The Stata Journal (2013)
13, Number 2, pp. 242–286

Creating and managing spatial-weighting matrices with the spmat command

David M. Drukker
StataCorp
College Station, TX
ddrukker@stata.com

Hua Peng
StataCorp
College Station, TX
hpeng@stata.com

Ingmar R. Prucha
Department of Economics
University of Maryland
College Park, MD
prucha@econ.umd.edu

Rafal Raciborski
StataCorp
College Station, TX
rraciborski@stata.com

Abstract. We present the `spmat` command for creating, managing, and storing spatial-weighting matrices, which are used to model interactions between spatial or more generally cross-sectional units. `spmat` can store spatial-weighting matrices in a general and banded form. We illustrate the use of the `spmat` command and discuss some of the underlying issues by using United States county and postal-code-level data.

Keywords: st0292, spmat, spatial-autoregressive models, Cliff–Ord models, spatial lag, spatial-weighting matrix, spatial econometrics, spatial statistics, cross-sectional interaction models, social-interaction models

1 Introduction

Building on Whittle (1954), Cliff and Ord (1973, 1981) developed statistical models that not only accommodate forms of cross-unit correlation but also allow for explicit forms of cross-unit interactions. The latter is a feature of interest in many social science, biostatistical, and geographic science models. Following Cliff and Ord (1973, 1981), much of the original literature was developed to handle spatial interactions. However, space is not restricted to geographic space, and many recent applications use spatial techniques in other situations of cross-unit interactions, such as social-interaction models and network models; see, for example, Kelejian and Prucha (2010) and Drukker, Egger, and Prucha (2013) for references. Much of the nomenclature still includes the adjective "spatial", and we continue this tradition to avoid confusion while noting the wider applicability of these models. For texts and reviews, see, for example, Anselin (1988, 2010), Arbia (2006), Cressie (1993), Haining (2003), and LeSage and Pace (2009).

The models derived and discussed in the literature cited above model cross-unit interactions and correlation in terms of spatial lags, which may involve the dependent variable, the exogenous variables, and the disturbances. A spatial lag of a variable is

 st0292

defined as a weighted average of observations on the variable over neighboring units. To illustrate, we after the rudimentary spatial-autoregressive (SAR) model

$$y_i = \lambda \sum_{j=1}^{n} w_{ij} y_j + \varepsilon_i, \qquad i = 1, \dots, n$$

where y_i denotes the dependent variable corresponding to unit i, the w_{ij} (with $w_{ii} = 0$) are nonstochastic weights, ε_i is a disturbance term, and λ is a parameter. In the above model, the y_i are determined simultaneously. The weighted average $\sum_{j=1}^{n} w_{ij} y_j$, on the right-hand side, is called a spatial lag, and the w_{ij} are called the spatial weights. It often proves convenient to write the model in matrix notation as

$$\mathbf{y} = \lambda \mathbf{W}\mathbf{y} + \varepsilon$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \qquad \mathbf{W} = \begin{bmatrix} 0 & w_{12} & \cdots & w_{1,n-1} & w_{1n} \\ w_{21} & 0 & \dots & w_{2,n-1} & w_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{n-1,1} & w_{n-1,2} & \cdots & 0 & w_{n-1,n} \\ w_{n1} & w_{n2} & \cdots & w_{n,n-1} & 0 \end{bmatrix}, \qquad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Again the $n \times 1$ vector $\mathbf{Wy}$ is typically referred to as the spatial lag in $\mathbf{y}$ and the $n \times n$ matrix $\mathbf{W}$ as the spatial-weighting matrix. More generally, as indicated above, the concept of a spatial lag can be applied to any variable, including exogenous variables and disturbances, which—as can be seen in the literature cited above—provides for a fairly general class of Cliff–Ord types of models.

Spatial-weighting matrices allow us to conveniently implement Tobler's first law of geography—"everything is related to everything else, but near things are more related than distant things" (Tobler 1970, 236)—which applies whether the space is geographic, biological, or social. The `spmat` command creates, imports, manipulates, and saves $\mathbf{W}$ matrices. The matrices are stored in a spatial-weighting matrix object (`spmat` object). The `spmat` object contains additional information about a spatial-weighting matrix, such as the identification codes of the cross-section units, and other items discussed below.[1]

The generic syntax of `spmat` is

`spmat` *subcommand* ...

where each *subcommand* performs a specific task. Some subcommands create `spmat` objects from a Stata dataset (`contiguity`, `idistance`, `dta`), a Mata matrix (`putmatrix`), or a text file (`import`). Other subcommands save objects to a disk (`save`, `export`) or read them back in (`use`, `import`). Still other subcommands summarize spatial-weighting

1. We use the term "units" instead of "places" because spatial-econometric methods have been applied to many cases in which the units of analysis are individuals or firms instead of geographical places; for example, see Leenders (2002).

matrices (`summarize`); graph spatial-weighting matrices (`graph`); manage them (`note`, `drop`, `getmatrix`); and perform computations on them (`lag`, `eigenvalues`). The remaining subcommands are used to change the storage format of the spatial-weighting matrices inside the `spmat` objects. As discussed below, matrices stored inside `spmat` objects can be general or banded, with general matrices occupying much more space than banded ones. The subcommand `permute` rearranges the matrix elements, and the subcommand `tobanded` is used to store a matrix in banded form.

`spmat contiguity` and `spmat idistance` create the frequently used inverse-distance and contiguity spatial-weighting matrices; Haining (2003, 83) and Cliff and Ord (1981, 17) discuss typical formulations of weights matrices. The import and management capabilities allow users to create spatial-weighting matrices beyond contiguity and inverse-distance matrices. Section 17.4 provides some discussion and examples.

Drukker, Prucha, and Raciborski (2013a, 2013b) discuss Stata commands that implement estimators for SAR models. These commands use spatial-weighting matrices previously created by the `spmat` command discussed in this article.

Before we describe individual subcommands in detail, we illustrate how to obtain and transform geospatial data into the format required by `spmat`, and we address computational problems pertinent to spatial-weighting matrices.

1.1 From shapefiles into Stata format

Many applications use geospatial data frequently made available in the form of shapefiles. Each shapefile is a pair of files: the database file and the coordinates file. The database file contains data on the attributes of the spatial units, while the coordinates file contains the geographical coordinates describing the boundaries of the spatial units. In the common case where the units correspond to nonzero areas instead of points, the boundary data in the coordinates file are stored as a series of irregular polygons.

The vast majority of geospatial data comes in the form of ESRI or MIF shapefiles.[2] There are user-written tools for translating shapefiles to Stata's `.dta` format and for mapping spatial data. `shp2dta` (Crow 2006) and `mif2dta` (Pisati 2005) translate ESRI and MIF shapefiles to Stata datasets. `shp2dta` and `mif2dta` translate the two files that make up a shapefile to two Stata `.dta` files. The database file is translated to the "attribute" `.dta` file, and the coordinates file is translated to the coordinates `.dta` file.[3,4]

2. Refer to http://www.esri.com for details about the ESRI format and to http://www.pbinsight.com for details about the MIF format. The ESRI format is much more common.
3. `shp2dta` and `mif2dta` save the coordinates data in the format required by `spmap` (Pisati 2007), which graphs data onto maps.
4. We use the term "attribute" instead of "database" because "database" does not adequately distinguish between attribute data and coordinates data.

The code below illustrates the use of `shp2dta` and `spmap` (Pisati 2007) on the county boundaries data for the continental United States; Crow and Gould (2007) provide a broader introduction to shapefiles, `shp2dta`, and `spmap`.

`shp2dta`, `mif2dta`, and `spmap` use a common set of conventions for defining the polygons in the coordinates data translated from the coordinates file. Crow and Gould (2007) discuss these conventions.

We downloaded `ts_2008_us_county00.db` and `ts_2008_us_county00.shp`, which are the attribute file and the coordinates file, respectively, and which make up the shapefile for U.S. counties from the U.S. Census Bureau.[5] We begin by using `shp2dta` to translate these files to the files `county.dta` and `countyxy.dta`.

```
. shp2dta using tl_2008_us_county00, database(county)
> coordinates(countyxy) genid(id) gencentroids(c)
```

`county.dta` contains the attribute information from the attribute file in the shapefile, and `countyxy.dta` contains the coordinates data from the shapefile. The attribute dataset `county.dta` has one observation per county on variables such as county name and state code. Because we specified the option `gencentroids(c)`, `county.dta` also contains the variables `x_c` and `y_c`, which contain the coordinates of the county centroids, measured in degrees. (See the help file for `shp2dta` for details and the x–y naming convention.) `countyxy.dta` contains the coordinates of the county boundaries in the long-form panel format used by `spmap`.[6]

Below we use `use` to read `county.dta` into memory and use `destring` (see [D] **destring**) to create a new, numeric state-code variable `st` from the original string state-identifying variable `STATEFP`. Next we use `drop` to drop the observations defining the coordinates of county boundaries in Alaska, Hawaii, and U.S. territories. Finally, we use `rename` to rename the variables containing coordinates of the county centroids and use `save` to save our changes into the `county.dta` dataset file.

```
. use county

. quietly destring STATEFP, generate(st)

. *keep continental US counties
. drop if st==2 | st==15 | st>56
(123 observations deleted)

. rename x_c longitude

. rename y_c latitude

. save county, replace
file county.dta saved
```

Having completed the translation and selected our subsample, we use `spmap` to draw the map, given in figure 1, of the boundaries in the coordinates dataset.

