STATA Resources, Unit 6, Section 1

Matrix construction and manipulation in STATA

Kyle Puetz • 4/28/13

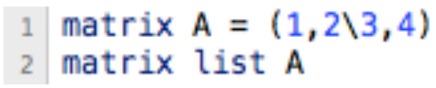
STATA provides a number of functions that allow users to operate it as a matrix calculator. Users can construct, edit, and analyze matrices all within the STATA interface.

The first thing I would like to demonstrate is how to construct matrices within STATA using the `matrix` command. The syntax for this command is:

1. Matrix input

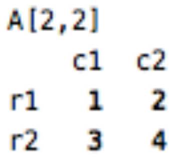
```
matrix [input] A = (#[,#...] [\ #[,#...] [\ [...]]])
```

Consequently, the following command (from which we've removed the unnecessary `input` aspect) …



```
matrix A = (1,2\3,4)
matrix list A
```

… provides us with this output:



```
A[2,2]
    c1  c2
r1   1   2
r2   3   4
```

It should be clear, then, that a comma indicates to STATA elements that should be separated *within the same row* while the backslash indicates that the following elements should be placed in a new, different row. It should also be clear that the default row names is r*n* where *n* is the number of the row, and, likewise, the default column name is c*n* where *n* the number of the column. Also note that the matrix list command provides us with information on the total number of rows and columns when it provides us with "A[2,2]." The first number here indicates the number of rows and the second the number of columns. Though with this example, we could clearly just tell how many rows and columns there are by looking at the matrix, this function becomes more helpful when dealing with matrices of exceptionally large size.

It is also noteworthy that we can create *vectors* using the set of matrix commands. We simply use the same syntax:

```
matrix V1 = (1,2,3,4)
matrix V2 = (1\2\3\4)
matrix list V1
matrix list V2
```

... which provides us with the following results:

```
. matrix list V1

V1[1,4]
    c1  c2  c3  c4
r1   1   2   3   4

. matrix list V2

V2[4,1]
    c1
r1   1
r2   2
r3   3
r4   4
```

There are a number of things you can do within the set of matrix commands that will generate an error. For example, ...

```
matrix drop X
matrix X = (1,2\3)
```

... generates the following error:

```
. matrix X = (1,2\3)
conformability error
r(503);
```

This is because if all the rows you have written do not share the same number of cells, STATA will fail to recognize it as a properly constructed matrix. This does bring up the question of what to do when you need to include missing or absent values within a matrix. We see, for instance, that merely using a comma, as one might do in R, doesn't work:

```
matrix drop X
matrix X = (1,2\3,)
```

```
. matrix X = (1,2\3,)
3, not found
r(111);
```

Just as in the conventional STATA interface, you will want to indicate missing data within the matrix by using a period:

```
matrix drop X
matrix X = (1,2\3,.)
```

This finally provides us with a properly functioning matrix:

```
X[2,2]
    c1  c2
r1   1   2
r2   3   .
```

Though the syntax provided by STATA itself suggests using letters as names for matrices, you are by no means required to do so. (You may have noticed the use of V1 and V2 to indicate the vertices in the previous example.) If you like you may use full names to indicate matrices. For example

```
matrix numbers = (1,2\3,4)
matrix list numbers
```

```
numbers[2,2]
    c1  c2
r1   1   2
r2   3   4
```

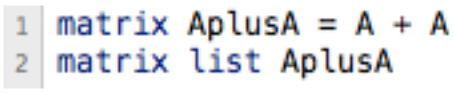
2. Matrix define

Thus far, I have shown how to create matrices in STATA using the `matrix input` command. There is another way, however, to create matrices in the conventional STATA program by using `matrix define`. The syntax for this command is

`matrix [define] A =` *`matrix_expression`*

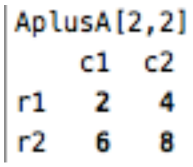
Note, however, that `define`, like `input`, does not have to be explicitly stated within the syntax. As such, you can create a matrix without having to specify whether you are using the `input` or `define` function.

The `matrix define` command essentially allows users to generate new matrices from the manipulation of previously constructed matrices. Let's use the first matrix we've generated to provide an example:



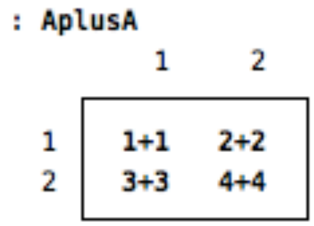
```
matrix AplusA = A + A
matrix list AplusA
```

This command — the use of the plus sign, essentially — tells STATA that you want to perform matrix addition. This provides us with the following result:



