Title

stata.com

eigensystem( ) — Eigenvectors and eigenvalues

Description

These routines calculate eigenvectors and eigenvalues of square matrix *A*.

`eigensystem(`*A*`,` *X*`,` *L*`,` *rcond*`,` *nobalance*`)` calculates eigenvectors and eigenvalues of a general, real or complex, square matrix *A*. Eigenvectors are returned in *X* and eigenvalues in *L*. The remaining arguments are optional:

1. If *rcond* is not specified, then reciprocal condition numbers are not returned in *rcond*.

   If *rcond* is specified and contains a value other than 0 or missing—*rcond* = 1 is suggested—in *rcond* will be placed a vector of the reciprocals of the condition numbers for the eigenvalues. Each element of the new *rcond* measures the accuracy to which the corresponding eigenvalue has been calculated; large numbers (numbers close to 1) are better and small numbers (numbers close to 0) indicate inaccuracy; see *Eigenvalue condition* below.

2. If *nobalance* is not specified, balancing is performed to obtain more accurate results.

   If *nobalance* is specified and is not zero nor missing, balancing is not used. Results are calculated more quickly, but perhaps a little less accurately; see *Balancing* below.

`lefteigensystem(`*A*`,` *X*`,` *L*`,` *rcond*`,` *nobalance*`)` mirrors `eigensystem()`, the difference being that `lefteigensystem()` solves for left eigenvectors solving *XA* = `diag(`*L*`)*`*X* instead of right eigenvectors solving *AX* = *X*`*diag(`*L*`)`.

`eigenvalues(`*A*`,` *rcond*`,` *nobalance*`)` returns the eigenvalues of square matrix *A*; the eigenvectors are not calculated. Arguments *rcond* and *nobalance* are optional.

`symeigensystem(`*A*`,` *X*`,` *L*`)` and `symeigenvalues(`*A*`)` mirror `eigensystem()` and `eigenvalues()`, the difference being that *A* is assumed to be symmetric (Hermitian). The eigenvalues returned are real. (Arguments *rcond* and *nobalance* are not allowed; *rcond* because symmetric matrices are inherently well conditioned; *nobalance* because it is unnecessary.)

The underscore routines mirror the routines of the same name without underscores, the difference being that *A* is damaged during the calculation and so the underscore routines use less memory.

`_eigen_la()` is the interface to the Netlib LAPACK routines used to implement the above functions.

`_symeigen_la()` is the interface to the LAPACK routines used to implement the above functions. It uses MKL LAPACK by default.

Direct use of `_eigen_la()` and `_symeigen_la()` is not recommended.

`_eigengelapacke()` is the interface to the MKL LAPACK routines that are used by default to compute the eigenvalues, eigenvectors, and condition numbers.

Syntax

```
void                  eigensystem(A, X, L [, rcond [, nobalance]])
void                lefteigensystem(A, X, L [, rcond [, nobalance]])
complex rowvector       eigenvalues(A       [, rcond [, nobalance]])

void                 symeigensystem(A, X, L)
real rowvector       symeigenvalues(A)

void                  _eigensystem(A, X, L [, rcond [, nobalance]])
void                _lefteigensystem(A, X, L [, rcond [, nobalance]])
complex rowvector       _eigenvalues(A       [, rcond [, nobalance]])
void                 _symeigensystem(A, X, L)
real rowvector       _symeigenvalues(A)
```

where inputs are

A:	*numeric matrix*
rcond:	*real scalar* (whether *rcond* desired)
nobalance:	*real scalar* (whether to suppress balancing)

and outputs are

X:	*numeric matrix* of eigenvectors
L:	*numeric vector* of eigenvalues
rcond:	*real vector* of reciprocal condition numbers

The columns of *X* will contain the eigenvectors except when using `_lefteigensystem()`, in which case the rows of *X* contain the eigenvectors.

The following routines are used in implementing the above routines:

real scalar `_eigen_la(`*real scalar todo*`,` *numeric matrix A*`,` *X*`,` *L*`,` *real scalar rcond*`,`
real scalar nobalance`)`

real scalar `_symeigen_la(`*real scalar todo*`,` *numeric matrix A*`,` *X*`,` *L*`)`

The following routine is an interface to the LAPACK routines that compute the eigenvalues and eigenvectors:

real scalar `_eigengelapacke(`*real scalar todo*`,` *numeric matrix A*`,` *X*`,` *L*`,`
real scalar rcond`,` *real scalar balancing*`)`

Remarks and examples

stata.com

Remarks are presented under the following headings:

Eigenvalues and eigenvectors

A scalar λ is said to be an eigenvalue of square matrix A: $n \times n$ if there is a nonzero column vector x: $n \times 1$ (called the eigenvector) such that

$$Ax = \lambda x \tag{1}$$

(1) can also be written as

$$(A - \lambda I)x = 0$$

where I is the $n \times n$ identity matrix. A nontrivial solution to this system of n linear homogeneous equations exists if and only if

$$\det(A - \lambda I) = 0 \tag{2}$$

This nth degree polynomial in λ is called the characteristic polynomial or characteristic equation of A, and the eigenvalues λ are its roots, also known as the characteristic roots.