5. Actually, we downloaded `ts_2008_us_county00.zip` from ftp://ftp2.census.gov/geo/tiger/TIGER2008/, and this `.zip` file contained the two files named in the text.
6. Crow and Gould (2007), the `shp2dta` help file, and the `spmap` help file provide more information about the input and output datasets.

```
. spmap using countyxy, id(id)
```

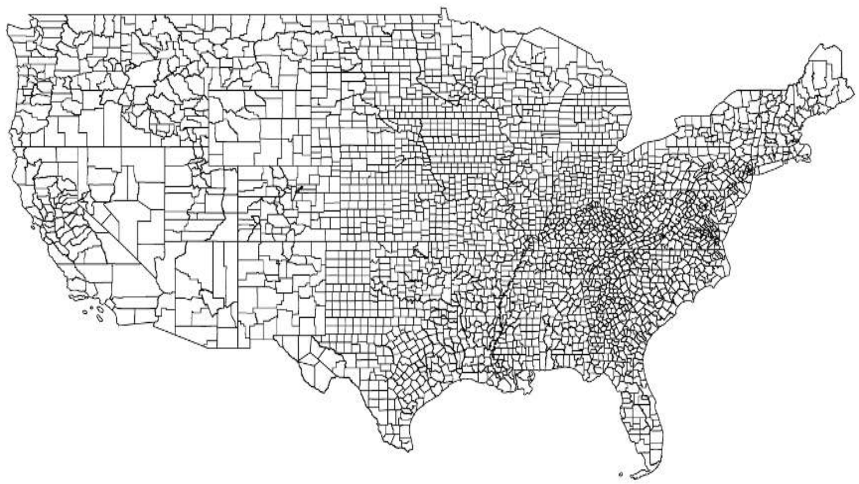

Figure 1. County boundaries for the continental United States, 2000

1.2 Memory considerations

The spatial-weighting matrix for the n units is an $n \times n$ matrix, which implies that memory requirements increase quadratically with data size. For example, a contiguity matrix for the 31,713 U.S. postal codes (five-digit zip codes) is a $31{,}713 \times 31{,}713$ matrix, which requires $31{,}713 \times 31{,}713 \times 8/2^{30} \approx 7.5$ gigabytes of storage space.

Many users do not have this much memory on their machines. However, it is usually possible to store spatial-weighting matrices more efficiently. Drukker et al. (2011) discuss how to judiciously reorder the observations so that many spatial-weighting matrices can be stored as banded matrices, thereby using less space than general matrices.

This subsection describes banded matrices and the potential benefits of using banded matrices for storing spatial-weighting matrices. If you do not have large datasets, you may skip this section and all future references to banded matrices.

A banded matrix is a matrix whose nonzero elements are confined to a diagonal band that comprises the main diagonal, zero or more diagonals above the main diagonal, and zero or more diagonals below the main diagonal. The number of diagonals above the main diagonal that contain nonzero elements is the upper bandwidth, say, b_U. The number of diagonals below the main diagonal that contain nonzero elements is the lower bandwidth, say, b_L. An example of a banded matrix having an upper bandwidth of 1 and a lower bandwidth of 2 is

$$\begin{bmatrix} \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{bmatrix}$$

We can save a lot of space by storing only the elements in the diagonal band because the elements outside the band are 0 by construction. Using this information, we can efficiently store this matrix without any loss of information as

$$\begin{bmatrix} 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \end{bmatrix}$$

The above matrix only contains the elements of the diagonals with nonzero elements. To store the elements in a rectangular array, we added zeros as necessary. The row dimension of the banded matrix is the upper bandwidth plus the lower bandwidth plus 1, or $b = b_U + b_L + 1$. We will use the $b \times n$ shorthand to refer to the dimensions of banded matrices.

Banded matrices require less storage space than general matrices. The `spmat` suite provides tools for creating, storing, and manipulating banded matrices. In addition, computing an operation on a banded matrix is much faster than on a general matrix.

Drukker et al. (2011) show that many spatial-weighting matrices have a banded structure after an appropriate reordering. In particular, a banded structure is often attained by sorting the data in an ascending order of the distance from a well-chosen place. In section 5, we will illustrate this method with data on U.S. counties and U.S. five-digit zip codes. In the case of U.S. five-digit zip codes, we show how to create a contiguity matrix with upper and lower bandwidths of 913. This allows us to store the data in a $1{,}827 \times 31{,}713$ matrix, which requires only $1{,}827 \times 31{,}713 \times 8/2^{30} \approx 0.43$ gigabytes instead of the 7.5 gigabytes required for the general matrix.

We are now ready to describe the `spmat` subcommands.

2 Creating a contiguity matrix from geospatial data

2.1 Syntax

`spmat contiguity` *objname* [*if*] [*in*] `using` *coordinates_file*`,` `id(`*varname*`)` [*options*]

options	Description
`normalize(`*norm*`)`	control the normalization method
`rook`	require that two units share a common border instead of just a common point to be neighbors
`banded`	store the matrix in the banded format
`replace`	replace existing `spmat` object
`saving(`*filename*[`, replace`]`)`	save the neighbor list to a file
`nomatrix`	suppress creations of `spmat` object
`tolerance(`#`)`	use when determining whether units share a common border

2.2 Description

`spmat contiguity` computes a contiguity or normalized-contiguity matrix from a coordinates dataset containing a polygon representation of geospatial data. More precisely, `spmat contiguity` constructs a contiguity matrix or normalized-contiguity matrix from the boundary information in a coordinates dataset and puts it into the new `spmat` object *objname*.

In a contiguity matrix, contiguous units are assigned weights of 1, and noncontiguous units are assigned weights of 0. Contiguous units are known as neighbors.

`spmat contiguity` uses the polygon data in *coordinates_file* to determine the neighbors of each unit. The *coordinates_file* must be a Stata dataset containing the polygon information in the format produced by `shp2dta` and `mif2dta`. Crow and Gould (2007) discuss the conventions used to represent polygons in the Stata datasets created by these commands.

2.3 Options

`id(`*varname*`)` specifies a numeric variable that contains a unique identifier for each observation. (`shp2dta` and `mif2dta` name this ID variable _ID.) `id()` is required.

`normalize(`*norm*`)` specifies one of the three available normalization techniques: `row`, `minmax`, and `spectral`. In a row-normalized matrix, each element in row i is divided by the sum of row i's elements. In a minmax-normalized matrix, each element is divided by the minimum of the largest row sum and column sum of the matrix. In a spectral-normalized matrix, each element is divided by the modulus of the largest eigenvalue of the matrix. See section 2.5 for details.

`rook` specifies that only units that share a common border be considered neighbors (edge or rook contiguity). The default is queen contiguity, which treats units that share a common border or a single common point as neighbors. Computing rook-contiguity matrices is more computationally intensive than the default queen-contiguity computation.[7]

`banded` requests that the new matrix be stored in a banded form. The banded matrix is constructed without creating the underlying $n \times n$ representation.

`replace` permits `spmat contiguity` to overwrite an existing `spmat` object.

`saving(`*filename*[`, replace`]`)` saves the neighbor list to a space-delimited text file. The first line of the file contains the number of units and, if applicable, bands; each remaining line lists a unit identification code followed by the identification codes of units that share a common border, if any. You can read the file back into an `spmat` object with `spmat import ..., nlist`. `replace` allows *filename* to be overwritten if it already exists.

`nomatrix` specifies that the `spmat` object *objname* and spatial-weighting matrix $\mathbf{W}$ not be created. In conjunction with `saving()`, this option allows for creating a text file containing a neighbor list without allocating space for the underlying contiguity matrix.

`tolerance(`#`)` specifies the numerical tolerance used in deciding whether two units are edge neighbors. The default is `tolerance(1e-7)`.

2.4 Examples

As discussed above, spatial-weighting matrices are used to compute weighted averages in which more weight is placed on nearby observations than on distant observations. While Haining (2003, 83) and Cliff and Ord (1981, 17) discuss formulations of weights matrices, contiguity and inverse-distance matrices are the two most common spatial-weighting matrices.

7. These definitions for rook neighbor and queen neighbor are commonly used; see, for example, Lai, So, and Chan (2009). (As many readers will recognize, the "rook" and "queen" terminology arises by analogy with chess, in which a rook may only move across sides of squares, whereas a queen may also move diagonally.)

In geospatial-type applications, researchers who want a contiguity matrix need to perform a series of complicated calculations on the boundary information in a coordinates dataset to identify the neighbors of each unit. `spmat contiguity` performs these calculations and stores the resulting weights matrix in an `spmat` object.

In contrast, some social-network datasets begin with a list of neighbors instead of the boundary information found in geospatial data. Section 15 discusses how to create a social-network matrix from a list of neighbors.

▷ **Example**

We continue the example from section 1.1 and assume both of the Stata datasets created in section 1.1 are in the current working directory. After loading the attribute dataset into memory, we create the `spmat` object `ccounty` containing a normalized-contiguity matrix for U.S. counties by typing

```
. use county, clear
. spmat contiguity ccounty using countyxy, id(id) normalize(minmax)
```

We use `spmat summarize`, discussed in section 4, to summarize the contents of the spatial-weighting matrix in the `ccounty` object we created above:

```
. spmat summarize ccounty, links
Summary of spatial-weighting object ccounty
```

Matrix	Description
Dimensions	3109 x 3109
Stored as	3109 x 3109
Links	
total	18474
min	1
mean	5.942104
max	14

The table shows basic information about the normalized contiguity matrix, including the dimensions of the matrix and its storage. The number of neighbors found is reported as 18,474, with each county having 6 neighbors on average.

◁

2.5 Normalization details

In this section, we present details about the normalization methods.[8] In each case, the normalized matrix $\widetilde{\mathbf{W}} = (\widetilde{w}_{ij})$ is computed from the underlying matrix $\mathbf{W} = (w_{ij})$, where the elements are assumed to be nonnegative; see, for example, Kelejian and Prucha (2010) for an introduction to the use and interpretation of these normalization methods.

8. The normalization methods are not restricted to contiguity matrices.

In a row-normalized matrix, the (i,j)th element of $\widetilde{\mathbf{W}}$ becomes $\widetilde{w}_{ij} = w_{ij}/r_i$, where r_i is the sum of the ith row of $\mathbf{W}$. After row normalization, each row of $\widetilde{\mathbf{W}}$ will sum to 1. Row normalizing a symmetric $\mathbf{W}$ produces an asymmetric $\widetilde{\mathbf{W}}$ except in very special cases. Kelejian and Prucha (2010) point out that normalizing by a vector of row sums needs to be guided by theory.

In a minmax-normalized matrix, the (i,j)th element of $\widetilde{\mathbf{W}}$ becomes $\widetilde{w}_{ij} = w_{ij}/m$, where $m = \min\{\max_i(r_i), \max_i(c_i)\}$, with $\max_i(r_i)$ being the largest row sum of $\mathbf{W}$ and $\max_i(c_i)$ being the largest column sum of $\mathbf{W}$. Normalizing by a scalar preserves symmetry and the basic model specification.

In a spectral-normalized matrix, the (i,j)th element of $\widetilde{\mathbf{W}}$ becomes $\widetilde{w}_{ij} = w_{ij}/v$, where v is the largest of the moduli of the eigenvalues of $\mathbf{W}$. As for the minmax norm, normalizing by a scalar preserves symmetry and the basic model specification.

