EIGENTEST
The Fortran 77 User's Guide

Che-Rung Lee
G. W. Stewart

Aug 20 2007

Introduction

EIGENTEST is a collection of routines to create and manipulate test matrices, called *eigenmats,* with known eigenvalues and eigenvectors. Eigenmats are real matrices maintained in factored form in such a way that storage and operation costs are proportional to the order of the eigenmat in question. The spectrum of an eigenmat consists of real eigenvalues, complex conjugate pairs of eigenvalues, and real Jordan blocks. The operations consist of multiplying a matrix by $(A - sI)$, $(A - sI)^{\mathrm{T}}$, $(A - sI)^{-1}$, and $(A - sI)^{-\mathrm{T}}$, where A is the eigenmat and s is a shift. In addition, EIGENTEST provides a function to compute individual eigenvectors and principal vectors, and functions to help with the creation of eigenmats.

This document is intended to provide a quick introduction to eigenmats, their operations, and their implementation in Fortran 77. For more details see the TOMS paper describing the EIGENTEST package.

Structure of an eigenmat

An eigenmat A has the factored form

$$A = YZLZ^{-1}Y^{-1} \equiv XLX^{-1}.$$

The matrix X consists of the eigenvectors and principal vectors of A. We will peel this factorization apart like an onion, beginning with the matrix Y.

The matrix Y is a special case of a *Householder-SVD matrix,* or *hsvdmat* for short. It has the form

$$Y = (I - uu^{\mathrm{T}})\Sigma(I - vv^{\mathrm{T}}), \qquad (1)$$

where

$$\|u\| = \|v\| = \sqrt{2}$$

and

$$\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n), \qquad \sigma_i > 0 \ \ (i = 1, \ldots, n).$$

The matrices $I - uu^{\mathrm{T}}$ and $I - vv^{\mathrm{T}}$ are called Householder transformations. They are orthogonal matrices, and hence the right-hand of (1) is the singular value decomposition of Y. By increasing the ratio of the largest to the smallest of the singular values $\sigma_1, \ldots, \sigma_n$, one can increase the ill-conditioning of the hsvdmat Y.

The matrix Z is the general case of an hsvdmat. It has the block diagonal form

$$Z = \mathrm{diag}(Z_1, \ldots, Z_{\mathrm{nblocks}}),$$

where each Z_i is an hsvdmat of the form (1).

The matrix L has the block structure

$$L = \mathrm{diag}(L_1, L_2, \ldots, L_m).$$

There are three kinds of blocks.

- **Real eigenvalue.** A real matrix of order one containing a real eigenvalue λ.
- **Complex conjugate eigenvalues.** A real matrix of order two having the form

$$L_i = \begin{pmatrix} \mu & \nu \\ -\nu & \mu \end{pmatrix}.$$

This is a normal matrix, whose eigenvalues are are $\mu \pm \nu i$ with eigenvectors

$$\begin{pmatrix} 1 \\ \pm i \end{pmatrix}.$$

- **Jordan block.** A real Jordan block of the form

$$\begin{pmatrix} \lambda & \eta_1 & 0 & \cdots & 0 \\ 0 & \lambda & \eta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda & \eta_{k-1} \\ 0 & 0 & \cdots & 0 & \lambda \end{pmatrix}. \tag{2}$$

The representation of an eigenmat

The implementation of Eigentest in Fortran 77 presents two problems not encountered by its Matlab, C, and Fortran 95 variants. First, Fortran 77 does not have dynamic memory allocation: all memory must be allocated statically at compile time. This means that the programmer must predetermine the maximum size of the eigenmats that will be needed and hard code that information into the declarations of the program. The second problem is that Fortran 77 has no way of creating structures or derived types. This means that either the components of an Eigemant will have to be represented by a host of individual variables, which will clutter the calling sequences of the EIGENTEST subroutines, or that it will have to be packed into arrays, which will obscure the underlying structure of the eigenmat.

EIGENTEST takes the second approach. The entire eigemat is represented by an integer array `iem` and a floating-point array `fem`. To help the programmer in setting up an eigenmat, we have provided a routine `eminit` to place the components of an eigenmat in the arrays. Since eigenmats are manipulated by functions provided by EIGENTEST, there is no need to worry about the details of how components are stored in the arrays.[1] However, to use `eminit` the programmer must know what the components are. In describing them it will be convenient to pretend that Fortran 77 has structures. We will use Matlab notation, since it is widely used and its indexing conventions are compatible with those of Fortran.

