance. This is based on a numerical approximation and does not necessarily correspond to the values for the expression given above.

Gauss

The Gauss procedure *maxlik* takes as input a N by K matrix **zz** with observations on all the variables in the problem, a matching K by 1 vector of variable names or variable index numbers **vars** (this vector is not essential to the routine, but must be passed), a pointer to a procedure for the computation of the log-likelihood **&lmlyx** and a vector of starting values for all parameters, **bstart**. The matrix **zz** should be structured as follows:

- First column: dependent variable, y
- Second column: spatially lagged dependent variable, Wy
- Third column: vector of ones (constant term)
- Fourth to next to last columns: explanatory variables (X)

- Last column: eigenvalues of the spatial weights matrix (rw)

Note that the inclusion of the eigenvalues as a pseudo variable allows the expression of the log likelihood in the standard format used by *maxlik.*

The easiest way to deal with the vector **vars** is to initialize it as a vector of sequential numbers, up to the dimension of the problem, as in:

vars=*seqa*(1,1,*cols*(**zz**));

The *maxlik* procedure returns a vector of estimates, **b**, the value of the likelihood function, **lik**, the value of the gradient at the final iteration, **gra**, and an approximation to the asymptotic variance matrix, **hh**. The calling statement is:

{ **b,lik,gra,hh** }=*maxlik*(**zz,vars,&lmlyx,bstart**);

Note that in order for this to work, the procedure *maxlik* must be contained in the current library or must be included explicitly by means of a use *maxlik* statement as the first line of the program.

There are three global variables that are often useful to control the maximization routine: **__row**, **__btol**, and **__gdproc**. When set equal to one (**__row = 1;**), the first option specifies that the log likelihood will be evaluated for a single observation. The second option sets the convergence criterion for the parameter values, while the third is a pointer to the procedure in which the partial derivatives are computed, e.g., **__gdproc=&glmlyx;**, where *glmlyx* is the procedure with the partial derivatives.

Limdep

In addition to the usual commands to load the data and the spatial weights matrix (see Chapter 4), the eigenvalues of the weights matrix must be input as well, by means of a *read* command (see Table 5.2). As before, the *namelist* command is used to specify the dependent variable and the set of explanatory variables.

The *create* command (line 21) copies the dependent variable into a variable, **yvar**, which is a name that is not application specific. As a result, the dependent variable is stored in two forms, the standard variable form and the namelist form. This is necessary because some Limdep commands require the standard variable form, while others require the namelist form.

The final command of the user input section is *calc* (line 22). This command specifies the convergence criterion, **epsilon**, for the optimization procedure.

Following the user input section, the spatially lagged dependent variable, **wy** is computed internally (line 28). In the listing of Table 5.2, the coefficient starting values are obtained from an OLS regression. The sum of squared deviations, **sumsqdev** (line 35), is a temporary variable generated by the *regress* command. It is used to form the starting value for σ^2. A vector containing all of the starting values needed for the log likelihood function is constructed in lines 37-39. Note that this is not required, and other initial values may be set as well. However, this must be done explicitly.

The subcommand *fcn* of *minimize* (lines 44-48) represents the negative of the log likelihood expression as Limdep uses a *minimize* routine. The *dot*[**namelist**] function of Limdep provides an efficient way to generate the σ_j label$_j$ * variable$_j$. The labels are the names of the starting values. Therefore, the bracketed term of the log likelihood function above can be expressed as **yvar** - *dot*[**z**] (line 48). The *namelist* **z** (line 41) starts with a zero term because the first starting value, σ^2, does not appear in the bracketed term. The label **rols** (line 46) represents the OLS estimate for the parameter ρ. The dot function multiplies **rols**, by the second element in **z**, the **wy** variable. In this manner, the only subcommand of *minimize* that needs to be changed to set up the minimization for a given application is *labels* in line 54.

The number of coefficient labels, **bt*** (line 54) should equal the number of variables entered into **xmat** in the user input section. The *tlb* subcommand of *minimize* (line 51) enters the parameter convergence criterion set by the user in the declaration of **epsilon** (line 22). The default setting for the maximum number of iterations is 50 in Limdep.

Rats

There are three commands in the input section of *mlyx.rat* (Table 5.3) in addition to those outlined for the Rats routines in Chapters 3 and 4. The eigenvalues are loaded (lines 14-15) in essentially the same way as the weights matrix. The re-naming of the dependent variable (line 16) is a straightforward application of the *set* command. The spatially lagged dependent variable (line 18) is created as a typical matrix product.

After the user input section, an OLS regression is computed (line 26) to obtain starting values for the nonlinear estimation. The set of explanatory variables, including the spatially lagged dependent variable and constant term, is saved by the *make* command (line 27). This allows the explanatory variables to be expressed in the *frml* command, discussed below, without the user having to explicitly enter the variable names.

In order to run a minimization in Rats, four commands are needed: *nonlin*; *frml*; the group of *eval* statements; and *maximize*. The *nonlin* command in Rats (line 31) serves to specify the parameters that are to be estimated. For the Columbus data set, there are two explanatory variables, **income** and **hoval**, in addition to the constant term and the lagged dependent variable. Therefore, five parameter names appear in the *nonlin* command. The final two labels listed, **bt1** and **bt2**, are the associated coefficients of **income** and **hoval**.

The *frml* command (lines 34-40) specifies the log likelihood function. Rats maximizes the log likelihood. Therefore, the specification shown on the *frml* line has the same sign as the log likelihood function listed earlier in this chapter. The last line (line 40) of the *frml* command will also require editing if you are not using exactly two explanatory variables.

The starting values for the optimization are entered in the two lines of *eval* statements (lines 44 and 46). The *eval* statements take OLS estimates of the parameters and rename them to correspond to the parameter names used in the *frml* command. Once again, the number of **bt** coefficients should be increased by modifying line 46 if additional explanatory variables are used.

Finally, the optimization is invoked by the *maximize* command (line 49). Several optional values are specified in the parentheses placed next to the *maximize* command. The *vcv* option allows the estimated variance-covariance matrix of the coefficients to be printed. The *trace* option displays intermediate results at each iteration. In the *iteration* option, the number of iterations is limited to 50.

Shazam

For Shazam (Table 5.4), the only additions to the user input section of Chapter 4 are the loading of the eigenvalues and the selection of the convergence criterion, **epsilon**. The eigenvalues are loaded into Shazam in the same manner as the spatial weights matrix (lines 13 and 18). **Epsilon** is assigned a value by the user in line 22. However, Shazam will not allow the criterion to be less than 0.000001. Specifying six or more zeroes to the right of the decimal place will result in an error message stating that the convergence has been set to zero.

Next, the spatially lagged dependent variable is created as a basic matrix product, **wy** (line 32). OLS is then run to obtain starting values (line 35). By using a temporary variable from the OLS output, the number of estimated parameters, **nest**, is calculated (line 36).

The *nl* command in Shazam serves to set up the nonlinear optimization. Since one equation is being optimized, a one immediately follows on the *nl* command line (line 47). The options set behind the / specify the number of coefficients to be estimated, the "*log density*" format of the function, the starting vector name, and the convergence criterion.

The *logden* command allows the likelihood function to be estimated as the sum of the single observation's log densities.

Shazam provides a maximization routine, so the equation specified in the *eq* command line (line 48) has the same sign as the log likelihood function at the beginning of this section. The equation is listed in much the same manner as Limdep and Rats. A difference exists, however, in that the starting values for the coefficients are not linked to the parameter names used in the *eq* command. Using a starting value vector instead of a list requires that the parameter names appear in the *eq* command in the order of the starting value vector.

Splus

The syntax for *mlyx.spl* is: *mlyx*(**xmat**,**depvar**,**wmat**,**wroot**,**epsilon**), where **xmat** is a matrix of explanatory variables, **depvar** is a vector of the dependent variable, **wmat** is the spatial weight matrix, **wroot** is a vector of eigenvalues of **wmat**, and **epsilon** is the convergence criterion (a single number). The function returns the output from the nonlinear estimation, and additional information (starting values and values of the objective function at each iteration) is output to the screen. Use *sink*("**<filename>**") to capture all output to the ASCII file **<filename>**.

If you are not using exactly two explanatory variables, then lines 19, 39, and 45 of the code in Table 5.5 must be modified so that they reference the correct number of variables. The comments in the source code describe the specific modifications you need to make. In addition, the starting value for **ro** (line 38) may need to be changed in order to achieve convergence. See the next section, "Performance evaluations" for a discussion of the sensitivity of Splus to the starting value for **ro**.

The Splus optimization routine *ms* must be called with an explicit formulation of the likelihood expression and all parameters. The *ms* command estimates the log likelihood function as the sum of the log densities of single observations. The data for *ms* must be placed in a data frame **wyframe**. A data frame is an Splus data object which contains observations on the variables used by the model. The entries to the data frame are the matrix **wyxmat**, the eigenvalue vector, and the dependent variable. The starting values are entered when specifying the parameters to be included in the data frame.

The ~ symbol is used to denote the formula (objective function) used by the *ms* command. The formula is followed by the name of the data frame, the convergence criterion, and the *trace* option. The underlined items in the objective function must be modified. The *trace* option is set to true so that the intermediate results will be displayed for each iteration. Similar to the *ls.print* command to display the OLS regression output, *summary.ms* is used to print the output from *ms* in tabular form.

Performance evaluations

As evident from Tables 5.1-5.5, the routines do not require a great deal of code. However, the models are difficult to specify within the packages, due in part to the lengthy nature of the right-hand side expressions. The commercial software packages have geared the nonlinear routines to interactive use. The packages are sensitive to three issues: the scaling of the variables; the starting values; and the estimated variance covariance matrix. Rats and Shazam appear to be the most robust with regard to these issues on the basis of the work conducted in this study.

As discussed previously, Limdep is particularly sensitive to the scaling of the variables.

Of the five packages, Splus is the most sensitive to starting values. For example, in the *mlyx* model with the Columbus data set, the OLS starting value of 0.56 for ρ generates an error "NAs in the log term" (i.e. negative values). However, a starting value of 0.25 for ρ will allow the *mlyx* routine to run. For all of the packages, when the autoregressive coefficient exceeds the acceptable range at a given iteration, the model will not converge. Typically, an error message is generated that the estimated variance covariance matrix is singular. In Splus, a message will be generated stating that there are NA values in the $\log(1 - \rho\,\omega)$ term.

REFERENCES

Anselin, Luc, 1980. *Estimation Methods for Spatial Autoregressive Structures.* Regional Science Dissertation and Monograph Series, Cornell University, Ithaca, NY.

Anselin, Luc, 1982. A note on small sample properties of estimators in a first-order spatial autoregressive model, *Environment and Planning A*, 14, 1023-30.

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

Anselin, Luc, 1990. Some robust approaches to testing and estimation in spatial econometrics, *Regional Science and Urban Economics*, 20, 141-163.

Bivand, Roger, 1991. Systat-compatible software for modelling spatial dependence among observations, *Paper Presented at the 7th European Colloquium on Theoretical and Quantitative Geography*, Stockholm, Sweden.

Cressie, Noel, 1991. *Statistics for Spatial Data* (New York: Wiley).

Getis, Arthur, 1990. Screening for spatial dependence in regression analysis, *Papers, Regional Science Association*, 69, 69-81.

Griffith, Daniel, 1988. Estimating spatial autoregressive model parameters with commercial statistical packages, *Geographical Analysis*, 20, 176-86.

Griffith, Daniel, Robine Lewis, Bin Li, Irina Vasiliev, Susanna McKnight and Xiamoming Yang, 1990. Developing Minitab software for spatial statistical analysis: a tool for education and research, *The Operational Geographer*, 8, 28-33.

Ord, Keith, 1975. Estimation methods for models of spatial interaction, *Journal of the American Statistical Association*, 70, 120-26.

Press, William H., Brian P. Flannery, Saul A. Teukolski and William T. Vetterling, 1988. *Numerical Recipes in C: The Art of Scientific Computing* (Cambridge: Cambridge University Press).

Ripley, Brian, 1988. *Statistical Inference for Spatial Processes* (Cambridge: Cambridge University Press).

THE MLYX.* PROGRAMS: SPATIAL LAG MODEL

Table 5.1 - Gauss: MLYX.GAS

```
/* LMLYX log likelihood for spatial autoregressive model
**       for input to MAXLIK
**
** FORMAT: llik=lmlyx(b,zz);
**
** INPUT:  b   -- vector with parameter values
**                b[1]   -- rho, spatial autoregressive parameter
**                b[1:next to last] -- regression coefficients
**                                     includes constant
**                b[last] -- sigma squared, error variance
**         zz  -- row vector with single observation on variables
**                first element  -- y, dependent variable
**                second element -- Wy, spatial lagged dependent var
**                third to next to last -- X, explanatory variables,
**                                           including constant term
**                last element   -- eigenvalue of spatial weights matrix
**                                     (used in Jacobian term)
**
** OUTPUT: llik -- log likelihood evaluated for one observation
**
** REMARK: log likelihood includes ln(2pi)
**         input and output structured to conform to call from proc maxlik
*/

proc(1)=lmlyx(b,zz);
local k,k1,e,llik,rho,bb,sig2,y,Wy,x,rw;
/* dimensions of parameter vectors */
k=rows(b)-1;                  /* k does not included sigma squared */
k1=rows(b);                   /* k1 includes all elements */
/* initialize parameters in standard notation */
rho=b[1];                     /* spatial autoregressive coefficient */
bb=b[2:k];                    /* regression coefficients */
sig2=b[k1];                   /* error variance */
/* initialize variables */
y=zz[1,1];                    /* dependent variable */
Wy=zz[1,2];                   /* spatially lagged dependent variable */
x=zz[1,3:cols(zz)-1];         /* explanatory variables */
rw=zz[1,cols(zz)];            /* eigenvalue of spatial weights matrix */
e=y - rho*Wy - x*bb;          /* residual */
/* log likelihood for single observation */
llik=-0.5.*ln(2.0.*pi) -0.5.*ln(sig2)-0.5.*(e'e)./sig2 + ln(1.0-rho*rw);
retp(llik);
endp;

/* GLMLYX gradient of loglikelihood in spatial autoregressive model
**        for input into MAXLIK with global option __gdproc=&glmlyx;
**
** FORMAT: glik=glmlyx(b,zz);
**
** INPUT:  b   -- vector with parameter values
**                b[1]   -- rho, spatial autoregressive parameter
**                b[1:next to last] -- regression coefficients
**                                     includes constant
**                b[last] -- sigma squared, error variance
**         zz  -- row vector with single observation on variables
**                first element  -- y, dependent variable
**                second element -- Wy, spatial lagged dependent var
**                third to next to last -- X, explanatory variables,
**                                           including constant term
**                last element   -- eigenvalue of spatial weights matrix
**                                     (used in Jacobian term)
**
** OUTPUT: glik -- row vector with partial derivatives of log likelihood
**                  evaluated for a single observation
**                  elements in same order as b, but organized as row
**
** REMARK: input and output structured to conform to call from proc maxlik
*/


proc(1)=glmlyx(b,zz);
local k,k1,rho,bb,sig2,y,Wy,x,rw,e,dldrho,dldb,dldsig,glik;
/* dimensions of parameter vectors */
k=rows(b)-1;                  /* k does not included sigma squared */
k1=rows(b);                   /* k1 includes all elements */
/* initialize parameters in standard notation */
```

```
rho=b[1];                     /* spatial autoregressive coefficient */
bb=b[2:k];                    /* regression coefficients */
sig2=b[k1];                   /* error variance */
/* initialize variables */
y=zz[1,1];                    /* dependent variable */
Wy=zz[1,2];                   /* spatially lagged dependent variable */
x=zz[1,3:cols(zz)-1];         /* explanatory variables */
rw=zz[1,cols(zz)];            /* eigenvalue of spatial weights matrix */
e=y - rho*Wy - x*bb;          /* residual */
/* partial derivatives of log likelihood for single observation */
dldrho=Wy*e./sig2 - rw./(1.0-rho*rw);  /* partial derivative for rho */
dldb=(x.*e)./sig2;                     /* partial derivative for b */
dldsig=0.5 .* (e^2/sig2 - 1.0)./sig2;  /* partial derivative for sig2 */
glik=dldrho~dldb~dldsig;               /* partial derivatives as row vector */
retp(glik);
endp;
```

Table 5.2 - Limdep: MLYX.LIM

```
?
??????????????????????????????USER INPUT SECTION??????????????????????
[CHANGE ONLY UNDERLINED ITEMS]
?
LOAD ; FILE = COLWS.LIM $                       [enter output file from weights preprocessing
                                                                                     routine]
READ ; NOBS = 49                                                   [number of observations]
     ; NVAR = 1                                                      [no change this line]
     ; NAMES = WROOT                                                 [no change this line]
     ; FORMAT = BINARY                                               [no change this line]
     ; FILE = COLWS.EIG $                                       [file name for eigenvalues]
READ ; NOBS = 49                                                    [number of observations
     ; NVAR = 3                                            [number of variables in data file]
     ; NAMES = 1                          [variable names/or # lines names occupy in file]
     ; FILE = COLREG.ASC $                                                 [data file name]
NAMELIST ; DEPVAR = CRIME                                           [dependent variable name]
         ; XMAT = INCOME,HOVAL $                 [explanatory variable names, do not add
                                                                                   constant

                                                           term or lagged dependent variable]
CREATE ; YVAR = CRIME $                                [dependent variable name for minimize]
CALC; NOLIST; EPSILON = 0.0000001 $                   [convergence criterion for parameters]
??????????????????????????????????END USER INPUT????????????????????????
?
NAMELIST; XMAT1 = ONE,XMAT $                     [add column of ones to explanatory variables]
MATRIX ; NOLIST; DEPVARV = COPY(DEPVAR) $              [move dependent variable to matrix
                                                                                 work area]
MATRIX ; NOLIST; WY = WMAT | DEPVARV $                   [spatially lagged dependent variable]
CREATE ; NOLIST; WY1 = 0 $
NAMELIST; WY2 = WY1 $
CREATE;  WY2 = DTA(WY) $                                           [move Wy to data work area]
NAMELIST ; WYXMAT = WY2,XMAT1 $                       [add Wy to explanatory variable list]
?
REGRESS; LHS = DEPVAR; RHS = WYXMAT $                                [OLS estimation of model]
CALC;  SIGSQML = SUMSQDEV / NREG $                          [ML estimate of the error variance]
?
MATRIX ; NOLIST; SLIST = SIGSQML*IDEN(1,1)                        [sigsqml & betas are starting
                                                                       values for minimize]

        ; SLIST = SLIST/B $
CREATE ; ZERO = 0 $                                               [create variable of zeroes]
NAMELIST ; Z = ZERO,WY2,ONE,XMAT $                     [set up for dot product in minimize]
CALC; NOLIST; PIE2 = 2*PI $
?
MINIMIZE ; FCN = 0.5*LOG(PIE2)                                [set up function to be minimized]
                + 0.5*LOG(SIGSQ)
                - LOG(ONE-ROLS*WROOT)                               [1 - rho*eigenvalues of W]
                + (0.5/SIGSQ)
                * (YVAR - DOT[Z])^2                          [(y - rho*Wy - alpha - bt1*x1
                                                                                - bt2*x2)^2]
         ; START = SLIST                                    [starting values for parameters]
         ; TLB = EPSILON                                [convergence criterion for parameters]
???????????????????????????USER INPUT SECTION???????????????????????
[CHANGE ONLY UNDERLINED ITEMS]
         ; LABELS = SIGSQ,ROLS,ALPHA,BT1,BT2 $                       [have as many bt's as
                                    variables entered in xmat (don't count constant term)]
????????????????????????????????END USER INPUT???????????????????????
STOP
```