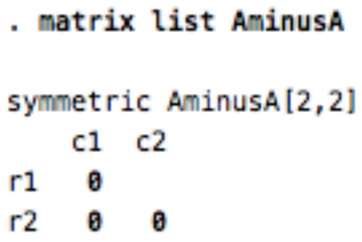
```
AplusA[2,2]
    c1  c2
r1   2   4
r2   6   8
```

With matrix addition, each cell in matrix 1 is added to the corresponding cell in matrix 2. What we're basically performing is this[1]:



```
: AplusA
          1     2

  1 |  1+1   2+2 |
  2 |  3+3   4+4 |
```

Likewise, it is possible to perform matrix subtraction using `matrix define` in a similarly intuitive way:



```
. matrix list AminusA

symmetric AminusA[2,2]
    c1  c2
r1   0
r2   0   0
```

It is also possible to perform matrix multiplication in STATA. Let's first create a new matrix to use within the command:

[1] To demonstrate how this works, I've provided a matrix of *strings* created via `mata` (which I assume is a portmanteau of "matrix" and "STATA"). Within `mata`, it is possible to edit matrices with a great deal more precision and options; it is important to note that the conventional matrix commands will not allow you to generate matrices of strings.

```
matrix B = (5,6,7\8,9,10)
matrix list B
```

```
B[2,3]
    c1  c2  c3
r1   5   6   7
r2   8   9  10
```

Now, let’s use matrix multiplication to generate a new matrix:

```
matrix AB = A*B
matrix list AB
```

Let’s see the results:

```
. matrix list AB

AB[2,3]
    c1  c2  c3
r1  21  24  25
r2  47  54  57
```

To understand how this matrix multiplication works, it is helpful to look at both matrices A and B as well as the process by which the matrix was created:

```
A[2,2]           B[2,3]
    c1  c2           c1  c2  c3
r1   1   2       r1   5   6   7
r2   3   4       r2   8   9  10
```

```
AB
            1          2          3

1      1*5+2*8    1*6+2*9    1*7+2*9
2      3*5+4*8    3*6+4*9    3*7+4*9
```

Keep in mind that when performing matrix addition and subtraction your matrices *must* have the exact same dimensions. In contrast, your matrices in matrix multiplication do not have to have the same dimensions, but the number of columns of the matrix functioning as the premultiplier (the first matrix mentioned within the syntax) must match the number of rows within the second matrix. For example, what if we were to use B rather than A as the premultiplier?

```
matrix BA = B*A
```

```
. matrix BA = B*A
conformability error
r(503);
```

We're confronted, once again, with a conformability error. That said, keep in mind that, while STATA will let you know when matrices do not fit together due to differences in dimension, it is possible for you to introduce errors into your analyses if matrices are the same dimensions and you do not specify the correct matrix as the premultiplier.

3. Creating network matrices

At this point, we've learned the basics of matrix construction and manipulation in STATA. Now I'd like to focus on matrix creation in a way amenable to enabling basic network analysis to occur. First, I'd like to focus upon *adjacency networks*. These are networks that represent the relationships between objects in a single mode — for instance, individuals.

Let's say that we wanted to generate an asymmetric matrix of friendship ties. This will basically represent every instance *ego* claimed *alter* as a friend and allows for differentiation within the social relation. (That is, that Person 1 claims Person 2 as a friend does not necessitate that Person 2 reciprocate.) Let's also assume for the purposes of this exercise that *ego* could not name *ego* as a friend. (This is a fairly common presupposition within social network analysis.)

So let's create some hypothetical network:

```
matrix relations = (0,0,1,1,0\1,0,1,1,0\1,0,0,1,0\1,1,0,0,1\0,1,0,0,0)
matrix list relations
```

```
relations[5,5]
    c1  c2  c3  c4  c5
r1   0   0   1   1   0
r2   1   0   1   1   0
r3   1   0   0   1   0
r4   1   1   0   0   1
r5   0   1   0   0   0
```

This looks great, but with social network data, we often want to *identify* the actual individuals. We can do this using the `rownames` and `colnames` functions:

```
matrix colnames relations = Attila Beksahn Cindy Dee Eliza
matrix rownames relations = Attila Beksahn Cindy Dee Eliza
matrix list relations
```

Keep in mind that this is an adjacency matrix — a matrix with one mode of objects. This is what allows us to use the same objects in both the `rownames` and `colnames` commands. This is probably obvious, but if you have a two-mode matrix of objects, you will want to include different names (as we will see in the next section). At any rate, this command provides us with a new matrix with object names attached:

```
relations[5,5]
          Attila  Beksahn    Cindy      Dee    Eliza
 Attila        0        0        1        1        0
Beksahn        1        0        1        1        0
  Cindy        1        0        0        1        0
    Dee        1        1        0        0        1
  Eliza        0        1        0        0        0
```

