spmatrix userdefined — Create custom weighting matrix

Description

`spmatrix userdefined` is one way of creating custom spatial weighting matrices. The function you write need not be based on coordinate locations.

Quick start

Having written the Mata function `SinvD()`, create new spatial weighting matrix `C` with default spectral normalization

```
spmatrix userdefined C = SinvD(_CX _CY)
```

Menu

Statistics > Spatial autoregressive models

Syntax

`spmatrix userdefined` *Wmatname* = *fcnname*`(`*varlist*`)` [*if*] [*in*] [, *options*]

Wmatname is a weighting matrix name, such as `W`.

fcnname is the name of a Mata function you have written, such as `SinvD` or `Awind`.

1. *fcnname* must start with the letter `S` or `A`, which indicates whether the function produces a symmetric or an asymmetric result.
2. *fcnname* receives two row-vector arguments and returns a scalar result. For example,

```
function SinvD(v1, v2)
{
      return(1/sqrt((v1-v2)*(v1-v2)'))
}
```

Function `SinvD()` starts with `S` because for all x and y, `SinvD(`x`,` y`)` = `SinvD(`y`,` x`)`.

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

Options

`normalize(`*normalize*`)` specifies how the resulting matrix is to be scaled. `normalize(spectral)` is the default. `normalize(minmax)`, `normalize(row)`, and `normalize(none)` are also allowed. See [SP] **spmatrix create** for full details of the option and *Choosing weighting matrices and their normalization* in [SP] **spregress** for details about normalization.

`replace` specifies to overwrite matrix *spmatname* if it already exists.

Remarks and examples

Sp provides five ways to create spatial weighting matrices:

1. [SP] **spmatrix create** creates standard weighting matrices. No programming and little effort is required.
2. [SP] **spmatrix import** imports weighting matrices produced by others.
3. [SP] **spmatrix fromdata** lets you create custom weighting matrices without Mata programming.
4. [SP] **spmatrix userdefined** lets you create custom weighting matrices. Some Mata programming is required.
5. [SP] **spmatrix spfrommata** leaves it to you to create matrices from start to finish, which you can do in Mata with or without programs. Once created, you use `spmatrix spfrommata` to store it.

This manual entry concerns method 4. Remarks are presented under the following headings:

Overview

Consider a cross-sectional spatial dataset of N observations. Each observation contains data on a place. We say place, but it need not be a physical place such as census block 060670011011085, zip code 77845, city College Station, county Brazos, state Texas, or country United States. It could be a network node or anything else.

A weighting matrix $\mathbf{W}$ is $N \times N$. Its elements $W_{i,j}$ record the potential spillover from place j to i. For simplicity and without loss of generality, we will assume that places i and j correspond to observations i and j.

`spmatrix userdefined` handles situations when $W_{i,j}$ is a function of $\mathbf{v}_i$ and $\mathbf{v}_j$, where $\mathbf{v}$ is a vector. $W_{i,j}$ could be a function of all the variables in observations i and j, but probably it is a function of a subset of them. For instance, if the spatial weighting matrix were based on locations, $W_{i,j}$ would be a function of variables `_CX` and `_CY` in the two observations. Or if the weighting were based on industry output, it might be a function of variables `f1`, `f2`, …, `f12` in observations i and j. The variables might contain the fraction of output within industrial group and so sum to 1, or they might record total dollar output.

Whatever the relevant variables are, let's just call them *varlist*. Then

$\mathbf{v}_i$ = row vector of values of *varlist* in observation i

$\mathbf{v}_j$ = row vector of values of *varlist* in observation j

The formula for the elements of a spatial weighting matrix is

$$W_{i,j} = \begin{cases} 0 & \text{if } i = j \\ f(v_i, v_j) & \text{otherwise} \end{cases}$$

`spmatrix userdefined` handles this problem when you type

`spmatrix userdefined` *Wmatname* `=` *fcnname*`(`*varlist*`)`

The mapping from the command's syntax to the mathematics is

Syntax element	Corresponding mathematical element
Wmatname	$\mathbf{W}$
fcnname`()`	$f(\cdot)$
varlist	$\mathbf{v}_i$, $\mathbf{v}_j$

Here is an example:

```
. mata:
------------------------------------------------- mata (type end to exit) -----
: function SinvD(vi, vj)
> {
>         return (1/sqrt( (vi-vj)*(vi-vj)' ) )
> }
: end
--------------------------------------------------------------------------------

. spmatrix userdefined W = SinvD(_CX _CY)
```

The above produces the matrix that could also be created by typing

```
. spmatrix create idistance W
```

Here is an example that `spmatrix create` cannot duplicate:

```
. mata:
------------------------------------------------- mata (type end to exit) -----
: function Sdistance(vi, vj)
> {
>         return ( sqrt( (vi-vj)*(vi-vj)' ) )
> }
: end
--------------------------------------------------------------------------------

. spmatrix userdefined W = Sdistance(f*)
```

The above code calculates the distance between the `f*` variables for all i and j. The "farther" apart industrial output shares are, the greater is the incentive for places i and j to engage in trade.