3 Creating an inverse-distance matrix from data

3.1 Syntax

`spmat idistance` *objname cvarlist* [*if*] [*in*], `id(`*varname*`)` [*options*]

where *cvarlist* is the list of coordinate variables.

options	Description
`dfunction(`*function*[`, miles`]`)`	specify the distance function
`normalize(`*norm*`)`	specify the normalization method
truncmethod	specify the truncation method
`banded`	store the matrix in the banded format
`replace`	replace an existing `spmat` object

where *function* is one of `euclidean`, `rhaversine`, `dhaversine`, or p;
`miles` may only be specified with `rhaversine` or `dhaversine`; and
truncmethod is one of `btruncate(`b B`)`, `dtruncate(`d_L d_U`)`, or
`vtruncate(`v`)`.

3.2 Description

An inverse-distance spatial-weighting matrix is composed of weights that are inversely related to the distances between the units. `spmat idistance` uses the coordinate variables from the attribute data in memory and the specified distance measure to compute the distances between units, to create an inverse-distance spatial-weighting matrix, and to store the result in an `spmat` object.

3.3 Options

`id(`*varname*`)` specifies a numeric variable that contains a unique identifier for each observation. `id()` is required.

`dfunction(`*function*[`, miles`]`)` specifies the distance function. *function* may be one of `euclidean` (default), `dhaversine`, `rhaversine`, or the Minkowski distance of order p, where p is an integer greater than or equal to 1.

When the default `dfunction(euclidean)` is specified, a Euclidean distance measure is applied to the coordinate variable list *cvarlist*.

When `dfunction(rhaversine)` or `dfunction(dhaversine)` is specified, the haversine distance measure is applied to the two coordinate variables *cvarlist*. (The first coordinate variable must specify longitude, and the second coordinate variable must specify latitude.) The coordinates must be in radians when `rhaversine` is specified. The coordinates must be in degrees when `dhaversine` is specified. The haversine distance measure is calculated in kilometers by default. Specify `dfunction(rhaversine, miles)` or `dfunction(dhaversine, miles)` if you want the distance returned in miles.

When `dfunction(`p`)` (p is an integer) is specified, a Minkowski distance measure of order p is applied to the coordinate variable list *cvarlist*.

The formulas for the distance measure are discussed in section 3.5.

`normalize(`*norm*`)` specifies one of the three available normalization techniques: `row`, `minmax`, and `spectral`. In a row-normalized matrix, each element in row i is divided by the sum of row i's elements. In a minmax-normalized matrix, each element is divided by the minimum of the largest row sum and column sum of the matrix. In a spectral-normalized matrix, each element is divided by the modulus of the largest eigenvalue of the matrix. See section 2.5 for details.

truncmethod options specify one of the three truncation criteria. The values of the spatial-weighting matrix $\mathbf{W}$ that meet the truncation criterion will be changed to 0. Only apply truncation methods when supported by theory.

`btruncate(`b B`)` partitions the values of $\mathbf{W}$ into B equal-length bins and truncates to 0 entries that fall into bin b or below, $b < B$.

`dtruncate(`d_L d_U`)` truncates to 0 the values of $\mathbf{W}$ that fall more than d_L diagonals below and d_U diagonals above the main diagonal. Neither value can be greater than $\lfloor$`(cols(W)`$-$`1)`$/4\rfloor$.[9]

`vtruncate(`v`)` truncates to 0 the values of $\mathbf{W}$ that are less than or equal to v.

See section 3.6 for more details about the truncation options.

9. This limit ensures that a cross product of the spatial-weighting matrix is stored more efficiently in banded form than in general form. The limit is based on the cross product instead of the matrix itself because the generalized spatial two-stage least-squares estimators use cross products of the spatial-weighting matrices.

`banded` requests that the new matrix be stored in a banded form. The banded matrix is constructed without creating the underlying $n \times n$ representation. Note that without `banded`, a matrix with truncated values will still be stored in an $n \times n$ form.

`replace` permits `spmat idistance` to overwrite an existing `spmat` object.

3.4 Examples

As discussed above, spatial-weighting matrices are used to compute weighted averages in which more weight is placed on nearby observations than on distant observations. Haining (2003, 83) and Cliff and Ord (1981, 17) discuss formulations of weights matrices, contiguity matrices, inverse-distance matrices, and combinations thereof.

In inverse-distance spatial-weighting matrices, the weights are inversely related to the distances between the units. `spmat idistance` provides several measures for calculating the distances between the units.

The coordinates may or may not be geospatial. Distances between geospatial units are commonly computed from the latitudes and longitudes of unit centroids.[10] Social distances are frequently computed from individual-person attributes.

In much of the literature, the attributes are known as coordinates because the nomenclature has developed around the common geospatial case in which the attributes are map coordinates. For ease of use, `spmat idistance` follows this convention and refers to coordinates, even though coordinate variables specified in *cvarlist* need not be spatial coordinates.

The (i, j)th element of an inverse-distance spatial-weighting matrix is $1/d_{ij}$, where d_{ij} is the distance between unit i and j computed from the specified coordinates and distance measure. Creating spatial-weighting matrices with elements of the form $1/f(d_{ij})$, where $f(\cdot)$ is some function, is described in section 17.4.

▷ Example

`county.dta` from section 1.1 contains the coordinates of the centroids of each county, measured in degrees, in the variables `longitude` and `latitude`. To get a feel for the data, we create an unnormalized inverse-distance spatial-weighting matrix, store it in the `spmat` object `dcounty`, and summarize it by typing

10. The word "centroid" in the literature on geographic information systems differs from the standard term in geometry. In the geographic information systems literature, a centroid is a weighted average of the vertices of a polygon that approximates the center of the polygon; see Waller and Gotway (2004, 44–45) for the formula and some discussion.

```
. spmat idistance dcounty longitude latitude, id(id) dfunction(dhaversine)
. spmat summarize dcounty
Summary of spatial-weighting object dcounty
```

Matrix	Description
Dimensions	3109 x 3109
Stored as	3109 x 3109
Values	
min	0
min>0	.0002185
mean	.0012296
max	1.081453

From the summary table, we can see that the centroids of the two closest counties lie within less than one kilometer of each other (1/1.081453), while the two most distant counties are 4,577 kilometers apart (1/0.0002185).

Below we compute a minmax-normalized inverse-distance matrix, store it in the `spmat` object `dcounty2`, and summarize it by typing

```
. spmat idistance dcounty2 longitude latitude, id(id) dfunction(dhaversine)
> normalize(minmax)
. spmat summarize dcounty2
Summary of spatial-weighting object dcounty2
```

Matrix	Description
Dimensions	3109 x 3109
Stored as	3109 x 3109
Values	
min	0
min>0	.0000382
mean	.0002151
max	.1892189

◁

3.5 Distance calculation details

Specifying q variables in the list of coordinate variables *cvarlist* implies that the units are located in a q-dimensional space. This space may or may not be geospatial. Let the q variables in the list of coordinate variables *cvarlist* be $\mathtt{x}_1$, $\mathtt{x}_2$, ..., $\mathtt{x}_q$, and denote the coordinates of observation i by ($\mathtt{x}_1[i]$, $\mathtt{x}_2[i]$, ..., $\mathtt{x}_q[i]$).

The default behavior of `spmat idistance` is to calculate the Euclidean distance between units s and t, which is given by

$$d_{st} = \sqrt{\sum_{j=1}^{q} \left(\mathtt{x}_j[s] - \mathtt{x}_j[t]\right)^2}$$

for observations s and t.

The Minkowski distance of order p is given by

$$d_{st} = \sqrt[p]{\sum_{j=1}^{q} \left| \mathtt{x}_j\,\mathtt{[}s\mathtt{]} - \mathtt{x}_j\,\mathtt{[}t\mathtt{]} \right|^p}$$

for observations s and t. When $p = 2$, the Minkowski distance is equivalent to the Euclidean distance.

The haversine distance measure is useful when the units are located on the surface of the earth and the coordinate variables represent the geographical coordinates of the spatial units. In such cases, we usually wish to calculate a spherical (great-circle) distance between the spatial units. This is accomplished by the haversine formula given by

$$d_{st} = r \times c$$

where

r is the mean radius of the Earth ($6,371.009$ km or $3,958.761$ miles)

$c = 2 \arcsin\{\min(1, \sqrt{a})\}$

$a = \sin^2 \phi + \cos(\phi_1) \cos(\phi_2) \sin^2 \lambda$

$\phi = \frac{1}{2}(\phi_2 - \phi_1) = \frac{1}{2}(\mathtt{x}_2\,\mathtt{[}t\mathtt{]} - \mathtt{x}_2\,\mathtt{[}s\mathtt{]})$

$\lambda = \frac{1}{2}(\lambda_2 - \lambda_1) = \frac{1}{2}(\mathtt{x}_1\,\mathtt{[}t\mathtt{]} - \mathtt{x}_1\,\mathtt{[}s\mathtt{]})$

$\mathtt{x}_1\,\mathtt{[}s\mathtt{]}$ and $\mathtt{x}_1\,\mathtt{[}t\mathtt{]}$ are the longitudes of point s and point t, respectively

$\mathtt{x}_2\,\mathtt{[}s\mathtt{]}$ and $\mathtt{x}_2\,\mathtt{[}t\mathtt{]}$ are the latitudes of point s and point t, respectively

Specify `dfunction(dhaversine)` to compute haversine distances from coordinates in degrees, and specify `dfunction(rhaversine)` to compute haversine distances from coordinates in radians. Both `dfunction(dhaversine)` and `dfunction(rhaversine)` by default use r = 6,371.009 to compute results in kilometers. To compute haversine distances in miles, with r = 3,958.761, instead specify `dfunction(dhaversine, miles)` or `dfunction(rhaversine, miles)`.