Table 5.3 - Rats: MLYX.RAT

```
OUTPUT NOECHO                                           [no command echo in output file]
*
*****************************USER INPUT SECTION*************************
*(CHANGE ONLY UNDERLINED ITEMS)
*
ALLOCATE 0 49                                           [enter number of observations]
OPEN DATA COLREG.RAT                                          [enter data file name]
DATA(FORMAT=RATS) / CRIME INCOME HOVAL                  [enter file format, all variable
                                                                         names in file]
DECLARE RECTANGULAR WMAT(49,49)                     [enter dimensions (= # observations)]
OPEN DATA COLWS.BIN                                        [enter weights matrix file]
READ(FORMAT=BINARY) WMAT                                                [file format]
OPEN DATA COLWS.EIG                                [file containing eigenvalues of W]
DATA(FORMAT=BINARY) / WROOT                                             [file format]
SET DEPVAR = CRIME(T)                                      [name of dependent variable]
DECLARE SERIES WY                                                [no change this line]
MATRIX WY = WMAT * DEPVAR                                        [no change this line]
EQUATION EQ1 DEPVAR                                              [no change this line]
#WY CONSTANT INCOME HOVAL                   [enter names of explantory variables, do not
                                          change the name or placement of WY or Constant]
*
******************************END USER INPUT***************************
*
OUTPUT NOTSTAT                                                   [no t-stat printing]
LINREG(EQUATION=EQ1) DEPVAR                                         [OLS regression]
MAKE(LASTREG) WYXMAT                                [save LHS variables in a matrix]
EVAL PI2 = 2*PI                                                            [scalar]
*
*************************************************************************
NONLIN SIGSQ ALPHA RO BT1 BT2               [enter # of bts as explanatory variables,
                              do not count the spatially lagged variable, or the constant term]
*************************************************************************
FRML MLYX = -0.5*LOG(PI2) - $
    0.5*LOG(SIGSQ) + $
    LOG(1.0-RO*WROOT(T)) - $
    (0.5/SIGSQ) *  $
    (DEPVAR(T) - RO*WYXMAT(T,1) - ALPHA*WYXMAT(T,2) $
*************************************************************************
    - BT1*WYXMAT(T,3) - BT2*WYXMAT(T,4))**2                 [enter number of bt*wyxmat
                      terms as in the nonlin command, bt3*(wyxmat(t,5) is next term if needed]
*************************************************************************
*
EVAL SIGSQ = RSS/NOBS ; EVAL RO = BETA(1) ; EVAL ALPHA = BETA(2)
*************************************************************************
EVAL BT1 = BETA(3) ; EVAL BT2 = BETA(4)                          [see nonlin command]
*************************************************************************
*
MAXIMIZE(VCV,TRACE,ITERATIONS=50) MLYX /
*
END
```

Table 5.4 - Shazam: MLYX.SHA

```
=*
=*[notes are contained in brackets]
=*[* denotes comments, = means no echo of command line in output file, and ? suppresses
output from these lines]
=*
=************************ USER INPUT SECTION ************************
[CHANGE ONLY UNDERSCORED ITEMS]
=*
=?FILE 11 COLREG.ASC                                            [enter data file name]
=?FILE 12. COLWS.BIN                         [weights matrix file name, . after the 12 if
                                                                     in binary format]
=?FILE 13. COLWS.EIG                                   [file containing eigenvalues]
=?READ(11) CRIME INCOME HOVAL /CLOSE                   [enter all variable names for
                                                                 data in above file]
=?READ(12) WMAT / BINARY ROWS=49 COLS=49 CLOSE                [binary if necessary,
                                                                  # rows & # columns]
=?READ(13) WROOT / BINARY ROWS=49 COLS=1 BYVAR CLOSE              [number of observ]
=?COPY INCOME HOVAL XMAT                    [names of explanatory variables, do not add a
                                             constant term or lagged dependent variables]
=?COPY CRIME DEPVAR                                      [name of dependent variable]
=GEN1 EPSILON = 0.000001                     [convergence criterion, 0.000001 is smallest
                                                                             allowed]
*
=**************************** END USER INPUT ***************************
*
=SET NOCC                                                  [turn off carriage control]
=SET NODOECHO                                               [turn off do-loop echoing]
=SET NOWARN                                        [turn off messages in output file]
*
=GENR ONEVEC = 1.0                                                   [vector of ones]
=MATRIX WY = WMAT * DEPVAR                         [spatially lagged dependent variable]
=MATRIX WYXMAT = WY | ONEVEC | XMAT                                     [concatenate]
*
=OLS DEPVAR WYXMAT /COEF=BETAS NOCONSTANT                              [OLS regression]
=GEN1 NEST = $K + 1                          [number of parameters to be estim by NL]
*
=GEN1 SIGSQML = $SSE/$N                                  [ML estimate of sigma-squared]
=PRINT SIGSQML
=DIM SQML 1 1
=MATRIX SQML = SIGSQML                                           [move to matrix area]
=MATRIX SLIST = (SQML'|BETAS')'                                         [concatenate]
=GEN1 PI2 = 2*3.14159                                                       [scalar]
*
*
*
=NL 1 / NCOEF=NEST LOGDEN START=SLIST CONV=EPSILON PCOV
=EQ -0.5*LOG(PI2) - 0.5*LOG(SIGSQ)+LOG(1.0-RO*WROOT)-(0.5/SIGSQ) &
**********************************************************************
 *(DEPVAR-RO*WY-ALPHA*1.0-BT1*INCOME-BT2*HOVAL)**2                    [add variables and
                                  associated constants for as many variables in xmat above]
**********************************************************************
*
STOP
```

Table 5.5 - Splus: MLYX.SPL

```
# FUNCTION MLYX
#
# Purpose: estimation of the mixed regressive spatial autoregressive model
#
# Inputs:  xmat    matrix of explanatory variables (NO column of ones)
#          depvar  the dependent variable
#          wmat    the spatial weights matrix
#          wroot   the eigenvalues of wmat
#          epsilon the convergence criterion
#
# Returns: output from ms.summary plus output to the screen.
#          Use sink("<filename>") to capture all output to <filename>.
#
mlyx_function(xmat,depvar,wmat,wroot,epsilon) {
#
nobs_nrow(xmat)
###For extra variables, add 'var3','var4',etc. below:
dimnames(xmat)_list(NULL,c('var1','var2'))
wy_wmat%*%depvar
dimnames(wy)_list(NULL,c('ylag'))
wyxvar_cbind(wy,xmat)
#
reg1_lsfit(wyxvar,depvar,intercept=T)   #OLS regression
cat("--- OLS OUTPUT ---\n")
ls.print(reg1)
betas_cbind(reg1$coef)                 #coefficients
resids_reg1$residuals                  #residuals fr reg output
epe_t(resids)%*%resids                 #[e'*e]
sigsqml_epe/nobs                       #ML estimate of error variance
slist_c(sigsqml,betas)                 #list of starting values
cat("\n--- Starting values:  sigsq, alpha, ro, bt1, bt2, ... ---\n")
print(slist)
#
wyframe_data.frame(cbind(1,wyxvar),wroot,depvar)
###For extra variables, add "bt3=slist[6], bt4=slist[7]" etc. below:
###Also, starting value for "ro" below may need adjustment:
parameters(wyframe)_list(sigsq=slist[1], alpha=slist[2], ro=0.25,
    bt1=slist[4], bt2=slist[5])
#
cat("--- MLYX OUTPUT ---\n")
#
###For extra variables, add "- bt3*var3 - bt4*var4" etc. below:
mlyx.fit_ms(~0.5*log(6.283185) + 0.5*log(sigsq)-log(1.0-ro*wroot)
  + (0.5/sigsq)* (depvar-ro*ylag - alpha*1.0 - bt1*var1-bt2*var2)^2,
  wyframe, control=list(tolerance=epsilon),trace=TRUE)
summary.ms(mlyx.fit)
}
```

CHAPTER 6
MAXIMUM LIKELIHOOD ESTIMATION OF MODELS WITH SPATIAL AUTOREGRESSIVE ERRORS

INTRODUCTION

In this chapter, routines are presented to estimate a model with a spatial autoregressive error term by means of maximum likelihood techniques. These routines follow the same approach as outlined in Chapter 5 for the spatial lag model and only the differences will be discussed here.

METHODOLOGICAL BACKGROUND

Model Specification

A regression model with spatial autoregressive errors is but a special case of what econometricians refer to as non-spherical error terms. In contrast to what holds for the mixed regressive spatial autoregressive model outlined in the previous chapter, ordinary least squares (OLS) is not a biased estimator for this specification. However, it is an inefficient estimator, since it ignores the special variance structure for the errors. In other words, the variance of the OLS estimates for this model is larger than that obtained by other estimators. More seriously, however, if the standard OLS variance estimates (such as those provided by most econometric software) are used to compute measures of fit (such as R^2) and to carry out inference (such as t-tests and F-tests), the results will be biased. The latter is potentially the most serious problem encountered when ignoring spatial autocorrelation in regression analysis.

Formally, the regression model with spatial autoregressive errors is:

$$y = X\beta + \varepsilon$$

and

$$\varepsilon = \lambda W\varepsilon + \mu$$

where y is a N by 1 vector of observations on the dependent variable, X is a N by K matrix of observations on the explanatory variables, ε is a N by 1 vector of error terms that follow a spatial autoregressive process, with λ as coefficient, β is a K by 1 vector of regression coefficients and μ is a N by 1 error vector with classical properties (homoskedastic and uncorrelated). The spatial autoregressive structure for the errors yields a covariance matrix of the following form:

$$E[\varepsilon\varepsilon'] = E[(I-\lambda W)^{-1}\mu\mu'(I-\lambda W')^{-1}]$$

or

$$E[\varepsilon\varepsilon'] = \sigma^2[(I-\lambda W)'(I-\lambda W)]^{-1}$$

where σ^2 is the variance of the error μ. In the usual notation, the error variance Ω is thus expressed in function of two parameters, the variance σ^2 and the autoregressive coefficient λ.

A standard result for linear regression with non-spherical errors is that the so-called Generalized Least Squares (GLS) is most efficient. However, GLS assumes that the error variance matrix Ω is known, whereas in practice this matrix has to be estimated. As long as a constistent estimate can be found for the parameters of the error variance, so-called Estimated (EGLS) or Feasible (FGLS) generalized least squares still yield consistent estimates for the regression coefficients. Moreover, if certain conditions are satisfied, EGLS or FGLS estimates may be asymptotically most efficient as well. However, most of the simple two-step estimation techniques suggested for the analysis of serial error dependence in the time domain do not have immediate counterparts for spatial dependence (for technical details, see Anselin, 1988, Chapter 8).

Likelihood and Concentrated Likelihood Function

Following the general approach for a model with a non-spherical error term, the likelihood function takes on the form:

$$L = \ln |I-\lambda W| - N/2\ln(2\pi) - N/2 \ln(\sigma^2) - (y-X\beta)'(I-\lambda W)'(I-\lambda W)(y-X\beta)/2\sigma^2$$

As in the previous chapter, this is a function of the parameters λ, σ^2 and β. The maximum likelihood estimates for the parameters are found by solving the usual first order conditions. As before, the estimates for β and σ^2 are conditional upon the value of the autoregressive coefficient λ:

$$\beta = [(X-\lambda WX)'(X-\lambda WX)]^{-1}(X-\lambda WX)'(y-\lambda Wy)$$

$$\sigma^2 = (y-\lambda Wy-X\beta+\lambda WX\beta)'(y-\lambda Wy-X\beta+\lambda WX\beta)/N$$

These are the usual expressions for EGLS estimates. After substituting these formulations back into the likelihood, the following concentrated log likelihood function is obtained:

$$L_C = - (N/2) \ln [(\varepsilon'\varepsilon)/N] + \Sigma_i \ln (1-\lambda\omega_i)$$

with

$$\varepsilon'\varepsilon = (y-\lambda Wy)'(y-\lambda Wy)-(y-\lambda Wy)'(X-\lambda WX)[(X-\lambda WX)'(X-\lambda WX)]^{-1}(X-\lambda WX)'(y-\lambda Wy)$$

As in the previous chapter, the concentrated log likelihood is solely a function of the autoregressive parameter and thus lends itself well to the application of simple search procedures.

Asymptotic Variance Matrix

The variance for the ML estimates of the parameters in a model with spatial autoregressive errors is also obtained from the second order conditions for maximizing the log likelihood. This variance matrix has asymptotic validity, but may not be very reliable as an indicator of the precision of estimates in finite samples. An expression for the needed information matrix can be derived from the second partial derivatives of the log likelihood function. As a result, the asymptotic variance matrix can be calculated by substituting the values for the ML estimates of the coefficients into the elements of the information matrix and taking the inverse.

In the spatial error model, the information matrix is block diagonal between the regression coefficients and the parameters of the error term (Breusch, 1980). The resulting variance for the regression coefficients has the familiar form:

$$Var(\beta)=\sigma^2[(X-\lambda WX)'(X-\lambda WX)]^{-1}$$

with σ^2 replaced by the estimate for the residual variance.

The 2 by 2 block of the information matrix that pertains to the σ^2 and λ parameters turns out to be:

$$\begin{matrix} N/2\sigma^4 & trW_B/\sigma^2 \\ trW_B/\sigma^2 & tr\,(W_B)^2 + tr\,(W_B'W_B) \end{matrix}$$

where $W_B = W(I-\lambda W)^{-1}$ to simplify notation, and tr stands for the trace of a matrix. As in the spatial lag case, the traces of W_B and $(W_B)^2$ simplify to $\Sigma_i\ \omega_i/(1-\lambda\omega_i)$ and $\Sigma_i\ [\omega_i/(1-\lambda\omega_i)]^2$ respectively (see Anselin, 1988, for technical details).

The asymptotic variance and standard error for the ML estimates of σ^2 and λ are the diagonal elements of the inverse of the corresponding block in the information matrix.

Estimation Strategies

As in the spatial lag model, there are two common strategies to obtain ML estimates for the spatial autoregressive model: a search strategy and the explicit nonlinear optimization of the log likelihood function. The search strategy works in the same manner as outlined in the previous chapter and will not be further considered here.

Full Nonlinear Optimization of the Log Likelihood Function

The nonlinear optimization of the log likelihood function in the spatial error model is very similar to that in the spatial lag model. Again, the simplifying results of Ord (1975) are used to create the eigenvalue of the spatial weights matrix as an additional pseudo variable. Consequently, the log likelihood can be expressed as a sum of terms obtained for each observation. The relevant expression is:

$$L = \Sigma_i \ln(1-\lambda\omega_i) - 0.5\ln(2\pi) - 0.5\ln(\sigma^2) - 0.5(y_i-\lambda Wy_i-x_i\beta+\lambda Wx_i\beta)^2/\sigma^2$$

in the same notation as in the previous chapter, and with Wx_i as a row vector with the observations on the spatially lagged explanatory variables for i.

This expression conforms to the usual format, provided that we create a pseudo-variable that contains the eigenvalues.

The partial derivatives of the log likelihood with respect to the parameters are (for detailed derivation, see Anselin, 1980):

$$\partial L/\partial\lambda = \Sigma_i \{ (Wy_i-Wx_i\beta).\varepsilon_i/\sigma^2 - \omega_i/(1-\lambda\omega_i) \}$$

$$\partial L/\partial\beta_k = \Sigma_i \{ (x_{ki}-\lambda Wx_{ki}).\varepsilon_i/\sigma^2 \}$$

$$\partial L/\partial\sigma^2 = 0.5\ \Sigma_i \{ -1/\sigma^2 + \varepsilon_i^2/\sigma^4 \}$$

where ε_i is the residual for observation i, $y_i-\lambda Wy_i-x_i\beta+\lambda Wx_i\beta$, x_{ki} and Wx_{ki} are the k-th term of row vectors x_i and Wx_i respectively, and β_k is the k-th regression coefficient.

The expression for the log likelihood terms and the first partial derivatives are the only information needed to implement the optimization with the five software packages considered in this report.

Starting Values

The starting values for the estimates of the coefficients in the spatial error model must be chosen with care. A simple approach is to take the OLS estimates for the regression coefficients and zero for the spatial autoregressive coefficient. However, in a number of instances this may turn out to be unsatisfactory and lead to explosive iterations of the numerical optimization routines. An alternative that sometimes fixes this problem is to use an OLS estimate in a regression of the residuals on the spatially lagged residuals (note that this is NOT a consistent estimate). A slightly more complex procedure is to estimate the autoregressive coefficient l in the so-called spatial Durbin form of the model:

$$y = \lambda Wy + X\beta - \lambda WX\beta + \varepsilon$$

This may be done by means of the approaches to estimate a spatial lag model.

OVERVIEW OF THE ROUTINES

User Input Section

The user input for the nonlinear optimization in the spatial error model is identical to the one described in the previous chapter, except for the inclusion of a set of spatially lagged explanatory variables. As before, the way in which this is accomplished varies slightly between the software packages.

Gauss

As for the spatial lag model, the Gauss procedure *maxlik* takes as input a N by K matrix **zz** with observations on all the variables in the problem, a matching K by 1 vector of variable names or variable index numbers **vars**, a pointer to a procedure for the computation of the log-likelihood and a vector of starting values for all parameters, **bstart**. The procedure for the log-likelihood is called *lmler* (with pointer **&lmler**). The matrix **zz** should be structured as follows:

- First column: dependent variable, y
- Second column: spatially lagged dependent variable, Wy
- Next set of columns: explanatory variables (including a vector of ones as the first)
- Following set of columns: spatially lagged explanatory variables, WX (includes a vector of ones as the first column)
- Last column: eigenvalues of the spatial weights matrix (rw)

Note that the inclusion of the eigenvalues as a pseudo variable allows the expression of the log likelihood in the standard format used by *maxlik*.

The easiest way to deal with the vector **vars** is to initialize it as a vector of sequential numbers, up to the dimension of the problem, as in:

vars=*seqa*(1,1,*cols*(**zz**));

The *maxlik* procedure returns a vector of estimates, **b**, the value of the likelihood function, **lik**, the value of the gradient at the final iteration, **gra**, and an approximation to the asymptotic variance matrix, **hh**. The calling statement is:

{ **b,lik,gra,hh** }=*maxlik*(**zz,vars,&lmler,bstart**);

Note that in order for this to work, the procedure *maxlik* must be contained in the current library or must be included explicitly by means of a *use maxlik* statement as the first line of the program.