An eigenmat A is represented by the structure

```
struct('n',    <Order of the matrix> ...
       'eig',  <Array containing the eigenvalues of A or
                 the superdiagonals of a Jordan block> ...
       'type', <The type of the entry in eig> ...                (3)
       'Z'     <The block hsvdmat Z> ...
       'Y',    <The block hsvdmat Y> ...
)
```

The contents of the `type` and `eig` arrays are determined as follows.

Real eigenvalue
$\mathtt{type(i)} = 1$ $\mathtt{eig(i)} = \lambda$

Complex eigenvalue
$\mathtt{type(i)} = 2$ $\mathtt{eig(i)} = \mu$
$\mathtt{type(i+1)} = 3$ $\mathtt{eig(i+1)} = \nu$

Jordan block
$\mathtt{type(i)} = -\mathtt{k}$ $\mathtt{eig(i)} = \lambda$
$\mathtt{type(i+j)} = -1$ $\mathtt{eig(i+j)} = \eta_{\mathtt{j}}$ $(\mathtt{j} = 1, \ldots, \mathtt{k}-1)$

The matrix Z is composed of hsvdmats Z_i of, say, order k_i of the form[2]

$$Z_i = (I - u_i u_i^{\mathrm{T}})\Sigma_i(I - v_i v_i^{\mathrm{T}}), \qquad i = 1, \ldots, \text{nblocks}.$$

It is stored as follows. The vectors u_i are packed in a floating-point array `u` of length n in their natural order. Likewise, the vectors v_i are packed in a floating-point array

[1]For those who do like to worry about such things, the details are given in an appendix to this guide.

[2]There is no necessary correspondence between the blocks of Z and the blocks of L. But since the purpose of a block of Z is to combine blocks of L, it is to be expected that a block of Z will exactly contain a contiguous sequence of blocks of L.

v, and the singular values σ_i are stored in a floating-point array `sig`. These arrays are accompanied by an integer array `bs` (for block start) of length `nblocks+1`. The absolute value of `i`th entry of `bx` contains the starting index for the `i`th block; i.e.,

$$\texttt{b(1)} = 1 \quad \text{and} \quad \texttt{b(i)} = \pm(1 + k_1 + \cdots + k_{i-1}) \ (i > 1).$$

Since $1 + k_1 + \cdots + k_{\text{nblocks}} = n + 1$, we have

$$\texttt{b(nblocks+1)} = \pm(n + 1).$$

The matrix Z is implemented by the structure

```
struct('n',       <The order of the matrix> ...
       'nblocks', <The number of blocks> ...
       'bs',      <The block start indices> ...
       'sig',     <The diagonal elements of the Sigma's> ...      (4)
       'u',       <The components of the u's> ...
       'v',       <The components of the v's> ...
)
```

It may happen that Y or some of the Z_i must be identity matrices. One way to create an identity is to set $u_i = v_i$ and $\Sigma_i = I$; but this is an inefficient way to compute $b = Z_i b$. Consequently, Eigentest adopts the following convention.

$$\text{If } \texttt{bs(i+1)} < 0, \text{ then } Z_i = I.$$

Thus if we wish to make Z an identity matrix, we simply set

```
Z = struct('n', n, 'nblocks', 1, 'bs', [1;-(n+1)], ...
           'sig', [], 'u', [], 'v', []);
```

The matrix Y is represented as an hsvdmat with only one block.

As mentioned above, all of the components of an eigenmat are stored in two singly subscripted arrays: an integer array `iem`, and a double-precision array `fem`. Eigentest provides a subroutine `eminit` to initialize and eigenmat. It's calling sequence is

```
eminit(nmax, n, iem, fem,  nblk, idy, idz, ia, fa, job)
   integer nmax, n, iem(*), nblk, idy, idz, ia(*)
   double precision fem(*), fa(*)
   character job*(*)
```

`Eminit` takes information from its calling sequence and places it in the appropriate places in `iem` and `fem` as specified by `job`. Its useage is simple but best explained by a running example. We will consider the generation of an eigenmat of order eight. It has three real eigenvalues $(1, 2, 3)$, a pair of complex conjugate eigenvalues $(1 \pm 12i)$, and a Jordan block of order 3 with eigenvalue $\sqrt{2}$ and superdiagonal elements of 10^{-3}. The hsvdmat Z mixes the real and complex eigenvalues and leaves the Jordan block alone (i.e., $Z_2 = I$). The hsvdmat Y mixes everything.