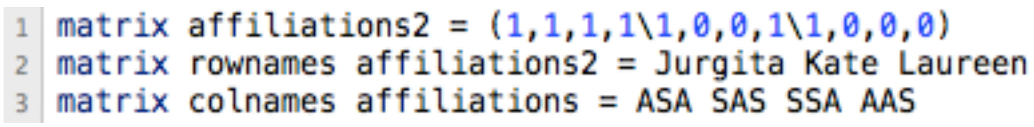
So next, let's try *affiliations* or *two-mode networks*. These are networks that have two different modes of objects. Let's say, for instance, that we wanted to look at the memberships of these individuals in four different professional organizations.

```
matrix affiliations = (1,1,1,0\1,1,0,0\1,1,1,1\1,0,1,0\1,0,0,0)
matrix rownames affiliations = Attila Beksahn Cindy Dee Eliza
matrix colnames affiliations = ASA SAS SSA AAS
matrix list affiliations
```

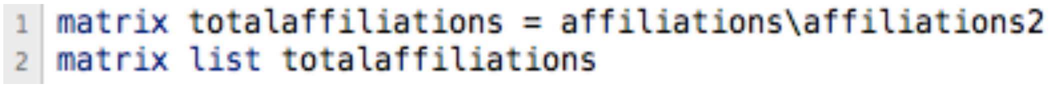
```
. matrix list affiliations

affiliations[5,4]
         ASA  SAS  SSA  AAS
 Attila    1    1    1    0
Beksahn    1    1    0    0
  Cindy    1    1    1    1
    Dee    1    0    1    0
  Eliza    1    0    0    0
```

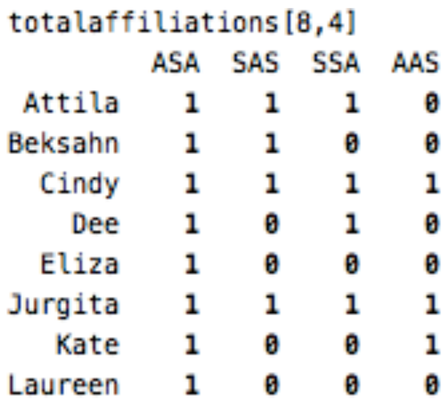
Here, we see that all of them belong to ASA, but membership in the other professional groups is somewhat less consistent.

I'd like to break in the middle of this discussion to demonstrate some commands for manipulating matrices we've already created. Let's say that after we created this small sample, we created some more data by looking at the professional memberships of some other people in the department:


```
matrix affiliations2 = (1,1,1,1\1,0,0,1\1,0,0,0)
matrix rownames affiliations2 = Jurgita Kate Laureen
matrix colnames affiliations = ASA SAS SSA AAS
```

So, now, we would like to append this new matrix to the existing relational dataset. We can add new rows by using the following syntax:


```
matrix totalaffiliations = affiliations\affiliations2
matrix list totalaffiliations
```


```
totalaffiliations[8,4]
         ASA  SAS  SSA  AAS
 Attila    1    1    1    0
Beksahn    1    1    0    0
  Cindy    1    1    1    1
    Dee    1    0    1    0
  Eliza    1    0    0    0
Jurgita    1    1    1    1
   Kate    1    0    0    1
Laureen    1    0    0    0
```

Similarly, it is possible to append new columns as well. Let's say in the course of our data collection we find a heretofore unknown professional organization that seems relevant to the professional development of our sample and we want to include it in our dataset. It's possible to do so using this syntax:

```
matrix forgotanorg = (0\0\1\0\1\0\0\0)
matrix colnames forgotanorg = SSS
matrix totalaffiliations = totalaffiliations,forgotanorg
matrix list totalaffiliations
```

The first two lines generate a new matrix and give it a name. The third line demonstrates how to append a new column. Rather than the backslash we used to append new rows, we use a comma to append new columns. Note that we do not even need to create a new matrix but can use "totalaffiliations" as both a source of data and as the ultimate dataset. Let's examine the results:

```
totalaffiliations[8,5]
         ASA  SAS  SSA  AAS  SSS
 Attila    1    1    1    0    0
Beksahn    1    1    0    0    0
  Cindy    1    1    1    1    1
    Dee    1    0    1    0    0
  Eliza    1    0    0    0    1
Jurgita    1    1    1    1    0
   Kate    1    0    0    1    0
Laureen    1    0    0    0    0
```

One particularly interesting aspect of affiliations networks is that they can be exploited to evince the *duality* between two orders (see Breiger 1974). In other words, an individual may be defined by the organizations to which he or she belongs, and, similarly, an organization may be defined by the individuals who claim membership to it. This duality allows us to generate two new matrices from the original affiliations matrix: one that represents relationships between individuals as mediated by groups and another that depicts relationships between groups as mediated by the people who claim membership therein.