The same three global variables can be manipulated as in the previous chapter. The pointer to the procedure in which the partial derivatives are computed should be set to **&glmler** (**_gdproc=&glmlyx**;).

Limdep

The Limdep user input is essentially the same as for the spatial lag case, except for the inclusion of spatially lagged explanatory variables. In the user input section of Table 6.2, this is shown by the *create* command to initialize the spatially lagged variables **wx1** and **wx2**. The spatially lagged explanatory variables are created by copying the namelist of explanatory variables to the matrix work area followed by a simple matrix multiplication. Once the variables are formed, they are returned to the data work area by the *dta* command so that they can be used in the *regress* command.

The OLS estimates from the spatial Durbin form of the model are used as starting values. As in the routine for the spatial lag model, the *labels* command needs to reflect the proper number of explanatory variables.

Rats

In Rats (Table 6.3), the spatially lagged explanatory variables are created in the user input section as well. The other minor change is the use of the *rhs* formula. This is implemented to decrease the length of the log likelihood expression. As for the spatial lag implementation, the user must enter the proper number of coefficients in the final section of the *frml* command.

Shazam

The only addition to the user input section compared to the spatial lag model is a constant, **ncol**, which facilitates record keeping of the explanatory variables. The spatially lagged explanatory variables are constructed by a single matrix multiplication. Subsequently, the columns of the resulting **wx** matrix are split into separate vectors using a do-loop. As in the Limdep section, only a subset of the OLS parameters are needed in ML estimation. These are selected by using a *sample* command, since the unneeded coefficients have been placed at the end of the vector. The user must specify the appropriate βX and λWXβ terms of the **eq** command line.

Splus

The *mler* Splus function takes the same arguments as in the spatial lag case, i.e. *mler*(**xmat,depvar,wmat,wroot,epsilon**). The function proceeds as described for the *mlyx* function in the previous chapter until the specification of the list of starting values. A subset of the vector is selected using brackets after the variable name. The data frame is set up exactly as in the previous chapter.

If you are not using exactly two explanatory variables, then you must modify lines 18, 23, 43, 50, and 51 of Table 6.5 so that they reference the correct number of variables. See the comments in the source code for specific instructions. Also, the starting value for **lambda** (line 42) may need to be changed in order to achieve convergence. For the Columbus dataset, the OLS value for **lambda** fails as a starting value. A starting value of -0.5 was found to lead to convergence.

REFERENCES

Anselin, Luc, 1980. *Estimation Methods for Spatial Autoregressive Structures.* Regional Science Dissertation and Monograph Series, Cornell University, Ithaca, NY.

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

Breusch, T., 1980. Useful invariance results for generalized regression models. *Journal of Econometrics* 13, 327-40.

Ord, Keith, 1975. Estimation methods for models of spatial interaction, *Journal of the American Statistical Association*, 70, 120-26.

THE MLER.* PROGRAMS: SPATIAL ERROR MODEL

Table 6.1 - Gauss: MLER.GAS

```
/* LMLYX log likelihood for spatial autoregressive model
**       for input to MAXLIK
**
** FORMAT: llik=lmler(b,zz);
**
** INPUT:  b   -- vector with parameter values
**                b[1]   -- rho, spatial autoregressive parameter
**                b[1:next to last] -- regression coefficients
**                                     includes constant
**                b[last] -- sigma squared, error variance
**         zz  -- row vector with single observation on variables
**                first element  -- y, dependent variable
**                second element -- Wy, spatial lagged dependent var
**                third to next to last -- X, explanatory variables,
**                                         including constant term
**                                         followed by WX
**                last element   -- eigenvalue of spatial weights matrix
**                                  (used in Jacobian term)
**
** OUTPUT: llik -- log likelihood evaluated for one observation
**
** REMARK: log likelihood includes ln(2pi)
**         input and output structured to conform to call from proc maxlik
*/

proc(1)=lmler(b,zz);
local k,k1,kk,e,llik,rho,bb,sig2,y,Wy,x,xx,Wx,rw;
/* dimensions of parameter vectors */
k=rows(b)-1;                  /* k does not included sigma squared */
k1=rows(b);                   /* k1 includes all elements */
/* initialize parameters in standard notation */
rho=b[1];                     /* spatial autoregressive coefficient */
bb=b[2:k];                    /* regression coefficients */
sig2=b[k1];                   /* error variance */
/* initialize variables */
y=zz[1,1];                    /* dependent variable */
Wy=zz[1,2];                   /* spatially lagged dependent variable */
xx=zz[1,3:cols(zz)-1];        /* explanatory variables, including lags */
kk=cols(xx)./2;               /* columns for X and WX matrix */
x=xx[1,1:kk];                 /* x matrix */
Wx=xx[1,kk+1:cols(xx)];       /* WX matrix */
rw=zz[1,cols(zz)];            /* eigenvalue of spatial weights matrix */
e=y - rho*Wy - x*bb - rho.*(Wx*bb);          /* residual */
/* log likelihood for single observation */
llik=-0.5.*ln(2.0.*pi) -0.5.*ln(sig2)-0.5.*(e'e)./sig2 + ln(1.0-rho*rw);
retp(llik);
endp;

/* glmler gradient of log likelihood */
/* glik must be row vector */

/* GLMLER gradient of loglikelihood in spatial autoregressive model
**        for input into MAXLIK with global option __gdproc=&glmler;
**
** FORMAT: glik=glmler(b,zz);
**
** INPUT:  b   -- vector with parameter values
**                b[1]   -- rho, spatial autoregressive parameter
**                b[1:next to last] -- regression coefficients
**                                     includes constant
**                b[last] -- sigma squared, error variance
**         zz  -- row vector with single observation on variables
**                first element  -- y, dependent variable
**                second element -- Wy, spatial lagged dependent var
**                third to next to last -- X, explanatory variables,
**                                         including constant term
**                                         followed by WX
**                last element   -- eigenvalue of spatial weights matrix
**                                  (used in Jacobian term)
**
** OUTPUT: glik -- row vector with partial derivatives of log likelihood
**                 evaluated for a single observation
**                 elements in same order as b, but organized as row
**
** REMARK: input and output structured to conform to call from proc maxlik
*/
```

```


proc(1)=glmler(b,zz);
local k,k1,kk,e,rho,bb,sig2,y,Wy,x,xx,Wx,rw,
      dldrho,dldb,dldsig,glik;
/* dimensions of parameter vectors */
k=rows(b)-1;                  /* k does not included sigma squared */
k1=rows(b);                   /* k1 includes all elements */
/* initialize parameters in standard notation */
rho=b[1];                     /* spatial autoregressive coefficient */
bb=b[2:k];                    /* regression coefficients */
sig2=b[k1];                   /* error variance */
/* initialize variables */
y=zz[1,1];                    /* dependent variable */
Wy=zz[1,2];                   /* spatially lagged dependent variable */
xx=zz[1,3:cols(zz)-1];        /* explanatory variables, including lags */
kk=cols(xx)./2;               /* columns for X and WX matrix */
x=xx[1,1:kk];                 /* x matrix */
Wx=xx[1,kk+1:cols(xx)];       /* WX matrix */
rw=zz[1,cols(zz)];            /* eigenvalue of spatial weights matrix */
e=y - rho*Wy - x*bb - rho.*(Wx*bb);         /* residual */
/* partial derivatives of log likelihood for single observation */
dldrho=(Wy+Wx*bb)*e./sig2 - rw./(1.0-rho*rw);
                                        /* partial derivative for rho */
dldb=((x+rho.*WX).*e)./sig2;
                                        /* partial derivative for b */
dldsig=0.5 .* (e^2/sig2 - 1.0)./sig2;  /* partial derivative for sig2 */
glik=dldrho~dldb~dldsig;                /* partial derivatives as row vector */
retp(glik);
endp;
```

Table 6.2 - Limdep: MLER.LIM

```
?
?????????????????????????????USER INPUT SECTION??????????????????????
[CHANGE ONLY UNDERLINED ITEMS]
?
LOAD ; FILE = COLWS.LIM $                          [enter output file from weights preprocessing
                                                                                     routine]
READ ; NOBS = 49                                                   [number of observations]
     ; NVAR = 1                                                       [no change this line]
     ; NAMES = WROOT                                                  [no change this line]
     ; FORMAT = BINARY                                                [no change this line]
     ; FILE = COLWS.EIG $                                       [file name for eigenvalues]
READ ; NOBS = 49                                                    [number of observations
     ; NVAR = 3                                           [number of variables in data file]
     ; NAMES = 1                               [variable names/or # lines names occupy in file]
     ; FILE = COLREG.ASC $                                                  [data file name]
NAMELIST ; DEPVAR = CRIME                                        [dependent variable name]
         ; XMAT = INCOME,HOVAL $                    [explanatory variable names, do not add
                                                constant term or lagged dependent variable]
CREATE ; WX1 = 0 ; WX2 = 0 $                          [add as many wx* terms as variables in
                                                                                      xmat]
CREATE ; YVAR = CRIME $                             [dependent variable name for minimize]
CALC; NOLIST; EPSILON = 0.0000001 $                 [convergence criterion for parameters]
????????????????????????????????END USER INPUT????????????????????????
?
NAMELIST; XMAT1 = ONE,XMAT $                                [add column of ones to explanatory
                                                                                  variables]
MATRIX ; NOLIST ; XPLANV = COPY(XMAT) $                                        [copy explan.
                                                             variables to matrix work area]
MATRIX ; NOLIST ; WX = WMAT | XPLANV $                                    [spatially lagged
                                                                        explan. variables]
NAMELIST ; WXS = WX* $
CREATE ; WXS = DTA(WX)$                                      [move spatially lagged explan.
                                                                 variables to data area]
MATRIX ; NOLIST; DEPVARV = COPY(DEPVAR) $                       [move dependent variable to
                                                                          matrix work area]
MATRIX ; NOLIST; WY = WMAT | DEPVARV $                          [spatially lagged dependent
                                                                                 variable]
CREATE ; NOLIST; WY1 = 0 $
NAMELIST; WY2 = WY1 $
CREATE;  WY2 = DTA(WY) $                                          [move Wy to data work area]
NAMELIST ; WYXMAT = WY2,XMAT1,WXS $                             [add Wy and WX to explanatory
                                                                            variable list]
?
REGRESS; LHS = DEPVAR; RHS = WYXMAT $                                [OLS estimation of model]
CALC;  SIGSQML = SUMSQDEV / NREG $                      [ML estimate of the error variance]
CALC ; NOLIST; K2 = KREG/2.0 + 1 $                                     [number of parameters,
                                                              with common factor constraint]
MATRIX ; NOLIST ; BETAS = PART(B,1,K2) $                                          [extract
                                                                      coefficients needed]
?
MATRIX ; NOLIST; SLIST = SIGSQML*IDEN(1,1)                     [sigsqml & betas are starting
                                                                      values for minimize]
       ; SLIST = SLIST/BETAS $
CREATE ; ZERO = 0 $                                            [create variable of zeroes]
NAMELIST ; Z = ZERO,WY2,ZERO,XMAT $                    [set up for dot product in minimize]
NAMELIST ; Z2 = ZERO,ZERO,ZERO,WXS $                            [2nd set up for dot product]
CALC; NOLIST; PIE2 = 2*PI $
?
MINIMIZE ; FCN = 0.5*LOG(PIE2)                               [set up function to be minimized]
                 + 0.5*LOG(SIGSQ)
                 - LOG(ONE-LAMBDA*WROOT)                           [1 - rho*eigenvalues of W]
                 + (0.5/SIGSQ)
                 * (YVAR + (LAMBDA-1.0)*ALPHA*ONE
                 - DOT[Z] + LAMBDA*DOT[Z2])^2
         ; START = SLIST                               [starting values for parameters]
         ; TLB = EPSILON                            [convergence criterion for parameters]
?????????????????????????????USER INPUT SECTION??????????????????????
[CHANGE ONLY UNDERLINED ITEMS]
         ; LABELS = SIGSQ,LAMBDA,ALPHA,BT1,BT2 $                      [have as many bt's as
                                      variables entered in xmat (don't count constant term)]
????????????????????????????????END USER INPUT????????????????????????
STOP
```

Table 6.3 - Rats: MLER.RAT

```
OUTPUT NOECHO                                          [no command echo in output file]
*
****************************USER INPUT SECTION************************
*(CHANGE ONLY UNDERLINED ITEMS)
*
ALLOCATE 0 49                                           [enter number of observations]
OPEN DATA COLREG.RAT                                          [enter data file name]
DATA(FORMAT=RATS) / CRIME INCOME HOVAL                  [enter file format, all variable
                                                                     names in file]
DECLARE RECTANGULAR WMAT(49,49)                   [enter dimensions (= # observations)]
OPEN DATA COLWS.BIN                                         [enter weights matrix file]
READ(FORMAT=BINARY) WMAT                                                  [file format]
OPEN DATA COLWS.EIG                                   [file containing eigenvalues of W]
DATA(FORMAT=BINARY) / WROOT                                               [file format]
SET DEPVAR = CRIME(T)                                      [name of dependent variable]
DECLARE SERIES WY WX1 WX2                      [do not change wy, place as many wx's as
                            explanatory variables (explan. vars do not count the constant or
                                                              spatially lagged variables]
MATRIX WY = WMAT * DEPVAR                                           [no change this line]
MATRIX WX1 = WMAT * INCOME                            [create spatially lagged explanatory
                                                                               variable]
MATRIX WX2 = WMAT * HOVAL                             [create spatially lagged explanatory
                                 variable, add another line if another explan. variable]
EQUATION EQ1 DEPVAR                                                 [no change this line]
#WY CONSTANT INCOME HOVAL WX1 WX2                [enter names of explantory variables, do
                               not change the name or placement of WY or Constant, or WX*
                                                          additonal WX terms may be needed]
*
******************************END USER INPUT**************************
*
OUTPUT NOTSTAT                                                     [no t-stat printing]
LINREG(EQUATION=EQ1) DEPVAR                                           [OLS regression]
MAKE(LASTREG) WYXMAT                                    [save LHS variables in a matrix]
EVAL PI2 = 2*PI                                                               [scalar]
*
**********************************************************************
NONLIN SIGSQ ALPHA LAMBDA BT1 BT2                  [enter # of bt terms as explanatory
                         variables, do not count the spatially lagged variable, or the constant
                                                                                  term]
**********************************************************************
FRML RHS = LAMBDA*WYXMAT(T,1) - (LAMBDA-1.0)*ALPHA*1.0
                                                                           [subformula]
FRML MLER = -0.5*LOG(PI2) - $
    0.5*LOG(SIGSQ) + $
    LOG(1.0-LAMBDA*WROOT(T)) - $
    (0.5/SIGSQ) *  $
**********************************************************************
    (DEPVAR(T)-RHS(T)-BT1*WYXMAT(T,3)-BT2*WYXMAT(T,4) + $                  [enter number
                                            of bt*wyxmat terms as explanatory variables]
    LAMBDA*BT1*WYXMAT(T,5) + LAMBDA*BT2*WYXMAT(T,6))**2
                           [enter number of lambda*bt*wyxmat terms as explanatory variables]
**********************************************************************
*
EVAL SIGSQ = RSS/NOBS ; EVAL LAMBDA = BETA(1) ; EVAL ALPHA = BETA(2)
**********************************************************************
EVAL BT1 = BETA(3) ; EVAL BT2 = BETA(4)                            [see nonlin command]
**********************************************************************
*
MAXIMIZE(VCV,TRACE,ITERATIONS=50) MLER /
*
END
```

Table 6.4 - Shazam: MLER.SHA

```
*
=*
=*[notes are contained in brackets]
=*[* denotes comments, = means no echo of command line in output file, and ? suppresses
output from these lines]
=*
=************************ USER INPUT SECTION *************************
[CHANGE ONLY UNDERSCORED ITEMS]
=*
=?FILE 11 COLREG.ASC                                              [enter data file name]
=?FILE 12. COLWS.BIN                       [weights matrix file name, . after the 12 if
                                                                    in binary format]
=?FILE 13. COLWS.EIG                                      [file containing eigenvalues]
=?READ(11) CRIME INCOME HOVAL /CLOSE                      [enter all variable names for
                                                                     data in above file]
=?READ(12) WMAT / BINARY ROWS=49 COLS=49 CLOSE                  [binary if necessary,
                                                                  # rows & # columns]
=?READ(13) WROOT / BINARY ROWS=49 COLS=1 BYVAR CLOSE                [number of observ]
=?COPY INCOME HOVAL XMAT                  [names of explanatory variables, do not add a
                                           constant term or lagged dependent variables]
=GEN1 NCOL = 2                              [enter number of variables placed in xmat]
=?COPY CRIME DEPVAR                                         [name of dependent variable]
=GEN1 EPSILON = 0.000001                 [convergence criterion, 0.000001 is smallest
                                                                              allowed]
*
=***************************** END USER INPUT ***************************
*
=SET NOCC                                                   [turn off carriage control]
=SET NODOECHO                                                [turn off do-loop echoing]
=SET NOWARN                                          [turn off messages in output file]
*
=GENR ONEVEC = 1.0                                                     [vector of ones]
=MATRIX WY = WMAT * DEPVAR                         [spatially lagged dependent variable]
=MATRIX WX = WMAT * XMAT                                   [spatially lagged explanatory
                                                                            variables]
=?DO # = 1,NCOL
=    COPY WX WX# /FCOL=#;#                                   [make each column of WX into a
                                                                     separate vector]
=ENDO
=MATRIX WYXMAT = WY | ONEVEC | WX                                          [concatenate]
*
=OLS DEPVAR WYXMAT /COEF=BETAS NOCONSTANT                               [OLS regression]
=GEN1 NEST1 = $K - NCOL                        [number of betas going in starting vector]
=GEN1 NOBS = $N                                                [number of observations]
=?SAMPLE 1 NEST1                                             [take a subset of the betas]
=COPY BETAS BETA
=SAMPLE 1 NOBS                                                          [reset sample]
*
=GEN1 SIGSQML = $SSE/$N                                   [ML estimate of sigma-squared]
=PRINT SIGSQML
=DIM SQML 1 1
=MATRIX SQML = SIGSQML                                               [move to matrix area]
=MATRIX SLIST = (SQML'|BETA')'                                             [concatenate]
=GEN1 PI2 = 2*3.14159                                                           [scalar]
=GEN1 NEST = NEST1 + 1                          [number of estimated parameters for NL]
*
*
=NL 1 / NCOEF=NEST LOGDEN START=SLIST CONV=EPSILON PCOV
=EQ -0.5*LOG(PI2)-0.5*LOG(SIGSQ)+LOG(1.0-LAMBDA*WROOT)-(0.5/SIGSQ) &
*(DEPVAR-LAMBDA*WY-(LAMBDA-1)*ALPHA*1.0 &
**********************************************************************
 - BT1*INCOME-BT2*HOVAL &                                             [add variables and
                              associated constants for as many variables in xmat above]
 + LAMBDA*BT1*WX1 + LAMBDA*BT2*WX2)**2                              [same as previous line]
***********************************************************************
*
STOP
```

