11 Eigenproblems

Functions:

jacobi: eigenvalues and vectors of a symmetric matrix
eigsrt: sorts eigenvectors into order by eigenvalue
tred2: Householder reduction of a real symmetric matrix
tqli: eigenvalues and vectors of a symmetric tridiagonal matrix
balanc: balance a non-symmetric matrix
elmhes: reduce a general matrix to Hessenberg form
hqr: eigenvalues of a Hessenberg matrix

---

jacobi

Description:

This function computes the eigenvalues and eigenvectors of a real symmetric matrix.

jacobi(a) $\longrightarrow$ **[d, v, nrot]**

Where:

a is a real symmetric matrix,
d a floating point list representing the eigenvalues of a,
v is a floating point matrix with columns representing the normalized eigenvectors of a,
nrot is the number of Jacobi rotations which were required.

Example:

In this example the eigenvalues and corresponding matrix of eigenvectors for a 3 by 3 matrix are computed using `jacobi`.

```
-> a : matrix([1.0, 2.0, 3.0],
              [2.0, 2.0, 3.0],
              [3.0, 3.0, 3.0]);
-> res : jacobi(a)?
                                          | 0.4250902   0.4827395  -0.7656768 |
                                          |                                   |
[[-0.3389217, 7.5165385, -1.1776168],  | -0.8291529  0.5469631  -0.1154850 |,
                                          |                                   |
                                          | 0.3630478   0.6839547  0.6327735  |
11]
_> trans(x):=transpose(x);
-> d : first(res); v : proj(res,2);
-> Test solution (should get zero vector in each case): #
-> column(1,
  a*trans(matrix(column(1,v))) - proj(d,1)*trans(matrix(column(1,v))))?
[-0.0000000, -0.0000000, 0.0000000]
-> column(1,
   a*trans(matrix(column(2,v))) - proj(d,2)*trans(matrix(column(2,v))))?
[-0.0000000, 0.0000000, 0.0000000]
-> column(1,
   a*trans(matrix(column(3,v))) - proj(d,3)*trans(matrix(column(3,v))))?
[0.0000000, 0.0000000, 0.0000000]
```

eigsrt

Description:

This function sorts eigenvectors into order by eigenvalue.

eigsrt(d, v) ⟶ [d, v]

Where:

d is a real list representing the eigenvalues as returned by *jacobi* on input and the list sorted into descending order on output,
v is a real matrix representing the corresponding eigenvectors as returned by *jacobi* on input and the sorted matrix on output.

Example:

The eigenvalues and corresponding eigenvectors returned by `jacobi` above are sorted by `eigsrt`.

```
-> Use the values of d and v from the example above #
-> eigsrt(d,v)?
                                          | 0.4827395  0.4250902   -0.7656768 |
                                          |                                   |
[[7.5165385, -0.3389217, -1.1776168],     | 0.5469631 -0.8291529   -0.1154850 |]
                                          |                                   |
                                          | 0.6839547  0.3630478   0.6327735  |
```

tred2

Description:

This function performs a Householder reduction of a real symmetric matrix.

tred2(a) $\longrightarrow$ **[a, d, e]**

Where:

a is a real symmetric matrix on input and an orthogonal matrix which effects the reduction on output if *tred2_eigenvectors* is set to *true*,
d is a floating point list representing the diagonal elements of the tridiagonal matrix,
e is a floating point list representing the off-diagonal elements of the tridiagonal matrix with e_1 set to zero.

Pre-variables:

tred2_eigenvectors [true] is either *true* or *false* depending on whether a is to contain the eigenvectors on output,

Example:

A Householder reduction of a matrix `a` is performed by `tred2` and the result is then tested.

```
-> Use same matrix again #
-> a : matrix([1.0, 2.0, 3.0],
              [2.0, 2.0, 3.0],
              [3.0, 3.0, 3.0]);
-> res : tred2(a)?
 | 0.7071068   -0.7071068  0.0000000 |
 |                                   |
[| -0.7071068  -0.7071068  0.0000000 |, [-0.5000000, 3.5000000, 3.0000000],
 |                                   |
 | 0.0000000   0.0000000   1.0000000 |
[0.0000000, 0.5000000, -4.2426407]]
-> aa : first(res); d : proj(res,2); e : last(res);
-> Test Solutions (should get tridiagonal matrix, d as leading diag): #
-> transpose(aa)*a*aa?
| -0.5000000  0.5000000   0.0000000  |
|                                    |
| 0.5000000   3.5000000   -4.2426407 |
|                                    |
| 0.0000000   -4.2426407  3.0000000  |
```

tqli

Description:

This fuction computes the eigenvalues and eigenvectors of a real symmetric tridiagonal matrix.

$$\textbf{tqli(d, e, z)} \longrightarrow \textbf{[d, z]}$$

Where:

d is a real list on input representing the diagonal elements of a tridiagonal matrix and on output representing the eigenvalues,
e is a real list representing the sub-diagonal elements of the matrix with e_1 arbitary,
z is a real matrix which on input is either the identity matrix or the matrix output by *tred2* and on output a floating point matrix such that the k'th column represents the eigenvector corresponding to d_k.

Pre-variables:

tqli_eigenvectors [true] is either *true* or *false* depending on whether or not the eigenvectors are to be computed.

