Title

stata.com

spmatrix create — Create standard weighting matrices

Description

`spmatrix create` creates standard-format spatial weighting matrices.

Quick start

Create contiguity spatial weighting matrix `M` with default spectral normalization

```
spmatrix create contiguity M
```

Same as above

```
spmatrix create contiguity M, normalize(spectral)
```

Create row-standardized contiguity spatial weighting matrix `M`

```
spmatrix create contiguity M, normalize(row)
```

Create contiguity spatial weighting matrix `M` without normalization

```
spmatrix create contiguity M, normalize(none)
```

Create spectral-normalized inverse-distance spatial weighting matrix `W`

```
spmatrix create idistance W
```

Menu

Statistics > Spatial autoregressive models

Syntax

```
spmatrix create contiguity spmatname [if] [in] [, contoptions stdoptions]

spmatrix create idistance spmatname [if] [in] [, idistoption stdoptions]
```

spmatname is a weighting matrix name.

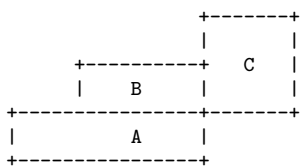
contoptions	Description
`rook`	share a border and not just a vertex
`first`	first-order neighbors
`second[(#)]`	second-order neighbors

idistoption	Description
`vtruncate(#)`	set (i, j) element to 0 if $1/\text{distance} \leq \#$

stdoptions	Description
`normalize(`*normalize*`)`	type of normalization; default is `normalize(spectral)`
`replace`	replace existing weighting matrix

Options for spmatrix create contiguity

`rook` specifies that areas that share just a vertex not be treated as neighbors. For instance, consider the following map:



If `rook` is not specified, A and C are neighbors because they have a vertex (corner) in common. If `rook` is specified, A and C are not neighbors. Regardless of whether `rook` is specified, A and B are neighbors and B and C are neighbors because they share a border (line segment).

`first` specifies that first-order neighbors be assigned 1. If areas i and j are neighbors, then $\textit{spmatname}_{i,j} = \textit{spmatname}_{j,i} = 1$. `first` is the default unless `second` or `second(#)` is specified.

`second[(#)]` specifies that the second-order neighbors—neighbors of neighbors—be assigned a nonzero value. `second` specifies that they be assigned 1. `second(#)` specifies that they be assigned #.

If you also specify option `first`, then the matrix created will set first-order neighbors to contain 1. For instance, if you specify `first second`, both kinds of neighbors will be set to 1. If you specify `first second(.5)`, first-order neighbors are set to 1 and second-order neighbors are set to 0.5.

Option for spmatrix create idistance

`vtruncate(#)` specifies that areas farther apart than # be set to 0. Type `spset` without arguments to determine the units in which # is specified. The `Coordinates` line of `spset`'s output will be one of the following:

```
Coordinates:  _CX, _CY (planar)
Coordinates:  _CY, _CX (latitude and longitude, kilometers)
Coordinates:  _CY, _CX (latitude and longitude, miles)
```

Units of # will be planar, kilometers, or miles. If planar, see [SP] **spdistance** for advice on determining the units.

If `spset` reports

```
Coordinates:  none
```

then you cannot use `spmatrix create`.

Options for both contiguity and idistance

`normalize(`*normalize*`)` specifies how the resulting matrix is to be scaled.

`normalize(spectral)` is the default. The matrix will be normalized so that its largest eigenvalue is 1.

`normalize(minmax)` specifies that the matrix elements be divided by the smaller of the largest row or column sum of absolute values. The min–max calculation is much quicker than the spectral calculation and in most cases gives similar results as the spectral normalization.

`normalize(row)` specifies that each row of the matrix be divided by the row's sum (not absolute values). This adjustment can be performed even more quickly than the min–max adjustment.

`normalize(none)` specifies that the matrix not be rescaled. This option has one use: To store the matrix in unadjusted form so that you can fetch it later, make changes to it while the matrix is still in its original units, and then repost the matrix, at which point it will be rescaled. See *Choosing weighting matrices and their normalization* in [SP] **spregress** for details about normalization.

`replace` specifies that matrix *spmatname* may be replaced if it already exists.

Remarks and examples

stata.com

See [SP] **Intro 1** about the role spatial weighting matrices play in SAR models and see [SP] **Intro 2** for a thorough discussion of the matrices. To remind you, the (i, j) element of a weighting matrix specifies the potential spillover from area j to i.

