



1 SUMMARY

HSL_EA20 is a suite of Fortran 95 procedures for computing the product of the s -root of a sparse self-adjoint positive definite matrix by a vector using a scalar product derived by a second symmetric positive definite matrix. Given two $n \times n$ symmetric positive definite matrices \mathbf{A} and \mathbf{M} , and a vector \mathbf{u} , the package uses the Lanczos method, applied to the matrix pencil (\mathbf{M}, \mathbf{A}) , to approximate

$$(\mathbf{M}^{-1}\mathbf{A})^s \mathbf{u}, \quad s \in (-1, 1). \quad (1.1)$$

Reverse communication is used. Control is returned to the user for the products of \mathbf{A} with a vector \mathbf{z} , of \mathbf{M} with a vector \mathbf{x} , or of \mathbf{M}^{-1} with a vector \mathbf{w} .

ATTRIBUTES — Version: 1.1.0 (18 July 2013) **Types:** Real, Double. **Calls:** _DOT, _PTEQR, _GEMV. **Original date:** July 2010. **Origin:** M. Arioli, Rutherford Appleton Laboratory. **Language:** Fortran 95.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a USE statement such as

Single precision version

```
USE HSL_EA20_single
```

Double precision version

```
USE HSL_EA20_double
```

If it is required to use both modules at the same time, the derived types (Section 2.3) and the subroutine (Section 2.2) must be renamed on one of the USE statements.

2.2 Argument lists and calling sequences

There is one procedure that the user may call:

- EA20 uses the Lanczos method to approximate $(\mathbf{M}^{-1}\mathbf{A})^s \mathbf{u}$. The subroutine uses a reverse communication mechanism for matrix-vector products.

2.3 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types EA20_control, EA20_info, and EA20_reverse. The following pseudocode illustrates this.

```
use HSL_EA20_double
...
type (EA20_control) :: control
type (EA20_info) :: info
type (EA20_reverse) :: reverse
```

The components of EA20_reverse are private and are used for communication between calls during the reverse communication process.

2.3.1 The reverse communication subroutine

The reverse communication subroutine is called as follows:

```
CALL EA20(n,u,s,ido,w,control,info,reverse)
```

`n` is a scalar of type `INTEGER` that must hold the order of A and M . **Restriction:** `data%n` ≥ 1 .

`u` is an array of type `REAL` (double precision in `HSL_EA20_double`) and size at least `data%n`. On input, `u` must be allocated and holds the vector to which we want to apply the s -root of $M^{-1}A$. On exit, `u` contains the result. **Restriction:** `size(data%u)` \geq `data%n`.

`s` is a scalar of type `REAL` (double precision in `HSL_EA20_double`) that must hold the value of the s -root desired, **Restriction:** $-1 < \text{data}\%s < 1$.

`ido` is a scalar of type `INTEGER` that must be initially set by the user to `-1`. On exit from `EA20_main`, it is set to a value that informs about which operation is required. Its values are `1`, `2`, `3`. Depending on the value of `info%ido` the user must supply

```
info%ido = 1 matrix A by vector w(:,1) (w(:,2) = Aw(:,1));
info%ido = 2 matrix M by vector w(:,1) (w(:,2) = Mw(:,1));
info%ido = 3 inverse of M by vector w(:,1) (w(:,2) = M^-1w(:,1)).
```

`w` is an array of type `REAL` (double precision in `HSL_EA20_double`) and dimensions `(n,2)` that is used as working space: `w(:,1)` holds the input for the matrix by vector or the inverse of M by vector that the user must supply following `ido`; `w(:,2)` holds the answers of the operation required by `ido` on `w(:,1)`. The user does not need to allocate it.

`control` is a scalar `INTENT(IN)` argument of type `EA20_control` that contains the control parameters. (see Section 2.3.2).

`info` is a scalar `INTENT(OUT)` argument of type `EA20_info`. Its components provide information about the execution of the subroutine, as explained in Section 2.3.3.

`reverse` is a scalar `INTENT(INOUT)` argument of type `(EA20_reverse)`. It is used to hold data about the internal status of the variables and must be passed unchanged during the reverse communication process.