There are, in fact, n solutions (λ_i, x_i) that satisfy (1)—although some can be repeated—and we can compactly write the full set of solutions as

$$AX = X * \mathrm{diag}(L) \tag{3}$$

where

$$X = (x_1, x_2, \ldots) \qquad (X : n \times n)$$
$$L = (\lambda_1, \lambda_2, \ldots) \qquad (L : 1 \times n)$$

For instance,

```
: A = (1, 2 \ 9, 4)
: X = .
: L = .
: eigensystem(A, X, L)
: X
                  1             2
  +-------------------------------+
1 |  -.316227766   -.554700196    |
2 |  -.948683298    .832050294    |
  +-------------------------------+

: L
       1    2
  +-----------+
1 |   7   -2  |
  +-----------+
```

The first eigenvalue is 7, and the corresponding eigenvector is $(-0.316 \setminus -0.949)$. The second eigenvalue is -2, and the corresponding eigenvector is $(-0.555 \setminus 0.832)$.

In general, eigenvalues and vectors can be complex even if A is real.

Left eigenvectors

What we have defined above is properly known as the right-eigensystem problem:

$$Ax = \lambda x \tag{1}$$

In the above, x is a column vector. The left-eigensystem problem is to find the row vector x satisfying

$$xA = \lambda x \tag{1'}$$

The eigenvalue λ is the same in (1) and (1′), but x can differ.

The n solutions (λ_i, x_i) that satisfy (1′) can be compactly written as

$$XA = \text{diag}(L) * X \tag{3'}$$

where

$$X = \underset{n\times n}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}} \qquad L = \underset{n\times 1}{\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}}$$

For instance,

```
: A = (1, 2 \ 9, 4)
: X = .
: L = .
: lefteigensystem(A, X, L)
: X
                  1             2
    ┌──────────────────────────────┐
  1 │  -.832050294   -.554700196   │
  2 │  -.948683298    .316227766   │
    └──────────────────────────────┘

: L
          1
    ┌──────┐
  1 │   7  │
  2 │  -2  │
    └──────┘
```

The first eigenvalue is 7, and the corresponding eigenvector is $(-0.832, -0.555)$. The second eigenvalue is -2, and the corresponding eigenvector is $(-0.949, 0.316)$.

The eigenvalues are the same as in the previous example; the eigenvectors are different.

Symmetric eigensystems

Below we use the term symmetric to encompass Hermitian matrices, even when we do not emphasize the fact.

Eigensystems of symmetric matrices are conceptually no different from general eigensystems, but symmetry introduces certain simplifications:

1. The eigenvalues associated with symmetric matrices are real, whereas those associated with general matrices may be real or complex.
2. The eigenvectors associated with symmetric matrices—which may be real or complex—are orthogonal.
3. The left and right eigenvectors of symmetric matrices are transposes of each other.
4. The eigenvectors and eigenvalues of symmetric matrices are more easily, and more accurately, computed.

For item 3, let us begin with the right-eigensystem problem:

$$AX = X * \text{diag}(L)$$

Taking the transpose of both sides results in

$$X'A = \text{diag}(L) * X'$$

because $A = A'$ if A is symmetric (Hermitian).

`symeigensystem(`A`,` X`,` L`)` calculates right eigenvectors. To obtain the left eigenvectors, you simply transpose X.

Normalization and order

If x is a solution to

$$Ax = \lambda x$$

then so is cx, c: 1×1, $c \neq 0$.

The eigenvectors returned by the above routines are scaled to have length (norm) 1.

The eigenvalues are combined and returned in a vector (L) and the eigenvectors in a matrix (X). The eigenvalues are ordered from largest to smallest in absolute value (or, if the eigenvalues are complex, by modulus). The eigenvectors are ordered to correspond to the eigenvalues.

Eigenvalue condition

Optional argument *rcond* may be specified as a value other than 0 or missing—*rcond* = 1 is suggested—and then *rcond* will be filled in with a vector containing the reciprocals of the condition numbers for the eigenvalues. Each element of *rcond* measures the accuracy with which the corresponding eigenvalue has been calculated; large numbers (numbers close to 1) are better and small numbers (numbers close to 0) indicate inaccuracy.

