Title

stata.com

Permutation — An aside on permutation matrices and vectors

Description Syntax Remarks and examples Also see

Description

Permutation matrices are a special kind of orthogonal matrix that, via multiplication, reorder the rows or columns of another matrix. Permutation matrices cast the reordering in terms of multiplication.

Permutation vectors also reorder the rows or columns of another matrix, but they do it via subscripting. This alternative method of achieving the same end is, computerwise, more efficient, in that it uses less memory and less computer time.

The relationship between the two is shown below.

Syntax

Action	Permutation matrix notation	Permutation vector notation
permute rows	$B = P*A$	$B = A[p,.]$
permute columns	$B = A*P$	$B = A[.,p]$
unpermute rows	$B = P'A$	$B = A\ ;\ B[p,.] = A$ or $B = A$`[invorder(`p`),.]`
unpermute columns	$B = A*P'$	$B = A\ ;\ B[,.p] = A$ or $B = A$`[., invorder(`p`)]`

A *permutation matrix* is an $n \times n$ matrix that is a row (or column) permutation of the identity matrix.

A *permutation vector* is a $1 \times n$ or $n \times 1$ vector of the integers 1 through n.

The following permutation matrix and permutation vector are equivalent:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \iff p = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

Either can be used to permute the rows of

$$A = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} \text{ to produce } \begin{bmatrix} e & f & g & h \\ i & j & k & l \\ a & b & c & d \end{bmatrix}$$

and to permute the columns of

$$P = \begin{bmatrix} m & n & o \\ p & q & r \\ s & t & u \\ v & w & x \end{bmatrix} \text{ to produce } \begin{bmatrix} n & o & m \\ q & r & p \\ t & u & s \\ w & x & v \end{bmatrix}$$

Remarks and examples

stata.com

Remarks are presented under the following headings:

Permutation matrices

A permutation matrix is a square matrix whose rows are a permutation of the identity matrix. The following are the full set of all 2×2 permutation matrices:

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

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2}$$

Let P be an $n \times n$ permutation matrix. If $n \times m$ matrix A is premultiplied by P, the result is to reorder its rows. For example,

$$\begin{matrix} P & & A & & PA \end{matrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \end{bmatrix} \tag{3}$$

Above, we illustrated the reordering using square matrix A, but A did not have to be square.

If $m \times n$ matrix B is postmultiplied by P, the result is to reorder its columns. We illustrate using square matrix A again:

$$\begin{matrix} A & & P & & AP \end{matrix}$$
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 2 \\ 6 & 4 & 5 \\ 9 & 7 & 8 \end{bmatrix} \tag{4}$$

Say that we reorder the rows of A by forming PA. Obviously, we can unreorder the rows by forming $P^{-1}PA$. Because permutation matrices are orthogonal, their inverses are equal to their transpose. Thus the inverse of the permutation matrix `(0,1,0\0,0,1\1,0,0)` we have been using is `(0,0,1\1,0,0\0,1,0)`. For instance, taking our results from (3)

$$\begin{matrix} P' & * & PA & = & A \end{matrix}$$
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{3′}$$

Allow us to summarize:

1. A permutation matrix P is a square matrix whose rows are a permutation of the identity matrix.
2. PA = a row permutation of A.
3. AP = a column permutation of A.
4. The inverse permutation is given by P'.
5. $P'PA = A$.
6. $APP' = A$.

How permutation matrices arise

Some of Mata's matrix functions implicitly permute the rows (or columns) of a matrix. For instance, the LU decomposition of matrix A is defined as

$$A = LU$$

where L is lower triangular and U is upper triangular. For any matrix A, one can solve for L and U, and Mata has a function that will do that (see [M-5] **lud( )**). However, Mata's function does not solve the problem as stated. Instead, it solves

$$P'A = LU$$

where P' is a permutation matrix that Mata makes up! Just to be clear; Mata's function solves for L and U, but for a row permutation of A, not A itself, although the function does tell you what permutation it chose (the function returns L, U, and P). The function permutes the rows because, that way, it can produce a more accurate answer.

You will sometimes read that a function engages in pivoting. What that means is that, rather than solving the problem for the matrix as given, it solves the problem for a permutation of the original matrix, and the function chooses the permutation in a way to minimize numerical roundoff error. Functions that do this invariably return the permutation matrix along with the other results, because you are going to need it.

For instance, one use of LU decomposition is to calculate inverses. If $A = LU$ then $A^{-1} = U^{-1}L^{-1}$. Calculating the inverses of triangular matrices is an easy problem, so one recipe for calculating inverses is

1. decompose A into L and U,
2. calculate U^{-1},
3. calculate L^{-1}, and
4. multiply the results together.