2.3.2 The derived data type for holding control parameters

The derived data type `EA20_control` is used to hold controlling data. The components, which are automatically given default values in the definition of the type, are:

Printing controls

`diagnostics_level` is a scalar of type `INTEGER` that is used to control the level of diagnostic printing. The different levels are:

- `< 0` No printing.
- `= 0` Error and warning messages only.
- `>= 1` As 0, plus some additional diagnostic printing.

The default is `diagnostics_level=0`.

`unit_diagnostics` is a scalar of type `INTEGER` that holds the unit number for diagnostic printing. Printing is suppressed if `unit_diagnostics < 0`. The default is `unit_diagnostics=6`.

`unit_error` is a scalar of type `INTEGER` that holds the unit number for error messages. Printing of error messages is suppressed if `unit_error < 0`. The default is `unit_error=6`.

`unit_warning` is a scalar of type `INTEGER` that holds the unit number for warning messages. Printing of warning messages is suppressed if `unit_warning < 0`. The default is `unit_warning=6`.

Other controls

`tol` is a scalar of type `REAL` (double precision in `HSL_EA20_double`) that holds the relative accuracy on the norm of the error between step i and $i + d$ where d is a parameter fixed by the user (see next item `d`). **Restriction:** $0 < \text{control}\%tol < 1$.

`d` is a scalar of type `INTEGER` that holds the delay parameter. **Restriction:** `control%d ≥ 1`.

`maxit` is a scalar of type `INTEGER` that holds the maximum number of Lanczos iterations allowed. **Restriction:** `control%maxit ≥ 1`.

2.3.3 The derived info type for holding information

The derived data type `EA20_info` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `EA20_info` (in alphabetical order) are:

`flag` is a scalar of type `INTEGER` that gives the exit status of the algorithm (details in Section 2.4).

`conv` is a scalar of type `LOGICAL` that informs if the convergence is achieved. If `conv = .true.` then a reasonable approximation is achieved, otherwise the result hold in `data%u` could be inaccurate.

`error` is a scalar of type `REAL` (double precision in `HSL_EA20_double`) that holds the error estimate (see Section 4.2).

2.3.4 The derived reverse type for holding information during reverse communication

The derived data type `EA20_reverse` is used to hold the parameters that must be saved during reverse communication. The user must not modify it and does not need to be set.

2.4 Warning and error messages

If `info%flag` has zero value on output, the code did not detect any error. Otherwise, a negative value indicates one of the following errors:

- 1 `data%n ≤ 0`;
- 2 `size(data%u) ≤ data%n`
- 3 failure in allocating `w`.
- 4 failure in allocating `reverse%v` the space necessary to store the Lanczos vectors. If `control%diagnostic_level > 0` additional diagnostics are printed.
- 5 failure in allocating internal workspace in EA20 If `control%diagnostic_level > 0` additional diagnostics are printed.

- 6 failure in allocating the vectors storing the information of the tridiagonal matrix and of the working vectors used by **LAPACK** routine `_PTEQR`. `info%LAP` holds the original error messages given by `_PTEQR`.
- 7 **M** is not positive definite.
- 8 error in `_PTEQR` suggesting that **A** is not positive definite. If `control%diagnostic_level > 0` additional diagnostics are printed. `info%LAP` holds the original error messages given by `_PTEQR`.
- 9 error in the deallocation. .
- 10 `data%tol ≤ 0` or `data%tol ≥ 1`.
- 11 `data%d ≤ 0`.
- 12 `data%s ≤ -1` or `data%s ≥ 1`.

A positive value of `info%flag` gives the following warning message:

- 1 Lanczos convergence is slow; the result may not be accurate.

3 GENERAL INFORMATION

Workspace: HSL_EA20 handles its own memory allocations. **BLAS** routines `_DOT`, `_GEMV`. **LAPACK** routines `_PTEQR`

Input/output: Output is provided under the control of `control%diagnostics_level`, which allows error, warning and diagnostics messages to be printed on units `control%unit_error`, `control%unit_warning` and `control%unit_diagnostics`, respectively.

Restrictions: `data%n ≥ 1`, `control%maxit > 0`, `data%d ≥ 1`, `0 < data%tol < 1`, `-1 < data%s < 11`.