The reciprocal condition number is calculated as `abs(y*x)`, where y: $1 \times n$ is the left eigenvector and x: $n \times 1$ is the corresponding right eigenvector. Since y and x each have norm 1, `abs(y*x)` = `abs(cos(theta))`, where *theta* is the angle made by the vectors. Thus $0 \leq$ `abs(y*x)` ≤ 1. For symmetric matrices, y^*x will equal 1. It can be proved that `abs(y*x)` is the reciprocal of the condition number for a simple eigenvalue, and so it turns out that the sensitivity of the eigenvalue to a perturbation is a function of how far the matrix is from symmetric on this scale.

Requesting that *rcond* be calculated increases the amount of computation considerably.

Balancing

By default, balancing is performed for general matrices. Optional argument *nobalance* allows you to suppress balancing.

Balancing is related to row-and-column equilibration; see [M-5] **_equilrc( )**. Here, however, a diagonal matrix D is found such that DAD^{-1} is better balanced, the eigenvectors and eigenvalues for DAD^{-1} are extracted, and then the eigenvectors and eigenvalues are adjusted by D so that they reflect those for the original A matrix.

There is no gain from these machinations when A is symmetric, so the symmetric routines do not have a *nobalance* argument.

Some additional balancing modes are available with `_eigengelapacke()`. It allows for permuting the rows and columns of the matrix in addition to scaling by a diagonal matrix D.

In `_eigengelapacke(...,` *real scalar balancing*`)`, if balancing is 0, then no balancing is performed; if balancing is 1, only scaling with a diagonal matrix D is performed; if balancing is 2, only permutation of the rows and columns is performed; in all other cases, both scaling and permutation are performed.

eigensystem( ) and eigenvalues( )

1. Use $L =$ `eigenvalues(`A`)` and `eigensystem(`A`, `X`, `L`)` for general matrices A.

2. Use $L =$ `eigenvalues(`A`)` when you do not need the eigenvectors; it will save both time and memory.

3. The eigenvalues returned by $L =$ `eigenvalues(`A`)` and by `eigensystem(`A`, `X`, `L`)` are of storage type complex even if the eigenvalues are real (that is, even if `Im(`L`)==0`). If the eigenvalues are known to be real, you can save computer memory by subsequently coding

   ```
   L = Re(L)
   ```

   If you wish to test whether the eigenvalues are real, examine `mreldifre(`L`)`; see [M-5] **reldif( )**.

4. The eigenvectors returned by `eigensystem(`A`, `X`, `L`)` are of storage type complex even if the eigenvectors are real (that is, even if `Im(`X`)==0`). If the eigenvectors are known to be real, you can save computer memory by subsequently coding

   ```
   X = Re(X)
   ```

   If you wish to test whether the eigenvectors are real, examine `mreldifre(`X`)`; see [M-5] **reldif( )**.

5. If you are using `eigensystem(A, X, L)` interactively (outside a program), X and L must be predefined. Type

```
: eigensystem(A, X=., L=.)
```

lefteigensystem( )

What was just said about `eigensystem()` applies equally well to `lefteigensystem()`.

If you need only the eigenvalues, use $L =$ `eigenvalues(A)`. The eigenvalues are the same for both left and right systems.

symeigensystem( ) and symeigenvalues( )

1. Use $L =$ `symeigenvalues(A)` and `symeigensystem(A, X, L)` for symmetric or Hermitian matrices A.
2. Use $L =$ `symeigenvalues(A)` when you do not need the eigenvectors; it will save both time and memory.
3. The eigenvalues returned by $L =$ `symeigenvalues(A)` and by `symeigensystem(A, X, L)` are of storage type real. Eigenvalues of symmetric and Hermitian matrices are always real.
4. The eigenvectors returned by `symeigensystem(A, X, L)` are of storage type real when A is of storage type real and of storage type complex when A is of storage type complex.
5. If you are using `symeigensystem(A, X, L)` interactively (outside a program), X and L must be predefined. Type

```
: symeigensystem(A, X=., L=.)
```

Conformability

`eigensystem(`*A*`, `*X*`, `*L*`, `*rcond*`, `*nobalance*`)`:

input:			
	A:	$n \times n$	
	rcond:	1×1	(optional, specify as 1 to obtain *rcond*)
	nobalance:	1×1	(optional, specify as 1 to prevent balancing)
output:			
	X:	$n \times n$	(columns contain eigenvectors)
	L:	$1 \times n$	
	rcond:	$1 \times n$	(optional)
	result:	*void*	

`lefteigensystem(`*A*`, `*X*`, `*L*`, `*rcond*`, `*nobalance*`)`:

input:

A:	$n \times n$	
rcond:	1×1	(optional, specify as 1 to obtain *rcond*)
nobalance:	1×1	(optional, specify as 1 to prevent balancing)

output:

