Title

stata.com

_equilrc( ) — Row and column equilibration

Description

`_equilrc(`*A*`, `*r*`, `*c*`)` performs row and column equilibration (balancing) on matrix *A*, returning the equilibrated matrix in *A*, the row-scaling factors in *r*, and the column-scaling factors in *c*.

`_equilr(`*A*`, `*r*`)` performs row equilibration on matrix *A*, returning the row-equilibrated matrix in *A* and the row-scaling factors in *r*.

`_equilc(`*A*`, `*c*`)` performs column equilibration on matrix *A*, returning the column-equilibrated matrix in *A* and the column-scaling factors in *c*.

`_perhapsequilrc(`*A*`, `*r*`, `*c*`)` performs row and/or column equilibration on matrix *A*—as is necessary and which decision is made by `_perhapsequilrc()`—returning the equilibrated matrix in *A*, the row-scaling factors in *r*, the column-scaling factors in *c*, and returning 0 (no equilibration performed), 1 (row equilibration performed), 2 (column equilibration performed), or 3 (row and column equilibration performed).

`_perhapsequilr(`*A*`, `*r*`)` performs row equilibration on matrix *A*—if necessary and which decision is made by `_perhapsequilr()`—returning the equilibrated matrix in *A*, the row-scaling factors in *r*, and returning 0 (no equilibration performed) or 1 (row equilibration performed).

`_perhapsequilc(`*A*`, `*c*`)` performs column equilibration on matrix *A*—if necessary and which decision is made by `_perhapsequilc()`—returning the equilibrated matrix in *A*, the column-scaling factors in *c*, and returning 0 (no equilibration performed) or 1 (column equilibration performed).

`rowscalefactors(`*A*`)` returns the row-scaling factors of *A*.

`colscalefactors(`*A*`)` returns the column-scaling factors of *A*.

Syntax

void	`_equilrc(`*numeric matrix A*`, `*r*`, `*c*`)`
void	`_equilr(`*numeric matrix A*`, `*r*`)`
void	`_equilc(`*numeric matrix A*`, `*c*`)`
real scalar	`_perhapsequilrc(`*numeric matrix A*`, `*r*`, `*c*`)`
real scalar	`_perhapsequilr(`*numeric matrix A*`, `*r*`)`
real scalar	`_perhapsequilc(`*numeric matrix A*`, `*c*`)`
real colvector	`rowscalefactors(`*numeric matrix A*`)`
real rowvector	`colscalefactors(`*numeric matrix A*`)`

The types of *r* and *c* are irrelevant because they are overwritten.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

Equilibration (also known as balancing) takes a matrix with poorly scaled rows and columns, such as

```
                 1              2
    ┌──────────────────────────────┐
1   │  1.00000e+10   5.00000e+10   │
2   │  2.00000e-10   8.00000e-10   │
    └──────────────────────────────┘
```

and produces a related matrix, such as

```
          1      2
    ┌─────────────┐
1   │   .2     1  │
2   │  .25     1  │
    └─────────────┘
```

that will yield improved accuracy in, for instance, the solution to linear systems. The improved matrix above has been row equilibrated. All we did was find the maximum of each row of the original and then divide the row by its maximum. If we were to take the result and repeat the process on the columns—divide each column by the column's maximum—we would obtain

```
          1      2
    ┌─────────────┐
1   │  .8     1   │
2   │   1     1   │
    └─────────────┘
```

which is the row-and-column equilibrated form of the original matrix.

In terms of matrix notation, equilibration can be thought about in terms of multiplication by diagonal matrices. The row-equilibrated form of A is RA, where R contains the reciprocals of the row maximums on its diagonal. The column-equilibrated form of A is AC, where C contains the reciprocals of the column maximums on its diagonal. The row-and-column equilibrated form of A is RAC, where R contains the reciprocals of the row maximums of A on its diagonal, and C contains the reciprocals of the column maximums of RA on its diagonal.