We begin with the arrays `iam` and `fem`. Their size will depend on the order of the largest eigenmat to be represented, which must be known before compilation. Then we define storage for `iam`, `fem`, `ia`, and `fa` by the statements by the statements

```
parameter nmax = <The order of the largest matrix>
integer iem(2*nmax+12), ia(nmax+1)
double precision fem(8*nmax), fa(nmax)
```

The next thing is to set up the basic eigenmat. In the matrix described above, the order of the matrix is eight, the number of blocks in Z is two, the second block being an identity. Therefore, we set up the block structure of Z in `ia` and call `eminit` as follows.[3]

```
ia(1)=1; ia(2)=6; ia(3)=-9;
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'setup')
```

We now proceed to set the `type` and `eig` components. First we compute and store them in the `ia` and `fa` arrays.

```
ia(1) = 1; ia(2) = 1; ia(3) = 1;
fa(1) = 1; fa(2) = 2; fa(3) = 3;
ia(4) = 2; ia(5) = 2;
fa(4) = 1; fa(5) = 12;
ia(6) = -3; ia(7) = -1; ia(8) = -1;
fa(6) = sqrt(2); fa(7) = 1e-3; fa(8) = 1e-3;
```

We then enter them into the eigenmat as follows.

```
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'eig')
```

We now turn to the matrix Z. It's block structure has already been initialized in the `'setup'` phase. All we need to do is import the `u`, `v`, and `sig` arrays. This is done as follows.

[3]For the sake of brevity, we use ';' as a (nonstandard) statement separator

```
do i=1,5
   fa(i) = duni() - 0.5
end do
call hscal(5, fa)
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'zu')
do i=1,5
   fa(i) = duni() - 0.5
end do                                                        (5)
call hscal(5, fa)
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'zv')
do i=1,5
   fa(i) = 1
end do
fa(5) = 1.e-1
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'sig')
```

The vectors `u` and `v` are random vectors in $[-.5, .5)$ normalized so that their norms are $\sqrt{2}$. The singular values are all one, except for the last, which is 10^{-1}. Since $Z_2 = I$, there is no need to initialize that block.

Note the two utility routines in this sequence. The function `duni()` [by D. Kahaner and G. Marsaglia] produces a pseudo-random double-precision number that is uniformly distributed in $[0, 1)$. For more on its usage, see the comments in the code. The subroutine `hscal` scales a nonzero vector so that its norm is $\sqrt{2}$, which is required of the generating vector of a Householder transformation. Its calling sequence is

```
call hscal(n, u)
   u   a nonzero vector of length n.  On return,
       u is scaled so that its norm is sqrt(2).
```

The initialization of Y is similar.

```
do i=1,8
   fa(i) = duni() - 0.5
end do
call hscal(8, fa)
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'yu')
do i=1,8
   fa(i) = duni() - 0.5
end do
call hscal(8, fa)
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'yv')
do i=1,8
```

```
   fa(i) = 1
end do
fa(8) = 1.0e-1
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fa, 'ysig')
```

This completeness the construction of our eigenmat. There are four points to be made about the process.

- If during `'setup'` the parameter `idy` is nonzero, Y is initialized as an identity matrix and needs no further initialization. Similarly for `idz`.

- Once the `'setup'` process is complete, the other entries can be made in any order. However, be sure not to change the first five arguments.

- One can use arrays other than `ia` or `fa` in calls to `eminit`. For example, you may prefer the following code for initializing Z over the code in (5).

```
do i=1,5
   fau(i) = duni() - 0.5
   fav(i) = duni() - 0.5
   fas(i) = 1
end do
call hscal(5, fau)
call hscal(5, fav)
fas(6) = 1.0e-1
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fau, 'zu')
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fav, 'zv')
call eminit(nmax, 8, iem, fem, 2, 0, 0, ia, fas, 'sig')
```

- In complicated experiments, you may want to write a function to generate your eigenmat having the parameters you want to vary as its arguments. For example, if the problem is to study the effects of ill-conditioning of Y and Z, it will pay to encapsulate the above code in a function with an argument `sigmin` that replaces `1.0e-1`.

Manipulating eigenmats

EIGENTEST has two functions to work with eigenmats.