Sfcnname() versus Afcnname()

The Mata functions you write start with the letter S when

S*fcnname*(*v_i*, *v_j*) == S*fcnname*(*v_j*, *v_i*)

Commutative functions produce symmetric matrices. `spmatrix userdefined` runs faster in this case because when it calls the function to calculate $W_{i,j}$, it also stores the result in $W_{j,i}$.

If the function is not commutative, it produces asymmetric matrices. Name such functions A*fcnname*(). Then, the function will be called separately to calculate $W_{i,j}$ and $W_{j,i}$.

Programming style

We wrote inverse distance as

```
. mata:
: program SinvD(vi, vj)
> {
>         return( 1/sqrt( (vi-vj)*(vi-vj)' ) )
> }
: end
```

There are other programming styles we could have used. The above used vector and matrix notation. Here is the same calculation written even more densely:

```
: program SinvD(vi, vj)
> {
>         delta = vi - vj
>         return( 1/sqrt( delta*delta' ) )
> }
: end
```

And here is the calculation again in more traditional scalar notation:

```
. mata:
: program SinvD(vi, vj)
> {
>         delta_x = vi[1] - vj[1]
>         delta_y = vi[2] - vj[2]
>         return( 1/sqrt(delta_x^2 + delta_y^2) )
> }
: end
```

You can write code in whichever style you find easiest.

Advanced programs

The Mata program you write is not required to be simple. If you were an epidemiologist, you could write a program that accounted for prevailing wind direction so that communicable diseases were more likely to spillover when location j is west of i and j's prevailing winds are out of the west. The program would look something like this:

```
: mata
: program Awind(vi, vj)
> {
>         locj      = (vj[1], vj[2])
>         loci      = (vi[1], vi[2])
>         windfromi =  vi[3]    // 1=N, 2=E, 3=S, 4=W
>
>         j_rel_i   = ...       // 1 if j N of i,
>                               // 2 if j E of i,
>                               // ..
>
>         if (j_rel_i == windfrom)  c = 1.5
>         else                      c = 0.5
>
>         return(SinvD(loci, locj)*c)
> }
: end
```

We omitted lines, and if we were going to use this approach, we would further complicate the program by considering the directions N, NE, E, SE, S, SW, W, and NW.

However complicated the code might be, the weighting matrix would be calculated by typing

```
. spmatrix userdefined Wadj = Awind(_CX _CY winddir)
```

`Wadj` would contain a wind-adjusted distance matrix, which will also be spectral normalized.

And then, we would be tempted to convert `Wadj` to be a wind-adjusted distance matrix for areas that bordered on each other. Let us show you how.

Mixed approaches

You do not have to calculate everything in your program. Let's imagine that in the above example, the researcher only wants to use the wind-adjusted calculation when i and j are first-order neighbors. Otherwise, the spillover is to be 0. We can use the same approach explained in more detail in [SP] **spmatrix create**:

1. Create the wind-adjusted matrix as shown above, but do not normalize it.
2. Create the first-order neighbor matrix using `spmatrix create contiguity`, also unnormalized.
3. Multiply the matrices element by element in Mata.
4. Store the Mata result in Sp.

The code is

```
. // -------------------------- create the matrices separately ---
. spmatrix userdefined Wadj = Awind(_CX _CY winddir), normalize(none)
. spmatrix create contiguity C, first normalize(none)
.
.
. // ------------------------------------- load them into Mata ---
. spmatrix matafromsp W1 v = Wadj
. spmatrix matafromsp W2 v = C
.
.
. // -------------------------multiply them element by element ---
. mata: W3 = W1 :* W2
```

```
.
. // ------------------------------------save the result in Sp ---
.
. spmatrix spfrommata Wfinal = W3 v
.
. // ------------------------------------------------clean up ---
.
. mata: mata drop  W1 W2 W3
. spmatrix drop Wadj
. spmatrix drop C
.
. // ---------------------------------------------final result ---
. spmatrix dir
```

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

In the above, :∗ (colon-asterisk) is Mata's element-by-element multiply function. See [SP] **spmatrix create** for more explanation.

Also see

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

[SP] **spmatrix spfrommata** — Copy Mata matrix to Sp

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

Mata Reference Manual