Portability: Fortran 95, plus allocatable components of derived types.

4 METHOD

4.1 Algorithms for the matrix EA20

Given a pair of symmetric and positive-definite matrices (\mathbf{M}, \mathbf{A}) , the generalised Lanczos algorithm constructs a set of \mathbf{M} -orthogonal vectors \mathbf{v}_i such that

$$\mathbf{A}\mathbf{V}_k = \mathbf{M}\mathbf{V}_k\mathbf{T}_k + \beta_{k+1}\mathbf{M}\mathbf{v}_{k+1}\mathbf{e}_k^T, \quad \mathbf{V}_k^T\mathbf{M}\mathbf{V}_k = \mathbf{I}_k$$

where the columns \mathbf{v}_i of $\mathbf{V}_k = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k]$ are known as the Lanczos vectors and $\mathbf{I}_k \in \mathbb{R}^{k \times k}$ is the identity matrix with k th column denoted by \mathbf{e}_k , while the matrix $\mathbf{T}_k \in \mathbb{R}^{k \times k}$ is a symmetric and tridiagonal matrix [2]. The standard algorithm corresponds to the case $\mathbf{M} = \mathbf{I}$. Note that \mathbf{T}_k can be seen as a projection of \mathbf{A} onto the space spanned by the \mathbf{M} -orthogonal columns of \mathbf{V}_k

$$\mathbf{V}_k^T\mathbf{A}\mathbf{V}_k = \mathbf{T}_k, \quad \mathbf{V}_k^T\mathbf{M}\mathbf{V}_k = \mathbf{I}_k. \quad (4.1)$$

In exact arithmetic, when $k = n$, the algorithm can be seen as providing simultaneous factorisations of the matrix pair (\mathbf{M}, \mathbf{A}) as

$$\mathbf{A} = \mathbf{V}_n^{-T}\mathbf{T}_n\mathbf{V}_n^{-1}, \quad \mathbf{M} = \mathbf{V}_n^{-T}\mathbf{V}_n^{-1}.$$

We can immediately derive that:

$$(\mathbf{M}^{-1}\mathbf{A})^s = (\mathbf{V}\mathbf{T}^s\mathbf{V}^T), \quad (4.2)$$

and

$$(\mathbf{M}^{-1}\mathbf{A})^{-s} = (\mathbf{V}\mathbf{T}^{-s}\mathbf{V}^T). \quad (4.3)$$

4.2 Sparse evaluation of $(\mathbf{M}^{-1}\mathbf{A})^s\mathbf{z}$

The complexity of the full ($k = n$) generalised Lanczos algorithm is in general $O(n^3)$. However, in many applications of interest we do not need to compute $(\mathbf{M}^{-1}\mathbf{A})^s$, but simply apply it (or its inverse) to a given vector $\mathbf{z} \in \mathbb{R}^n$. In such cases, a truncated version of the algorithm is used in practice with only k Lanczos vectors being constructed. As we are interested in approximations of $(\mathbf{M}^{-1}\mathbf{A})^s\mathbf{z}$ we note first that if we start the Lanczos process with $\mathbf{v} = \mathbf{z}$ then

$$\mathbf{V}_k^T \mathbf{M} \mathbf{z} = \mathbf{e}_1 \|\mathbf{z}\|_{\mathbf{M}} \quad (4.4)$$

where $\mathbf{e}_1 \in \mathbb{R}^k$ is the first column of the identity \mathbf{I}_k . This leads us to consider the following approximations of the matrix-vector products:

$$(\mathbf{M}^{-1}\mathbf{A})^s \mathbf{z} \approx \mathbf{V}_k \mathbf{T}_k^s \mathbf{e}_1 \|\mathbf{z}\|_{\mathbf{M}}.$$

Similarly, if we wish to apply the inverse of $(\mathbf{M}^{-1}\mathbf{A})^s$ to a given vector \mathbf{z} we first note that if we start the iteration with $\mathbf{v} = \mathbf{M}^{-1}\mathbf{z}$ then

$$\mathbf{V}_k^T \mathbf{z} = \mathbf{V}_k^T \mathbf{M} (\mathbf{M}^{-1}\mathbf{z}) = \mathbf{e}_1 \|\mathbf{M}^{-1}\mathbf{z}\|_{\mathbf{M}} = \mathbf{e}_1 \|\mathbf{z}\|_{\mathbf{M}^{-1}}.$$