3.6 Truncation details

Unlike contiguity matrices, inverse-distance matrices cannot naturally yield a banded structure because the off-diagonal elements are never exactly 0. Consider an example in which we have nine units arranged on the real line with x denoting the unit locations.

```
. use truncex, clear
. list
```

	id	x
1.	1	0
2.	2	1
3.	3	2
4.	4	503
5.	5	504
6.	6	505
7.	7	1006
8.	8	1007
9.	9	1008

The units are grouped into three clusters. The units belonging to the same cluster are close to one another, while the distance between the units belonging to different clusters is large. For real-world data, the units may represent, for example, cities in different states. We use `spmat idistance` to create the `spmat` object `ex` from the data:

```
. spmat idistance ex x, id(id)
```

The resulting spatial-weighting matrix of inverse distances is

$$\begin{bmatrix} \mathbf{0} & 1 & 0.5 & 0.00199 & 0.00198 & 0.00198 & 0.00099 & 0.00099 & 0.00099 \\ 1 & \mathbf{0} & 1 & 0.00199 & 0.00199 & 0.00198 & 0.001 & 0.00099 & 0.00099 \\ 0.5 & 1 & \mathbf{0} & 0.002 & 0.00199 & 0.00199 & 0.001 & 0.001 & 0.00099 \\ 0.00199 & 0.00199 & 0.002 & \mathbf{0} & 1 & 0.5 & 0.00199 & 0.00198 & 0.00198 \\ 0.00198 & 0.00199 & 0.00199 & 1 & \mathbf{0} & 1 & 0.00199 & 0.00199 & 0.00198 \\ 0.00198 & 0.00198 & 0.00199 & 0.5 & 1 & \mathbf{0} & 0.002 & 0.00199 & 0.00199 \\ 0.00099 & 0.001 & 0.001 & 0.00199 & 0.00199 & 0.002 & \mathbf{0} & 1 & 0.5 \\ 0.00099 & 0.00099 & 0.001 & 0.00198 & 0.00199 & 0.00199 & 1 & \mathbf{0} & 1 \\ 0.00099 & 0.00099 & 0.00099 & 0.00198 & 0.00198 & 0.00199 & 0.5 & 1 & \mathbf{0} \end{bmatrix}$$

Theoretical considerations may suggest that the weights should actually be 0 below a certain threshold. For example, choosing the threshold value of $1/500 = 0.002$ for our matrix results in the following structure:

$$\begin{bmatrix} \mathbf{0} & 1 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & \mathbf{0} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 1 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{0} & 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \mathbf{0} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 1 & \mathbf{0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & 1 & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{0} & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 1 & \mathbf{0} \end{bmatrix}$$

Now the matrix with the truncated values can be stored more efficiently in a banded form:

$$\begin{bmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 \end{bmatrix}$$

`spmat idistance` provides tools for truncating the values of an inverse-distance matrix and storing the truncated matrix in a banded form. Like `spmat contiguity`, `spmat idistance` is capable of creating a banded matrix without creating the underlying $n \times n$ representation of the matrix. The user must specify a theoretically justified truncation criterion for such an application.

Here we illustrate how one could apply each of the truncation methods mentioned in section 3.3 to our hypothetical inverse-distance matrix. The most natural way is to use value truncation. In the code below, we create a new `spmat` object `ex1` with the values of $\mathbf{W}$ that are less than or equal to 1/500 set to 0.[11] We also request that $\mathbf{W}$ be stored in banded form.

```
. spmat idistance ex1 x, id(id) banded vtruncate(1/500)
```

The same outcome can be achieved with bin truncation. In bin truncation, we find the maximum value in $\mathbf{W}$ denoted by m, divide the interval $(0, m]$ into B bins of equal length, and then truncate to 0 elements that fall into bins $1, \dots, b$; see *Bin truncation details* below for a more technical description. In our hypothetical matrix, the largest element of $\mathbf{W}$ is 1. If we divide the values in $\mathbf{W}$ into three bins, the bins will be defined by $(0, 1/3]$, $(1/3, 2/3]$, $(2/3, 1]$. The values we wish to round to 0 fall into the first bin.

11. `vtruncate()` accepts any expression that evaluates to a number.

In the code below, we create a new `spmat` object `ex2` with the values of $\mathbf{W}$ that fall into the first bin set to 0. We also request that $\mathbf{W}$ be stored in banded form.

```
. spmat idistance ex2 x, id(id) banded btruncate(1 3)
```

Diagonal truncation is not based on value comparison; therefore, in general, we will not be able to replicate exactly the results obtained with bin or value truncation. In the code below, we create a new `spmat` object `ex3` with the values of $\mathbf{W}$ that fall more than two diagonals below and above the main diagonal set to 0. We also request that $\mathbf{W}$ be stored in banded form.

```
. spmat idistance ex3 x, id(id) banded dtruncate(2 2)
```

The resulting matrix based on diagonal truncation is shown below. No values in $\mathbf{W}$ have been changed; instead, we copied the requested elements from $\mathbf{W}$ and stored them in banded form, padding the banded format with 0s when necessary (see section 1.2).

Diagonal truncation can be hard to justify on a theoretical basis. It can retain irrelevant neighbors, as in this example, or it can wipe out our relevant ones. Its use should be limited to situations in which one has a good knowledge of the underlying structure of the spatial-weighting matrix. Bin or value truncation will generally be easier to apply.

$$\begin{bmatrix} 0 & 0 & 0.5 & 0.00199 & 0.00199 & 0.5 & 0.00199 & 0.00199 & 0.5 \\ 0 & 1 & 1 & 0.002 & 1 & 1 & 0.002 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 0.002 & 1 & 1 & 0.002 & 1 & 1 & 0 \\ 0.5 & 0.00199 & 0.00199 & 0.5 & 0.00199 & 0.00199 & 0.5 & 0 & 0 \end{bmatrix}$$

A word of warning: While truncation leads to matrices that can be stored more efficiently, truncation should only be applied if supported by theory. Ad hoc truncation may lead to a misspecification of the model and a subsequent inconsistent inference.

Bin truncation details

Formally, letting m be the largest element in $\mathbf{W}$, `btruncate(`b B`)` divides the interval $(0, m]$ into B equal-length subintervals and sets elements in $\mathbf{W}$ whose value falls in the b smallest subintervals to 0. We partition the interval $(0,m]$ into B intervals $(a_{kL}, a_{kU}]$, where $k = \{1, \ldots, B\}$, $a_{kL} = (k-1)m/B$, and $a_{kU} = km/B$. We set $w_{ij} = 0$ if $w_{ij} \in (a_{kL}, a_{kU}]$ for $k \leq b$.

4 Summarizing an existing spatial-weighting matrix

4.1 Syntax

`spmat summarize` *objname* [, `links detail` {`banded` | *truncmethod*}]

where *truncmethod* is one of `btruncate(`b B`)`, `dtruncate(`d_L d_U`)`, or `vtruncate(`v`)`.

4.2 Description

`spmat summarize` reports summary statistics about the elements in the spatial-weighting matrix in the existing `spmat` object *objname*.

4.3 Options

`links` is useful when *objname* contains a contiguity or a normalized-contiguity matrix. Rather than the default summary of the values in the spatial-weighting matrix, `links` causes `spmat summarize` to summarize the number of neighbors.

`detail` requests a tabulation of links for a contiguity or a normalized-contiguity matrix. The values of the identifying variable with the minimum and maximum number of links will be displayed.

`banded` reports the bands for the matrix that already has a (possibly) banded structure but is stored in an $n \times n$ form.

truncmethods are useful when you want to see summary statistics calculated on a spatial-weighting matrix after some elements have been truncated to 0. `spmat summarize` with a *truncmethod* will report the lower and upper band based on a matrix to which the specified truncation criterion has been applied. (Note: No data are actually changed by selecting these options. These options only specify that `spmat summarize` calculate results as if the requested truncation criterion has been applied.)

`btruncate(`b B`)` partitions the values of $\mathbf{W}$ into B bins and truncates to 0 entries that fall into bin b or below.

`dtruncate(`d_L d_U`)` truncates to 0 the values of $\mathbf{W}$ that fall more than d_L diagonals below and d_U diagonals above the main diagonal. Neither value can be greater than $\lfloor$`(cols(W)`$-$`1)`$/4\rfloor$.

`vtruncate(`v`)` truncates to 0 the values of $\mathbf{W}$ that are less than or equal to v.

4.4 Saved results

Only `spmat summarize` returns saved results.

Let $\mathbf{W}_c$ be a contiguity or normalized-contiguity matrix and $\mathbf{W}_d$ be an inverse-distance matrix.

`spmat summarize` saves the following results in `r()`:

Scalars

`r(b)`	number of rows in $\mathbf{W}$	`r(lmean)`	mean number of neighbors in $\mathbf{W}_c$
`r(n)`	number of columns in $\mathbf{W}$	`r(lmax)`	maximum number of neighbors in $\mathbf{W}_c$
`r(lband)`	lower band, if $\mathbf{W}$ is banded	`r(ltotal)`	total number of neighbors in $\mathbf{W}_c$
`r(uband)`	upper band, if $\mathbf{W}$ is banded	`r(eig)`	`1` if object contains eigenvalues, `0` otherwise
`r(min)`	minimum of $\mathbf{W}_d$	`r(canband)`	`1` if object can be banded based on `r(lband)` and `r(uband)`, `0` otherwise
`r(min0)`	minimum element > 0 in $\mathbf{W}_d$		
`r(mean)`	mean of $\mathbf{W}_d$		
`r(max)`	maximum of $\mathbf{W}_d$		
`r(lmin)`	minimum number of neighbors in $\mathbf{W}_c$		

4.5 Examples

It is generally useful to know some summary statistics for the elements of your spatial-weighting matrices. In sections 2.4 and 3.4, we used `spmat summarize` to report summary statistics for spatial-weighting matrices.

Many spatial-weighting matrices contain many elements that are not 0 but are very small. At times, theoretical considerations such as threshold effects suggest that these small weights should be truncated to 0. In these cases, you might want to summarize the elements of the spatial-weighting subject to different truncation criteria as part of some sensitivity analysis.

▷ Example

In section 3.4, we stored an unnormalized inverse-distance spatial-weighting matrix in the `spmat` object `dcounty`. In this example, we find the summary statistics of the elements of a truncated version of this matrix.

For each county, we set to 0 the weights for counties whose centroids are farther than 250 km from the centroid of that county. Because we are operating on inverse distances, we specify 1/250 as the truncation criterion. The summary of the matrix calculated after applying the truncation criterion is reported in the `Truncated matrix` column. We can see that now the minimum nonzero distance is reported as 0.004.

```
. spmat summarize dcounty, vtruncate(1/250)
Summary of spatial-weighting object dcounty
```

	Current matrix	Truncated matrix
Dimensions	3109 x 3109	3109 x 3109
Bands		(3098, 3098)
Values		
min	0	0
min>0	.0002185	.004
mean	.0012296	.0002583
max	1.081453	1.081453

The `Bands` row reports the lower and upper bands with nonzero values. Those values tell us whether the matrix can be stored in banded form. As mentioned in section 3.3, neither value can be greater than $\lfloor$`(cols(W)`$-$`1)/4`$\rfloor$. In our case, the maximum values for bands is $\lfloor(3{,}109-1)/4\rfloor = 777$; therefore, if we truncated the values of the matrix according to our criterion, we would not be able to store the matrix in banded form.[12] In section 5.1, we show how we can use the sorting tricks of Drukker et al. (2011) to store this matrix in banded form.