X:	$n \times n$	(rows contain eigenvectors)
L:	$n \times 1$	
rcond:	$n \times 1$	(optional)
result:	*void*	

`eigenvalues(`*A*`, `*rcond*`, `*nobalance*`)`:

input:

A:	$n \times n$	
rcond:	1×1	(optional, specify as 1 to obtain *rcond*)
nobalance:	1×1	(optional, specify as 1 to prevent balancing)

output:

rcond:	$1 \times n$	(optional)
result:	$1 \times n$	(contains eigenvalues)

`symeigensystem(`*A*`, `*X*`, `*L*`)`:

input:

A:	$n \times n$	

output:

X:	$n \times n$	(columns contain eigenvectors)
L:	$1 \times n$	
result:	*void*	

`symeigenvalues(`*A*`)`:

input:

A:	$n \times n$	

output:

result:	$1 \times n$	(contains eigenvalues)

`_eigensystem(`*A*`, `*X*`, `*L*`, `*rcond*`, `*nobalance*`)`:

input:

A:	$n \times n$	
rcond:	1×1	(optional, specify as 1 to obtain *rcond*)
nobalance:	1×1	(optional, specify as 1 to prevent balancing)

output:

A:	0×0	
X:	$n \times n$	(columns contain eigenvectors)
L:	$1 \times n$	
rcond:	$1 \times n$	(optional)
result:	*void*	

_lefteigensystem(*A*, *X*, *L*, *rcond*, *nobalance*):

input:

A:	$n \times n$	
rcond:	1×1	(optional, specify as 1 to obtain *rcond*)
nobalance:	1×1	(optional, specify as 1 to prevent balancing)

output:

A:	0×0	
X:	$n \times n$	(rows contain eigenvectors)
L:	$n \times 1$	
rcond:	$n \times 1$	(optional)
result:	*void*	

_eigenvalues(*A*, *rcond*, *nobalance*):

input:

A:	$n \times n$	
rcond:	1×1	(optional, specify as 1 to obtain *rcond*)
nobalance:	1×1	(optional, specify as 1 to prevent balancing)

output:

A:	0×0	
rcond:	$1 \times n$	(optional)
result:	$1 \times n$	(contains eigenvalues)

_symeigensystem(*A*, *X*, *L*):

input:

A:	$n \times n$	

output:

A:	0×0	
X:	$n \times n$	(columns contain eigenvectors)
L:	$1 \times n$	
result:	*void*	

_symeigenvalues(*A*):

input:

A:	$n \times n$	

output:

A:	0×0	
result:	$1 \times n$	(contains eigenvalues)

_eigen_la(*todo*, *A*, *X*, *L*, *rcond*, *nobalance*):

input:

todo:	1×1	
A:	$n \times n$	
rcond:	1×1	
nobalance:	1×1	

output:

A:	0×0	
X:	$n \times n$	
L:	$1 \times n$	or $n \times 1$
rcond:	$1 \times n$	or $n \times 1$ (optional)
result:	1×1	(return code)

_symeigen_la(*todo*, *A*, *X*, *L*):

input:			
todo:	1×1		
A:	$n \times n$		
output:			
A:	0×0		
X:	$n \times n$		
L:	$1 \times n$		
result:	1×1	(return code)	

_eigengelapacke(*todo*, *A*, *X*, *L*, *rcond*, *balancing*):

input:				
todo:	1×1			
A:	$n \times n$			
rcond:	1×1			
balancing:	1×1			
output:				
A:	0×0			
X:	$n \times n$			
L:	$1 \times n$	or	$n \times 1$	
rcond:	$1 \times n$	or	$n \times 1$	(optional)
result:	1×1	(return code)		

Diagnostics

All functions return missing-value results if *A* has missing values.

symeigensystem(), symeigenvalues(), _symeigensystem(), and _symeigenvalues() use the lower triangle of *A* without checking for symmetry. When *A* is complex, only the real part of the diagonal is used.

_eigengelapacke() aborts with an error message when MKL is not available or is set to not use MKL.

References

Gould, W. W. 2011a. Understanding matrices intuitively, part 1. *The Stata Blog: Not Elsewhere Classified.* http://blog.stata.com/2011/03/03/understanding-matrices-intuitively-part-1/.

——. 2011b. Understanding matrices intuitively, part 2, eigenvalues and eigenvectors. *The Stata Blog: Not Elsewhere Classified.* http://blog.stata.com/2011/03/09/understanding-matrices-intuitively-part-2/.

Also see

[M-5] **matexpsym( )** — Exponentiation and logarithms of symmetric matrices

[M-5] **matpowersym( )** — Powers of a symmetric matrix

[M-4] **Matrix** — Matrix functions

For suggested citations, see the FAQ on citing Stata documentation.