This leads us to consider the following approximations

$$(\mathbf{M}^{-1}\mathbf{A})^{-s} \mathbf{z} \approx \mathbf{V}_k \mathbf{T}_k^{-s} \mathbf{V}_k^T \mathbf{z} = \mathbf{V}_k \mathbf{T}_k^{-s} \mathbf{e}_1 \|\mathbf{z}\|_{\mathbf{M}^{-1}}.$$

The complexity of the above operations depends on the complexity corresponding to the application of the inverse of \mathbf{M} . If this operation can be achieved in $O(n)$ operations, then the overall complexity of computing $(\mathbf{M}^{-1}\mathbf{A})^s\mathbf{z}$, or $(\mathbf{M}^{-1}\mathbf{A})^{-s}\mathbf{z}$ is of order $O(kn)$ for $k \ll n$, with storage requirements of the same order. We point out that $(\mathbf{M}^{-1}\mathbf{A})^{-s}\mathbf{z}$ can be also computed inverting the matrices \mathbf{M} and \mathbf{A} and applying the Lanczos algorithm as before. The tridiagonal matrix $\tilde{\mathbf{T}}_k$ that we will obtain will be a different approximation of the inverse of \mathbf{T} . Several applications of the algorithm are discussed in [1].

4.3 Stopping criterion

The scalar product

$$(\mathbf{z}, (\mathbf{M}^{-1}\mathbf{A})^s \mathbf{z})_{\mathbf{M}} \quad (4.5)$$

is always non negative and it is equal to if and only if $\mathbf{z} = 0$, owing to the self-adjoint property and positive definiteness of the matrix $\mathbf{M}^{-1}\mathbf{A}$ respect the \mathbf{M} -scalar product. Taking into account (4.4), we have that

$$(\mathbf{z}, (\mathbf{M}^{-1}\mathbf{A})^s \mathbf{z})_{\mathbf{M}} = \mathbf{e}_1^T \mathbf{T}^s \mathbf{e}_1 \|\mathbf{z}\|_{\mathbf{M}}^2.$$

The sequence of the approximations

$$\mathbf{e}_1^T \mathbf{T}_k^s \mathbf{e}_1 \rightarrow \mathbf{e}_1^T \mathbf{T}^s \mathbf{e}_1$$

for $k \rightarrow n$. Let $\varepsilon \in (0, 1)$ be a threshold fixed a priori (`data%tol` = ε). The stopping criterion

$$\text{if } |\mathbf{e}_1^T \mathbf{T}^s \mathbf{e}_1 - \mathbf{e}_1^T \mathbf{T}_k^s \mathbf{e}_1| \leq \varepsilon \mathbf{e}_1^T \mathbf{T}^s \mathbf{e}_1 \quad \text{then stop} \quad (4.6)$$

will ideally check that the actual approximation at the k -step of the Lanczos process is a good approximation. We substitute in the code the unknown value $\mathbf{e}_1^T \mathbf{T}^s \mathbf{e}_1$ with $\mathbf{e}_1^T \mathbf{T}_{k+d}^s \mathbf{e}_1$, where d (`data%d`) is a small integer (in [1] a small value of k and d are enough to give a good approximation in the case of elliptic differential problems). Therefore, the practical stopping criterion used in the code is

$$\text{if } |\mathbf{e}_1^T \mathbf{T}_{k+d}^s \mathbf{e}_1 - \mathbf{e}_1^T \mathbf{T}_k^s \mathbf{e}_1| \leq \varepsilon \mathbf{e}_1^T \mathbf{T}_{k+d}^s \mathbf{e}_1 \quad \text{then stop}, \quad (4.7)$$

i.e. we stop when we do not see any improvement in the norm after d additional steps. We point out that given the spectral decomposition of \mathbf{T}_k

$$\mathbf{T}_k = \mathbf{S}_k^T \Delta_k \mathbf{S}_k$$

we have that

$$\mathbf{e}_1^T \mathbf{T}_k^s \mathbf{e}_1 = \sum_j (\delta_k)_j^s (\mathbf{S}_k)_{1,j}^2.$$