Table 6.5 - Splus: MLER.SPL

```
# FUNCTION MLER
#
# Purpose: Estimate the spatial error model
# Inputs:  xmat    matrix of explanatory variables (NO column of ones)
#          depvar  the dependent variable
#          wmat    the spatial weights matrix
#          wroot   the eigenvalues of wmat
#          epsilon the convergence criterion
#
# Returns: output from ms.summary plus output to the screen.
#          Use sink("<filename>") to capture all output to <filename>.
#
mler_function(xmat,depvar,wmat,wroot,epsilon) {
#
nobs_nrow(xmat)
###For extra variables, add 'var3','var4',etc. below:
dimnames(xmat)_list(NULL,c('var1','var2'))
wy_wmat%*%depvar                          #construct lagged depvar
dimnames(wy)_list(NULL,c('ylag'))         #name the column of wy
wx_wmat%*%xmat                            #construct WX
###For extra variables, add 'wvar3','wvar4',etc. below:
dimnames(wx)_list(NULL,c('wvar1','wvar2'))  #name columns of WX
wyxvar_cbind(wy,xmat,wx)                  #matrix of Wy and explanatory vars.
k_ncol(wyxvar)+1
#
reg1_lsfit(wyxvar,depvar,intercept=T)     #OLS regression
cat("--- OLS OUTPUT ---\n")
betas_cbind(reg1$coef)                    #coefficients
ls.print(reg1)
resids_reg1$residuals                     #residuals fr reg output
epe_t(resids)%*%resids                    #[e'*e]
sigsqml_epe/nobs                          #ML estimate f error variance
kparam_k/2+1
slist_c(sigsqml,betas[1:kparam])          #list of starting values
cat("\n--- Starting values:  sigsq, alpha, lambda, bt1, bt2, ... ---\n")
print(slist)
#
wyframe_data.frame(wyxvar,wroot,depvar)
###For extra variables, add "bt3=slist[6], bt4=slist[7]" etc. below:
###Also, starting value for "lambda" below may need adjustment:
parameters(wyframe)_list(sigsq=slist[1], alpha=slist[2], lambda=-0.5,
     bt1=slist[4], bt2=slist[5])
#
cat("--- MLER OUTPUT ---\n")
#
###For extra variables, add "- bt3*var3 - bt4*var4" etc. and
###  "+ lambda*bt3*wvar3 + lambda*bt4*wvar4" etc. below:
mler.fit_ms(~0.5*log(6.283185)-log(1.0-lambda*wroot)+0.5*log(sigsq)
  +(0.5/sigsq)*(depvar-lambda*ylag-bt1*var1-bt2*var2
  +(lambda-1.0)*alpha*1.0 + lambda*bt1*wvar1 + lambda*bt2*wvar2)^2,
wyframe,control=list(tolerance=epsilon),trace=TRUE)
#
summary.ms(mler.fit)
}
```

APPENDICES

APPENDIX 1: CONTENTS OF THE INCLUDED DISKETTE

The diskette included with this report contains three directories: GIS, ROUTINES, and COLUMBUS. The contents of each directory are described below.

GIS: The GIS W-matrix extraction programs

The GIS directory contains both executable files (where appropriate) and source code for all of the programs described in Chapter 2. Files with a .EXE extension are DOS executable; files with a .C extension are C source code in ASCII format; and files with a .AML extension are Arc/Info AML source code:

```
RASDIST  EXE
RASCONT  EXE
SP2FULL  EXE
INF2SMF  C
RASCONT  C
RASDIST  C
SP2FULL  C
CONWEIGH AML
DISWEIGH AML
```

ROUTINES: The statistical routines

The ROUTINES directory contains the statistical routines described in Chapters 3-6:

```
WRSTD.*
EIGENW.*     [for Gauss and Splus only]
REGDIAG.*
MLYX.*
MLER.*
```

The following filename extensions apply to the above files:

```
.GAS - Gauss
.LIM - Limdep
.RAT - Rats
.SHA - Shazam
.SPL - Splus
```

COLUMBUS: The Columbus, Ohio dataset

The routines included with this report are set up to run with the Columbus, Ohio dataset. This dataset pertains to 49 planning neighborhoods in 1984, and contains three variables:

- CRIME - Number of residential burglaries and automobile thefts per 1,000 households.
- INCOME - Per-capita income in units of $1,000.
- HOVAL - Median housing value in units of $1,000.

The example regression model assumed in the routines is:

$$\text{CRIME} = \beta_0 + \beta_1(\text{INCOME}) + \beta_2(\text{HOVAL}) + \varepsilon$$

Files in the COLUMBUS directory on the diskette

The following is a description of the Columbus dataset files on the included diskette, in the directory COLUMBUS:

FILENAME	**DESCRIPTION**
COLREG.ASC	A three-column ASCII file containing the Columbus variables in the order CRIME, INCOME, HOVAL. The first line of this file contains the names of the variables.
COLWMAT.ASC	A full spatial weight matrix based on binary contiguity for the Columbus neighborhoods. This is an ASCII file containing 49x49 matrix elements one row at a time. A single row of the matrix occupies more than one line of the file. This matrix is NOT row-standardized.
COLWMAT.LIM	The same file as above, but this one contains a header which is specific to LIMDEP.
POLYID.IMG/.DOC	An Idrisi-format image of the Columbus neighborhoods. Each neighborhood has an integer polygon identifier (poly-ID) from 1 to 49. These poly-IDs can be associated with the regression data by numbering the lines of the file COLREG.ASC from 1 to 49.

Results of analyses on the Columbus dataset

The following analytical results should match the results you get from running the statistical routines on the Columbus dataset, using a row-standardized binary contiguity weight matrix (i.e. a row-standardized version of the COLWMAT.ASC matrix described above). These results were computed by the *SpaceStat* program (Anselin, 1992).

REGDIAG output

```
ORDINARY LEAST SQUARES ESTIMATION
DEPENDENT VARIABLE     CRIME      OBS  49      VARS   3      DF  46
R2           0.5524    R2-adj     0.5329
LIK        -187.377    AIC        380.754   SC          386.430
RSS         6014.86    F-test     28.3856   Prob  9.34081e-09
SIG-SQ      130.758   (     11.4349 ) SIG-SQ(ML)      122.752   (      11.0794 )

VARIABLE    COEFF       S.D.      t-value      Prob
CONSTANT     68.6189     4.73547   14.490397    0.000000
  INCOME     -1.5973     0.33413   -4.780490    0.000018
 HOUSING   -0.273931    0.103198   -2.654414    0.010874

REGRESSION DIAGNOSTICS FOR SPATIAL DEPENDENCE
FOR WEIGHTS MATRIX  COLWMAT (row-standardized weights)
TEST                             MI/DF        VALUE        PROB
Moran's I (error)               0.229641     2.883861     0.003928
Lagrange Multiplier (error)            1     5.405834     0.020070
Lagrange Multiplier (lag)              1     8.380236     0.003793
```

MLYX output

```
SPATIAL LAG MODEL - MAXIMUM LIKELIHOOD ESTIMATION
DATA SET COLUMBUS                 SPATIAL WEIGHTS MATRIX  COLWMAT
DEPENDENT VARIABLE     CRIME      OBS  49      VARS   4      DF  45
R2           0.6090    Sq. Corr.  0.6431
LIK        -182.909    AIC        373.818   SC          381.386
SIG-SQ      98.0100   (     9.90000 )

VARIABLE    COEFF       S.D.      z-value      Prob
 W_CRIME    0.410186    0.119193    3.441372    0.000579
CONSTANT     46.2543     7.29619    6.339510    0.000000
  INCOME    -1.07405    0.307577   -3.491975    0.000479
 HOUSING   -0.261133   0.0898871   -2.905126    0.003671
```

MLER output

```
SPATIAL ERROR MODEL - MAXIMUM LIKELIHOOD ESTIMATION
DATA SET COLUMBUS                 SPATIAL WEIGHTS MATRIX  COLWMAT
DEPENDENT VARIABLE     CRIME      OBS  49      VARS   3      DF  46
```

```
R2            0.3388     Sq. Corr.    0.5344     R2(Buse)      0.4239
LIK        -183.565      AIC        373.130      SC          378.806
SIG-SQ       96.5903   (       9.82804 )

VARIABLE      COEFF         S.D.       z-value        Prob
CONSTANT      60.5982      5.36611    11.292754     0.000000
  INCOME    -0.955021     0.329915    -2.894746     0.003795
 HOUSING    -0.313957    0.0924998    -3.394137     0.000688
  LAMBDA     0.552929     0.135462     4.081811     0.000045
```

APPENDIX 2: OVERVIEW OF STEPS FOR USING THIS REPORT

Point of departure

These steps assume that you are starting with a dataset containing several variables which are referenced to a set of polygons, and that the polygons are stored in one of the following GIS packages:

- ARC/INFO
- IDRISI
- OSU Map-for-the-PC
- Any raster GIS which supports an uncompressed, row-scan-order file format (stored either as ASCII or as 1, 2, or 4 byte binary). This is probably the simplest and most common raster format available, and is supported by ERDAS and GRASS, for example.

STEP 1: Extract a spatial weight matrix from your GIS (Chapter 2)

a) To create a contiguity-based matrix, use the program CONWEIGHT.AML for ARC/INFO or the program RASCONT.C for a raster GIS package.

b) For a distance-based matrix, use the program DISWEIGHT.AML for ARC/INFO or the program RASDIST.C for a raster GIS package.

c) Steps a) and b) produce the matrix in sparse format. Use the program SP2FULL.C to create the full-format matrix which is required by the statistical routines.

STEP 2: Process the spatial weight matrix (Chapter 3)

a) Use the appropriate WRSTD.* routine to row-standardize the matrix. For Limdep, Rats, and Shazam, the WRSTD.* routine computes eigenvalues as well. For Gauss and Splus, use the EIGENW.* routine to compute eigenvalues.

STEP 3: Diagnose your regression model for spatial effects (Chapter 4)

a) Run the REGDIAG.* routine on your regression model.

b) Inspect the output from REGDIAG.*. If the Moran's I and Lagrange Multiplier (LM) tests are significant, go on to step 4. If the tests are not significant, your model does not contain significant spatial effects.

STEP 4: Run the appropriate alternative regression model (Chapters 5-6)

a) If the LM Lag Test (computed in step 3 above) returned a larger value than the LM Error Test, run the MLYX.* routine (Chapter 5).

b) If the LM Error Test (computed in step 3 above) returned a larger value than the LM Lag Test, run the MLER.* routine (Chapter 6).

c) If desired, test the sensitivity of your results to the structure of the spatial weight matrix by creating a different matrix in step 1 and using it in steps 2-4.

APPENDIX 3: SPARSE MATRIX FILE FORMATS

For each sparse matrix file format (binary and general), the first line of the file contains a single integer, representing the number of observations (N) which are referenced in the file. The remainder of each file format is described below.

Sparse Binary Format

After the first line (described above), this file contains a pair of lines for each polygon-ID in the following format:

```
<N>
<polygon-ID>  <number_of_neighbors>
<neighbor-ID>  <neighbor-ID>  <neighbor-ID> etc.
```

For example, if polygon #10 were adjacent to polygons #3, #6, #7, and #13, then the following two lines would appear in the sparse matrix:

```
10  4
3  6  7  13
```

For polygons with no neighbors, the file will contain a blank line after the line with <polygon-ID> <number_of_neighbors>.

Sparse General Format

After the first line (described above), this file contains one line for each non-zero element of the matrix. Each line has three items as follows:

```
<N>
<polygon-ID>  <neighbor-ID>  <weight>
```

For example, if polygon #6 were a neighbor to polygon #13 with a weight value of 56.78, then the corresponding line in the sparse matrix file would read:

```
6  13  56.78
```

For polygons with no neighbors, the values for <neighbor-ID> and <weight> will be zero.

