1 SUMMARY

This subroutine uses the Lanczos algorithm to **compute in parallel the part of the spectrum of a large symmetric matrix A that lies in a specified interval,** that is, it computes eigenvalues without regard to multiplicities. The user is required to partition the vectors into contiguous sections of similar sizes, each residing on a separate process. He or she must provide parallel code that computes **u**+**Av** for any given vectors **u** and **v**. The partitions should be chosen to make this computation straightforward and rapid.

Auxiliary calls allow corresponding eigenvectors to be found. In this case, the user is responsible for storing each vector **v** and restoring it during the eigenvector calculation.

MPI is used for message passing. The system must be homogeneous, that is, all the processes must be identical.

ATTRIBUTES — **Version:** 2.0.0. (15 November 2022) **Types:** Real (double). **Calls:** FA14, FD15. **Original date:** April 2004. **Remark:** EP25 is a parallel version of EA25. **Origin:** J.K.Reid, Rutherford Appleton Laboratory.

2 HOW TO USE THE PACKAGE

2.1 Parallel programming

The user is responsible for initializing MPI by calling MPI_INIT on each process and defining a communicator. Note that many MPI calls require that INCLUDE 'mpif.h' be used. All calculations are performed redundantly on all processes, apart from those associated with distributed vectors. The processes synchronize within each iteration because they need to calculate inner products of distributed vectors.

It is required that the system be homogeneous. Therefore on each call and each return, each of the arguments EL, ER, ACC, LEIG, LX, LALFA, LP, NLAN, IFLAG, EIG, JEIG, NEIG, X, DEL, NU, ALFA, BETA, JLAN, and JFLAG should have identical values on all the processes. We have included checks in the sample code in Section 5 that all the values of IFLAG are identical since differences would seriously damage the flow of control. Such checks are not needed if the user is certain that the system is homogeneous.

2.2 Overall control

Rather than ask the user to furnish a subroutine to compute **u**+**Av** inside the Lanczos process, control is returned at each iteration so that code for computing **u**+**Av** can be written by the user with access to any other data needed. On such returns, those eigenvalues that have been so far computed will be available. Following successful termination for one interval, a new interval of the spectrum of the same matrix may be processed more economically by a first call with the ends of the new interval held in the arguments EL and ER and all other arguments unchanged.

Provided the user has stored all the vectors **v** for which **u**+**Av** was calculated, auxiliary calls to EP25ED may be made to find eigenvectors corresponding to computed eigenvalues. Again, control is returned at each iteration to give the user flexibility over how the vectors are stored.

Simple examples of suitable calls are given in Section 5.

2.3 The argument list and calling sequence (initialization)

The EP25ID entry must be called prior to the first call to the EP25AD entry to specify the matrix order and the communicator and to initialize private workspace. Any tasks that the user was previously performing with the communicator (or any communicator that overlaps it) must have completed.

```
CALL EP25ID(N,MPI_COMM,KEEP,RKEEP)
```

N is an INTEGER variable that must be set to the order n of the matrix **A**. N is not altered. **Restriction:** N > 0.

MPI_COMM is an INTEGER variable that must be set to the MPI communicator to be used. It is not altered.

KEEP is an INTEGER array of length 15 used by EP25 as private workspace and must not be altered by the user.

RKEEP is a DOUBLE PRECISION array of length 7 used by EP25 as private workspace and must not be altered by the user.

2.3 The argument list and calling sequence (main entry)

```
      CALL EP25AD(LOCALN,EL,ER,ACC,LEIG,LX,LALFA,LP,NLAN,IFLAG,U,V,
     *            EIG,JEIG,NEIG,X,DEL,NU,ALFA,BETA,KEEP,RKEEP)
```

LOCALN is an INTEGER variable that must be set to the size of that part of each vector that is held locally. The sum of all these must equal n. It is not altered. **Restriction:** LOCALN ≥ 0, $\sum$LOCALN $= n$.

EL,ER are DOUBLE PRECISION variables that must be set to indicate the range [EL,ER] within which the eigenvalues are required. EL may be set to a very large negative number if no lower bound is required and ER may be set to a very large positive number if no upper bound is required. If EL $\geq$ ER, it is assumed that all eigenvalues are wanted. EL, ER are not altered.

ACC is a DOUBLE PRECISION variable that must be set to the required relative precision (relative to the largest eigenvalue of **A**). If ACC is negative or very small, as much accuracy as the precision reasonably allows will be obtained. ACC is not altered.