◁

5 Examples of banded matrices

Thus far we have claimed that banded matrices are useful when handling spatial-weighting matrices, but we have not yet substantiated this point. To illustrate the usefulness of storing spatial-weighting matrices in a banded-matrix form, we revisit the U.S. counties data and introduce U.S. five-digit zip code data.

12. In practice, rather than calculating the maximum value for bands by hand, we would use the **r(canband)**, **r(lband)**, and **r(uband)** scalars returned by **spmat summarize**; see section 4.4 for details.

5.1 U.S. county data revisited

Recall that at the moment, we have the neighbor information stored in the `spmat` object `ccounty`. We use `spmat graph`, discussed in section 7, to produce an intensity plot of the $n \times n$ normalized contiguity matrix contained in the object `ccounty` by typing

```
. spmat graph ccounty, blocks(10)
```

Figure 2 shows that zero and nonzero entries are scattered all over the matrix. This pattern arises because the original shapefiles had the counties sorted in an order unrelated to their distances from a common point.

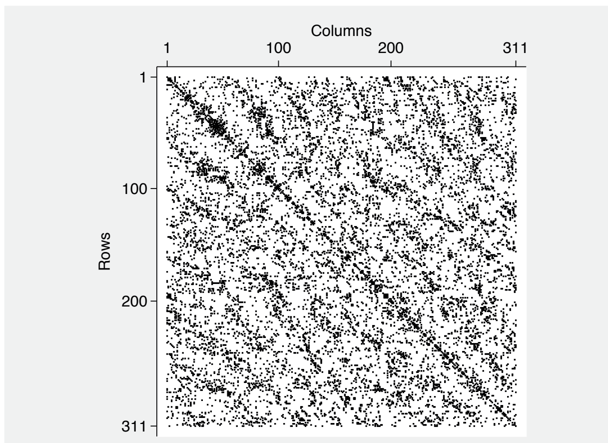

Figure 2. Normalized contiguity matrix for unsorted U.S. counties

We can store the normalized contiguity matrix more efficiently if we generate a variable containing the distance from a particular place to all the other places and then sort the data in an ascending order according to this variable.[13] We implement this method in four steps: 1) we sort the county data by the longitude and latitude of the county centroids contained in `longitude` and `latitude`, respectively, so that the first observation will be the corner observation of Curry County, OR; 2) we calculate the distance of each county from the corner county in the first observation; 3) we sort on the variable containing the distances calculated in step 2; and 4) we recompute and summarize the normalized contiguity matrix.

13. For best results, pick a place located in a remote corner of the map; see Drukker et al. (2011) for further details.

```
. sort longitude latitude
. generate double dist =
> sqrt( (longitude-longitude[1])^2 + (latitude-latitude[1])^2 )
. sort dist
. spmat contiguity ccounty2 using countyxy, id(id) normalize(minmax) banded
> replace
. spmat summ ccounty2, links
Summary of spatial-weighting object ccounty2
```

Matrix	Description
Dimensions	3109 x 3109
Stored as	465 x 3109
Links	
total	18474
min	1
mean	5.942104
max	14

```
. spmat graph ccounty2, blocks(10)
```

Specifying the option `banded` in the `spmat contiguity` command caused the contiguity matrix to be stored as a banded matrix. The summary table shows that the contiguity information is now stored in a $465 \times 3{,}109$ matrix, which requires much less space than the original $3{,}109 \times 3{,}109$ matrix. Figure 3 clearly shows the banded structure.

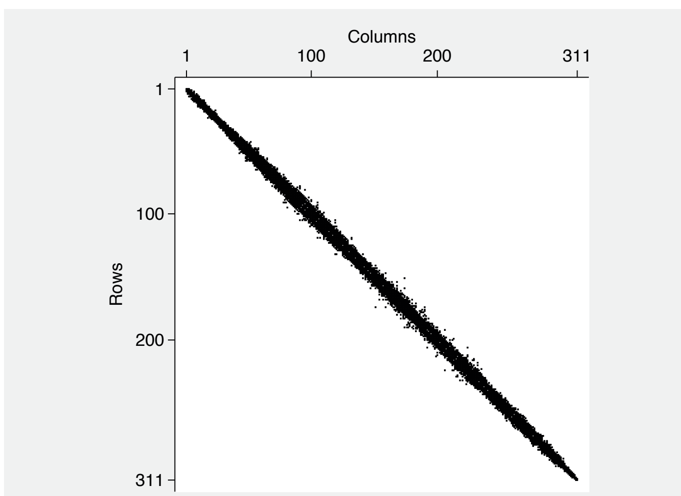

Figure 3. Normalized contiguity matrix for sorted U.S. counties

Similarly, we can re-create the `dcounty` object calculated on the sorted data and see whether the inverse-distance matrix can be stored in banded form after applying a truncation criterion.

```
. spmat idistance dcounty longitude latitude, id(id) dfunction(dhaversine)
> vtruncate(1/250) banded replace
. spmat summ dcounty
Summary of spatial-weighting object dcounty
```

Matrix	Description
Dimensions	3109 x 3109
Stored as	769 x 3109
Values	
min	0
min>0	.004
mean	.0002583
max	1.081453

We can see that the `Values` summary for this matrix and the matrix from section 4.5 is the same; however, the matrix in this example is stored in banded form.

5.2 U.S. zip code data

The real power of banded storage is unveiled when we lack the memory to store spatial data in an $n \times n$ matrix. We use the five-digit zip code level data for the continental United States.[14] We have information on 31,713 five-digit zip codes, and as was mentioned in section 1.2, we need 7.5 gigabytes of memory to store the normalized contiguity matrix as a general matrix.

14. Data are from the U.S. Census Bureau at ftp://ftp2.census.gov/geo/tiger/TIGER2008/.

Instead, we repeat the sorting trick and call `spmat contiguity` with option `banded`, hoping that we will be able to fit the banded representation into memory.

```
. use zip5, clear
. *keep continental US zip codes
. drop if latitude > 49.5 | latitude < 24.5 | longitude < -124
(524 observations deleted)
. sort longitude latitude
. generate double dist =
> sqrt( (longitude-longitude[1])^2 + (latitude-latitude[1])^2 )
. sort dist
. spmat contiguity zip5 using zip5xy, id(id) normalize(minmax) banded
warning: spatial-weighting matrix contains 131 islands
. spmat summarize zip5, links
Summary of spatial-weighting object zip5
```

Matrix	Description
Dimensions	31713 x 31713
Stored as	1827 x 31713
Links	
total	166906
min	0
mean	5.263015
max	26

```
warning: spatial-weighting matrix contains 131 islands
```

The output from `spmat summarize` indicates that the normalized contiguity matrix is stored in a $1{,}827 \times 31{,}713$ matrix. This fits into less than half a gigabyte of memory! All we did to store the matrix in a banded format was change the sort order of the data and specify the `banded` option. We discuss storing an existing $n \times n$ spatial-weighting matrix in banded form in sections 18.1 and 18.2.

Having illustrated the importance of banded matrices, we return to documenting the `spmat` commands.

6 Inserting documentation into your spmat objects

6.1 Syntax

`spmat note` *objname* [{ : "*text*", `replace`|`drop` }]

6.2 Description

`spmat note` creates and manipulates a note attached to the `spmat` object.

6.3 Options

`replace` causes `spmat note` to overwrite the existing note with a new one.

`drop` causes `spmat note` to clear the note associated with *objname*.

6.4 Examples

If you plan to use a spatial-weighting matrix outside a given do-file or session, you should attach some documentation to the `spmat` object.

`spmat note` stores the note in a string scalar; however, it is possible to store multiple notes in the scalar by repeatedly appending notes.

▷ **Example**

We attach a note to the `spmat` object `ccounty` and then display it by typing

```
. spmat note ccounty : "Source: Tiger 2008 county files."
. spmat note ccounty
   Source: Tiger 2008 county files.
```

As mentioned, we can have multiple notes:

```
. spmat note ccounty : "Created on 18jan2011."
. spmat note ccounty
   Source: Tiger 2008 county files. Created on 18jan2011.
```

◁

7 Plotting the elements of a spatial-weighting matrix

7.1 Syntax

`spmat` <u>`gr`</u>`aph` *objname* [`,` <u>`bl`</u>`ocks(`[`(`*stat*`)`] *p*`)` *twoway_options*]

7.2 Description

`spmat graph` produces an intensity plot of the spatial-weighting matrix contained in the `spmat` object *objname*. Zero elements are plotted in white; the remaining elements are partitioned into bins of equal length and assigned gray-scale colors `gs0`–`gs15` (see [G-4] ***colorstyle***), with darker colors representing higher values.

7.3 Options

`blocks(`[`(`*stat*`)`] *p*`)` specifies that the matrix be divided into blocks of size *p* and that block maximums be plotted. This option is useful when the matrix is large. To plot a statistic other than the default maximum, you can specify the optional *stat* argument. For example, to plot block medians, type `blocks((p50)` *p*`)`. The supported statistics include those returned by **`summarize, detail`**; see [R] **summarize** for a complete list.

twoway_options are any options other than `by()`; they are documented in [G-3] ***twoway_options***.

7.4 Examples

An intensity plot of a spatial-weighting matrix can reveal underlying structure. For example, if there is a banded structure to the spatial-weighting matrix, large amounts of memory may be saved.

See section 5.1 for an example in which we use **`spmat graph`** to reveal the banded structure in a spatial-weighting matrix.

8 Computing spatial lags

8.1 Syntax

`spmat lag` [*type*] *newvar objname varname*

8.2 Description

`spmat lag` uses a spatial-weighting matrix to compute the weighted averages of a variable known as the spatial lag of a variable.

More precisely, **`spmat lag`** uses the spatial-weighting matrix in the **`spmat`** object *objname* to compute the spatial lag of the variable *varname* and stores the result in the new variable *newvar*.

8.3 Examples

Spatial lags of the exogenous right-hand-side variables are frequently included in SAR models; see, for example, LeSage and Pace (2009).

Recall that a spatial lag is a weighted average of the variable being lagged. If `x_spl` denotes the spatial lag of the existing variable `x`, using the spatial-weighting matrix $\mathbf{W}$, then the algebraic definition is `x_spl` $= \mathbf{W}$`x`.