Example:

The eigenvalues and eigenvectors of a real symmetric tridiagonal matrix `a` are computed by `tqli`.

```
-> a : matrix([1.0, 2.0, 0.0],
              [2.0, 2.0, 3.0],
              [0.0, 3.0, 3.0]);
-> Set d & e to be diag. and off-diag. elements resp.: #
-> d : [1.0, 2.0, 3.0];
-> e : [0.0, 2.0, 3.0]; with the first, 0.0, being arbitrary. #
-> z : ident(3)?
| 1  0  0 |
|         |
| 0  1  0 |
|         |
| 0  0  1 |
-> res : tqli(d,e,z)?
                                     | 0.5480088   0.7907190   0.2728550 |
                                     |                                   |
[[-1.5483659, 1.6395110, 5.9088549], | -0.6982635  0.2528367   0.6697027 |]
                                     |                                   |
                                     | 0.4605589   -0.5575276  0.6906869 |
```

balanc

Description:

This function balances a real square matrix.

balanc(a) $\longrightarrow$ **a**

Where:

a is a real square matrix on input and the corresponding balanced matrix on output.

Pre-variables:

balanc_radix [2.0d0] is a floating point number representing the machines floatng point radix.

Example:

An unbalanced matrix `a` is balanced using `balanc`.

```
-> balance the following (unbalanced) matrix (w.r.t. norms) #
-> a : matrix([1.0, 100.0, 1.0, 100.0, 1.0],
              [1.0,   1.0, 1.0,   1.0, 1.0],
              [1.0, 100.0, 1.0, 100.0, 1.0],
              [1.0,   1.0, 1.0,   1.0, 1.0],
              [1.0, 100.0, 1.0, 100.0, 1.0]);
-> Norms at present:  rows: 203.0, 5.0, 203.0, 5.0, 203.0 #
->  columns: 5.0, 302.0, 5.0, 302.0, 5.0 #
-> ba : balanc(a)?
| 1.000 12.500 1.000 12.500 1.000 |
|                                 |
| 8.000 1.000  8.000 1.000  8.000 |
|                                 |
| 1.000 12.500 1.000 12.500 1.000 |
|                                 |
| 8.000 1.000  8.000 1.000  8.000 |
|                                 |
| 1.000 12.500 1.000 12.500 1.000 |
-> Norms now:          rows: 28.0, 26.0, 28.0, 26.0, 28.0 #
->                  columns: 19.0, 39.5, 19.0, 39.5, 19.0 #
```

elmhes

Description:

This function reduces a general matrix to Hessenberg form.

elmhes(a) ⟶ a

Where:

a is on input a real matrix and on output the corresponding Hessenberg form.

Example:

A matrix `a` is first balanced using `balanc` and then converted into Hessenberg form using `elmhes`.

```
-> a : matrix([1.0, 2.0, 300.0, 4.0, 5.0],
              [2.0, 3.0, 400.0, 5.0, 6.0],
              [3.0, 4.0,   5.0, 6.0, 7.0],
              [4.0, 5.0, 600.0, 7.0, 8.0],
              [5.0, 6.0, 700.0, 8.0, 9.0]);
-> aa : balanc(a)?
| 1.0000000   2.0000000   37.5000000  4.0000000   5.0000000  |
|                                                            |
| 2.0000000   3.0000000   50.0000000  5.0000000   6.0000000  |
|                                                            |
| 24.0000000  32.0000000  5.0000000   48.0000000  56.0000000 |
|                                                            |
| 4.0000000   5.0000000   75.0000000  7.0000000   8.0000000  |
|                                                            |
| 5.0000000   6.0000000   87.5000000  8.0000000   9.0000000  |
-> elma : elmhes(aa)?
| 1.0000000   39.3750000  9.6178334    3.3333333   4.0000000  |
|                                                              |
| 24.0000000  27.3333333  116.0968004  48.0000000  48.0000000 |
|                                                              |
| 0.0833333   85.5138889  -4.7804667   -1.3333333  -2.0000000 |
|                                                              |
| 0.1666667   0.8617833   5.1884795    1.4471333   2.1707000  |
|                                                              |
| 0.2083333   0.5853500   0.3333333    -0.0000000  -0.0000000 |
```

hqr

Description:

This function computes the eigenvalues of a Hessenberg matrix.

$$\mathbf{hqr(a)} \longrightarrow \mathbf{[wr, wi]}$$

Where:

a is a real square matrix,
wr is a floating point list representing the real parts of the eigenvalues of a,
wi is a floating point list representing the imaginary parts of the eigenvalues of a.

Example:

A matrix `a` is first balanced using `balanc`. The balanced matrix `aa` is then converted into Hessenberg form using `elmhes` and the eigenvalues computed using `hqr`.

```
-> a : matrix([ 1,2,  0,0, 0],
              [-2,3,  0,0, 0],
              [ 3,4, 50,0, 0],
              [-4,5,-60,7, 0],
              [-5,6,-70,8,-9]);
-> aa : balanc(a);
-> elma : elmhes(aa);
-> hqr(elma)?
[[50.0000000, 2.0000000, 2.0000000, 7.0000000, -9.0000000], [0.0000000,
-1.7320508, 1.7320508, 0.0000000, 0.0000000]]
```