LEIG is an INTEGER variable that must be set to the length of array EIG. It must be as least as large as the number of distinct eigenvalues in [EL,ER]. LEIG is not altered.

LX is an INTEGER variable that must be set to the length of arrays X, DEL and NU. A value three times the number of distinct eigenvalues in [EL,ER] usually suffices. LX is not altered. **Restriction:** LX > 5.

LALFA is an INTEGER variable that must be set to the length of arrays ALFA and BETA. It limits the number of Lanczos steps possible. The number needed varies widely from problem to problem. In our experimental use of the code we have never needed more than $5n$, and when looking at a small interval near the end of the spectrum have sometimes needed less than $n/4$. LALFA is not altered.

LP is an INTEGER variable that must be set to the unit number for diagnostic messages, which are printed only on process 0. If LP ≤ 0, no messages are printed. LP is not altered.

NLAN is an INTEGER variable that need not be set initially. On return, it holds the index of the Lanczos vector of which part is stored in V. NLAN must not be altered by the user.

IFLAG is an INTEGER variable that must be set prior to the first call for a particular matrix to one of the following values:

- 2 The user does not want to specify a start vector.
- 3 The user has distributed a start vector **v** in V.

It should not otherwise be changed by the user. In particular it should be left at the value

- 0 if the user wishes to follow a successful call for one interval of the spectrum of a matrix by a call for another interval of the spectrum of the same matrix.

On output it has one of the following values:

- 0 Successful completion of calculation.
- 1 Normal intermediate return.
- –1 The sum of all the values of LOCALN does not equal the value of N sent to EP25ID.
- –2 LEIG is too small. This error return permits subsequent recall with a reduced range [EL,ER], IFLAG reset to zero, and other arguments unchanged.

–3 LX is too small. This error return permits subsequent recall with a reduced range [EL,ER], IFLAG reset to zero, and other arguments unchanged.

–4 LALFA is too small.

–5 N ≤ 0, LX ≤ 5, or LALFA ≤ 0.

–6 Premature termination, probably caused by the start vector **v** being an eigenvector. EIG will hold NEIG eigenvalues but there may be more in the given interval. If the user began with IFLAG=3, he or she is recommended to restart with IFLAG=2.

–7 The input value of IFLAG is outside the range [0,3].

U,V are DOUBLE PRECISION arrays of length LOCALN. On a return with IFLAG the user must compute the local part of **u**+**Av** for the vectors **u** and **v** distributed in U and V, without altering V. If eigenvectors are required, the user must store V (perhaps in a direct-access file) and restore it during the eigenvector calculation. This is illustrated in Section 5. If the user wishes to specify the Lanczos starting vector, it should be distributed in V before the first (IFLAG=3) call for the matrix **A**. Otherwise, the user must never set or alter U or V.

EIG is a DOUBLE PRECISION array of length LEIG. It should never be set or altered by the user. On any return it contains NEIG computed eigenvalues, stored in increasing order. If IFLAG ≠ 0, there may be more in the requested interval.

JEIG is an INTEGER array of size (2,LEIG). It should never be set or altered by the user. On any return JEIG(1,I) contains the Lanczos step at which EIG(I) was accepted and JEIG(2,I) contains a value needed for the recurrences to get the eigenvector, for I = 1, 2,..., NEIG.

NEIG is an INTEGER variable that should never be set or altered by the user. On return it holds the number of eigenvalues in EIG.

X,DEL are DOUBLE PRECISION arrays of length LX, that should never be set or altered by the user.

NU is an INTEGER array of length LX that should never be set or altered by the user.

ALFA,BETA are DOUBLE PRECISION arrays of length LALFA used to hold the tridiagonal matrix generated by the Lanczos process. The user should never set or alter these arguments.

KEEP is an INTEGER array of length 15 used by EP25 as private workspace, see Section 2.4.

RKEEP is a DOUBLE PRECISION array of length 7 used by EP25 as private workspace, see Section 2.4.

2.3 The argument list and calling sequence (eigenvector entry)

```
      CALL EP25ED(LOCALN,LALFA,LP,JLAN,EIG,JEIG,NEIG,
     *            ALFA,BETA,LY,LZ,IFLAG,Y,W,Z,KEEP)
```

LOCALN is an INTEGER variable that must be set to the size of that part of each vector that is held locally. It is not altered. **Restriction:** LOCALN ≥ 0, $\sum$LOCALN = *n*.