The code below generates the new variable `x_spl`, which contains the spatial lag of `x`, using the spatial-weighting matrix **W**, which is contained in the `spmat` object `ccounty`:

```
. clear all
. use county
. spmat contiguity ccounty using countyxy, id(id) normalize(minmax)
. generate x = runiform()
. spmat lag x_spl ccounty x
```

We could now include both `x` and `x_spl` in our model.

9 Computing the eigenvalues of a spatial-weighting matrix

9.1 Syntax

`spmat eigenvalues` *objname* [, `eigenvalues(`*vecname*`)` `replace`]

9.2 Description

`spmat eigenvalues` calculates the eigenvalues of the spatial-weighting matrix contained in the `spmat` object *objname* and stores them in *vecname*. The maximum-likelihood estimator implemented in the `spreg ml` command, as described in Drukker, Prucha, and Raciborski (2013b), uses the eigenvalues of the spatial-weighting matrix during the optimization process. If you are estimating several models by maximum likelihood with the same spatial-weighting matrix, computing and storing the eigenvalues in an `spmat` object will remove the need to recompute the eigenvalues.

9.3 Options

`eigenvalues(`*vecname*`)` stores the user-defined vector of eigenvalues in the `spmat` object *objname*. *vecname* must be a Mata row vector of length n, where n is the dimension of the spatial-weighting matrix in the `spmat` object *objname*.

`replace` permits `spmat eigenvalues` to overwrite existing eigenvalues in *objname*.

9.4 Examples

Putting the eigenvalues into the `spmat` object can dramatically speed up the computations performed by the `spreg ml` command; see Drukker, Prucha, and Raciborski (2013b) for details and references therein.

We can calculate the eigenvalues of the spatial-weighting matrix contained in the `spmat` object `ccounty` and store them in the same object by typing

```
. spmat eigenvalues ccounty
Calculating eigenvalues.... finished.
```

10 Removing an spmat object from memory

10.1 Syntax

`spmat drop` *objname*

10.2 Description

`spmat drop` removes the `spmat` object *objname* from memory.

10.3 Examples

To drop the `spmat` object `dcounty` from memory, we type

```
. spmat drop dcounty
(note: spmat object dcounty not found)
```

11 Saving an spmat object to disk

11.1 Syntax

`spmat save` *objname* `using` *filename* [`, replace`]

11.2 Description

`spmat save` saves the `spmat` object *objname* to a file in a native Stata format.

11.3 Option

`replace` permits `spmat save` to overwrite *filename*.

11.4 Examples

Creating a spatial-weighting matrix, and perhaps its eigenvalues as well, can be a time-consuming process. If you are going to repeatedly use a spatial-weighting matrix, you probably want to save it to a disk and read it back in for subsequent uses. `spmat save` will save the `spmat` object to disk for you. Section 12 discusses `spmat use`, which reads the object from disk into memory.

If you are going to save an `spmat` object to disk, it is a good practice to use `spmat note` to attach some documentation to the object before saving it. Section 6 discusses `spmat note`.

Just like with Stata datasets, you can save your `spmat` objects to disk and share them with other Stata users. The file format is platform independent. So, for example, a Mac user could save an `spmat` object to disk and email it to a coauthor, and the Windows-using coauthor could read in this `spmat` object by using `spmat use`.

We can save the information contained in the `spmat` object `ccounty` in the file `ccounty.spmat` by typing

```
. spmat save ccounty using ccounty.spmat
```

12 Reading spmat objects from disk

12.1 Syntax

`spmat use` *objname* `using` *filename* [`, replace`]

12.2 Description

`spmat use` reads into memory an `spmat` object from a file created by `spmat save`; see section 11 for a discussion of `spmat save`.

12.3 Option

`replace` permits `spmat use` to overwrite an existing `spmat` object.

12.4 Examples

As mentioned in section 11, creating a spatial-weighting matrix can be time consuming. When repeatedly using a spatial-weighting matrix, you might want to save it to disk with `spmat save` and read it back in with `spmat use` for subsequent uses.

In section 11, we saved the `spmat` object `ccounty` to the file `ccounty.spmat`. We now drop the existing `ccounty` object from memory and read it back in with `spmat use`:

```
. spmat drop ccounty
. spmat use ccounty using ccounty.spmat
. spmat note ccounty
   Source: Tiger 2008 county files. Created on 18jan2011.
```

13 Writing a spatial-weighting matrix to a text file

13.1 Syntax

`spmat export` *objname* `using` *filename* [`, noid nlist replace`]

13.2 Description

`spmat export` saves the spatial-weighting matrix contained in the `spmat` object *objname* to a space-delimited text file. The matrix is written in a rectangular format with unique place identifiers saved in the first column. `spmat export` can also save lists of neighbors to a text file.

13.3 Options

`noid` causes `spmat export` not to save unique place identifiers, only matrix entries.

`nlist` causes `spmat export` to write the matrix in the neighbor-list format described in section 2.3.

`replace` permits `spmat export` to overwrite *filename*.

13.4 Examples

The main use of `spmat export` is to export a spatial-weighting matrix to a text file that can be read by another program. Long (2009, 336) recommends exporting all data to text files that will be read by future software as part of archiving one's research work.

Another use of `spmat export` is to review neighbor lists from a contiguity matrix. Here we illustrate how one can export the contiguity matrix in the neighbor-list format described in section 2.3.

```
. spmat export ccounty using nlist.txt, nlist
```

We call the Unix command `head` to list the first 10 lines of `nlist.txt`:[15]

```
. !head nlist.txt
3109
1 1054 1657 2063 2165 2189 2920 2958
2 112 2250 2277 2292 2362 2416 3156
3 2294 2471 2575 2817 2919 2984
4 8 379 1920 2024 2258 2301
5 6 73 1059 1698 2256 2886 2896
6 5 1698 2256 2795 2886 2896 3098
7 517 1924 2031 2190 2472 2575
8 4 379 1832 2178 2258 2987
9 413 436 1014 1320 2029 2166
```

The first line of the file indicates that there are 3,109 total spatial units. The second line indicates that the unit with identification code 1 is a neighbor of units with identification codes 1054, 1657, 2063, 2165, 2189, 2920, and 2958. The interpretation of the remaining lines is analogous to that for the second line.

14 Getting a spatial-weighting matrix from an spmat object

14.1 Syntax

`spmat getmatrix` *objname* [*matname*] [, `id(`*vecname*`)` `eig(`*vecname*`)`]

14.2 Description

`spmat getmatrix` copies the spatial-weighting matrix contained in the `spmat` object *objname* and stores it in the Mata matrix *matname*; see [M-0] **intro** for an introduction to using Mata. If specified, the vector of unique identifiers and the eigenvalues of the spatial-weighting matrix will be stored in Mata vectors.

14.3 Options

`id(`*vecname*`)` specifies the name of a Mata vector to contain IDs.

`eig(`*vecname*`)` specifies the name of a Mata vector to contain eigenvalues.

15. Users of other operating systems should open the file in a text editor.

14.4 Examples

If you want to make changes to an existing spatial-weighting matrix, you need to retrieve it from the `spmat` object, store it in Mata, make the desired changes, and store the new matrix back in the `spmat` object by using `spmat putmatrix`. (See section 17 for a discussion of `spmat putmatrix`.)

`spmat getmatrix` performs the first two tasks: it makes a copy of the spatial-weighting matrix from the `spmat` object and stores it in Mata.

As we discussed in section 3, `spmat idistance` creates a spatial-weighting matrix of the form $1/d_{ij}$, where d_{ij} is the distance between units i and j. In section 17.4, we use `spmat getmatrix` in an example in which we change a spatial-weighting matrix to the form $1/\exp(0.1 \times d_{ij})$ instead of just $1/d_{ij}$.

15 Importing spatial-weighting matrices

15.1 Syntax

`spmat import` *objname* `using` *filename* [`, noid nlist geoda idistance`
`normalize(`*norm*`) replace`]

15.2 Description

`spmat import` imports a spatial-weighting matrix from a space-delimited text file and stores it in a new `spmat` object.

15.3 Options

`noid` specifies that the first column of numbers in *filename* does not contain unique place identifiers and that `spmat import` should create and use the identifiers $1, \ldots, n$.

`nlist` specifies that the text file to be imported contain a list of neighbors in the format described in section 2.3.

`geoda` specifies that *filename* be in the `.gwt` or `.gal` format created by the GeoDa$^{\mathrm{TM}}$ software.

`idistance` specifies that the file contains raw distances and that the raw distances should be converted to inverse distances. In other words, `idistance` specifies that the (i, j)th element in the file be d_{ij} and that the (i, j)th element in the spatial-weighting matrix be $1/d_{ij}$, where d_{ij} is the distance between units i and j.

`normalize(`*norm*`)` specifies one of the three available normalization techniques: `row`, `minmax`, and `spectral`. In a row-normalized matrix, each element in row i is divided by the sum of row i's elements. In a minmax-normalized matrix, each element is divided by the minimum of the largest row sum and column sum of the matrix. In a spectral-normalized matrix, each element is divided by the modulus of the largest eigenvalue of the matrix. See section 2.5 for details.

`replace` permits `spmat import` to overwrite an existing `spmat` object.

15.4 Examples

One frequently needs to import a spatial-weighting matrix from a text file. `spmat import` supports three of the most common formats: simple text files, GeoDa™ text files, and text files that require minor changes such as converting from raw to inverse distances.

By default, the unique place-identifying variable is assumed to be stored in the first column of the file, but this can be overridden with the `noid` option.

In section 17.4, we provide an extended example that begins with using `spmat import` to import a spatial-weighting matrix.

16 Obtaining a spatial-weighting matrix from a Stata dataset

16.1 Syntax

`spmat dta` *objname varlist* [*if*] [*in*] [, `id(`*varname*`)` `idistance` `normalize(`*norm*`)` `replace`]

16.2 Description

`spmat dta` imports a spatial-weighting matrix from the variables in a Stata dataset and stores it in an `spmat` object.

The number of variables in *varlist* must equal the number of observations because spatial-weighting matrices are $n \times n$.

16.3 Options

`id(`*varname*`)` specifies that the unique place identifiers be contained in *varname*. The default is to create an identifying vector containing $1, \dots, n$.

`idistance` specifies that the variables contain raw distances and that the raw distances be converted to inverse distances. In other words, `idistance` specifies that the ith observation on the jth variable be d_{ij} and that the (i, j)th element in the spatial-weighting matrix be $1/d_{ij}$, where d_{ij} is the distance between units i and j.