That would be the solution except that the LU decomposition function does not decompose A into L and U; it decomposes $P'A$, although the function does tell us P. Thus we can write,

$$\begin{aligned} P'A &= LU \\ A &= PLU \qquad (\text{remember } P'^{-1} = P) \\ A^{-1} &= U^{-1}L^{-1}P' \end{aligned}$$

Thus the solution to our problem is to use the U and L just as we planned—calculate $U^{-1}L^{-1}$—and then make a column permutation of that, which we can do by postmultiplying by P'.

There is, however, a detail that we have yet to reveal to you: Mata's LU decomposition function does not return P, the permutation matrix. It instead returns p, a permutation vector equivalent to P, and so the last step—forming the column permutation by postmultiplying by P'—is done differently. That is the subject of the next section.

Permutation vectors are more efficient than permutation matrices, but you are going to discover that they are not as mathematically transparent. Thus when working with a function that returns a permutation vector—when working with a function that permutes the rows or columns—think in terms of permutation matrices and then translate back into permutation vectors.

Permutation vectors

Permutation vectors are used with Mata's subscripting operator, so before explaining permutation vectors, let's understand subscripting.

Not surprisingly, Mata allows subscripting. Given matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Mata understands that

$$A\texttt{[2,3]} = 6$$

Mata also understands that if one or the other subscript is specified as `.` (missing value), the entire column or row is to be selected:

$$A\texttt{[.,3]} = \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}$$

$$A\texttt{[2,.]} = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

Mata also understands that if a vector is specified for either subscript

$$A\texttt{[}\begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix}\texttt{, .]} = \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{bmatrix}$$

In Mata, we would actually write the above as `A[(2\3\2),.]`, and Mata would understand that we want the matrix made up of rows 2, 3, and 2 of A and all columns. Similarly, we can request all rows and columns 2, 3, and 2:

$$A[.,(2,3,2)] = \begin{bmatrix} 2 & 3 & 2 \\ 5 & 6 & 5 \\ 8 & 9 & 8 \end{bmatrix}$$

In the above, we wrote $(2,3,2)$ as a row vector because it seems more logical that way, but we could just as well have written `A[.,(2\3\2)]`. In subscripting, Mata does not care whether the vectors are rows or columns.

In any case, we can use a vector of indices inside Mata's subscripts to select rows and columns of a matrix, and that means we can permute them. All that is required is that the vector we specify contain a permutation of the integers 1 through n because, otherwise, we would repeat some rows or columns and omit others.

A permutation vector is an $n \times 1$ or $1 \times n$ vector containing a permutation of the integers 1 through n. For example, the permutation vector equivalent to the permutation matrix

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

is

$$p = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

p can be used with subscripting to permute the rows of A

$$A[p,.] = \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 1 & 2 & 3 \end{bmatrix}$$

and similarly, `A[.,p]` would permute the columns.

Also subscripting can be used on the left-hand side of the equal-sign assignment operator. So far, we have assumed that the subscripts are on the right-hand side of the assignment operator, such as

```
B = A[p,.]
```

We have learned that if `p = (2\3\1)` (or `p = (2,3,1)`), the result is to copy the second row of A to the first row of B, the third row of A to the second row of B, and the first row of A to the third row of B. Coding

```
B[p,.] = A
```

does the inverse: it copies the first row of A to the second row of B, the second row of A to the third row of B, and the third row of A to the first row of B. `B[p,.] = A` really is the inverse of `C = A[p,.]` in that, if we code

```
C = A[p,.]
B[p,.] = C
```

B will be equal to *A*, and if we code

```
C[p,.] = A
B = C[p,.]
```

B will also be equal to *A*.

There is, however, one pitfall that you must watch for when using subscripts on the left-hand side: the matrix on the left-hand side must already exist and it must be of the appropriate (here same) dimension. Thus when performing the inverse copy, it is common to code

```
B = C
B[p,.] = C
```

The first line is not unnecessary; it is what ensures that *B* exists and is of the proper dimension, although we could just as well code

```
B = J(rows(C), cols(C), .)
B[p,.] = C
```

The first construction is preferred because it ensures that *B* is of the same type as *C*. If you really like the second form, you should code

```
B = J(rows(C), cols(C), missingof(C))
B[p,.] = C
```

Going back to the preferred code

```
B = C
B[p,.] = C
```

some programmers combine it into one statement:

```
(B=C)[p,.] = C
```

Also Mata provides an `invorder(`*p*`)` (see [M-5] **invorder( )**) that will return an inverted *p* appropriate for use on the right-hand side, so you can also code

```
B = C[invorder(p),.]
```

Also see

[M-5] **invorder( )** — Permutation vector manipulation

[M-1] **Intro** — Introduction and advice