Remarks are presented under the following headings:

Creating contiguity matrices

`spmatrix create contiguity` is mostly used to create matrices with elements equal to 1 or 0 (before normalization). $spmatname_{i,j}$ is 1 when areas i and j are neighbors. The matrix is symmetric.

Creation of contiguity matrices requires that the Sp data in memory be linked to a shapefile. The data must look like this:

```
. spset
      Sp dataset: irrelevant
Linked shapefile: irrelevant
            Data: Cross sectional or Panel
 Spatial-unit ID: _ID
     Coordinates: _CY, _CX (latitude and longitude, miles)
```

Determining whether places are neighbors requires a linked shapefile. Knowing their locations is not sufficient.

To create a contiguity matrix named `F` of first-order neighbors, type

```
. spmatrix create contiguity F
```

The contiguity matrix `F` is automatically normalized using the spectral normalization; see *Choosing weighting matrices and their normalization* in [SP] **spregress** for details about normalization.

In [SP] **Intro 1**, we discussed a spatial weighting matrix containing 1s for first-order neighbors and 0.5s for second-order neighbors. Such a matrix could be created by typing

```
. spmatrix create contiguity W, first second(0.5)
```

Also in the introduction, we considered making two weighting matrices, `W` for first-order neighbors and `V` for second. To create `W` and `V`, type

```
. spmatrix create contiguity W
. spmatrix create contiguity V, second
```

The syntax of the `spmatrix create contiguity` command is as follows:

Command	Meaning
1. `spmatrix create contiguity`	1st-order neighbors
2. `spmatrix create contiguity, first`	same as command 1
3. `spmatrix create contiguity, second`	2nd-order neighbors
4. `spmatrix create contiguity, second(1)`	same as command 3
5. `spmatrix create contiguity, first second`	1st- and 2nd-order neighbors
6. `spmatrix create contiguity, first second(1)`	same as command 5
7. `spmatrix create contiguity, first second(0.5)`	1st- and 2nd-order neighbors, 1st set to 1, 2nd set to 0.5

Creating inverse-distance matrices

`spmatrix create idistance` creates matrices with elements equal to the reciprocal of distance between places (before normalization). The matrix is symmetric.

Creation of inverse-distance matrices requires that the Sp data have coordinates, but a shapefile is not required. The data must look like this:

```
. spset
      Sp dataset: irrelevant
Linked shapefile: irrelevant
            Data: Cross sectional or Panel
 Spatial-unit ID: _ID
     Coordinates: _CY, _CX (latitude and longitude, miles)
```

Coordinates must be defined, although they are not required to be latitude and longitude. If they are latitude and longitude, however, Sp needs to know; see [SP] **Intro 4**. Whether units are miles or kilometers is irrelevant.

To create an inverse-distance matrix named `Idist`, type

```
. spmatrix create idistance Idist
```

The inverse-distance matrix `Idist` is automatically normalized using the spectral normalization; see *Choosing weighting matrices and their normalization* in [SP] **spregress** for details about normalization.

`spmatrix create idistance` allows option `vtruncate(#)`, which sets spillovers less than or equal to # to 0. To create an inverse-distance matrix `I0` with places more than 100 apart, you could type

```
. spmatrix create idistance I0, vtruncate(.01)
```

Note that you specify $\# = 1/\text{distance}$.

See the description of the `vtruncate()` option above for the meaning of how far apart 100 means.

Creating inverse-distance contiguity matrices

An inverse-distance contiguity matrix is a weighting matrix that contains inverse distance for neighbors and 0 otherwise. Here is how you create such a matrix:

1. Create the inverse-distance and contiguity matrices separately.
2. Multiply them element by element in Mata. The result is a matrix containing inverse distance for neighbors because the contiguity matrix contains 1s and 0s.
3. Store the Mata result as an Sp spatial weighting matrix.