`normalize(`*norm*`)` specifies one of the three available normalization techniques: `row`, `minmax`, and `spectral`. In a row-normalized matrix, each element in row i is divided by the sum of row i's elements. In a minmax-normalized matrix, each element is divided by the minimum of the largest row sum and column sum of the matrix. In a spectral-normalized matrix, each element is divided by the modulus of the largest eigenvalue of the matrix. See section 2.5 for details.

`replace` permits `spmat dta` to overwrite an existing `spmat` object.

16.4 Examples

People have created Stata datasets that contain spatial-weighting matrices. Given the power of `infile` and `infix` (see [D] **infile (fixed format)** and [D] **infix (fixed format)**), it is likely that more such datasets will be created. `spmat dta` imports these spatial-weighting matrices and stores them in an `spmat` object.

Here we illustrate how we can create an `spmat` object from a Stata dataset. The dataset `schools.dta` contains the distance in miles between five schools in the variables `c1-c5`. The unique school identifier is recorded in the variable `id`. In Stata, we type

```
. use schools, clear
. list
```

	id	c1	c2	c3	c4	c5
1.	101	0	5.9	8.25	6.22	7.66
2.	205	5.9	0	2.97	4.87	7.63
3.	113	8.25	2.97	0	4.47	7
4.	441	6.22	4.87	4.47	0	2.77
5.	573	7.66	7.63	7	2.77	0

```
. spmat dta schools c*, id(id) idistance normalize(minmax)
```

17 Storing a Mata matrix in an spmat object

17.1 Syntax

`spmat putmatrix` *objname* [*matname*] [, `id(`*varname*|*vecname*`)` `eig(`*vecname*`)` `idistance` `bands(`*l u*`)` `normalize(`*norm*`)` `replace`]

17.2 Description

`spmat putmatrix` puts Mata matrices into an existing `spmat` object *objname* or into a new `spmat` object if the specified object does not exist. The optional unique place identifiers can be provided as a Mata vector or a Stata variable. The optional eigenvalues of the Mata matrix can be provided in a Mata vector.

17.3 Options

`id(`*varname* | *vecname*`)` specifies a Mata vector *vecname* or a Stata variable *varname* that contains unique place identifiers.

`eig(`*vecname*`)` specifies a Mata vector *vecname* that contains the eigenvalues of the matrix.

`idistance` specifies that the Mata matrix contains raw distances and that the raw distances be converted to inverse distances. In other words, `idistance` specifies that the (i, j)th element in the Mata matrix be d_{ij} and that the (i, j)th element in the spatial-weighting matrix be $1/d_{ij}$, where d_{ij} is the distance between units i and j.

`bands(`*l u*`)` specifies that the Mata matrix *matname* be banded with l lower and u upper diagonals.

`normalize(`*norm*`)` specifies one of the three available normalization techniques: `row`, `minmax`, and `spectral`. In a row-normalized matrix, each element in row i is divided by the sum of row i's elements. In a minmax-normalized matrix, each element is divided by the minimum of the largest row sum and column sum of the matrix. In a spectral-normalized matrix, each element is divided by the modulus of the largest eigenvalue of the matrix. See section 2.5 for details.

`replace` permits `spmat putmatrix` to overwrite an existing `spmat` object.

17.4 Examples

`spmat contiguity` and `spmat idistance` create spatial-weighting matrices from raw data. This section describes situations in which we have the spatial-weighting matrix precomputed and simply want to put it in an `spmat` object. The spatial-weighting matrix can be any matrix that satisfies the conditions discussed, for example, in Kelejian and Prucha (2010).

In this section, we show how to create an `spmat` object from a text file by using `spmat import` and how to use `spmat getmatrix` and `spmat putmatrix` to generate an inverse-distance matrix according to a user-specified functional form.

The file `schools.txt` contains the distance in miles between five schools. We call the Unix command `cat` to print the contents of the file:

```
. !cat schools.txt
5
101 0 5.9 8.25 6.22 7.66
205 5.9 0 2.97 4.87 7.63
113 8.25 2.97 0 4.47 7
441 6.22 4.87 4.47 0 2.77
573 7.66 7.63 7 2.77 0
```

The school ID is recorded in the first column of the file, and column i records the distance from school i to all the other schools, including itself. We can use `spmat import` to create a spatial-weighting matrix from this file:

```
. spmat import schools using schools.txt, replace
```

The resulting spatial-weighting matrix is

$$\begin{bmatrix} 0 & 5.9 & 8.25 & 6.22 & 7.66 \\ 5.9 & 0 & 2.97 & 4.87 & 7.63 \\ 8.25 & 2.97 & 0 & 4.47 & 7.0 \\ 6.22 & 4.87 & 4.47 & 0 & 2.77 \\ 7.66 & 7.63 & 7.0 & 2.77 & 0 \end{bmatrix}$$

We now illustrate how to create a spatial-weighting matrix with the distance declining in an exponential fashion, $\exp(-0.1 d_{ij})$, where d_{ij} is the original distance from school i to school j.

```
. spmat getmatrix schools x
. mata: x = exp(-.1:*x)
. mata: _diag(x,0)
. spmat putmatrix schools x, normalize(minmax) replace
```

Thus we read in the original distances, extract the distance matrix with `spmat getmatrix`, use Mata to transform the matrix entries according to our specifications, and reset the diagonal elements to 0. Finally, we use `spmat putmatrix` to put the transformed matrix into an `spmat` object. The resulting minmax-normalized spatial-weighting matrix is

$$\begin{bmatrix} 0 & 0.217 & 0.172 & 0.211 & 0.182 \\ 0.217 & 0 & 0.292 & 0.241 & 0.183 \\ 0.172 & 0.292 & 0 & 0.251 & 0.195 \\ 0.211 & 0.241 & 0.251 & 0 & 0.297 \\ 0.182 & 0.183 & 0.195 & 0.297 & 0 \end{bmatrix}$$

18 Converting general matrices into banded matrices

This section shows how to transform a spatial-weighting matrix stored as a general matrix in an `spmat` object in a banded format. If this topic is not of interest, you can skip this section.

The easy case is when the matrix already has a banded structure so that we can simply use `spmat tobanded`.

Now consider the more difficult case in which we have a spatial-weighting matrix stored in an `spmat` object and we would like to use the sorting method described in Drukker et al. (2011) to store this matrix in a banded format. This transformation requires 1) permuting the elements of the existing spatial-weighting matrix to correspond to a new row sort order and then 2) storing the spatial-weighting matrix in banded format. We accomplish step 1 by storing the new row sort order in a permutation vector, as explained below, and then by using `spmat permute`. We use `spmat tobanded` to perform step 2.

Note that most of the time, it is more convenient to sort the data as described in section 5.1 and to call `spmat contiguity` or `spmat idistance` with a truncation criterion. With very large datasets, `spmat contiguity` and `spmat idistance` will be the only choices because they are capable of creating banded matrices from data without first storing the matrices in a general form.

18.1 Permuting a spatial-weighting matrix stored in an spmat object

Syntax

`spmat permute` *objname pvarname*

Description

`spmat permute` permutes the rows and columns of the $n \times n$ spatial-weighting matrix stored in the `spmat` object *objname*. The permutation vector stored in *pvarname* contains a permutation of the integers $\{1, \ldots, n\}$, where n is both the sample size and the dimension of $\mathbf{W}$. That the value of the ith observation of *pvarname* is j specifies that we must move row j to row i in the permuted matrix. After moving all the rows as specified in *pvarname*, we move the columns in an analogous fashion. See *Permutation details: Mathematics* below for a more thorough explanation.

Examples

`spmat permute` is illustrated in the *Examples* section of section 18.2.

Permutation details: Mathematics

Let `p` be the permutation vector created from *pvarname*, and let `W` be the spatial-weighting matrix contained in the specified `spmat` object. The $n \times 1$ permutation vector `p` contains a permutation of the integers $\{1, \ldots, n\}$, where n is the dimension of `W`.

The permutation of `W` is obtained by reordering the rows and columns of `W` as specified by the elements of `p`. Each element of `p` specifies a row and column reordering of `W`. That element i of `p` is j—that is, `p[i]=j`—specifies that we must move row j to row i in the permuted matrix. After moving all the rows according to `p`, we move the columns analogously.

Here is an illustrative example. We have a matrix `W`, which is not banded:

```
. mata: W
[symmetric]
       1   2   3   4   5
    +---------------------+
  1 |  0                  |
  2 |  1   0              |
  3 |  0   0   0          |
  4 |  0   1   0   0      |
  5 |  1   0   1   0   0  |
    +---------------------+
```

Suppose that we also have a permutation vector `p` that we could use to permute `W` to a banded matrix.

```
. mata: p
       1   2   3   4   5
    +---------------------+
  1 |  3   5   1   2   4  |
    +---------------------+
```

See *Permutation details: An example* below to see how we used the sorting trick of Drukker et al. (2011) to obtain this `p`. See *Examples* in section 18.2 for an example with real data.

The values in the permutation vector `p` specify how to permute (that is, reorder) the rows and the columns of `W`. Let's start with the rows. That 3 is element 1 of `p` specifies that row 3 of `W` be moved to row 1 in the permuted matrix. In other words, we must move row 3 to row 1.

Applying this logic to all the elements of `p` yields that we must reorder the rows of `W` by moving row 3 to row 1, row 5 to row 2, row 1 to row 3, row 2 to row 4, and row 4 to row 5. In the output below, we use Mata to perform this operation on `W`, store the result in `A`, and display `A`. If the Mata code is confusing, just check that `A` contains the described row reordering of `W`.

```
. mata: A = W[p,.]

. mata: A
       1   2   3   4   5
    +---------------------+
  1 |  0   0   0   0   1  |
  2 |  1   0   1   0   0  |
  3 |  0   1   0   0   1  |
  4 |  1   0   0   1   0  |
  5 |  0   1   0   0   0  |
    +---------------------+
```

Having reordered the rows, we reorder the columns in the analogous fashion. Operating on `A`, we move column 3 to column 1, column 5 to column 2, column 1 to column 3, column 2 to column 4, and column 4 to column 5. In the output below, we use Mata to perform this operation on `A`, store the result in `B`, and display `B`. If the Mata code is confusing, just check that `B` contains the reordering of `A` described above.

```
. mata: B = A[.,p]
. mata: B
[symmetric]
       1   2   3   4   5
    +---------------------+
  1 |  0                  |
  2 |  1   0              |
  3 |  0   1   0          |
  4 |  0   0   1   0      |
  5 |  0   0   0   1   0  |
    +---------------------+
```

Note that B is the desired banded matrix. For Mata aficionados, typing W[p,p] would produce this permutation in one step.