First, let's move from the two-mode affiliations network to a one-mode network of individuals. In order to do this, we are going to need to generate the *transpose* of the existing matrix. We indicate we want the transpose of a matrix simply by adding an apostrophe to the end of an existing matrix.

```
matrix affiltranspose = totalaffiliations'
matrix list affiltranspose
```

The transpose of a matrix is one in which the rows and columns have been switched:

```
affiltranspose[5,8]
     Attila  Beksahn    Cindy      Dee    Eliza  Jurgita     Kate  Laureen
ASA       1        1        1        1        1        1        1        1
SAS       1        1        1        0        0        1        0        0
SSA       1        0        1        1        0        1        0        0
AAS       0        0        1        0        0        1        1        0
SSS       0        0        1        0        1        0        0        0
```

When we multiply the original matrix by its transpose, we exploit the duality of individuals and groups to demonstrate how groups mediate relations between individuals. (This is because individuals were in the rows in the first matrix.)

```
matrix PtoP = totalaffiliations * totalaffiliations'
matrix list PtoP
```

```
symmetric PtoP[8,8]
          Attila  Beksahn    Cindy      Dee    Eliza  Jurgita     Kate  Laureen
 Attila        3
Beksahn        2        2
  Cindy        3        2        5
    Dee        2        1        2        2
  Eliza        1        1        2        1        2
Jurgita        3        2        4        2        1        4
   Kate        1        1        2        1        1        2        2
Laureen        1        1        1        1        1        1        1        1
```

If we understand organizations as network foci, then we can understand them as potentially connecting individuals, so these relations are of substantive interest. We see that Attila and Beksahn, for instance, share membership in two organizations, whereas Cindy and Jurgita share membership in four organizations. The diagonal represents the number of organizations ego shares membership with him- or herself — in other words, the total number of organizations to which he or she belongs.

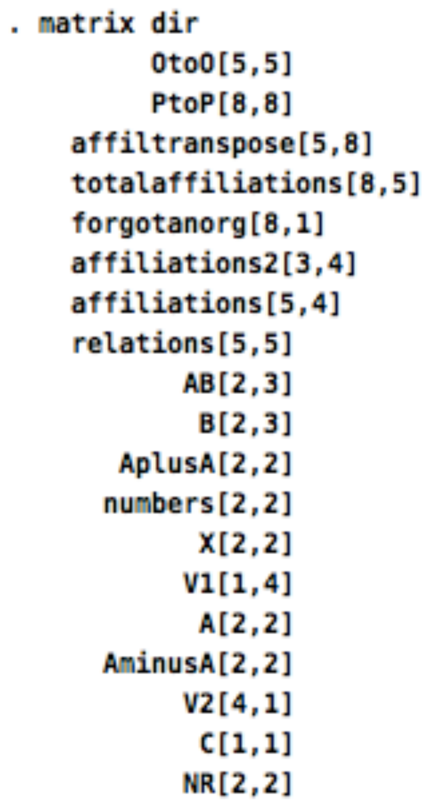
Let us also look at the relationships between the organizations. Researchers have historically looked at this type of data as representing the structure of the "organizational field" and to generate measures of interorganizational competition within organizational ecology. Here, we exploit the duality of individuals and groups to show how individuals mediate the interorganizational relations. (In this example, the transposed matrix is the premultiplier and therefore the organizations are in the rows.)

```
matrix OtoO = totalaffiliations' * totalaffiliations
matrix list OtoO
```

```
symmetric OtoO[5,5]
     ASA  SAS  SSA  AAS  SSS
ASA    8
SAS    4    4
SSA    4    3    4
AAS    3    2    2    3
SSS    2    1    1    1    2
```

We can interpret this matrix in a manner similar to the last. ASA shares four members with SAS, while SAS shares one member with SSS. Likewise, the diagonal represents the number of members of each organization within our sample.

Finally, I'd like to point out that all the matrices we have created at this point are not available to us in the places we would traditionally look for our data in STATA's data editor. To look at what matrices we've created thus far, you'll need to remember the `matrix dir` command. This command provides the list of matrices you've constructed, along with each matrix's dimensions.



```
. matrix dir
         OtoO[5,5]
         PtoP[8,8]
    affiltranspose[5,8]
    totalaffiliations[8,5]
    forgotanorg[8,1]
    affiliations2[3,4]
    affiliations[5,4]
    relations[5,5]
           AB[2,3]
            B[2,3]
       AplusA[2,2]
      numbers[2,2]
            X[2,2]
           V1[1,4]
            A[2,2]
      AminusA[2,2]
           V2[4,1]
            C[1,1]
           NR[2,2]
```