EIGENTEST
The Fortran 95 User's Guide

Che-Rung Lee
G. W. Stewart

Aug 20 2007

Introduction

EIGENTEST is a collection of functions 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 95. 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

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

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

$$Z = \mathrm{diag}(Z_0, \ldots, Z_{\mathrm{nblocks}-1}),$$

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

In Fortran 95, an eigenmat A is represented by the structure

```
type eigenmat
   integer                 :: n        ! The order of the matrix.
   real(wp), allocatable :: eig(:)   ! Array containing the
                                       ! eigenvalues of A
                                       ! or the superdiagonals
```

```
                                        ! of a Jordan block.
   integer, allocatable  :: type(:) ! The type of the entry
                                        ! in eig.
   type(hsvdmat)          :: Y, Z   ! The outer and inner hsvd
                                        ! transformations.
end type eigenmat
```

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

Real eigenvalue

$\texttt{type(i)} = \texttt{1}$	$\texttt{eig(i)} = \lambda$

Complex eigenvalue

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

Jordan block

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

The matrix Z is composed of hsvdmats Z_i of, say, order k_i of the form[1]

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

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 `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

[1] 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.

```
   type hsvdmat
   integer               :: n       ! The order of the matrix.
   integer               :: nblocks ! The number of blocks
                                    ! in the hsvdmat.
   integer, allocatable  :: bs(:)   ! abs(bs(i)) is the index of the
                                    ! start of the i-th block.
                                    ! abs(bs(nblocks+1)) = n+1.
                                    ! If bs(i+1)<0, the i-th block
                                    ! is an identity.
   real(wp), allocatable :: u(:)    ! The vectors generating the left
                                    ! Householder transformations.
   real(wp), allocatable :: v(:)    ! The vectors generating the right
                                    ! Householder transformations.
   real(wp), allocatable :: sig(:)  ! The diagonals of the Sigma_i.
end type hsvdmat
```

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.

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

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

```
Z.nblocks = 1
Z.bs[0] = 0
Z.bs[1] = -n
```

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

EIGENTEST provides a function to allocate storage for an eigenmat and its associated hsvdmats. It has the form

```
subroutine EigenmatAlloc(A, n, nblocks, yident, zident)
   type(eigenmat), intent(inout) :: A
   integer, intent(in)           :: n
   integer, intent(in)           :: nblocks
   logical                       :: yident, zident

   A        The eigenmat to be initialized.
   n        The order of A.
   nblocks  The number of blocks in the hsvd matrix Z.
```

```
yident   If yident is .true., Y is declared to be an identity
         matrix and only Y%nblock and Y%bs are initialized.
zident   If zident is .true., Z is declared to be an identity
         matrix and only Z%nblock and Z%bs are initialized.
```

`EigenmatAlloc` allocates memory for the arrays in `A`, `A%Y`, and `A%Z`. In addition it initializes `A%n`, `A%Z%n`, `A%Z%nblocks`, `A%Z%bs(1)`, `A%Z%bs(nblocks+1)`, `A%Y%n`, `A%y%nblocks`, and `A%Y%bs`. All other arrays are initialized to zero.

`EigematFree(A)` dealocates the storage of the eigenmat A.

A utility subroutine routine, `hscale`, that scales a vector to have norm $\sqrt{2}$ is provided to aid in setting up hsvdmats. Its calling sequence is

```
call hscal(u)
   u    A nonzero vector.  On return the
        norm of u is sqrt(2).
```

Figure 1 shows how to set up an eigenmat. `A` has three real eigevalues $(1, 2, 3)$, 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. The hsvdmat Y mixes everything.