Say we wished to find the solution x to

$$Ax = b$$

We could compute the solution by solving for y in the equilibrated system

$$(RAC)y = Rb$$

and then setting

$$x = Cy$$

Thus routines that perform equilibration need to return to you, in some fashion, R and C. The routines here do that by returning r and c, the reciprocals of the maximums in vector form. You could obtain R and C from them by coding

$$R = \texttt{diag}(r)$$
$$C = \texttt{diag}(c)$$

but that is not in general necessary, and it is wasteful of memory. In code, you will need to multiply by R and C, and you can do that using the `:*` operator with r and c:

$$RA \leftrightarrow r\texttt{:*}A$$
$$AC \leftrightarrow A\texttt{:*}c$$
$$RAC \leftrightarrow r\texttt{:*}A\texttt{:*}c$$

Is equilibration necessary?

Equilibration is not a panacea. Equilibration can reduce the condition number of some matrices and thereby improve the accuracy of the solution to linear systems, but equilibration is not guaranteed to reduce the condition number, and counterexamples exist in which equilibration actually decreases the accuracy of the solution. That said, you have to look long and hard to find such examples.

Equilibration is not especially computationally expensive, but neither is it cheap, especially when you consider the extra computational costs of using the equilibrated matrices. In statistical contexts, equilibration may buy you little because matrices are already nearly equilibrated. Data analysts know variables should be on roughly the same scale, and observations are assumed to be draws from an underlying distribution. The computational cost of equilibration is probably better spent somewhere else. For instance, consider obtaining regression estimates from $X'X$ and $X'y$. The gain from equilibrating $X'X$ and $X'y$, or even from equilibrating the original X matrix, is nowhere near that from the gain to be had in removal of the means before $X'X$ and $X'y$ are formed.

In the example in the previous section, we showed you a matrix that assuredly benefited from equilibration. Even so, after row equilibration, column equilibration was unnecessary. It is often the case that solely row or column equilibration is sufficient, and in those cases, although the extra equilibration will do no numerical harm, it will burn computer cycles. And, as we have already argued, some matrices do not need equilibration at all.

Thus programmers who want to use equilibration and obtain the best speed possible examine the matrix and on that basis perform (1) no equilibration, (2) row equilibration, (3) column equilibration, or (4) both. They then write four branches in their subsequent code to handle each case efficiently.

In terms of determining whether equilibration is necessary, measures of the row and column condition can be obtained from `min(`r`)/max(`r`)` and `min(`c`)/max(`c`)`, where r and c are the scaling factors (reciprocals of the row and column maximums). If those measures are near 1 (LAPACK uses $\geq .1$), then equilibration can be skipped.

There is also the issue of the overall scale of the matrix. In theory, the overall scale should not matter, but many packages set tolerances in absolute rather than relative terms, and so overall scale does matter. In most of Mata's other functions, relative scales are used, but provisions are made so that you can specify tolerances in relative or absolute terms. In any case, LAPACK uses the rule that equilibration is necessary if the matrix is too small (its maximum value is less than `epsilon(100)`, approximately 2.22045e–14) or too large (greater than `1/epsilon(100)`, approximately 4.504e+13).

To summarize,

1. In statistical applications, we believe that equilibration burns too much computer time and adds too much code complexity for the extra accuracy it delivers. This is a judgment call, and we would probably recommend the use of equilibration were it computationally free.

2. If you are going to use equilibration, there is nothing numerically wrong with simply equilibrating matrices in all cases, including those in which equilibration is not necessary. The advantages of this is that you will gain the precision to be had from equilibration while still keeping your code reasonably simple.

3. If you wish to minimize execution time, then you want to perform the minimum amount of equilibration possible and write code to deal efficiently with each case: (1) no equilibration, (2) row equilibration, (3) column equilibration, and (4) row and column equilibration. The defaults used by LAPACK and incorporated in Mata's `_perhapsequil*()` routines are

    a. Perform row equilibration if `min(`r`)/max(`r`)` $<$ .1, or if `min(abs(`A`))` $<$ `epsilon(100)`, or if `min(abs(`A`))` $>$ `1/epsilon(100)`.

    b. After performing row equilibration, perform column equilibration if `min(`c`)/max(`c`)` $<$ .1, where c is calculated on r`*`A, the row-equilibrated A, if row equilibration was performed.