- The subroutine `emprod` computes the products involving an eigenmat. Its calling sequence is

```
emprod(iem, fem, ncols, B, ldb, C, ldc, shift, op)
   integer iem(*), ncols, ldb, ldc
   double precision fem(*), B(ldb,*), C(ldc,*), shift
   character op*(*)

   iem,fem    Arrays containing the eigenmat.
   ncols      Number of columns in the matrix B.
   B          Pointer to an array containing the matrix B.
   ldb        The leading dimension of the array B.
   C          Pointer to an array containing the matrix C.
   ldc        The leading dimension of C.
   shift      A shift.
   op         A string specifying the operation to be performed.

              'ab'    C = (A - shift*I)*B
              'atb'   C = (A - shift*I)'*B
              'aib'   C = (A - shift*I)\B
              'aitb'  C = (A - shift*I)'\B
```

• The subroutine `emvecs` computes specified eigenvectors or, in the case of a Jordan block, principal vectors. Its calling sequence is

```
subroutine emvecs(iem, fem, eignum, eig, x, y, cond, job)
   integer          iem(*), eignum
   double precision fem(*), cond
   complex*16       eig, x(*), y(*)
   character        job


   iem, fem Arrays containing the eigenmat whose vectors
            are to be computed.
   eignum   The position in A%eig of the eigenvalue.
   eig      The eigenvalue.
   x(:)     The right eigenvector or principal vector.
   y(:)     The left eigenvector or principal vector.
   cond     The condition number of the eigenvalue.
            (or -1, if the eigenvalue belongs to a
            Jordan block).
   job      A string specifying what to compute.
```

'r'	Compute the right eigenvector or principal vector.
'l'	Compute the left eigenvector or principal vector..
'b'	Compute both and the condition number. (Note: For Jordan blocks, principal vectors are computed and -1 is returned for the condition number.)

Appendix: The storage of an Eigenmat

In this appendix we give the details of the storage of an eigenmat in the arrays `iem` and `fem`. The reader should refer to (3) and (4) for the components of an eigenmat. We also give the calling sequence for `hsvdpr`, a utility routine for computing products with hsvdmats.

The components of an hsvdmat are stored as follows.

iem(1)	nmax	The maximum size of the matrix.
iem(2)	n	The order of the matrix.
iem(3)	type	The beginning of an array containing the types of the eigenvalues in eig.
iem(nmax+3)		The beginning of the integer array for the hsvdmat Y.
iem(nmax+8)		The beginning of the integer array for the hsvdmat Z.
fem(1)	eig	The array containing the eigenvalues of the matrix.
fem(nmax+1)		The beginning of the double array for the hsvdmat Y.
fem(4*nmax+1)		The beginning of the double array for the hsvdmat Z.
fem(7*nmax+1)		The beginning of a work array of length nmax.

(6)

An hsvdmat is stored as follows.

iem(1)	nmax	maximum value of n.
iem(2)	n	The order of the hsvd.
iem(3)	nblk	The number of blocks in the hsvd.
iem(4)	bs	Beginning of the block start array. abs(bs(5))=1 and abs(bs(i+1))-abs(bs(i))

```
                            is the size of the ith block.
                            If bs(i+1) is negative, block i is
                            an identity.
fem(1)          u           start of u.
fem(1+nmax)     v           start of v.
fem(1+2*nmax)   sig         start of the singular values.
```

Here the indexing is relative to be beginning of of the hsvdmat in (6). Thus the start of the `bs` array for Z is `iem((nmax+8)+4-1)` or `iem(nmax+11)`. Again, the array `sig` for Y begins at `fem((nmax+1)+(1+2*nmax)-1)` or `fem(3*nmax+1)`.

EIGENTEST provides a function to manipulate the hsvdmats Y and Z. Its calling sequence is

```
subroutine hsvdpr(ihsm, fhsm, ncols, B, ldb, op)
integer ihsm(*), ncols, ldb
double precision fhsm(*), B(ldb,*)
character op*(*)

   ihsm, fhsm    The hsvd
   ncols         The number of columns in B.
   B             The array B.
   ldb           The leading dimension of B.
   op            A string specifying the operation to be performed.
                    'ab'   B <- X*B
                    'atb'  B <- X^t*B
                    'aib'  B <- X^-1*B
                    'aitb' B <- X^-1T*B
```

In its ordinary use, `hsvdpr` would be called with the location in `iem` and `fem` at which Y and Z begins. For example, to perform an operation with Z, one would code [see (6)]

```
call hsvdpr(iem(nmax+8), fhsm(4*nmax+1), ncols, B, ldb, op)
```

The Fortran 77 package

The Fortran 77 version of the eigetest package comes with the following files.

`README` A brief introductory file.

`eigentest.f` The EIGENTEST program. This file can be compiled to compute an object file suitable for linking to an application.

`testeigentest.f` A test program for Eigentest that runs 64 test cases probing various aspects of the package. The numbers in the output should be within two or so orders of magnitude of the rounding unit.

`Eigentest.pdf` The technical report describing eigentest.

`F77UsersGuide.pdf` This user's guide.