We do that below to create an inverse-distance first-order neighbor matrix named `CN`.

```
. // -------------------------- create the matrices separately ---
.
. spmatrix create idistance  N, normalize(none)          // note 1
. spmatrix create contiguity C, first normalize(none)    // note 2
.
. // ------------------------------------ load them into Mata ---
.
. spmatrix matafromsp Wn v = N
. spmatrix matafromsp Wc v = C
.
.
. // ------------------------ multiply them element by element ---
. mata: Wcn = Wc :* Wn                                   // note 3
.
.
```

```
. // ----------------------------------- save the result in Sp ---
.
. spmatrix spfrommata CN = Wcn v                          // note 4
.
.
. // ------------------------------------------------ clean up ---
.
. mata: mata drop Wcn Wc Wn                                // note 5
. spmatrix drop C
. spmatrix drop N
. // ---------------------------------------- the final result ---
.
. spmatrix dir
```

Weighting matrix name	N x N	Type	Normalization
CN	254 x 254	custom	spectral

Notes:

1. We specify `normalize(none)` when we create the matrices separately for speed, not because it is necessary. Normalization amounts to multiplying the matrices by a constant, and that will not matter. Calculating the constant, however, takes considerable time.
2. We created `C` to be first-order neighbors. We could have included second-order neighbors as well by adding option `second` to the command.
3. Colon-asterisk (`:*`) is Mata's element-by-element matrix multiplication operator. It is called colon-multiply.
4. `spmatrix spfrommata` allows the `normalize()` option and defaults to `normalize(spectral)`, just as `spmatrix create` does. Thus, the matrix stored in Sp is normalized.
5. Do not skip the clean-up step. Spatial weighting matrices are $N \times N$ and can consume considerable amounts of memory.

   It is also important that we cleared the Mata matrices by dropping them and not by typing `clear mata`. Sp stores the matrices you create in Mata and, if you cleared Mata, the new weighting matrix `CN` would also be dropped!

See [SP] **spmatrix matafromsp** and [SP] **spmatrix spfrommata**.

The normalize() option

We have hardly mentioned the `normalize()` option so far, because `spmatrix create` normalizes matrices by default. Normalization is important. All the Sp commands that create spatial weighting matrices normalize by default and include the `normalize()` option for cases in which you want to modify how or whether it is done.

Sp provides three normalizations:

`normalize(spectral)`	the default
`normalize(minmax)`	min–max
`normalize(row)`	row

`normalize()` provides a fourth setting to skip normalization altogether:

`normalize(none)`	do not perform normalization

The Sp commands are so determined you do not forget to normalize spatial weighting matrices at the last step that you must not forget to specify `normalize(none)` when you are building a custom matrix from ingredients. The spectral and min–max normalizations merely change the scale of the matrix.

`normalize(row)`, however, is a normalization of a different ilk from the others. The others merely change the scale of the matrix. Changing a matrix's scale is performed by dividing the elements by a constant. `normalize(row)` divides each row by a different constant. Doing this transformation on the matrix changes the model specification.

See [SP] **spmatrix normalize** and *Choosing weighting matrices and their normalization* in [SP] **spregress** for details.

Panel data

If you have panel data and want to create a weighting matrix, you must use an `if` statement with `spmatrix create` to restrict the data to a single time value.

Here is an example. We load an Sp panel dataset and type `spset` to see the Sp settings:

```
. copy https://www.stata-press.com/data/r18/homicide_1960_1990.dta .
. copy https://www.stata-press.com/data/r18/homicide_1960_1990_shp.dta .
. use homicide_1960_1990
(S.Messner et al.(2000), U.S southern county homicide rate in 1960-1990)
. xtset _ID year
Panel variable: _ID (strongly balanced)
 Time variable: year, 1960 to 1990, but with gaps
         Delta: 1 unit
. spset
      Sp dataset: homicide_1960_1990.dta
Linked shapefile: homicide_1960_1990_shp.dta
            Data: Panel
 Spatial-unit ID: _ID
         Time ID: year (see xtset)
     Coordinates: _CX, _CY (planar)
```

If we tried to create a weighting matrix the usual way, we would not be successful:

```
. spmatrix create contiguity W
variable _ID does not uniquely identify observations in the master data
r(459);
```

We get an error message because `spmatrix create` needs to know which observations to use. We must restrict `spmatrix create` to one observation per panel, which is easy to do using an `if` statement:

```
. spmatrix create contiguity W if year == 1990
```

Do not misinterpret the purpose of `if year == 1990`. The matrix created will be appropriate for creating spatial lags for any year, because if two spatial units share a border in 1990, they will share it in the other years too. The map does not change.

Also see

[SP] **spmatrix** — Categorical guide to the spmatrix command

[SP] **Intro** — Introduction to spatial data and SAR models

Mata Reference Manual