For those whose intuition is grounded in linear algebra, here is the permutation-matrix explanation. The permutation vector p defines the permutation matrix E, where E is obtained by performing the row reordering described above on the identity matrix of dimension 5. Then the permuted form of W is given by E*W*E', as we illustrate below:

```
. mata: E = I(5)
. mata: E
[symmetric]
       1   2   3   4   5
    +---------------------+
  1 |  1                  |
  2 |  0   1              |
  3 |  0   0   1          |
  4 |  0   0   0   1      |
  5 |  0   0   0   0   1  |
    +---------------------+
. mata: E = E[p,.]
. mata: E
       1   2   3   4   5
    +---------------------+
  1 |  0   0   1   0   0  |
  2 |  0   0   0   0   1  |
  3 |  1   0   0   0   0  |
  4 |  0   1   0   0   0  |
  5 |  0   0   0   1   0  |
    +---------------------+
. mata: E*W*E´
[symmetric]
       1   2   3   4   5
    +---------------------+
  1 |  0                  |
  2 |  1   0              |
  3 |  0   1   0          |
  4 |  0   0   1   0      |
  5 |  0   0   0   1   0  |
    +---------------------+
```

permutation (see [M-1] **permutation**) provides further details on permutation vectors and permutation matrices.

Permutation details: An example

`spmat permute` requires that the permutation vector be stored in the Stata variable *pvarname*. Assume that we now have the unpermuted matrix **W** stored in the `spmat` object `cobj`. The matrix represents contiguity information for the following data:

```
. list

     +----------------+
     |  id   distance |
     |----------------|
  1. |  79       5.23 |
  2. |  82      27.56 |
  3. | 100          0 |
  4. | 114       1.77 |
  5. | 140      20.47 |
     +----------------+
```

The variable `distance` measures the distance from the centroid of the place with `id=100` to the centroids of all the other places. We sort the data on `distance` and generate the permutation vector `p`, which is just a running index $1, \ldots, 5$:

```
. sort distance
. generate p = _n
. list

     +--------------------+
     |  id   distance   p |
     |--------------------|
  1. | 100          0   1 |
  2. | 114       1.77   2 |
  3. |  79       5.23   3 |
  4. | 140      20.47   4 |
  5. |  82      27.56   5 |
     +--------------------+
```

We obtain our permutation vector by sorting the data back to the original order based on the `id` variable:

```
. sort id
. list

     +--------------------+
     |  id   distance   p |
     |--------------------|
  1. |  79       5.23   3 |
  2. |  82      27.56   5 |
  3. | 100          0   1 |
  4. | 114       1.77   2 |
  5. | 140      20.47   4 |
     +--------------------+
```

Now coding `spmat permute cobj p` will reorder the rows and columns of **W** in exactly the same way as the Mata code did above.

18.2 Banding a spatial-weighting matrix

Syntax

`spmat tobanded` *objname1* [*objname2*] [`,` *truncmethod* `replace`]

where *truncmethod* is one of `btruncate(`b B`)`, `dtruncate(`d_L d_U`)`, or `vtruncate(`#`)`.

Description

`spmat tobanded` stores an existing, general-format spatial-weighting matrix in a banded format. `spmat tobanded` has truncation options for inducing a banded structure in spatial-weighting matrices that are not already in banded form.

More precisely, `spmat tobanded` stores the spatial-weighting matrix in an `spmat` object in banded format.

Options

truncmethod specifies one of the three truncation criteria. The values of $\mathbf{W}$ that meet the truncation criterion will be changed to 0.

`btruncate(`b B`)` partitions the values of $\mathbf{W}$ into B bins and truncates to 0 entries that fall into bin b or below.

`dtruncate(`d_L d_U`)` truncates to 0 the values of $\mathbf{W}$ that fall more than d_L diagonals below and d_U diagonals above the main diagonal. Neither value can be greater than $\lfloor$`(cols(W)`$-$`1)`$/4\rfloor$.

`vtruncate(`#`)` truncates to 0 the values of $\mathbf{W}$ that are less than or equal to #.

`replace` allows *objname1* or *objname2* to be overwritten if it already exists.

Examples

Sometimes, we have large spatial-weighting matrices that fit in memory, but they take up so much space that there is too little room to do anything else. In these cases, we are better off storing these spatial-weighting matrices in a banded format when possible.

`spmat tobanded` stores existing spatial-weighting matrices in a banded format. The two allowed syntaxes are

`spmat tobanded` *objname1* `, replace`

and

`spmat tobanded` *objname1 objname2* [`, replace`]

The first syntax replaces the general-form spatial-weighting matrix in the `spmat` object *objname1* with its banded form.

The second syntax stores the general-form spatial-weighting matrix in the `spmat` object *objname1* in banded form in the `spmat` object *objname2*. You must specify `replace` if *objname2* already exists.

We continue with the example from section 2.4, where we have the $3{,}109 \times 3{,}109$ normalized-contiguity matrix stored in the `spmat` object `ccounty`. In section 5.1, we showed that if we sort the data on a distance variable, we can call `spmat contiguity` again and get a banded matrix. Here we show that we can achieve the same result by 1) creating a permutation vector, 2) calling `spmat permute`, and 3) running `spmat tobanded` on the existing `spmat` object.

We begin by generating a permutation vector and storing it in the Stata variable `p`. Recall that we want the ith element of `p` to contain the observation number that it will have under the new sort order. This process is given in the code below and is analogous to the one discussed in the subsections *Permutation details: Mathematics* and *Permutation details: An example* in section 18.1. Because the data are already sorted by ID, we begin by sorting them by longitudes and latitudes of the centroids so that the first observation will contain a corner place. Next we generate the distance from the corner place. After sorting the data in ascending order from the distance to the corner observation, we generate our permutation vector `p` and finally put the data back in the original sort order.

```
. use county, clear
. generate p = _n
. sort longitude latitude
. generate double dist =
> sqrt( (longitude-longitude[1])^2 + (latitude-latitude[1])^2 )
. sort dist
```

We can now use this permutation vector and `spmat permute` to perform the permutation, and we can finally call `spmat tobanded` to band the spatial-weighting matrix stored inside the `spmat` object `ccounty`. Note that the reported summary is identical to the one in section 5.1.

```
. spmat permute ccounty p
. spmat tobanded ccounty, replace
. spmat summarize ccounty, links
Summary of spatial-weighting object ccounty
```

Matrix	Description
Dimensions	3109 x 3109
Stored as	465 x 3109
Links	
total	18474
min	1
mean	5.942104
max	14

```
(object contains eigenvalues)
```

19 Conclusion

We discussed the `spmat` command for creating, managing, importing, manipulating, and storing spatial-weighting matrix objects. In future work, we will consider additional subcommands for creating specific types of spatial-weighting matrices.

20 Acknowledgment

We gratefully acknowledge financial support from the National Institutes of Health through the SBIR grants R43 AG027622 and R44 AG027622.

21 References

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

———. 2010. Thirty years of spatial econometrics. *Papers in Regional Science* 89: 3–25.

Arbia, G. 2006. *Spatial Econometrics: Statistical Foundations and Applications to Regional Convergence*. Berlin: Springer.

Cliff, A. D., and J. K. Ord. 1973. *Spatial Autocorrelation*. London: Pion.

———. 1981. *Spatial Processes: Models and Applications*. London: Pion.

Cressie, N. A. C. 1993. *Statistics for Spatial Data*. Revised ed. New York: Wiley.

Crow, K. 2006. shp2dta: Stata module to convert shape boundary files to Stata datasets. Statistical Software Components S456718, Department of Economics, Boston College. http://ideas.repec.org/c/boc/bocode/s456718.html.

Crow, K., and W. Gould. 2007. FAQ: How do I graph data onto a map with spmap? http://www.stata.com/support/faqs/graphics/spmap-and-maps/.

Drukker, D. M., P. Egger, and I. R. Prucha. 2013. On two-step estimation of a spatial autoregressive model with autoregressive disturbances and endogenous regressors. *Econometric Reviews* 32: 686–733.

Drukker, D. M., H. Peng, I. R. Prucha, and R. Raciborski. 2011. Sorting induces a banded structure in spatial-weighting matrices. Working paper, Department of Economics, University of Maryland.

Drukker, D. M., I. R. Prucha, and R. Raciborski. 2013a. A command for estimating spatial-autoregressive models with spatial-autoregressive disturbances and additional endogenous variables. *Stata Journal* 13: 287–301.

———. 2013b. Maximum likelihood and generalized spatial two-stage least-squares estimators for a spatial-autoregressive model with spatial-autoregressive disturbances. *Stata Journal* 13: 221–241.

Haining, R. 2003. *Spatial Data Analysis: Theory and Practice*. Cambridge: Cambridge University Press.

Kelejian, H. H., and I. R. Prucha. 2010. Specification and estimation of spatial autoregressive models with autoregressive and heteroskedastic disturbances. *Journal of Econometrics* 157: 53–67.

Lai, P.-C., F.-M. So, and K.-W. Chan. 2009. *Spatial Epidemiological Approaches in Disease Mapping and Analysis*. Boca Raton, FL: CRC Press.

Leenders, R. T. A. J. 2002. Modeling social influence through network autocorrelation: Constructing the weight matrix. *Social Networks* 24: 21–47.

LeSage, J., and R. K. Pace. 2009. *Introduction to Spatial Econometrics*. Boca Raton: Chapman & Hall/CRC.

Long, J. S. 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.

Pisati, M. 2005. mif2dta: Stata module to convert MapInfo Interchange Format boundary files to Stata boundary files. Statistical Software Components S448403, Department of Economics, Boston College.
http://ideas.repec.org/c/boc/bocode/s448403.html.

———. 2007. spmap: Stata module to visualize spatial data. Statistical Software Components S456812, Department of Economics, Boston College.
http://ideas.repec.org/c/boc/bocode/s456812.html.

Tobler, W. R. 1970. A computer movie simulating urban growth in the Detroit region. *Economic Geography* 46: 234–240.

Waller, L. A., and C. A. Gotway. 2004. *Applied Spatial Statistics for Public Health Data*. Hoboken, NJ: Wiley.

Whittle, P. 1954. On stationary processes in the plane. *Biometrika* 41: 434–449.

About the authors

David Drukker is the director of econometrics at StataCorp.

Hua Peng is a senior software engineer at StataCorp.

Ingmar Prucha is a professor of economics at the University of Maryland.

Rafal Raciborski is an econometrician at StataCorp.