The _equil*() family of functions

The `_equil*()` family of functions performs equilibration as follows:

`_equilrc(`A`, `r`, `c`)` performs row equilibration followed by column equilibration; it returns in r and in c the row- and column-scaling factors, and it modifies A to be the fully equilibrated matrix r`:*`A`:*`c.

`_equilr(`A`, `r`)` performs row equilibration only; it returns in r the row-scaling factors, and it modifies A to be the row-equilibrated matrix r`:*`A.

`_equilc(`A`, `c`)` performs column equilibration only; it returns in c the row-scaling factors, and it modifies A to be the row-equilibrated matrix A`:*`c.

Here is code to solve $Ax = b$ using the fully equilibrated form, which damages A in the process:

```
_equilrc(A, r, c)
x = c:*lusolve(A, r:*b)
```

The _perhapsequil*() family of functions

The `_perhapsequil*()` family of functions mirrors `_equil*()`, except that these functions apply the rules mentioned above for whether equilibration is necessary.

Here is code to solve $Ax = b$, which may damage A in the process:

```
result = _perhapsequilrc(A, r, c)
if (result==0)          x = lusolve(A, b)
else if (result==1)     x = lusolve(A, r:*b)
else if (result==2)     x = c:*lusolve(A, b)
else if (result==3)     x = c:*lusolve(A, r:*b)
```

As a matter of fact, the _perhapsequil*() family returns a vector of 1s when equilibration is not performed, so you could code

```
(void) _perhapsequilrc(A, r, c)
x = c:*lusolve(A, r:*b)
```

but that would be inefficient.

rowscalefactors() and colscalefactors()

rowscalefactors(*A*) and colscalefactors(*A*) return the scale factors (reciprocals of row and column maximums) to perform row and column equilibration. These functions are used by the other functions above and are provided for those who wish to write their own equilibration routines.

Conformability

_equilrc(*A*, *r*, *c*):

input:			
	A:	$m \times n$	
output:			
	A:	$m \times n$	
	r:	$m \times 1$	
	c:	$1 \times n$	

_equilr(*A*, *r*):

input:			
	A:	$m \times n$	
output:			
	A:	$m \times n$	
	r:	$m \times 1$	

_equilc(*A*, *c*):

input:			
	A:	$m \times n$	
output:			
	A:	$m \times n$	
	c:	$1 \times n$	

_perhapsequilrc(*A*, *r*, *c*):

input:			
	A:	$m \times n$	
output:			
	A:	$m \times n$	(unmodified if *result* $= 0$)
	r:	$m \times 1$	
	c:	$1 \times n$	
	result:	1×1	

`_perhapsequilr(`*A*`, `*r*`)`:

input:

A:	$m \times n$	

output:

A:	$m \times n$	(unmodified if *result* $= 0$)
r:	$m \times 1$	
result:	1×1	

`_perhapsequilc(`*A*`, `*c*`)`:

input:

A:	$m \times n$	

output:

A:	$m \times n$	(unmodified if *result* $= 0$)
c:	$1 \times n$	
result:	1×1	

`rowscalefactors(`*A*`)`:

A:	$m \times n$
result:	$m \times 1$

`colscalefactors(`*A*`)`:

A:	$m \times n$
result:	$1 \times n$

Diagnostics

Scale factors used and returned by all functions are calculated by `rowscalefactors(`*A*`)` and `colscalefactors(`*A*`)`. The functions are defined as `1:/rowmaxabs(`*A*`)` and `1:/colmaxabs(`*A*`)`, with missing values changed to 1. Thus rows or columns that contain missing or are entirely zero are defined to have scale factors of 1.

Equilibration functions do not equilibrate rows or columns that contain missing or all zeros.

The `_equil*()` functions convert *A* to an array if *A* was a view. The Stata dataset is not changed.

The `_perhapsequil*()` functions convert *A* to an array if *A* was a view and returned is nonzero. The Stata dataset is not changed.

Also see

[M-4] **matrix** — Matrix functions