LALFA is an INTEGER variable that must be set to the length of arrays ALFA and BETA. LALFA is not altered.

LP is an INTEGER variable that must be set to the unit number for diagnostic messages, which are printed only on process 0. If LP ≤ 0, no messages are printed. LP is not altered.

JLAN is an INTEGER variable that need not be set initially. On return with IFLAG=1, it holds the index of the Lanczos vector of which the local part must be recoved and placed in W. JLAN must not be altered by the user.

EIG is a DOUBLE PRECISION array of length NEIG. It must hold some or all of the eigenvalues computed on a call of EP25AD. Note that where EP25AD finds a large number of eigenvalues, the corresponding eigenvectors may be found by a sequence of calls of EP25ED each of which finds some of them, so avoiding massive storage demands in arrays Y and Z. EIG is not altered.

JEIG is an INTEGER array of size (2,NEIG). It must hold the information on the eigenvalues whose eigenvectors are

wanted, as returned by EP25AD, namely JEIG(1,I) must contain the Lanczos step at which EIG(I) was accepted and JEIG(2,I) must contain the value needed for the recurrences to get the eigenvector, for I = 1, 2,..., NEIG. JEIG is not altered.

NEIG is an INTEGER variable that must be set to the number of eigenvalues in EIG. NEIG is not altered.

ALFA,BETA are DOUBLE PRECISION arrays of length LALFA that must be as left by EP25AD. The user should never set or alter these arguments.

LY is an INTEGER variable that must be set to the first dimension of array Y. It is not altered. **Restriction:** LY ≥ LOCALN.

LZ is an INTEGER variable that must be set to the first dimension of array Z. It is not altered. **Restriction:** LZ ≥ max(JEIG(1,I), I = 1, 2, ..., NEIG).

IFLAG is an INTEGER variable that must be set to the value 2 on first entry for a set of eigenvectors. On return it has one of the following values:

- 0 Successful completion of calculation.
- 1 Normal intermediate return.
- –1 The sum of all the values of LOCALN does not equal the value of N sent to EP25ID.
- –2 LY < LOCALN.
- –3 LZ < max(JEIG(1,I), I = 1, 2, ..., NEIG).
- –4 LOCALN ≤ 0, LALFA ≤ 0, or NEIG ≤ 0.
- –5 The input value of IFLAG is neither 1 nor 2.

Y is a DOUBLE PRECISION array of size (LY,NEIG) used to return eigenvectors. The local part of the eigenvector corresponding to EIG(I) is placed in Y(K,I), K = 1, 2, ..., LOCALN.

W is a DOUBLE PRECISION array of length LOCALN into which the local part of the Lanczos vector must be placed on an entry with IFLAG = 1.

Z is a DOUBLE PRECISION array of size (LZ,NEIG) used for workspace.

KEEP is an INTEGER array of length 15 used by EP25 as private workspace, see Section 2.4.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided in the arguments X, DEL, NU, ALFA, BETA, W and Z.

Other routines called directly: FA14AD, FD15AD, EP25CD, EP25DD, EP25GD.

Input/output: Diagnostic messages on unit LP (see Section 2.5 and Section 2.6).

Restrictions: LOCALN ≥ 0, LX > 5, LY ≥ LOCALN, LZ ≥ max(JEIG(1,I), I = 1, 2,..., NEIG), $\sum$LOCALN = n.

4 METHOD

The Lanczos algorithm generates a sequence of symmetric tridiagonal matrices $\mathbf{T}_j$, each of which is the j-th order leading principal submatrix of all subsequent tridiagonal matrices. Each eigenvalue of **A** is the limit of a sequence constructed by taking a particular eigenvalue from each $\mathbf{T}_j$. The subroutine aims to follow the convergence of these sequences by constructing intervals containing the eigenvalues of $\mathbf{T}_j$ and having lengths comparable with the separation from corresponding eigenvalues of $\mathbf{T}_{j-1}$, continuing until it is convinced that all eigenvalues have been found. For further details see 'Tracking the progress of the Lanczos algorithm for large symmetric eigenproblems', by

B. N. Parlett and J. K. Reid, IMA Journal of Numerical Analysis (1981) **1**, 135-155.

We would like to thank I. M. Smith, M. Pettipher and L. Margetts of Manchester University for suggesting the extension to a parallel code.