APPENDIX 4: SOURCE CODE FOR CHAPTER 2 ROUTINES

RASCONT.C

```
/****************************************************************************
RASCONT - reads a raster polygon image and creates a binary  contiguity matrix
By Rusty Dodson, NCGIA Santa Barbara.
****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#define FULL 1000  /* Size of storage buffer.  Must be larger than number */
                   /* of polygons in the raster image. */

/* global variables */
int  *above, *center, *below,   /* integer arrays (rows of the raster) */
     *top, *mid, *bot,          /* pointers to above arrays */
     lastpoly, lastneib;        /* last adjacency written to buffer array */


typedef struct  /* this structure holds one poly/neighbor set */
{
    int   poly;         /* the target polygon */
    int   neib;         /* the target's neighbor */
} RECORD;

RECORD  *buffer;        /* buffer array of poly/neib records */
int     count = 0;      /* counter to keep track of buffer's fullness */

main(argc, argv) /* main program */
int argc;
char *argv[];
{
    int i;              /* counter */
    int rows, cols;     /* rows and columns in the image file */
    long num_obs;       /* number of polygons in raster file */
    int *tmp;           /* temp pointer to hold places for top, mid, bot */
    int row_cntr=1;     /* counter to keep track of which row array to fill */
    int *poly_ids;      /* array of unique polygon ID's */
    FILE *raster,       /* raster image file */
         *outfile,      /* temp file to deal with memory constraints */
         *scm_file;     /* final Sparse Contiguity Matrix file */
    long position;      /* stores position of file pointer */
    int  file_type;     /* bytes per cell (0 for ascii) */
    FILE *Idrisi(), *Osumap(), *Generic(); /* image-reading funcions */
    long nsquared, longi; /* added for DOS compatibility */

    if(argc != 2) { /* if invalid # of arguments, display usage */
        Usage();
    }

    switch(argv[1][0]) {  /* switch on the command-line argument */
        case 'i': case 'I':
            raster = Idrisi(&file_type, &rows, &cols);
            break;
        case 'o': case 'O':
            raster = Osumap(&file_type, &rows, &cols);
            break;
        case 'g': case 'G':
            raster = Generic(&file_type, &rows, &cols);
            break;
        default:
            Usage();
    } /* end switch */

    if( rows < 3 || cols < 3 ) {
        printf("Error:  The raster dimension must be at least 3 by 3.\n");
        exit(-1);
    }

    /* remember the position of the raster file pointer */
    if((position = ftell(raster)) < 0) printf("\nftell failed.\n"), exit(-1);

    printf("\nThe raster has %d rows, %d columns, %d bytes/cell (0=ascii).\n",
        rows, cols, file_type);

    /* get memory for arrays */
    Alloc_mem(cols); /* this function allocates global arrays */
    if((poly_ids = (int *) malloc( FULL * sizeof(int))) == NULL) mem_error();

    /* scan the raster file and get a list of all unique polygon ID's */
    printf("\nFIRST PASS:  Scanning the image for unique polygons...\n");
    Scan_rast(raster, poly_ids, &num_obs, rows, cols, file_type);
    if(num_obs > (long)FULL) {
        printf("\nERROR: Too many polygons (%ld) for this version \n", num_obs);
        printf("  of the program.  Raise the value of the constant 'FULL'\n");
        printf("  in the source code and re-compile.\n");
        exit(-1);
    }
    printf("\n  The raster has %ld polygons.  Polygon IDs:\n", num_obs);
    for(longi=0;longi<num_obs;longi++)
      printf(" %d",poly_ids[longi]);
    printf("\n");

    /* return raster pointer to point back to the first cell */
    if((fseek(raster, position, 0)) != 0) printf("\nfseek failed.\n"), exit(-1);

    if((outfile = fopen("full.asc", "w+")) == NULL)
        File_error("full.asc");


    /* initialize the temporary NxN array to the temp file */
    fprintf(outfile, "%10ld", num_obs);
    nsquared = num_obs * num_obs;
    for(longi=0; longi < nsquared; longi++)
            fprintf(outfile, " 0");


    /* check first row of raster (special case) */
    Read_a_row(1, raster, cols, file_type);
    top = above, mid = top;
    Read_a_row(2, raster, cols, file_type);
    bot = center;
    Check(poly_ids, num_obs, outfile, cols);
    bot = below, mid = center;
    Read_a_row(3, raster, cols, file_type);


    /* check middle rows */
    printf("\nSECOND PASS:  Checking for adjacencies...\n");
    printf("  Rows remaining to be processed:\n");
    for(i=0; i<(rows-3); i++) {
        if (i%50 == 0) printf("    %d\n", rows-i);
        Check(poly_ids, num_obs, outfile, cols);
        tmp = top, top = mid;
        mid = bot, bot = tmp;
        Read_a_row(row_cntr, raster, cols, file_type);
        if(++row_cntr == 4) row_cntr = 1;
    }

    /* check last row (special case) and free up memory */
    Check(poly_ids, num_obs, outfile, cols);
    top = mid, mid = bot;
    Check(poly_ids, num_obs, outfile, cols);
    free(above), free(center), free(below);

    /* flush the buffer */
    Sortbuf(count);
    Writedsk(outfile, count, poly_ids, num_obs);

    /* open the final output file */
    if((scm_file = fopen("sparse.asc", "w")) == NULL)
        File_error("sparse.asc");
    fprintf(scm_file, "%d\n", num_obs);

    /* convert the NxN matrix file (outfile) to Spacestat format */
    rewind(outfile);
    SSformat(outfile, scm_file, num_obs, poly_ids);

    printf("\nDone.  Output files are 'full.asc' and 'sparse.asc'.\n");
    return(0);

}
/**************************************************************************
ALLOC_MEM - allocates the needed memory objects.
**************************************************************************/
Alloc_mem(cols)
int cols;       /* number of columns in raster image */
{
  /* allocate a buffer array of poly/neib structures (RECORDS) */
  if((buffer = (RECORD *) malloc(FULL * sizeof(RECORD))) == NULL) mem_error();

  /* allocate three arrays with room for duplicate endpoints */
  if((above  = (int *) malloc( (cols+2) * sizeof(int))) == NULL) mem_error();
  if((center = (int *) malloc( (cols+2) * sizeof(int))) == NULL) mem_error();
  if((below  = (int *) malloc( (cols+2) * sizeof(int))) == NULL) mem_error();
}
mem_error()
{
    printf("Memory allocation error.");
    printf("  Try again after freeing up some memory.\n");
    exit(-1);
}
/***************************************************************************
CHECK.C:  compares cells & neibs for one row
***************************************************************************/
Check(poly_ids, num_obs, file, cols)
int poly_ids[];          /* sorted array of unique polygon ID's */
long num_obs;            /* size of above array */
FILE *file;              /* file for writing adjacency info */
int cols;       /* columns in raster image */
{
    int off,    /* offset from beg. of array */
        r, a,
        *row;

    for(off=1; off<=cols; off++) {       /* scan along the row */
        for(r=1; r<=3; r++) {            /* scan each row of a 3x3 window */
            if(r == 1) row = &top[0];            /* scan top row of 3x3 */
            if(r == 2) row = &mid[0];            /* scan middle row of 3x3 */
            if(r == 3) row = &bot[0];            /* scan botton row of 3x3 */

            for(a=(-1); a<=1; a++) {     /* scan 3 cells in a row of the 3x3 */

              /* check for an adjacency, and check that the same adjacency was
                 not just written */
              if ( (mid[off] != row[off+a]) &&
                 ((mid[off] != lastpoly) || (row[off+a] != lastneib)) &&
                 (mid[off] != 0) && (row[off+a] != 0) )
              {
                  /* write adjacency to buffer and update last vars. */
                  Writebuf(mid[off], row[off+a], poly_ids, num_obs, file);
                  lastpoly = mid[off], lastneib = row[off+a];
              }/* end if */
            } /* end for a */
        } /* end for r */
    } /* end for off */
}
/***************************************************************************
READ_A_ROW - fills a row with values from the raster file.  Note that the first and last
elements are duplicates of their neighboring values.  This is to avoid the border problem
when checking raster cell neighbors.
***************************************************************************/
Read_a_row(row, file, cols, file_type)
int row;        /* flag value (1,2,3) representing which array to fill */
FILE *file;     /* raster image file */
int cols;       /* number of columns in the raster image */
int file_type;  /* bytes per cell (0 for ascii) */
{
    int   i, result;
    int   *array;          /* pointer to one of the global arrays */
    unsigned char  one;
    unsigned short two;    /* for reading 1, 2, or 4-byte binary images */
    unsigned long four;

    /* find out which row to fill */
    if(row == 1) array = above;
    if(row == 2) array = center;
    if(row == 3) array = below;

    /* fill the array (excluding the endpoints) from the raster file */
    for(i=1;i<=cols;i++)
    {
        if(file_type == 0)              /* if ascii file */
            result = fscanf( file, "%d", &array[i]);
        else if(file_type == 1) {       /* if one-byte binary file */
            result = fread(&one, sizeof(char), 1, file);
            array[i] = (int) one;
        }
        else if(file_type == 2) {       /* if two-byte bin. file */
            result = fread(&two, sizeof(short), 1, file);
            array[i] = (int) two;
        }
        else {                          /* else 4-byte binary file */
```

```
            result = fread(&four, sizeof(long), 1, file);
            array[i] = (int) four;
        }
        if(result != 1) {
            printf("ERROR:  bad data read from raster image file.\n");
            exit(-1);
        }
    }   /* end for i */

    /* duplicate the endpoints */
    array[0] = array[1];
    array[cols+1] = array[cols];
}       /* end function */
/***************************************************************************
IDRISI - gets file parameters for an IDRISI image and returns a file
pointer to the raster data.
***************************************************************************/
#include <stdio.h>

FILE *Idrisi(ftype, rows, cols)
int *ftype,
    *rows,
    *cols;
{
    char docfile[84], imgfile[84]; /* extra 4 chars for file extensions */
    FILE *doc_file,        /* file pointer to idrisi '.DOC' file */
         *img_file;      /* raster image file */
    char c, byte_flag, ascii;
    char ver_flag;  /* r = idrisi version 3; c = version 4 */

    /* get filename from user */
    printf("Enter the name of the Idrisi image (no extension): ");
    gets(docfile);
    strcpy(imgfile,docfile);

    /* open the .DOC file */
    strcat(docfile,".doc");
    if((doc_file = fopen(docfile, "r")) == NULL)
        File_error(docfile);

    /* now get parameters from the .DOC file */
    while((c = getc(doc_file)) != '\n') ;  /* skip to end of current line */
    fseek(doc_file, 14, 1);  /* seek to appropriate place and get */
    fscanf(doc_file, "%c", &byte_flag); /* flag value (should be 'b' or 'i') */

    if(byte_flag != 'b' && byte_flag != 'i') {
         printf("ERROR:  data type must be integer or byte.\n");
         exit(-1);
    }

    while((c = getc(doc_file)) != '\n') ; /* skip to end of current line */
    fseek(doc_file, 14, 1);          /* get the ascii flag value */
    fscanf(doc_file, "%c", &ascii);      /* should be 'a' or 'b'     */

    if(ascii != 'a' && ascii != 'b') {
        printf("ERROR:  image file can not be packed binary format.");
        printf("  Use CONVERT.\n");
        exit(-1);
    }

    /* get row and column values for image */
    while((c = getc(doc_file)) != '\n') ; /* skip to end of current line */
    fscanf(doc_file, "%c", &ver_flag);     /* should be 'r' or 'c'     */
    if(ver_flag != 'r' && ver_flag != 'c') {
        printf("ERROR:  bad read in line 4 of the .DOC file.\n");
        exit(-1);
    }
    fseek(doc_file, 13, 1);
    fscanf(doc_file, "%d", ver_flag == 'r'? rows:cols);
    while((c = getc(doc_file)) != '\n') ;
    fseek(doc_file, 14, 1);
    fscanf(doc_file, "%d", ver_flag == 'r'? cols:rows);

    fclose(doc_file);     /* don't need this file again */

    /* assign the ftype variable */
    if(ascii == 'a') *ftype = 0;           /* ascii file */
    else {
        if(byte_flag == 'b') *ftype = 1;   /* binary byte file */
        if(byte_flag == 'i') *ftype = 2;   /* binary integer file */
    }
    /* now that ftype is known, open the .IMG file in ascii or bin mode */
    strcat(imgfile,".img");
    if((img_file = fopen(imgfile,ftype == 0 ? "r":"rb")) == NULL)
        File_error(imgfile);

    printf("\nThe IDRISI image is in version %c format.\n",
        ver_flag == 'r'? '3':'4');

    return(img_file);
}
/***************************************************************************
SSFORMAT - converts the temporary 0/1 matrix file to SPACESTAT sparse
binary contiguity format.
***************************************************************************/
SSformat(infile, scm_file, n, poly_ids)
FILE *infile,           /* NxN binary contiguity file (0/1) */
     *scm_file;         /* Sparse Contiguity Matrix file for Spacestat */
long n;                 /* number of observations (polygons) */
int  *poly_ids;         /* array of unique polygon ID's */
{
    int  num_neibs=0,   /* number of neighbors for a given polygon */
         i, j, k;       /* loop counters */
    int *one_row;       /* array to hold one row of the NxN bin. cont. matrix */

    /* allocate memory for one row of the NxN matrix */
    if( (one_row = (int *) malloc((int)n * sizeof(int))) == NULL)
        mem_error();

    /* seek over the first element of this file (number of observations) */
    fseek(infile, 10, 0);

    for(j=0; j<n; j++) {
       for(i=0; i<n; i++) {
           /* read in one row from file and increment num_neibs when approp.*/
           fscanf(infile, "%d", &one_row[i]);
           if( (one_row[i] == 1) && (poly_ids[i] != 0) ) num_neibs++;
       } /* end for i */

       /* now print info to the S.C.M. file */
       if(poly_ids[j] != 0) {
            fprintf(scm_file, "%d %d\n", poly_ids[j], num_neibs);
            for(k=0; k<n; k++)
                if(one_row[k] == 1) fprintf(scm_file, "%d ", poly_ids[k]);
            fprintf(scm_file, "\n");
        }       /* end if */
        if(num_neibs == 0)
            printf("\nWARNING!  Polygon %d has no neighbors.\n", poly_ids[j]);
        num_neibs = 0;
    } /* end for j */
} /* end function */
/***************************************************************************
SCAN_RAST - goes through each cell of the raster image and fills an array
with unique polygon ID's.  The global array 'center' is used as a buffer.
(This array is later used by other functions.)
***************************************************************************/
Scan_rast(rast_file, poly_ids, n, rows, cols, file_type)
FILE *rast_file;        /* pointer to raster image file */
int  poly_ids[];        /* array to be filled with unique polygon ID's */
long *n;                /* number of polygons found to exist in image */
int  rows, cols,        /* number of rows/cols in image */
     file_type;   /* format of image file (0=ascii, 1=bin/byte, 2=bin/integ) */
{
    int i, k,
        Comp(); /* comparison criteria function for qsort and bsearch */

    *n = 0;     /* initialize number of polygons found */

    printf("  Rows remaining to be processed:\n");
    for(k=0; k<rows; k++) {     /* for each row */
        /* read a row from image into center[] */
        if (k%50 == 0) printf("    %d\n", rows-k);
        Read_a_row(2, rast_file, cols, file_type);
        qsort(&center[0], cols+2, sizeof(int), Comp); /* sort the row array */
        for(i=0; i<(cols+1); i++) {     /* for each element in sorted array */
           if( center[i] == center[i+1] )
               continue;        /* if same as next element, skip it */
           else {
               /* if current poly id not already known and not zero */
               if( (center[i] != 0) && ((bsearch(&center[i], &poly_ids[0],
                 (int)*n, sizeof(int), Comp)) == NULL)) {
                   poly_ids[(*n)++] = center[i];  /* add it to poly array */
                   qsort(&poly_ids[0], (int)*n, sizeof(int), Comp);/*sort it*/
               } /* end if */
           } /* end else */
        } /* end for i */
        /* process last cell in the row */
        if( (center[cols+1] != 0) &&
          ((bsearch(&center[cols+1], &poly_ids[0],(int)*n, sizeof(int), Comp))
          == NULL)) {
            poly_ids[(*n)++] = center[cols+1];  /* add it to poly array */
            qsort(&poly_ids[0], (int)*n, sizeof(int), Comp);/* sort array */
        }  /* end if */
    } /* end for k */
} /* end function */

Comp(elem1, elem2)      /* compare function which is passed to qsort */
int *elem1, *elem2;
{
        if (*elem1 < *elem2) return(-1);
        else if (*elem1 > *elem2) return(1);
        else return(0);
}
/***************************************************************************
SORTBUF - sorts an array of poly/neib records.
***************************************************************************/
Sortbuf(size)
int size;       /* number of records currently in buffer */
{
    int Compare();      /* function containing comparison criteria for qsort */

    qsort( &buffer[0], size, sizeof(RECORD), Compare);
}

Compare(elem1, elem2) /* compare function which is passed to qsort */
RECORD *elem1, *elem2;
{
        if (elem1->poly < elem2->poly) return(-1);
        else if (elem1->poly > elem2->poly) return(1);
        else
        {
            if (elem1->neib < elem2->neib) return(-1);
            else if (elem1->neib > elem2->neib) return(1);
            else return(0);
        }
}
/***************************************************************************
WRITEBUF - this function is passed two integers which represent a
polygon/neighbor adjacency.  The integers are written to a buffer array,
which is periodically (when full) sorted, thinned, and written to disk.
***************************************************************************/
Writebuf(poly, neib, poly_ids, num_obs, file)
int poly, neib,
    poly_ids[];         /* sorted array of unique polygon ID's */
long num_obs;           /* size of above array */
FILE *file;             /* file for writing */
{
    /* write the adjacency to the buffer */
    buffer[count].poly = poly, buffer[count].neib = neib;

    /* if buffer is full, sort it and write it to disk (the 'writedsk'
       function thins out redundant adjacencies) */
    if (++count >= FULL) {
        Sortbuf(count);
        Writedsk(file, count, poly_ids, num_obs);
        count = 0;
    } /* end if */
}
/***************************************************************************
WRITEDSK - when the poly/neib buffer is full, this function writes the
sorted and thinned adjacency information to hard disk.
***************************************************************************/
Writedsk(file, num_items, poly_ids, num_obs)
FILE *file;
int num_items,          /* number of items in buffer */
    poly_ids[];         /* sorted array of unique polygon ID's */
long num_obs;           /* size of above array */
{
    int i,
        Comp();
    int *poly_ptr, *neib_ptr,    /* pointers returned by bsearch */
        poly_index, neib_index;     /* positions of poly/neib in poly_ids[] */
    long seek_val;       /* distance to skip along the full matrix */

    /* write the first pair unless it has zeroes */
    if( (buffer[0].poly != 0) && (buffer[0].neib != 0) &&
        (buffer[0].poly != buffer[0].neib) )
    {
           poly_ptr = (int *) bsearch(&buffer[0].poly, &poly_ids[0],
               (int)num_obs, sizeof(int), Comp);
           neib_ptr = (int *) bsearch(&buffer[0].neib, &poly_ids[0],
               (int)num_obs, sizeof(int), Comp);
           poly_index = poly_ptr - poly_ids;
           neib_index = neib_ptr - poly_ids;
           seek_val = (long)(10+(2 * ( (poly_index * num_obs) + neib_index)));
           fseek(file, seek_val, 0);
           fprintf(file, " 1");
    }

    /* now write only non-redundant pairs which don't refer to the 0 poly */
    for(i=1; i<num_items; i++)
      if( ((buffer[i].poly != buffer[i-1].poly) ||
          (buffer[i].neib != buffer[i-1].neib)) &&
          (buffer[i].poly != 0) && (buffer[i].neib != 0) &&
          (buffer[i].poly != buffer[i].neib) )
      {
           poly_ptr = (int *) bsearch(&buffer[i].poly, &poly_ids[0],
               (int)num_obs, sizeof(int), Comp);
           neib_ptr = (int *) bsearch(&buffer[i].neib, &poly_ids[0],
               (int)num_obs, sizeof(int), Comp);
           poly_index = poly_ptr - poly_ids;
           neib_index = neib_ptr - poly_ids;
           seek_val = (long)(10+(2 * ( (poly_index * num_obs) + neib_index)));
           fseek(file, seek_val, 0);
           fprintf(file, " 1");
      }
} /* end function */
/*****************************************************************************
OSUMAP - gets file parameters for an OSU-MAP image and returns a file
pointer to the raster data.
*****************************************************************************/
FILE *Osumap(ftype, rows, cols)
int *ftype,
    *rows,
    *cols;
{
    char  filename[80];
    FILE  *osu_file;
    long  lrows, lcols; /* temp. variables to hold long values */

    /* set ftype (all osu-map files use 2-byte binary integer format) */
    *ftype = 2;

    /* get filename from user and open the file */
    printf("Enter the name of the OSU-MAP file (no extension): ");
    gets(filename);
    if((osu_file = fopen(filename, "r")) == NULL)
        File_error(filename);

    /* skip to the right place and read rows/columns */
    fseek(osu_file, 1792L, 0);
    fread(&lrows, sizeof(long), 1, osu_file);
    fread(&lcols, sizeof(long), 1, osu_file);
    *rows = (int) lrows;
    *cols = (int) lcols;

    /* skip to where the cell data begin and return file pointer */
    fseek(osu_file, 3840L, 0);
    return(osu_file);
}
/*****************************************************************************
GENERIC - Allows practically any raster file format to be read by
prompting the user for all needed parameters.  Returns a file pointer
to the raster data.
*****************************************************************************/
FILE *Generic(file_type, rows, cols)
int *file_type, *rows, *cols;
{
    char filename[80], input[20];
    FILE *gen_file;
    int  flag, bytes;
    long head_skip; /* header bytes to skip */
    int c, x;
    /* get filename from user and open the file */
    printf("Enter the name of the raster image file:  ");
    gets(filename);
    if((gen_file = fopen(filename, "r")) == NULL)
        File_error(filename);

    /* get the number of rows/cols */
    while(1) {
        printf("\nEnter the number of rows:  ");
        gets(input);
        if ((flag = sscanf(input, "%d", rows)) != 1)
            printf("Bad input.  Please re-enter...\n");
        else break;
    }
    while(1) {
        printf("\nEnter the number of cols:  ");
        gets(input);
        if ((flag = sscanf(input, "%d", cols)) != 1)
            printf("Bad input.  Please re-enter...\n");
        else break;
    }

    /* get file_type */
    while(1) {
        printf("\nEnter the file type (a = ascii, b = binary):  ");
        if((c = getchar()) != EOF) {
            while((x = getchar()) != '\n'); /* ignore stuff beyond one char */
            if(c == 'a' || c == 'A') {
                *file_type = 0;
                break;
            }
            else if(c == 'b' || c == 'B') {
                *file_type = 1;
                break;
            }
            else printf("Bad input.  Please re-enter...\n");
        }
    }

    /* if binary file, re-open as binary and get #bytes/cell */
    if(*file_type == 1) {
        fclose(gen_file);
        if((gen_file = fopen(filename, "rb")) == NULL)
            File_error(filename);
        while(1) {
            printf("\nEnter the number of bytes per cell (1, 2, or 4):  ");
            gets(input);
            if ((flag = sscanf(input, "%d", &bytes)) == 1) {
                if(bytes == 1 || bytes == 2 || bytes == 4) {
                    *file_type = bytes;
                    break;
                }
            }
            printf("Bad input.  Please re-enter...\n");
        }
    }

    /* skip over header if necessary */
    while(1) {
       printf("\nEnter the number of header bytes in %s that should be skipped",
filename);
       printf("\n  or press <Enter> for no header:  ");
       gets(input);
            if (input[0] == '\0') {
                head_skip = 0;
                break;
            }
            else if ((flag = sscanf(input, "%ld", &head_skip)) == 1)
                break;
            else printf("Bad input.  Please re-enter...\n");
    }

    /* skip to where the cell data begin and return file pointer */
    fseek(gen_file, head_skip, 0);
    return(gen_file);
}
/****************************************************************************
USAGE - Displays command usage
****************************************************************************/
Usage() {
    printf("\n      This program requires one of the following arguments:\n");
    printf("======================================");
    printf("======================================\n\n");
    printf("  'i' - Reads an IDRISI image.\n\n");
    printf("  'o' - Reads an OSU-MAP image.\n\n");
    printf("  'g' - Reads a generic raster format by prompting for all");
    printf(" needed parameters.\n\n");
    printf("======================================");
    printf("======================================\n\n");
    exit(-1);
}
/***************************************************************************
FILE_ERROR - reports error on file opening and exits
***************************************************************************/
File_error(string)
char *string; /* name of file which failed to open */
{
    printf("Error opening the file: '%s'.\n",string);
    exit(-1);
}
```