Finally, we advise the user that $d = 3$ is a reasonable choice. Moreover, if \mathbf{A} and \mathbf{M} are respectively the stiffness and the mass matrices of a finite-element approximation of differential operators the tolerance value ϵ can be of the order of the maximum diameter of the elements in the mesh [1]

References

- [1] M. Arioli, D. Loghin, Discrete interpolation norms with applications, *SIAM J. Num. Anal.* 47 (4) (2009) 2924–2951.
- [2] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Classics in Applied Mathematics 20, SIAM, Philadelphia, USA, 1998.

5 EXAMPLE OF USE

Suppose we wish to compute $(\mathbf{M}^{-1}\mathbf{A})^{1/2}\mathbf{u}$, where

$$\mathbf{M} = \mathbf{I}, \quad \mathbf{A} = \frac{1}{9} \begin{bmatrix} 2 & -1 & & & & & & & & \\ -1 & 2 & -1 & & & & & & & \\ & -1 & 2 & -1 & & & & & & \\ & & -1 & 2 & -1 & & & & & \\ & & & -1 & 2 & -1 & & & & \\ & & & & -1 & 2 & -1 & & & \\ & & & & & -1 & 2 & -1 & & \\ & & & & & & -1 & 2 & -1 & \\ & & & & & & & -1 & 2 & -1 \\ & & & & & & & & -1 & 2 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \quad (5.1)$$

The following program uses EA20 to compute the result.

```
program main
```

```
! Simple code to illustrate row entry to hsl_ea20
use HSL_EA20_double
implicit none
```

```
! Derived types
type (ea20_control) :: cntl
type (ea20_info)    :: info
type (ea20_reverse) :: rev
```

```
! Parameters
```

```
integer, parameter :: wp = kind(0.0d0)
integer, parameter :: n = 10
double precision, parameter :: zero = 0.0d0, one = 1.0d0, two = 2.0d0
```

```

integer :: i,ido
double precision, dimension(n) :: u
double precision, allocatable :: w(:, :)
double precision :: s

!! set u
u = 0.d0
u(2:n-1) = one

!! set data
s = 0.5d0

!! set cntl
cntl%d      = 3          !! delay
cntl%tol    = 1.d-2      !! convergece tolerance
cntl%maxit  = 10         !! max number iteration

cntl%diagnostics_level = 1 !! full error check

ido = -1

do while ( ido .ne. 0 .and. info%flag == 0)

    call EA20(n,u,s,ido,w,cntl,info,rev)

    select case ( ido )

    case (0)
        exit

    case (1) !! Matrix-Vector product w_out = A w(:,1)
        w(1,2) = two*w(1,1) - w(2,1)
        do i=2,n-1
            w(i,2) = -w(i-1,1)+two*w(i,1)-w(i+1,1)
        end do
        w(n,2) = -w(n-1,1)+two*w(n,1)

    case (2) !! Matrix-Vector product w(:,2) = M w(:,1)
        w(:,2) = w(:,1)

    case (3) !! Matrix-Vector product w_out = inv(M) w(:,1)
        w(:,2) = w(:,1)

    end select

end do
! end if

if (info%flag .ge. 0) then

```

```
write(*,'(a,i5)') 'error code = ',info%flag
!! print the final solution
write(*,'(a,i5)') 'number of iterations = ',info%iter
write(*,'(a,1x,1pe14.2)') 'estimated final error = ',info%error
write(*,'(a)') '      i      X(i)'
write(*,'(i5,1x,1pe14.7)') (i,u(i), i=1,n)
end if
```

```
end program main
```

This produces the following output:

```
error code =      0
number of iterations =      6
estimated final error =      2.14E-03
  i      X(i)
 1     -5.000E-01
 2      7.469E-01
 3      3.136E-01
 4      2.297E-01
 5      2.028E-01
 6      2.028E-01
 7      2.297E-01
 8      3.136E-01
 9      7.469E-01
10     -5.000E-01
```