5 EXAMPLE OF USE

Suppose the spectrum of the matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{T} & -\mathbf{I} & \\ -\mathbf{I} & \mathbf{T} & -\mathbf{I} \\ & -\mathbf{I} & \mathbf{T} \end{pmatrix}, \quad \text{where} \quad \mathbf{T} = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & -1 & \\ & -1 & 4 & -1 \\ & & -1 & 4 \end{pmatrix}$$

is required. The following code runs on three processes, partitioning the vectors into three segments of size 4. It finds the spectrum in two parts and finds the eigenvectors corresponding to the two smallest eigenvalues in the two parts. It uses three direct-access files with different names and different units to store and restore the Lanczos vectors.

```
      INTEGER LALFA,LEIG,LX,LY,LZ,N,LOCALN
      PARAMETER (LALFA=40,LEIG=10,LX=50,LY=4,LZ=LALFA,N=12,LOCALN=4)
      DOUBLE PRECISION
     +      ACC,ALFA(LALFA),BETA(LALFA),DEL(LX),EIG(LEIG),EL,ER,
     +      RKEEP(7),U(LOCALN),V(LOCALN),VKEEP(LOCALN,LALFA),X(LX),
     +      Y(LOCALN,6),Z(LZ,6)
      INTEGER I,IFLAG,IRANGE,ITER,UNIT,J,JEIG(2,LEIG),JFLAG,JLAN,K,
     +  KEEP(15),LEN,LP,NEIG,NLAN,NU(LX)
      INCLUDE 'mpif.h'
      INTEGER IER,LAST,MPI_COMM,STAT(MPI_STATUS_SIZE),NPROC,PROC
      CHARACTER*8 FILE
C NPROC Number of processors
C PROC  Index of this processor
      CALL MPI_INIT(IER)
      MPI_COMM=MPI_COMM_WORLD
      CALL MPI_COMM_RANK(MPI_COMM,PROC,IER)
      CALL MPI_COMM_SIZE(MPI_COMM,NPROC,IER)
      IF (NPROC.LT.3) THEN
         WRITE(6,*) 'This test needs at least 3 processes'
         GO TO 100
      END IF
C Initialize EP25
      CALL EP25ID(N,MPI_COMM,KEEP,RKEEP)
C Perform eigenvalue calculations for two different ranges
      ACC=1.0D-9
      LP=6
      IFLAG=2
      DO 60 IRANGE=1,2
        IF(IRANGE.EQ.1)THEN
          EL=0.0D0
          ER=4.0D0
        ELSE
          EL=4.0D0
          ER=8.0D0
        END IF
        DO 10 ITER=1,LALFA
          CALL EP25AD
     +         (LOCALN,EL,ER,ACC,LEIG,LX,LALFA,LP,NLAN,IFLAG,U,V,
     +          EIG,JEIG,NEIG,X,DEL,NU,ALFA,BETA,KEEP,RKEEP)
C Exit the loop if any process requests it
          CALL MPI_ALLREDUCE(IFLAG,JFLAG,1,MPI_INTEGER,MPI_MIN,
```

```
     +                   MPI_COMM,IER)
           IF(JFLAG.NE.1)GO TO 20
 C Store the Lanczos vector
           DO 5 I = 1, LOCALN
              VKEEP(I,NLAN) = V(I)
     5     CONTINUE
 C Form U=U+A*V
         CALL MULT(MPI_COMM,PROC,U,V)
    10   CONTINUE
         GO TO 70

 C Check that all processes have completed
    20   IF(JFLAG.NE.0) GO TO 70

 C Write out spectrum found
         IF(PROC.EQ.0)WRITE (LP,'(/,A,F5.1,A,F5.1,A,/,10F7.4)')
     +    ' Spectrum in interval (',EL,',',ER,') is',(EIG(I),I=1,NEIG)

 C Find two eigenvectors
         NEIG=2
         IFLAG = 2
         LAST = KEEP(1)
         DO 30 ITER=1,LALFA
            CALL EP25ED
     +       (LOCALN,LALFA,LP,JLAN,EIG,JEIG,NEIG,
     *       ALFA,BETA,LY,LZ,IFLAG,Y,V,Z,KEEP)
 C Exit the loop if any process requests it
            CALL MPI_ALLREDUCE(IFLAG,JFLAG,1,MPI_INTEGER,MPI_MIN,
     +                   MPI_COMM,IER)
           IF(JFLAG.NE.1)GO TO 35