RASDIST.C

```
/***************************************************************************
RASDIST - reads a raster polygon image and returns polygon centroids and
a distance-base spatial weight matrix.
By Rusty Dodson, NCGIA Santa Barbara.
***************************************************************************/
#include <stdio.h>
#include <math.h>
#define FULL 1000  /* Maximum allowable polygons.  Must be larger than number */
                   /* of polygons in the raster image. */

/* global variables */
float *x, *y;      /* arrays of x and y coordinates */
long   *pix_count; /* count of #pixels per polygon */
int  *rast_row;    /* stores one row of the raster */

main(argc, argv) /* main program */
int argc;
char *argv[];
{
    int i; /* counter */
    int rows, cols;     /* rows and columns in the image file */
    int num_obs;        /* number of polygons in raster file (excluding zeros)*/
    int *poly_ids;      /* array of unique polygon ID's */
    FILE *raster,       /* raster image file */
         *outfile, *outfile2, *outfile3;
    long position;      /* variable to hold position of file pointer */
    int  file_type;     /* bytes per cell (0 for ascii) */
    FILE *Idrisi(), *Osumap(), *Generic(); /* image-reading funcions */
    float dist_thresh, dist_decay; /* distance cutoff, distance decay */

    if(argc != 2) { /* if invalid # of arguments, display usage */
        Usage();
    }

    switch(argv[1][0]) {  /* switch on the command-line argument */
        case 'i': case 'I':
            raster = Idrisi(&file_type, &rows, &cols);
            break;
        case 'o': case 'O':
            raster = Osumap(&file_type, &rows, &cols);
            break;
        case 'g': case 'G':
            raster = Generic(&file_type, &rows, &cols);
            break;
        default:
            Usage();
    } /* end switch */

    /* get distance parameters from user */
    Get_dist_params(&dist_thresh, &dist_decay);

    if( rows < 3 || cols < 3 ) {
        printf("Error:  The raster dimension must be at least 3 by 3.\n");
        exit(-1);
    }

    /* remember the position of the raster file pointer */
    if((position = ftell(raster)) < 0) printf("\nftell failed.\n"), exit(-1);

    printf("\nThe raster has %d rows, %d columns, %d bytes/cell (0=ascii).\n",
        rows, cols, file_type);

    /* allocate some arrays */
    if((poly_ids = (int *) malloc( FULL * sizeof(int))) == NULL) mem_error();
    Alloc_mem(cols);

    /* scan the raster file and get a list of all unique polygon ID's */
    printf("\nFIRST PASS:  Scanning the image for unique polygons...\n");
    Scan_rast(raster, poly_ids, &num_obs, rows, cols, file_type);
    if(num_obs > FULL) {
        printf("\nERROR: Too many polygons (%d) for this version \n", num_obs);
        printf("  of the program.  Raise the value of the constant 'FULL'\n");
        printf("  in the source code and re-compile.\n");
        exit(-1);
    }
    printf("\n  The raster has %d polygons.  Polygon ID's:\n", num_obs);
    for(i=0;i<num_obs;i++)
      printf(" %d",poly_ids[i]);
    printf("\n");

    /* allocate global arrays */
    if((x = (float *) malloc( num_obs * sizeof(float))) == NULL) mem_error();
    if((y = (float *) malloc( num_obs * sizeof(float))) == NULL) mem_error();
    if((pix_count = (long *)malloc(num_obs * sizeof(long))) == NULL)mem_error();

    /* initialize above arrays */
    for(i=0;i<num_obs;i++) {
        x[i] = 0.0;
        y[i] = 0.0;
        pix_count[i] = 0;
    }

    /* return raster pointer to point back to the first cell */
    if((fseek(raster, position, 0)) != 0) printf("\nfseek failed.\n"), exit(-1);

    printf("\nSECOND PASS:  Calculating polygon centroids...\n");
    Get_centroids(raster, poly_ids, num_obs, rows, cols, file_type);

    if((outfile = fopen("centroid.asc", "w")) == NULL)
        File_error("centroid.asc");
    if((outfile2 = fopen("dmat.asc", "w")) == NULL)
        File_error("dmat.asc");

    /* if idrisi, open another output file */
    if(argv[1][0] == 'i' || argv[1][0] == 'I'){
        if((outfile3 = fopen("centroid.vec", "w")) == NULL)
            File_error("centroid.vec");
    }

    /* write a file of centroids */
    if(argv[1][0] == 'i' || argv[1][0] == 'I'){ /* write 2 files for idrisi */
        fprintf(outfile, "%d 3 ID X Y\n", num_obs);
        for(i=0; i<num_obs; i++) {
            fprintf(outfile3, "%d  1\n%f  %f\n", poly_ids[i], x[i], y[i]);
            fprintf(outfile, "%d %f %f\n", poly_ids[i], x[i], y[i]);
        }
        fprintf(outfile3, "0  0\n");
    }
    else for(i=0; i<num_obs; i++)
        fprintf(outfile, "%d %f %f\n", poly_ids[i], x[i], y[i]);

    /* make the sparse distance matrix */
    printf("\nCalculating the sparse distance matrix...\n");
    Make_dmat(poly_ids, dist_thresh, dist_decay, num_obs, outfile2);

    printf("\nDone.  Output files are 'centroid.asc' and 'dmat.asc'.\n");

    if(argv[1][0] == 'i' || argv[1][0] == 'I'){
        printf("\n  Also, a file called 'centroid.vec' was created for\n");
        printf("dislaying centroids with Idrisi.  Use Idrisi's DOCUMENT\n");
        printf("module to make a .DVC file for the 'centroid.vec'\n");
        printf("vector file.\n\n");
    }
return(0);

/*************************/
} /* end main function */
/***************************************************************************
```

```
ALLOC_MEM - allocates the needed memory objects.
****************************************************************************/
Alloc_mem(cols)
int cols;       /* number of columns in raster image */
{
  /* allocate an array to hold a row of the raster */
  if((rast_row = (int *)malloc(cols * sizeof(int))) == NULL) mem_error();
}
mem_error()
{
    printf("Memory allocation error.");
    printf("  Try again after freeing up some memory.\n");
    exit(-1);
}
/****************************************************************************
READ_A_ROW - fills a row with values from the raster file.
****************************************************************************/
Read_a_row(file, cols, file_type)
FILE *file;     /* raster image file */
int cols;       /* number of columns in the raster image */
int file_type;  /* bytes per cell (0 for ascii) */
{
    int   i, result;
    unsigned char  one;
    unsigned short two;   /* for reading 1, 2, or 4-byte binary images */
    long four;

    /* fill the array (excluding the endpoints) from the raster file */
    for(i=0;i<cols;i++)
    {
        if(file_type == 0)              /* if ascii file */
            result = fscanf( file, "%d", &rast_row[i]);
        else if(file_type == 1) {       /* if one-byte binary file */
            result = fread(&one, sizeof(char), 1, file);
            rast_row[i] = (int) one;
        }
        else if(file_type == 2) {       /* if two-byte bin. file */
            result = fread(&two, sizeof(short), 1, file);
            rast_row[i] = (int) two;
        }
        else {                          /* else 4-byte binary file */
            result = fread(&four, sizeof(long), 1, file);
            rast_row[i] = (int) four;
        }
        if(result != 1) {
            printf("ERROR:  bad data read from raster image file.\n");
            exit(-1);
        }
    }   /* end for i */
}       /* end function */
/****************************************************************************
IDRISI - gets file parameters for an IDRISI image and returns a file
pointer to the raster data.
****************************************************************************/
#include <stdio.h>

FILE *Idrisi(ftype, rows, cols)
int *ftype,
    *rows,
    *cols;
{
    char docfile[84], imgfile[84]; /* extra 4 chars for file extensions */
    FILE *doc_file,       /* file pointer to idrisi '.DOC' file */
         *img_file;      /* raster image file */
    char c, byte_flag, ascii;
    char ver_flag;  /* r = idrisi version 3; c = version 4 */

    /* get filename from user */
    printf("Enter the name of the Idrisi image (no extension): ");
    gets(docfile);
    strcpy(imgfile,docfile);

    /* open the .DOC file */
    strcat(docfile,".doc");
    if((doc_file = fopen(docfile, "r")) == NULL)
        File_error(docfile);

    /* now get parameters from the .DOC file */
    while((c = getc(doc_file)) != '\n') ;  /* skip to end of current line */
    fseek(doc_file, 14, 1);  /* seek to appropriate place and get */
    fscanf(doc_file, "%c", &byte_flag); /* flag value (should be 'b' or 'i') */
    if(byte_flag != 'b' && byte_flag != 'i') {
         printf("ERROR:  data type must be integer or byte.\n");
         exit(-1);
    }

    while((c = getc(doc_file)) != '\n') ; /* skip to end of current line */
    fseek(doc_file, 14, 1);          /* get the ascii flag value */
    fscanf(doc_file, "%c", &ascii);     /* should be 'a' or 'b'     */

    if(ascii != 'a' && ascii != 'b') {
        printf("ERROR:  image file can not be packed binary format.");
        printf("  Use CONVERT.\n");
        exit(-1);
    }

    /* get row and column values for image */
    while((c = getc(doc_file)) != '\n') ; /* skip to end of current line */
    fscanf(doc_file, "%c", &ver_flag);     /* should be 'r' or 'c'     */
    if(ver_flag != 'r' && ver_flag != 'c') {
        printf("ERROR:  bad read in line 4 of the .DOC file.\n");
        exit(-1);
    }
    fseek(doc_file, 13, 1);
    fscanf(doc_file, "%d", ver_flag == 'r'? rows:cols);
    while((c = getc(doc_file)) != '\n') ;
    fseek(doc_file, 14, 1);
    fscanf(doc_file, "%d", ver_flag == 'r'? cols:rows);

    fclose(doc_file);     /* don't need this file again */

    /* assign the ftype variable */
    if(ascii == 'a') *ftype = 0;           /* ascii file */
    else {
        if(byte_flag == 'b') *ftype = 1;   /* binary byte file */
        if(byte_flag == 'i') *ftype = 2;   /* binary integer file */
    }

    /* now that ftype is known, open the .IMG file in ascii or bin mode */
    strcat(imgfile,".img");
    if((img_file = fopen(imgfile,ftype == 0 ? "r":"rb")) == NULL)
        File_error(imgfile);

    printf("\nThe IDRISI image is in version %c format.\n",
        ver_flag == 'r'? '3':'4');

    return(img_file);
}
/**************************************************************************
SCAN_RAST - goes through each cell of the raster image and fills an array
with unique polygon ID's.  The global array 'rast_row' is used as a buffer.
(This array is later used by other functions.)
**************************************************************************/
Scan_rast(rast_file, poly_ids, n, rows, cols, file_type)
FILE *rast_file;        /* pointer to raster image file */
int  poly_ids[],        /* array to be filled with unique polygon ID's */
     *n,                /* number of polygons found to exist in image */
     rows, cols,        /* number of rows/cols in image */
     file_type;         /* format of image file (0=ascii, 1=bin/byte, 2=bin/integ) */
{
    int i, k,
        Comp(); /* comparison criteria function for qsort and bsearch */

    *n = 0;     /* initialize number of polygons found */

    printf("  Rows remaining to be processed:\n");
    for(k=0; k<rows; k++) {     /* for each row */
        /* read a row from image into rast_row[] */
        if (k%50 == 0) printf("    %d\n", rows-k);
        Read_a_row(rast_file, cols, file_type);
        qsort(&rast_row[0], cols, sizeof(int),Comp);/* sort the row array */
        for(i=0; i<(cols-1); i++) {     /* for each element in sorted array */
           if( rast_row[i] == rast_row[i+1] )
               continue;         /* if same as next element, skip it */
           else {
               /* if current poly id not already known and not zero */
               if( (rast_row[i] != 0) && ((bsearch(&rast_row[i], &poly_ids[0],
                 *n, sizeof(int), Comp)) == NULL)) {
                   poly_ids[(*n)++] = rast_row[i];  /* add it to poly array */
                   qsort(&poly_ids[0], *n, sizeof(int), Comp);/* sort array */
               } /* end if */
```

```
            } /* end else */
        } /* end for i */
        /* process last cell in the row */
        if( (rast_row[cols-1] != 0) &&
          ((bsearch(&rast_row[cols-1], &poly_ids[0], *n, sizeof(int), Comp))
          == NULL)) {
            poly_ids[(*n)++] = rast_row[cols-1];
            qsort(&poly_ids[0], *n, sizeof(int), Comp);
        } /* end if */
    } /* end for k */
} /* end function */

Comp(elem1, elem2)      /* compare function which is passed to qsort */
int *elem1, *elem2;
{
        if (*elem1 < *elem2) return(-1);
        else if (*elem1 > *elem2) return(1);
        else return(0);
}
/****************************************************************************
OSUMAP - gets file parameters for an OSU-MAP image and returns a file
pointer to the raster data.
****************************************************************************/
FILE *Osumap(ftype, rows, cols)
int *ftype,
    *rows,
    *cols;
{
    char  filename[80];
    FILE  *osu_file;
    long  lrows, lcols; /* temp. variables to hold long values */

    /* set ftype (all osu-map files use 2-byte binary integer format) */
    *ftype = 2;

    /* get filename from user and open the file */
    printf("Enter the name of the OSU-MAP file (no extension): ");
    gets(filename);
    if((osu_file = fopen(filename, "r")) == NULL)
        File_error(filename);

    /* skip to the right place and read rows/columns */
    fseek(osu_file, 1792L, 0);
    fread(&lrows, sizeof(long), 1, osu_file);
    fread(&lcols, sizeof(long), 1, osu_file);
    *rows = (int) lrows;
    *cols = (int) lcols;

    /* skip to where the cell data begin and return file pointer */
    fseek(osu_file, 3840L, 0);
    return(osu_file);
}
/****************************************************************************
GENERIC - Allows practically any raster file format to be read by
prompting the user for all needed parameters.  Returns a file pointer
to the raster data.
****************************************************************************/
FILE *Generic(file_type, rows, cols)
int *file_type, *rows, *cols;
{
    char filename[80], input[20];
    FILE *gen_file;
    int  flag, bytes;
    long head_skip; /* header bytes to skip */
    int c, x;

    /* get filename from user and open the file */
    printf("Enter the name of the raster image file:  ");
    gets(filename);
    if((gen_file = fopen(filename, "r")) == NULL)
        File_error(filename);

    /* get the number of rows/cols */
    while(1) {
        printf("\nEnter the number of rows:  ");
        gets(input);
        if ((flag = sscanf(input, "%d", rows)) != 1)
            printf("Bad input.  Please re-enter...\n");
        else break;
    }
    while(1) {
        printf("\nEnter the number of cols:  ");
        gets(input);
        if ((flag = sscanf(input, "%d", cols)) != 1)
            printf("Bad input.  Please re-enter...\n");
        else break;
    }

    /* get file_type */
    while(1) {
        printf("\nEnter the file type (a = ascii, b = binary):  ");
        if((c = getchar()) != EOF) {
            while((x = getchar()) != '\n'); /* ignore stuff beyond one char */
            if(c == 'a' || c == 'A') {
                *file_type = 0;
                break;
            }
            else if(c == 'b' || c == 'B') {
                *file_type = 1;
                break;
            }
            else printf("Bad input.  Please re-enter...\n");
        }
    }

    /* if binary file, re-open as binary and get #bytes/cell */
    if(*file_type == 1) {
        fclose(gen_file);
        if((gen_file = fopen(filename, "rb")) == NULL)
            File_error(filename);
        while(1) {
            printf("\nEnter the number of bytes per cell (1, 2, or 4):  ");
            gets(input);
            if ((flag = sscanf(input, "%d", &bytes)) == 1) {
                if(bytes == 1 || bytes == 2 || bytes == 4) {
                    *file_type = bytes;
                    break;
                }
            }
            printf("Bad input.  Please re-enter...\n");
        }
    }

    /* skip over header if necessary */
    while(1) {
       printf("\nEnter the number of header bytes in %s that should be skipped",
filename);
       printf("\n  or press <Enter> for no header:  ");
       gets(input);
            if (input[0] == '\0') {
                head_skip = 0;
                break;
            }
            else if ((flag = sscanf(input, "%ld", &head_skip)) == 1)
                break;
            else printf("Bad input.  Please re-enter...\n");
    }

    /* skip to where the cell data begin and return file pointer */
    fseek(gen_file, head_skip, 0);
    return(gen_file);
}
/***************************************************************************
USAGE - Displays command usage
***************************************************************************/
Usage() {
    printf("\n      This program requires one of the following arguments:\n");
    printf("=======================================");
    printf("=======================================\n\n");
    printf("  'i' - Reads an IDRISI image.\n\n");
    printf("  'o' - Reads an OSU-MAP image.\n\n");
    printf("  'g' - Reads a generic raster format by prompting for all");
    printf(" needed parameters.\n\n");
    printf("=======================================");
    printf("=======================================\n\n");
    exit(-1);
}
/***************************************************************************
FILE_ERROR - reports error on file opening and exits
***************************************************************************/
File_error(string)
char *string; /* name of file which failed to open */
```

```
{
    printf("Error opening the file: '%s'.\n",string);
    exit(-1);
}
/****************************************************************************
GET_CENTROIDS - scans each image pixel and tallys up x, y, and pix_count
arrays for the appropriate polygon-id.  Then x and y are divided by pix_count
thus calculating the centroid.
****************************************************************************/
Get_centroids(rast_file, poly_ids, n, rows, cols, file_type)
FILE *rast_file;        /* pointer to raster image file */
int  *poly_ids,         /* array of unique polygon ID's */
     n,                 /* length of above array */
     rows, cols,        /* number of rows/cols in image */
     file_type;   /* format of image file (0=ascii, 1=bin/byte, 2=bin/integ) */
{
    int i, k,
        cell, /* a cell value from the raster image */
        *index_ptr, /* pointer returned by bsearch() */
        index,      /* position of cell in poly_ids */
        Comp(); /* comparison criteria function for bsearch */

    printf("  Rows remaining to be processed:\n");
    for(k=0; k<rows; k++) {      /* for each row */
        if (k%50 == 0) printf("    %d\n", rows-k);
        for(i=0; i<cols; i++) {      /* for each column */
            /* read a cell value from the raster file */
            cell = Read_cell(rast_file, file_type);
            if(cell != 0) {
                /* find cell's position in poly_ids array */
                index_ptr = (int *) bsearch(&cell, &poly_ids[0], n,
                    sizeof(int), Comp);
                index = index_ptr - poly_ids;
                x[index] += (float) i;
                y[index] += (float) k;
                pix_count[index]++;
            } /* end if */
        } /* end for i */
    } /* end for k */

    for(i=0; i<n; i++) {  /* now calculate the centroids */
    /* 0.5 is added below to put centroids at pixel centers */
        x[i] /= (float) pix_count[i];
        x[i] += 0.5;
        y[i] /= (float) pix_count[i];
        y[i] = (float) (rows-1) - y[i]; /* this line flips the y-axis */
        y[i] += 0.5;
    }
} /* end function */
/****************************************************************************
READ_CELL - returns a single cell value from the current position in the
raster image file.
****************************************************************************/
Read_cell(file, file_type)
FILE *file;   /* file pointer */
int  file_type; /* file type:  (0=ascii, 1=bin/byte, 2=bin/integ) */
{
    int cell;
    unsigned char  one;
    unsigned short two;   /* for reading 1, 2, or 4-byte binary images */
    long four;

    if(file_type == 0) {              /* if ascii file */
        fscanf( file, "%d", &cell);
        return(cell);
    }
    else if(file_type == 1) {         /* if one-byte binary file */
        fread(&one, sizeof(char), 1, file);
        return (int) one;
    }
    else if(file_type == 2) {         /* if two-byte bin. file */
        fread(&two, sizeof(short), 1, file);
        return (int) two;
    }
    else {                            /* else 4-byte binary file */
        fread(&four, sizeof(long), 1, file);
        return (int) four;
    }
} /* end function */
/****************************************************************************
MAKE_DMAT - makes the sparse distance matrix.
```

```
****************************************************************************/
Make_dmat(poly_ids, thresh, decay, n, outfile)
int   *poly_ids;     /* array of unique polygon-ids */
float thresh, decay; /* cutoff distance and distance-decay parameters */
int   n;             /* number of observations */
FILE  *outfile;      /* output file */
{
    int i, j;
    double sqdist, sqthresh, newdist;
    float Dist_sqr();

    if(decay < 0) decay *= -1; /* make sure decay power is positive */
    fprintf(outfile, "%d\n", n); /* print number of obs. to top of file */
    sqthresh = thresh * thresh; /* square the threshold for efficiency */
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            if(i==j) fprintf(outfile, "%d %d 0.0\n", poly_ids[i], poly_ids[j]);
            else {
                sqdist = Dist_sqr(x[i], y[i], x[j], y[j]);
                if(sqdist <= sqthresh) {
                    if(decay == 2) newdist = 1/sqdist;
                    else newdist = 1/(pow(sqrt(sqdist),decay));
                    fprintf(outfile, "%d %d %E\n", poly_ids[i], poly_ids[j],
                        newdist);
                } /* end if */
            } /* end if */
        } /* end for j */
    } /* end for i */
} /* end function */
/****************************************************************************
DIST_SQR - returns squared Euclidean distance between two points.
****************************************************************************/
float Dist_sqr(x1, y1, x2, y2)
float x1, y1, x2, y2; /* two points */
{
    float sqr_dist; /* return value */

    sqr_dist = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2);
    if(sqr_dist < 1.0 && sqr_dist != 0.0)
        printf("WARNING:  distance less than 1.0 encountered.\n");
    if(sqr_dist == 0.0) {
        printf("WARNING:  zero distance encountered--");
        printf("using 0.0000000001 instead...\n");
        sqr_dist = 0.0000000001;
    }
    return(sqr_dist);
}
/****************************************************************************
GET_DIST_PARAMS - inputs distance parameters from user
****************************************************************************/
Get_dist_params(thresh, decay)
float *thresh, *decay;
{
    char input[20];
    int flag;

    while(1) {
        printf("\nEnter the distance threshold in units of cells:  ");
        gets(input);
        if ((flag = sscanf(input, "%f", thresh)) != 1)
            printf("Bad input.  Please re-enter...\n");
        else break;
    }
    while(1) {
        printf("\nEnter the distance-decay power (as a positive number):  ");
        gets(input);
        if ((flag = sscanf(input, "%f", decay)) != 1)
            printf("Bad input.  Please re-enter...\n");
        else break;
    }
} /* end function */
```