Note that EIGENTEST provides a utility subroutine to scale a nonzero vector so that its norm is $\sqrt{2}$. 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).
```

From the foregoing it is clear that setting up an eigenmat can be nontrivial. In complicated experiments, you may want to write a function, whose arguments are the parameters you want to vary, to generate your matrix. For example, if one were performing a series of experiments to determine the effects of the condition of Y and Z, one might turn the code in Figure 1 into a function with the argument `sigmin`.

Manipulating eigenmats

EIGENTEST has two functions to work with eigenmats and one to work with hsvdmats.

- `EigenmatProd` computes the the products involving an eigenmat. It has the form

```
subroutine EigenmatProd(A, ncols, B, ldb, C, ldc, shift, job)
   type(Eigenmat), intent(in) :: A
   integer, intent(in)        :: ncols, ldb, ldc
   real(wp), intent(in)       :: B(ldb,*)
```

```
sigmin = 1.0e-3_wp

call EigenmatAlloc(A, 8, 2, 0, 0);

A%type(1:3) = (/1,1,1/)
A%eig(1:3) = (/1,2,3/)
A%type(4:5) = (/2,3/)
A%eig(4:5) = (/1,12/)
A%type(6) = -3;
A%type(7:8) = -(/1,1/))
A%eig(6) = sqrt(2.0_wp);
A%eig(7:8) = 1e-3_wp*(/1,1/);

A%Z%bs(1) = 1;
A%Z%bs(2) = 6;
A%Z%bs(3) = -9;
call random(A%Z%u(1:5)); A%Z%u(1:5)=A%Z%u(1:5)-0.5
call hscal(5, A%Z%u(1:5))
call random(A%Z%v(1:5)); A%Z%v(1:5)=A%Z%v(1:5)-0.5
call hscal(5, A%Z%v(1:5))
A%Z%sig = (/(1, i=1,8)/)
A%Z%sig(8) = sigmin

call random(A%Y%u); A%Y%u=A%Y%u-0.5
call hscal(8, A%Y%u)
call random(A%Y%v); A%Y%u=A%Y%v-0.5
call hscal(8, A%Y%v)
A%Y%sig = (/(1, i=1,8)/)
A%Y%sig(8) = sigmin
```

Figure 1: Generating an eigenmat

```
   real(wp), intent(inout)    :: C(ldc,*)
   real(wp), intent(in)       :: shift
   character(*), intent(in)   :: job

   A       The eigenmat
   ncols   Number of columns in the matrix B.
   B       The array containing the matrix B.
   ldb     The leading dimension of B.
   C       The array array containing the matrix C.
   ldc     The leading dimension of C.
   shift   A shift.
   job     A string specifying the operation to be performed.

           "ab"    C = (A - shift*I)*B
           "atb"   C = (A - shift*I)^T*B
           "aib"   C = (A - shift*I)^-1*B
           "aitb"  C = (A - shift*I)^-T*B
```

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

```
subroutine EigenmatVecs(A, eignum, eig, x, y, cond, job)
   type(Eigenmat), intent(in) :: A
   integer, intent(in)        :: eignum
   complex(wp), intent(out)   :: eig
   complex(wp), intent(inout) :: x(:)
   complex(wp), intent(inout) :: y(:)
   real(wp), intent(out)      :: cond
   character, intent(in)      :: job

   A        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.
                  "l"  Compute the left eigenvector.
                  "b"  Compute both and the condition number.
                  (Note: For Jordan blocks, principal vectors
                  are computed and -1 is returned for the
                  condition number.)
```

All vectors returned have norm one.

• `HsvdProd` is a utility routine used by Eigentest to compute the products involving an hsvdmat. Its calling sequence is

```
subroutine HsvdProd(X, ncols, B, ldb, job)
   type(hsvdmat), intent(in) :: X
   integer, intent(in)       :: ncols, ldb
   real(wp), intent(inout)   :: B(ldb, *)
   character(*), intent(in)  :: job

   X       Pointer to the hsvdmat.
   ncols   The number of columns in B.
   B       The array B.
   ldb     The leading dimension of B.
   job     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^-T*B
```

The Fortran 95 package

The Fortran95 version of the eigetest package comes with the following files.

`README` A brief introductory file.

`eigentest.f95` The Eigentest module `eigentest`. This module must be `use`'d by any application that uses Eigentest (e.g., `testeigentest` below.)

`testeigentest.f95` 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.

`F95UsersGuide.pdf` This user's guide.