 C Recover the Lanczos vector
           DO 25 I = 1, LOCALN
              V(I) = VKEEP(I,JLAN)
    25     CONTINUE
    30   CONTINUE
    35   DO 50 I=1,2
           IF(PROC.EQ.0)THEN
             WRITE(LP,'(A,F7.4,A)')
     +      ' Eigenvector corresponding to eigenvalue',EIG(I), ' is'
           WRITE(LP,'(4F10.4)')(Y(K,I),K=1,LOCALN)
         END IF
         DO 40 J=1,2
           CALL MPI_BARRIER(MPI_COMM,IER)
           IF (PROC.EQ.J) CALL MPI_SEND(Y(1,I),LOCALN,
     +                    MPI_DOUBLE_PRECISION,0,1,MPI_COMM,IER)
           IF (PROC.EQ.0) THEN
             CALL MPI_RECV(Y(1,I),LOCALN,
     +              MPI_DOUBLE_PRECISION,J,1,MPI_COMM,STAT,IER)
           WRITE(LP,'(4F10.4)')(Y(K,I),K=1,LOCALN)
           END IF
    40     CONTINUE
    50   CONTINUE
    60 CONTINUE
       GO TO 100

 C EP25AD is signalling failure
    70 IF(PROC.EQ.0)WRITE(LP,'(A)')' EP25AD has failed.'

   100 CALL MPI_FINALIZE(IER)
       END
```

```
      SUBROUTINE MULT(MPI_COMM,PROC,U,V)
C Form U=U+A*V
      INTEGER LOCALN
      PARAMETER (LOCALN=4)
      INTEGER MPI_COMM,PROC
      DOUBLE PRECISION  U(LOCALN),V(LOCALN)

      INCLUDE 'mpif.h'
      DOUBLE PRECISION  VNEIB(LOCALN)
      INTEGER J,IER,STAT(MPI_STATUS_SIZE)
C First handle local part
      DO 10 J=1,4
         U(J) = U(J) + 4.0D0*V(J)
         IF(J.GT.1)U(J)=U(J)-V(J-1)
         IF(J.LT.4)U(J)=U(J)-V(J+1)
   10 CONTINUE
C Subtract contribution from the left
      IF (PROC.LT.2) CALL MPI_SEND(V,4,MPI_DOUBLE_PRECISION,
     +                    PROC+1,1,MPI_COMM,IER)
      IF (PROC.GT.0) CALL MPI_RECV(VNEIB,4,MPI_DOUBLE_PRECISION,
     +                    PROC-1,1,MPI_COMM,STAT,IER)
      IF (PROC.GT.0) THEN
         DO 20 J=1,4
            U(J) = U(J) - VNEIB(J)
   20    CONTINUE
      END IF
C Subtract contribution from the right
      IF (PROC.GT.0) CALL MPI_SEND(V,4,MPI_DOUBLE_PRECISION,
     +                    PROC-1,1,MPI_COMM,IER)
      IF (PROC.LT.2) CALL MPI_RECV(VNEIB,4,MPI_DOUBLE_PRECISION,
     +                    PROC+1,1,MPI_COMM,STAT,IER)
      IF (PROC.LT.2) THEN
         DO 30 J=1,4
            U(J) = U(J) - VNEIB(J)
   30     CONTINUE
      END IF
      END
```

This produces the following output

```
 Spectrum in interval (  0.0,  4.0) is
 0.9678 1.9678 2.3820 3.2038 3.3820 3.7962
 Eigenvector corresponding to eigenvalue 0.9678 is
   -0.1859   -0.3008   -0.3008   -0.1859
   -0.2629   -0.4253   -0.4253   -0.2629
   -0.1859   -0.3008   -0.3008   -0.1859
 Eigenvector corresponding to eigenvalue 1.9678 is
   -0.3008   -0.1859    0.1859    0.3008
   -0.4253   -0.2629    0.2629    0.4253
   -0.3008   -0.1859    0.1859    0.3008

 Spectrum in interval (  4.0,  8.0) is
 4.2038 4.6180 4.7962 5.6180 6.0322 7.0322
 Eigenvector corresponding to eigenvalue 4.2038 is
   -0.1859    0.3008   -0.3008    0.1859
   -0.2629    0.4253   -0.4253    0.2629
   -0.1859    0.3008   -0.3008    0.1859
 Eigenvector corresponding to eigenvalue 4.6180 is
   -0.4253    0.2629    0.2629   -0.4253
    0.0000    0.0000    0.0000    0.0000
    0.4253   -0.2629   -0.2629    0.4253
```