SP2FULL.C

```
/****************************************************************************

SP2FULL - converts a spatial weight matrix from sparse binary or general
    format to full matrix format.

  The first line of the input file must contain an integer designating the
degree of the resulting output matrix.  This number is the same as the
```

```
number of observations, or the number of unique polygon-ID values.

Rusty Dodson, NCGIA Santa Barbara, 4/92.
*************************************************************************/
#include <stdio.h>
#define ZERO 0.0

main()
{
    char infile[80], outfile[80], c;
    FILE *fp_in, *fp_out;
    double *array;       /* array of poly-ID's */
    double *poly_array;    /* array of polygons for general matrix case */
    double *neib_array;    /* array of neighbors       "        "        */
    double *wt_array=NULL; /* array of weight values   "        "        */
    double temp; /* temporary variable for holding fscanf values */
    long position; /* variable to hold position of file pointer */
    int numobs; /* number of unique polygon-ID's */
    int poly_prec; /* precision of poly-IDs and weights */
    int file_type = 0; /* '1' for binary input matrix, '2' for general */
    int result; /* return value from fscanf */
    int poly, neib_index; /* single poly-ID and neib_array position */
    int numneibs; /* # of neighbors for a polygon */
    int done_flag = 0; /* indicates when all non-zero elements are written */
    int i,j,rows; /* counters */

    /* get input file name and open it */
    printf("\nEnter the name of the input sparse matrix file:  ");
    gets(infile);
    if((fp_in = fopen(infile, "r")) == NULL)
        File_error(infile);

    /* get and open output file */
    printf("\nEnter the name of the full matrix file to be created:  ");
    gets(outfile);
    if((fp_out = fopen(outfile, "w")) == NULL)
        File_error(outfile);

    fscanf(fp_in, "%d", &numobs); /* get numobs from 1st line of file */
    position = ftell(fp_in); /* remember position of file pointer */
    file_type = Get_format(fp_in); /* get format of input file (bin or gen) */
    poly_prec = Get_prec(fp_in);
    fseek(fp_in, position, 0); /* return file ptr to earlier position */

    /* allocate memory for arrays */
    if((array = (double *) malloc(numobs * sizeof(double))) == NULL)
        Mem_error();
    if((neib_array = (double *) malloc((numobs+2) * sizeof(double))) == NULL)
        Mem_error();
    if(file_type == 2) { /* need more arrays for general input matrix case */
        if((poly_array = (double *) malloc((numobs+2) * sizeof(double)))
            == NULL) Mem_error();
        if((wt_array = (double *) malloc((numobs+2) * sizeof(double))) == NULL)
            Mem_error();
        /* skip pointer to the first weight */
        fscanf(fp_in, "%f %f", &temp, &temp);
    }

    /* go through the input file and get an array of all poly-ID's */
    fseek(fp_in, position, 0); /* return file ptr to earlier position */
    if(file_type == 1) /* if binary format input file */
    {
        for(i=0; i<numobs; i++)
        {
            result = fscanf(fp_in, "%lf", &array[i]);
            if(result == EOF) {
                printf("\nERROR: %d polygon-IDs not found in input file.\n",
                    numobs);
                exit(-1);
            }
            for(j=0; j<2; j++) /* skip next two newlines */
                while((c=getc(fp_in)) != '\n');
        }
    }
    else /* general format input file */
    {
        i = 0;
        result = fscanf(fp_in, "%lf", &array[i]); /* get first poly-ID */
        while(1)
        {
            if(result == EOF && (i+1) == numobs) break;
            else if (result == EOF) {
                printf("\nERROR: %d polygon-IDs not found in input file.\n",
                    numobs);
                exit(-1);
            }
            while((c = getc(fp_in)) != '\n'); /* skip to end of line */
            result = fscanf(fp_in, "%lf", &temp);
            if(temp == array[i]) continue;
            else array[++i] = temp;
        }
    }

    printf("Polygon-id's:\n"); /* print poly IDs for an error check */
    for(i=0;i<numobs;i++) printf(" %.*lf",poly_prec, array[i]);
    printf("\n");

    /* now make the full matrix */
    fseek(fp_in, position, 0); /* return file ptr to earlier position */
    if(file_type == 1) /* binary format input file */
    {
        for(rows=0; rows<numobs; rows++) {
            if((result = fscanf(fp_in, "%lf %d", &poly, &numneibs)) == EOF)
                return(0);
            for(j=0; j<numneibs; j++)
                fscanf(fp_in, "%lf", &neib_array[j]);
            neib_index = Get_index(array, neib_array[0], numobs);
            for(i=0, j=1; i<numobs; i++) {
                if(j<=numneibs && i==neib_index) {
                    fprintf(fp_out, " 1");
                    neib_index = Get_index(array, neib_array[j++], numobs);
                }
                else fprintf(fp_out, " 0");
                /* print carriage returns every screen line (80 chars) */
                if((i+1)%39 == 0 && (i+1) != numobs) fprintf(fp_out, "\n");
            } /* end for i */
            fprintf(fp_out, "\n");
        } /* end for */
    } /* end if */

    else /* general format input file */
    {
        numneibs = 0, rows = 0;
        result = fscanf(fp_in, "%lf %lf %lf", &poly_array[numneibs],
            &neib_array[numneibs], &wt_array[numneibs]);
        while(rows < numobs) {
            numneibs++;
            result = fscanf(fp_in, "%lf %lf %lf", &poly_array[numneibs],
                &neib_array[numneibs], &wt_array[numneibs]);
            if(poly_array[numneibs] != poly_array[0] || result == EOF) {
                /* print the row of the matrix */
                neib_index = Get_index(array, neib_array[0], numobs);
                for(i=0, j=0; i<numobs; i++) {
                    if(j<numneibs && i==neib_index && done_flag == 0) {
                        fprintf(fp_out, " %lE", wt_array[j]);
                        neib_index = Get_index(array, neib_array[++j], numobs);
                    }
                    else fprintf(fp_out, " %lE", ZERO);
                if((i+1)%5 == 0 && (i+1) != numobs) fprintf(fp_out, "\n");
                } /* end for i */
                fprintf(fp_out, "\n");
                poly_array[0] = poly_array[numneibs];
                neib_array[0] = neib_array[numneibs];
                wt_array[0] = wt_array[numneibs];
                numneibs = 0, rows++;
                if(result==EOF) done_flag = 1;
            } /* end if */
        } /* end while */
    } /* end else */
    return(0);
} /* end main program */
/***************************************************************************/
Get_index(array, item, numobs) /* returns index number of 'item' in 'array' */
/***************************************************************************/
double *array, item;
int numobs;
{
    double *item_ptr; /* pointer returned by bsearch() */
    int index; /* return value */
    int Compare();

    /* get a pointer to the desired item in the array */
    if((item_ptr = (double *) bsearch(&item, array, numobs, sizeof(double),
        Compare)) == NULL) return -1;

    /* subtract pointers to find the position number of item in array */
    index = item_ptr - array;

    return index;
}
/****************************************************************************/
Compare(elem1, elem2) /* compare function for use with bsearch() */
/****************************************************************************/
double *elem1, *elem2;
{
    if (*elem1 < *elem2) return(-1);
    else if (*elem1 > *elem2) return(1);
    else return(0);
}
/****************************************************************************/
File_error(string)
/****************************************************************************/
char *string; /* name of file which failed to open */
{
    printf("ERROR opening the file: '%s'.\n",string);
    exit(-1);
}
/****************************************************************************/
Mem_error()
/****************************************************************************/
{
    printf("\nMemory allocation error.\n");
    exit(1);
}
/****************************************************************************
  GET_PREC - reads first numeric item from current position of file pointer.
Returns the number of places right of the decimal point or 0 for integers.
****************************************************************************/
int Get_prec(fp)
FILE *fp; /* file pointer */
{
    int i=0, length=0, precision=0;
    char c, firstitem[50];

    while(isspace(c = getc(fp))) ; /* skip beginning white space */
    ungetc(c,fp);

    while(!(isspace(c = getc(fp)))) /* read number at current file position */
        firstitem[i++] = c;
    length = i; /* total length of item */
    for(i=0;i<length;i++) {
        if (firstitem[i] == '.') {
            if (i==(length-1)) return(0); /* decimal point w/ no precision */
            precision = (length - (i+1));
            return(precision); /* return # places right of decimal point */
        } /* end if */
    } /* end for */
    return(0); /* integer data type */
} /* end function */
/****************************************************************************
  GET_FORMAT - determines whether input file is in binary or general
format by examining the number of numeric items on the 2nd line of the file.
(2 items = binary format; 3 items = general format.  Returns 1 for binary,
2 for general format.
****************************************************************************/
Get_format(fp)
FILE *fp; /* file pointer */
{
    int items; /* number of items read from file */
    double a, b, c; /* items read from file */
    char string[80], *result;

    result = fgets(string, 80, fp); /* skip up to first newline */
    if((result = fgets(string, 80, fp)) == NULL) {
        printf("\nERROR:  bad format found in line 2 of input file.\n");
        exit(-1);
    }
    items = sscanf(string, "%f %f %f", &a, &b, &c);
    if(items == 2 || items == 3) return(items - 1);
    else {
        printf("\nERROR:  bad format found in line 2 of input file.\n");
        exit(-1);
    }
}
```

INF2SMF.C

INF2SMF.C - converts output from CONWEIGHT.AML to sparse matrix format

Synopsis

This program works in conjunction with the Arc/Info macro CONWEIGHT.AML. The program converts the raw output from CONWEIGHT.AML into sparse matrix format. CONWEIGHT.AML calls INF2SMF automatically, so after compiling and installation, the execution of INF2SMF should be transparent to the user. INF2SMF may, however, be run interactively if necessary. In order to be called automatically by the CONWEIGHT.AML routine, the executable file "inf2smf" must be in the current directory or accessible on the Unix search path.

Usage

Command syntax is:

inf2smf <format>

where <format> consists of the number 0, 1, or 2. This number matches the option number which was chosen when CONWEIGHT.AML was executed:

0 - Binary queen's case format

1 - Binary rook's case format

2 - General rook's case format

See the documenation for CONWEIGHT.AML for a description of these formats.

Input

The program reads a file called **arcnsp** from the INFO subdirectory of the current workspace directory (./info/arcnsp). The **arcnsp** file is an ASCII file with three columns which denote polygon-IDs, neighbor-IDs, and weights, respectively. In the case of queen's case format files, the latter portion of the arcnsp file may contain three columns in the following format: node identifier (expressed as a negative number), polygon-ID, neighbor-ID. Records of this format will only exist if the associated Arc/Info coverage has nodes with valency greater than three.

Output

The program output is an ASCII file called **con_mat.asc**, which is in sparse binary or general matrix format, depending on the value of the <format> argument.

Notes

- The maximum allowable node valency is 200. This value may be altered by changing the value for the MAX_VALENCE constant and then re-compiling the program.
- The program creates and later removes a temporary file called **tempdump.swm**. If this file already exists, it will be deleted.

Source code

```
/******************************************************************************
    INF2SMF.C - this program reads the file ./info/arcnsp, which is a
```

```
table of polygon contiguity information written by the Arc/Info macro
CONWEIGHT.AML.  This information is put in sparse matrix
format.  The program requires an argument of 0, 1, or 2, depending
on the format of the input file.  The output file is called "con_mat.asc".

By Rusty Dodson, NCGIA--Santa Barbara, Oct., 1991. (REVISED Mar., 1992)
***************************************************************************/
#include <stdio.h>
#include <ctype.h>
#define MAX_VALENCE 200 /* maximum number of arc records for one node */

        /* global structure definition */
        typedef struct /* this structure holds one poly/neighbor/weight set */
        {
            float   poly; /* the target polygon */
            float   neib; /* the target's neighbor */
            float   weight; /* optional weight value */
        } RECORD;

main(argc, argv)
int argc;
char *argv[];
{
        RECORD  *neiblist, *newlist, *queen_list; /* arrays of RECORDS */
        FILE    *fp, /* info file of polygon neighbors */
                *tempfile; /* used to dump queen's case permutations */
        float   *vector; /* to hold values for queen's case */
        float   last_node; /* last node index used during queen's case */
        int     Thin_list(); /* removes redundancies from list of RECORDS */
        int     newsize=0;  /* # of records in a list */
        int     i=0, j, k, m, n;     /* loop counters */
        int     numneibs;            /* total number of neighbors for a poly */
        int     current;             /* current position in list */
        int     matrix_type=0;       /* 0, 1, or 2.  see Usage below */
        int     Find_dtype();        /* finds data type of the poly ID's */
        int     precision=0;         /* floating point precision of poly ID's */
        int     Find_size();         /* determines size of arcnsp file */
        int     lines;               /* number of lines in arcnsp file */
        int     Comp(), Compare(); /* compare functions for sorting */

        if(argc != 2) {
            printf("Usage:  'inf2smf n' (n = 0, 1, or 2).\n");
            printf("  0:  binary queen's contiguity\n");
            printf("  1:  binary rook's contiguity\n");
            printf("  2:  general rook's contiguity (boundary length)\n");
            exit(1);
        }
        matrix_type = argv[1][0] - 48; /* converts ASCII# to int */
        if(matrix_type < 0 || matrix_type > 2)
            printf("Error:  Argument must be 0, 1, or 2.\n"), exit(1);

        if ((fp = (fopen("info/arcnsp", "r+"))) == NULL)
            printf("\n Error opening the info/arcnsp file.\n"), exit(1);

        lines = Find_size(fp), rewind(fp);       /* find # of lines in file */

        /* allocate memory for the RECORD lists */
        neiblist = (RECORD *)malloc(lines * sizeof(RECORD));
        newlist = (RECORD *)malloc(lines * sizeof(RECORD));
        vector = (float *)malloc(MAX_VALENCE * sizeof(float));

        precision = Find_dtype(fp); /* get datatype of poly ID numbers */
        rewind(fp);                 /* (prec. = 0 for integers) */

        /* read the datafile into neiblist */
        for(i=0; i<lines; i++)
           current = fscanf(fp, "%f %f %f",&neiblist[i].poly,&neiblist[i].neib,
                     &neiblist[i].weight);
        fclose(fp); /* done with info output file */

        /* thin out any redundant poly/neighbor records */
        newsize = Thin_list(&neiblist[0], &newlist[0], lines);

/*** QUEEN'S CASE ***********************************************************/
        if(matrix_type == 0) { /* if queen's case contiguity */

            /* open a temporary file and dump newlist into it */
            tempfile = fopen("tempdump.swm", "w+");
            for(i=0; i<newsize; i++)
                fprintf(tempfile, "%f %f\n", newlist[i].poly,newlist[i].neib);
            /* skip to queen's case part of neiblist (where "poly" is neg.) */
            for(i=0; neiblist[i].poly > 0.0; i++);

            /* now deal with the node-adjacent information */
            while(i < lines) {
                k = 0;
                last_node = neiblist[i].poly;

                /* put poly-id's into vector */
                while(neiblist[i].poly == last_node){ /* while same node */
                    vector[k++] = neiblist[i].neib;
                    vector[k++] = neiblist[i].weight;
                    i++;
                }

                /* sort and thin the vector */
                qsort(&vector[0], k, sizeof(float), Comp);
                for (m=0, n=1; n<k; ) {
                    while(vector[m] == vector[n]) n++;
                    vector[++m] = vector[n];
                }
                k = m;

                /* write the permutations of vector's contents to file */
                for(m=0; m<(k-1); m++) {
                    for(n=(m+1); n<k; n++) {
                        if (vector[m] != vector[n]) {
                          fprintf(tempfile, "%f %f\n", vector[m], vector[n]);
                          fprintf(tempfile, "%f %f\n", vector[n], vector[m]);
                        }
                    } /* end for(n) */
                } /* end for(m) */
            } /* end while(i) */

            /* get size of temp file and allocate new array based on size */
            rewind(tempfile);
            lines = Find_size(tempfile);
            rewind(tempfile);
            queen_list = (RECORD *)malloc(lines * sizeof(RECORD));

            /* read tempfile into queen_list */
            for( i=0; i<lines; i++ )
                fscanf(tempfile, "%f %f", &queen_list[i].poly,
                  &queen_list[i].neib);
            fclose(tempfile);
            remove("tempdump.swm");

            /* sort and thin the queen_list array to the newlist array */
            qsort( &queen_list[0], lines, sizeof(RECORD), Compare);
            newsize = Thin_list(&queen_list[0], &newlist[0], lines);

        } /***** END QUEEN'S CASE BLOCK *****/
/******************************************************************************/

        /* find and write # of observations to the output file */
        fp = fopen("con_mat.asc", "w");
        for(i=0, j=0; i<(newsize-1); i++) {
            if(newlist[i].poly == newlist[i+1].poly) continue;
            else j++;
        }
        if(newlist[newsize].poly != newlist[newsize - 1].poly) j++;
        fprintf (fp, "%d\n", j);

        for(i=0, j=1; i<newsize; i++)
        {
            if (matrix_type == 2) {  /* general matrix format */
              fprintf (fp, "%.*f %.*f %.4f\n", precision, newlist[i].poly,
                   precision, newlist[i].neib, newlist[i].weight);
            }
            else { /* binary matrix format */
              numneibs = 0, current = i;
              while (newlist[++i].poly == newlist[current].poly)
                numneibs++;
              fprintf (fp, "%.*f %d\n", precision, newlist[current].poly,
                (numneibs+1) );
              for (j=current; j<=(current + numneibs); j++) {
                fprintf(fp, "%.*f ", precision, newlist[j].neib);
              }
              fprintf(fp, "\n");
              i--;
            }
        }
        fclose(fp);
        printf("Done.  Output file is called 'con_mat.asc'.\n");
} /* end main program */
/************************************************************************
 Find_size - this function returns the number of lines in the arcnsp data
        file.  This information is passed to malloc.
************************************************************************/
int Find_size(fp)
FILE *fp;
{
  int c, lines = 0;

  while ((c = getc(fp)) != EOF)
    if(c == '\n') lines++;
  return(lines);
} /* end function */
/************************************************************************
 Find_dtype - determines the data type of the polygon ID numbers by examining
        the first number in the info output file.  This format is assumed to
        hold for all polygon ID numbers in the file.  Returns the number
        of places right of the decimal point or 0 for integers.
************************************************************************/
int Find_dtype(fp)
FILE *fp;
{
 int i=0, length=0, precision=0;
 char c, firstitem[50];

 while(isspace(c = getc(fp))) ; /* skip beginning white space */
 ungetc(c,fp);
 while(!(isspace(c = getc(fp)))) /* read first number in file */
    firstitem[i++] = c;
 length = i; /* total length of item */
 for(i=0;i<length;i++) {
    if (firstitem[i] == '.') {
        if (i==(length-1)) return(0); /* decimal point w/ no precision */
        precision = (length - (i+1));
        return(precision); /* return # places right of decimal point */
    }
 }
 return(0); /* integer data type */
} /* end function */
/****************************************************************************
COMPARE FUNCTIONS - these are passed to qsort
****************************************************************************/
Comp(elem1, elem2) /* compares elements of a simple array */
int *elem1, *elem2;
{
        if (*elem1 < *elem2) return(-1);
        else if (*elem1 > *elem2) return(1);
        else return(0);
}
Compare(elem1, elem2) /* compares elements in poly/neib structures */
RECORD *elem1, *elem2;
{
        if (elem1->poly < elem2->poly) return(-1);
        else if (elem1->poly > elem2->poly) return(1);
        else
        {
            if (elem1->neib < elem2->neib) return(-1);
            else if (elem1->neib > elem2->neib) return(1);
            else return(0);
        }
}
/****************************************************************************
THIN_LIST - reads records from list1, removes redundancies (summing
        weight values when necessary), and writes non-redundant
        information to list2.  Returns number of records in list2.
****************************************************************************/
Thin_list(list1, list2, size)
RECORD *list1, *list2; /* old list, new list */
int    size;           /* number of elements in old list */
{
        int i, j;

        /* thin out any redundant poly/neighbor records */
        for (j=0,i=0; j<size; j++)
        {
            if(list1[j].poly < 0.0) break;
            if ((list1[j].poly == list1[j+1].poly) &&
                (list1[j].neib == list1[j+1].neib) &&
                (list1[j].poly != 0) && (list1[j].neib != 0))
            {
                list1[j+1].weight += list1[j].weight;
                continue;
            }
            if ((list1[j].poly != 0) && (list1[j].neib != 0))
                list2[i++] = list1[j];
        }
        return i; /* this is the size of the "thinned" list */
}
```

