1 SUMMARY

This subroutine **calculates all the eigenvalues and eigenvectors of a complex Hermitian matrix.** Given an n by n matrix **H** with elements $h_{ij} = conj\{h_{ji}\}$, it finds solutions λ_i and $\mathbf{x}_i$, i=1,2,...,n, where $\mathbf{H}\mathbf{x}_i = \lambda_i \mathbf{x}_i$. The Jacobi method is used.

ATTRIBUTES — Version: 1.1.0. **Types:** Real (single, double). **Language:** `EC23AD` uses `COMPLEX*16` **Original date:** May 2001. **Remark:** `EC23` is a threadsafe version of `EC13`. **Origin:** S.Clough,Harwell.

2 HOW TO USE THE PACKAGE

2.1 The argument list and calling sequence.

The single precision version

```
CALL EC23A (N,A,ND,D,V,EIGVEC,IROT,IW,LP)
```

The double precision version

```
CALL EC23AD (N,A,ND,D,V,EIGVEC,IROT,IW,LP)
```

`N` is an `INTEGER` variable which must be set by the user to n the order of the matrix **H**. This argument is not altered by the subroutine.

`A` is a `COMPLEX` array (`COMPLEX*16` for the D version) of dimensions `(ND,N)`. On entry, the array elements `A(I,J)` for `I` ≤ `J, J=1,N` must contain the upper triangular part of the matrix whose eigensystem the user wishes to calculate. These values will be overwritten by the subroutine.

`ND` is an `INTEGER` variable which must be set by the user to the first dimension of arrays `A` and `V`. **Restriction:** `ND` ≥ `N` > 0. This argument is not altered by the subroutine.

`D` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `N`. On return from the subroutine it will contain the eigenvalues of the matrix **H**.

`V` is a `COMPLEX` array (`COMPLEX*16` for the D version) of dimensions `(ND,N)`. On return from the subroutine, it will hold the eigenvectors of the matrix **H**, its ith column holding the eigenvector corresponding to the eigenvalue λ_i of **H**, i=1,2,...,n.

`EIGVEC` is a `LOGICAL` variable which must be set by the user to `.TRUE.` if the eigenvectors as well as the eigenvalues of **H** are required or to `.FALSE.` if only the eigenvalues are required. This argument is not altered by the subroutine.

`IROT` is an `INTEGER` variable whose value should be set by the user to the maximum number of Jacobi rotations he or she wishes to allow. If this maximum is exceeded, an error message is printed (see §2.2). On successful completion, `IROT` is set to the number of rotations actually performed. Usually about $5n^2$ rotations suffice. **Restriction:** `IROT` > 0.

`IW` is an `INTEGER` work array of length `N`.

`LP` is an `INTEGER` variable to be used by the routine when writing error messages. It must be set to a valid unit number (normally 6), or set to zero to suppress messages.

2.2 Error messages.

There is only one error message which occurs if the algorithm has not converged after the `IROT` rotations specified by the user. In this case the message

```
TOO MANY ITERATIONS TAKEN BY EC23A/AD
```

will be output on the unit given by the argument LP and IROT will be set to its input value.

3 GENERAL INFORMATION

Use of common: None.

Workspace: INTEGER IW(N) (see §2.1).

Other routines called directly: None.

Input/output: In the event of non-convergence a diagnostic message is printed on the unit given by the argument LP, see section 2.2.

Restrictions:

ND $\geq$ N $>$ 0,

IROT $>$ 0.

Portability: EC23AD uses COMPLEX*16 facility.

4 METHOD

The method used is a version of Jacobi's method. At each stage h_{pq}, a large off-diagonal element of the modified matrix, is reduced to zero by a rotation in the $(p,q)^{\text{th}}$ plane. The off-diagonal element is chosen to maximize

$$(|\text{Re}(h_{ij})|,|\text{Im}(h_{ij})|)$$

(This method differs from the classical Jacobi method, in which the off-diagonal element with the largest absolute value is chosen as h_{pq} , due to efficiency considerations. Convergence is not significantly affected). The process stops when all the off-diagonal elements are small compared with the diagonal elements.

The method is quadratically convergent and will, in normal cases, require only a small multiple of n^2 rotations to converge.

5 EXAMPLE OF USE

Suppose that we are given a complex Hermitian matrix

$$\mathbf{H} = \begin{bmatrix} 1.2 & 0.5+2.0i & 1.3-0.3i & 0.7-0.2i \\ 0.5-2.0i & 0.9 & 2.1+1.3i & 1.7+1.7i \\ 1.3-0.3i & 2.1-1.3i & 1.5 & 0.5-1.3i \\ 0.7+0.2i & 1.7-1.7i & 0.5+1.3i & 0.2 \end{bmatrix}$$

and we wish to compute all the eigenvalues. The Fortran code for doing this might be

```
      COMPLEX*16 A(4,4),V(4,4)
      DOUBLE PRECISION D(4)
      INTEGER IROT,IW(4),J,N,ND
      LOGICAL EIGVEC
      DATA A(1,1),A(1,2),A(1,3),A(1,4)
     * /(1.2D0,0.0D0),(0.5D0,2.0D0),(1.3D0,0.3D0),(0.7D0,-0.2D0)/
      DATA A(2,2),A(2,3),A(2,4)
     *                 /(0.9D0,0.0D0),(2.1D0,1.3D0),(1.7D0,1.7D0)/
      DATA A(3,3),A(3,4)           /(1.5D0,0.0D0),(0.5D0,-1.3D0)/
      DATA A(4,4)                                 /(0.2D0,0.0D0)/
      N=4
      ND=4
      EIGVEC=.TRUE.
      IROT=50
      CALL EC23AD (N,A,ND,D,V,EIGVEC,IROT,IW,6)
      WRITE (6,'(A,1P,10E12.4)') ' Eigenvalues are',(D(J),J=1,N)
      END
```

The output from the code is

```
 Eigenvalues are  2.1672E+00 -3.5233E+00  5.3072E+00 -1.5115E-01
```