CONWEIGHT.AML

```
/*
/* CONWEIGHT.AML - dumps an ascii file (info/arcnsp) which contains the
/*  information necessary to create a contiguity-based spatial weight
/*  matrix.  An associated C-program (inf2smf) is used to convert this file into
/*  sparse matrix format.
/*
/*  By Rusty dodson, NCGIA Santa Barbara, Feb, 1992
/*
/*  Portions of the info code below (for making the .PNL, .NAL, and
/*  .VAL files) were provided by Dale Honeycutt of ESRI.
/*
&args cov item
/* set args to upper case
&s cov = [translate %cov%]
&s item = [translate %item%]
/*
/* check for bad input:
&if [null %cov%] or [null %item%] &then &do
  &type Usage: CONWEIGHT <cover> <item>
  &return
&end
&if not [exists %cov% -coverage] &then &do
  &type Error:  The coverage "%cov%" does not exist.
  &return
&end
&if not [exists %cov% -line] or not [exists %cov% -poly] &then &do
  &type Error:  %cov% must have polygon and line topology.
  &type Use the BUILD command.
  &return
&end
&if not [iteminfo %cov%.PAT -info %item% -exists] &then &do
  &type Error:  The item "%item%" does not exist in "%cov%.PAT".
  &return
&end
/*
&type '\  ============================================================='
&type '        SELECT THE DESIRED TYPE OF SPATIAL WEIGHTS MATRIX:        '
&type '  ============================================================='
&type '  Contiguity type:                                         Press:'
&type '  ---------------                                          -----'
&type '  Binary Queen''s Case. . . . . . . . . . . . . . . . . . . . 0'
&type '  Binary Rook''s Case . . . . . . . . . . . . . . . . . . . . 1'
&type '  General Rook''s case based on common boundary length'
&type '    (weight = %(common boundary length/polygon perimeter)) . . 2\'
&s matrix [response 'Enter your selection (default is 0)' 0]
/*
/* check for valid user input, default to 0 if necessary
&if [type %matrix%] ne -1 &then &s matrix 3
&if %matrix% < 0 or %matrix% > 2 &then &do
  &s matrix 0
  &type 'Invalid selection.  Defaulting to binary queen''s case matrix.'
&end
/*
/* delete pre-existing info files which will be created below
&if [exists %cov%.PNL -INFO] &then &do
  &if [delete %cov%.PNL -INFO] ^= 0 &then &do
    &type Could not delete %cov%.PNL
    &return
  &end
&end
&if [exists %cov%.NAL -INFO] &then &do
  &if [delete %cov%.NAL -INFO] ^= 0 &then &do
    &type Could not delete %cov%.NAL
    &return
  &end
&end
&if [exists %cov%.VAL -INFO] &then &do
```

```
    &if [delete %cov%.VAL -INFO] ^= 0 &then &do
      &type Could not delete %cov%.VAL
      &return
    &end
  &end
  &if [exists %cov%.HVN -INFO] &then &do
    &if [delete %cov%.HVN -INFO] ^= 0 &then &do
      &type Could not delete %cov%.HVN
      &return
    &end
  &end
  /*
  /* go into INFO and whip it up
  &data ARC INFO
  ARC
  CALC $COMMA-SWITCH = -1
  REM ##### Build a polygon neighbor list (cov.PNL) #####
  DEFINE %cov%.PNL
  POLY#,4,5,B
  NEIGHBOR#,4,5,B
  WEIGHT,4,8,F,4
  [Unquote ' ']
  REM ### the above line sends a carriage return to INFO
  REDEFINE
  1,LPOLY#,4,5,B
  1,RPOLY#,4,5,B
  1,%cov%#,4,5,B
  [Unquote ' ']
  SEL %cov%.PAT
  ALTER %cov%#,,,,,NEIGHBOR#,,,
  SEL %cov%.AAT RO
  REL %cov%.PNL 1 BY LPOLY# APPEND
  CALC $1NEIGHBOR# = RPOLY#
  CALC $1WEIGHT = LENGTH
  REL %cov%.PNL 1 BY RPOLY# APPEND
  CALC $1WEIGHT = LENGTH
  CALC $1NEIGHBOR# = LPOLY#
  SEL %cov%.PNL
  REL %cov%.PAT 1 BY %cov%# ORDERED
  REL %cov%.PAT 2 BY NEIGHBOR# ORDERED
  REM reselect out the unwanted polygons
  RESEL POLY# NE 1 AND NEIGHBOR# NE 1
  RESEL $1%item% NE 0 AND $2%item% NE 0
  REM
  REM ##### This line establishes the type of boundary length weight.
  REM ##### It can be modified in several ways.
  CALC WEIGHT = ( WEIGHT / $1PERIMETER ) * 100
  REM
  DISPLAY 'Sorting'
  SORT ON $1%item%, $2%item%
  PRINT $1%item%, $2%item%, WEIGHT
  &if %matrix% = 1 or %matrix% = 2 &then &goto done
  REM
  REM ##### the following sections are only used with the queen's case option
  REM
  REM ##### Build a node arc list (cov.NAL) #####
  DEF %cov%.NAL
  NODE#,4,5,B,0
  %cov%#,4,5,B,0
  [Unquote ' ']
  REDEFINE
  1,FNODE#,4,5,B,0
  [Unquote ' ']
  ALTER FNODE#
  ,,,,,,TNODE#,,,,,
  ALTER %cov%#
  ,,,,ARC#,,,,,,,,,,
  SEL %cov%.AAT
  REL %cov%.NAL 1 BY FNODE# APPEND NUMERIC
  CALC $1ARC# = %cov%#
  REL %cov%.NAL 1 BY TNODE# APPEND NUMERIC
  CALC $1ARC# = %cov%#
  SEL %cov%.NAL
  SORT NODE#,ARC#
  REM
  REM ##### Build a valency table (cov.VAL) #####
  DEF %cov%.VAL
  NODE#,4,5,B,0
  VALENCE,2,5,B,0
  [Unquote ' ']
  REDEFINE
```

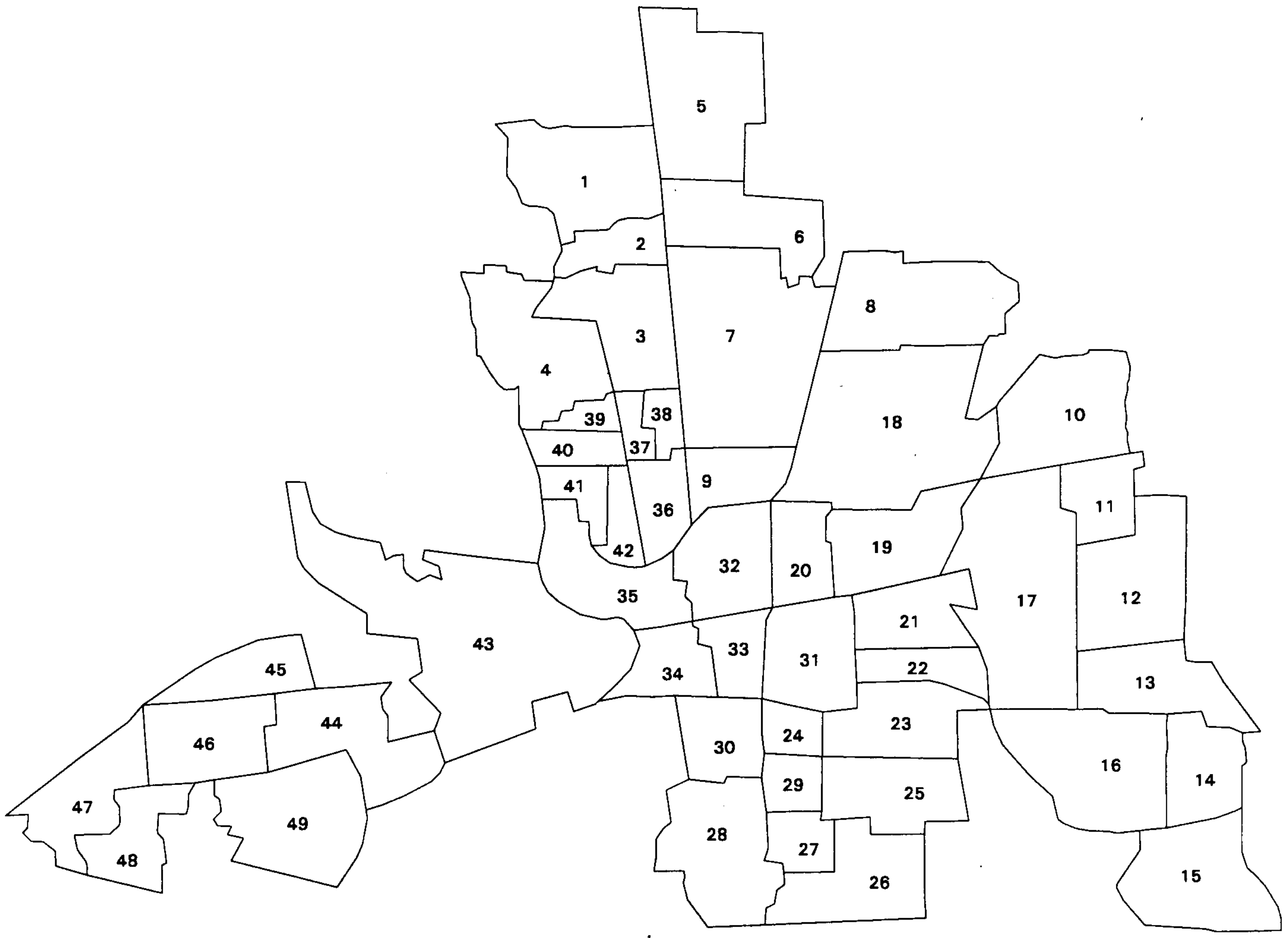
```
1,FNODE#,4,5,B,0
[Unquote ' ']
ALTER FNODE#
,,,,,,TNODE#,,,,,
SEL %cov%.NAL
REL %cov%.VAL 1 BY NODE# SUMMARY NUMERIC
CALC $NUM1 = $NUM1
SEL %cov%.VAL
CALC VALENCE = 0
SEL %cov%.NAL
REL %cov%.VAL 1 BY NODE# SEQUENTIAL NUMERIC
CALC $1VALENCE = $1VALENCE + 1
REM
REM ##### Build a high valency node table (cov.HVN) #####
DEFINE %cov%.HVN
NODE#,4,5,B
%COV%#,4,5,B
NEIGHBOR#,4,5,B
[Unquote ' ']
SEL %COV%.NAL
REL %COV%.VAL 1 BY NODE#
RESEL $1VALENCE > 3
REL %COV%.AAT 1 BY %COV%#
REL %COV%.HVN 2 BY NODE# APPEND
CALC $2%COV%# = $1LPOLY#
CALC $2NEIGHBOR# = $1RPOLY#
SEL %COV%.HVN
REL %COV%.PAT 1 BY %COV%# ORDERED
REL %COV%.PAT 2 BY NEIGHBOR# ORDERED
CALC NODE# = NODE# * -1
RESEL $1%ITEM% = 0 AND $2%ITEM% = 0
NSEL
PRINT NODE#, $1%ITEM%, $2%ITEM%
REM
&label done
QUIT
STOP
&end /* end &data section
&type 'Converting the ./info/arcnsp file to sparse matrix format...'
&type '  The executable program "inf2smf" must be available...'
&system inf2smf %matrix%
```

DISWEIGHT.AML

```
/* DISWEIGHT - creates a distance-based spatial weight matrix.
/*
/*   This macro uses an existing point or poly coverage with a .PAT.
/*  NOTE:  for polygon coverages, label points are used to represent polygon
/*  locations.  To use true geographic centroids, pre-process the coverage
/*  in ARC with the CENTROIDLABELS command.
/*
/*   R. Dodson, August, 1992
/*
&args cov item distance decay
/* set args to upper-case
&s cov = [translate %cov%]
&s item = [translate %item%]
/*
/* check for bad input
&if [null %cov%] or [null %item%] or [null %distance%] &then
    &return Usage: DISWEIGHT <cover> <item> <distance> {decay}
&if not [exists %cov% -poly] and not [exists %cov% -point] &then
   &return Error:  "%cov%" must exist and must have a .PAT file.
&if not [iteminfo %cov%.PAT -info %item% -exists] &then
  &return Error:  The item "%item%" does not exist in "%cov%.PAT".
&if [type %distance%] ^= -1 and [type %distance%] ^= -2 &then
  &return Error:  <distance> must be numeric.
&if [null %decay%] &then &s decay = 1
&else &if [type %decay%] ^= -1 and [type %decay%] ^= -2 &then
  &return Error:  {decay} must be numeric.
&if [exists CENT_DIST -info] &then
  &return The INFO file CENT_DIST exists.  Please delete or rename it.
/*
/* calculate distances between points
pointdistance %cov% %cov% cent_dist %distance%
/* go into INFO, modify the cent_dist file, relate to .PAT, and make
/*   the spatial weight matrix
&data ARC INFO
ARC
REM *** Suppress commas in output
CALC $COMMA-SWITCH = -1
```

```
REM *** .PAT needs this alternate item name:
SEL %cov%.PAT
ALTER %cov%#,,,,,NEIGHBOR#,,,
REM *** Now write # of observations to the output file
RESEL %item% NE 0 AND %cov%# NE 1
OUTPUT ../DIS_MAT.ASC INIT
PRINT $NOSEL
REM *** These alternate names are needed to relate to the .PAT
SEL CENT_DIST
ALTER %cov%#A,,,,,%cov%#,,,
ALTER %cov%#B,,,,,NEIGHBOR#,,,
REM *** Add more precision to the DISTANCE item
ALTER DISTANCE
,20,,15,,,,,
REM *** These calcs make the # items line up with the <cov># item in <cov>.PAT
CALC %cov%#A = %cov%#A + 1
CALC %cov%#B = %cov%#B + 1
REM
REM *** Now relate to the PAT, make the weight, and dump the file
RELATE %cov%.PAT 1 BY %cov%#
RELATE %cov%.PAT 2 BY NEIGHBOR#
REM
REM *** Assign the weight ****************************************************
RESEL DISTANCE > 0.0
CALC DISTANCE =  1 / ( DISTANCE ** %decay% )
REM *** line below is an example of distance weighted by poly areas.
REM CALC DISTANCE = ( 1 / DISTANCE ) * ( $2AREA / $1AREA ) * 10000
REM ***************************************************************************
ASEL
RESEL $1%item% NE 0 AND $2%item% NE 0
REM
SORT ON $1%item%, $2%item%
PRINT $1%item%, $2%item%, DISTANCE
QUIT
STOP
&end
&type 'Deleting temporary INFO file CENT_DIST...'
&if [delete CENT_DIST -info] ^= 0 &then &type Could not delete the INFO file ~
 'CENT_DIST'
&type Done.  The distance weight matrix file is "dis_mat.asc".
&return
```

APPENDIX 5: MAP OF COLUMBUS, OHIO NEIGHBORHOODS



Columbus, Ohio neighborhoods