1 SUMMARY

This routine uses the **MINRES method to solve the** $n \times n$ **symmetric but possibly indefinite linear system** $\mathbf{Ax} = \mathbf{b}$**, optionally using preconditioning.** If $\mathbf{M} = \mathbf{PP}^T$ is the preconditioning matrix, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{PAP}^T$ and $\bar{\mathbf{b}} = \mathbf{Pb}$ and recovers the solution $\mathbf{x} = \mathbf{P}^T\bar{\mathbf{x}}$. Reverse communication is used for preconditioning operations and matrix-vector products of the form $\mathbf{Az}$.

ATTRIBUTES — Version: 1.1.0 (20 March 2023). **Interfaces:** Fortran **Types:** Real (single, double). **Original date:** April 2015. **Origin:** T. Rees, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a `USE` statement

Single precision version

```
USE HSL_MI32_single
```

Double precision version

```
USE HSL_MI32_double
```

If it is required to use both modules at the same time, then the derived types (Section 2.4) must be renamed in one of the `USE` statements.

The following procedures are available to the user:

`mi32_minres` is called repeatedly to solve the system using a reverse communication interface. On each return, the user must provide additional information and, if necessary, recall the subroutine.

`mi32_finalize` deallocates array components of the private derived data type (allocated by `mi32_minres`), and should be called at the end of the solution process.

2.2 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types `mi32_keep`, `mi32_control`, and `mi32_info`.

The following pseudo-code illustrates this.

```
use hsl_mi32_double
...
type(mi32_keep) :: keep
type(mi32_control) :: control
type(mi32_info) :: info
```

The components of `mi32_control` and `mi32_info` are explained in Section 2.4.1 and 2.4.2. The components of `mi32_keep` are priavte and used to pass data between calls to the subroutines.

2.3 Argument lists and calling sequences

2.3.1 Integer, real and package types

INTEGER denotes default INTEGER.

REAL denotes default real if the single precision version is being used, and double precision real if the double precision version is being used.

We use the term **package type** to mean default real if the single precision version is being used, and double precision real for the double precision version.

2.3.2 The linear system solution subroutine

The linear system solution algorithm uses a reverse communication interface and must be called repeatedly (based on the value of action) as follows:

```
call mi32_minres( action, n, X, V_in, V_out, keep, control, info )
```

action is a scalar INTENT(INOUT) argument of type INTEGER that is used for reverse communication. On the first call, action must be set to 1. On each subsequent return it specifies the action the user must perform as follows:

- <0 : An error has occured, see Section 2.5 for details.
- 0 : MINRES has sucessfully converged to a solution that has been returned as the vector X.
- 2 : The user must perform the preconditioning operation

$$\mathbf{y} := \mathbf{P}\mathbf{P}^T\mathbf{z},$$

where $\mathbf{P}\mathbf{P}^T$ is the preconditioning matrix, and recall mi32_minres. The vector $\mathbf{z}$ is available as the first n components of the array V_out, and $\mathbf{y}$ must be placed in V_in. No argument except V_in should be altered before recalling mi32_minres.

- 3 : The user must perform the matrix-vector product

$$\mathbf{y} := \mathbf{A}\mathbf{z}$$

and recall mi32_minres. The vector $\mathbf{z}$ is available as the first n components of the array V_out, and $\mathbf{y}$ must be placed in V_in. No argument except V_in should be altered before recalling mi32_minres.

- 4 : The user should test for convergence. This value will only occur when the user has opted to test convergence by setting control%own_stopping_rule to .TRUE.. If the user does not wish to test for convergence (we do not recommend the user tests for convergence each time action $= 4$ is returned) or if convergence has not been achieved, the user must recall mi32_minres without changing any of the arguments.

n is a scalar INTENT(IN) argument of type INTEGER that must be set to the number of unknowns, n. **Restriction:** n>0.

X is an array INTENT(INOUT) argument of dimension n and package type that holds an estimate of the solution $\mathbf{x}$ of the linear system. On initial entry (action=1), X must contain an estimate of the solution. On exit, X contains the current best estimate of the solution.

V_in is an array INTENT(INOUT) argument of dimension n and package type that is used to pass information to mi32_minres. The required content of the array is under the control of the parameter action (see above). On initial entry (action=1), V_in must contain the residual $\mathbf{Ax} - \mathbf{b}$.

V_out is a one-dimensional POINTER array of package type that is used to pass information from mi32_minres. The actual content of the array depends on the value of the parameter action (see above). Its allocation status and value must not be altered by the user.

keep is a scalar INTENT(INOUT) argument of type mi32_keep. It is used to hold data about the system being solved.

control is a scalar INTENT(IN) argument of type mi32_control.

info is a scalar INTENT(INOUT) argument of type mi32_info.

2.3.3 The termination subroutine

Pointer arrays holding private data are deallocated as follows:

```
call mi32_finalize( keep )
```

keep is a scalar INTENT(INOUT) argument of type mi32_keep exactly as for mi32_minres. On exit, its array components will have been deallocated.

2.4 The derived types

2.4.1 The derived data type for holding control parameters

The derived data type mi32_control is used to hold controlling data. The components, which are automatically given default values in the definition of the type, are:

out is a scalar variable of type default INTEGER that holds the Fortran unit for diagnostic printing. Printing is suppressed if out $<$ 0. The default is -1.

error is a scalar variable of type default INTEGER that holds the Fortran unit for error messages. Printing of error messages is suppressed if error $\leq$ 0. The default is 6.

itmax is a scalar variable of type default INTEGER that holds the maximum number of iterations that will be allowed in mi32_minres. If itmax is set to a negative number, it will be reset by mi32_minres to n+1. The default is -1.

conv_test_norm is a scalar variable of type default INTEGER that allows the user to select whether the algorithm tests for convergence in the M-norm (1) or the two-norm (2). If conv_test_norm = 2, then four additional vectors of length n will be stored. In general, the choice conv_test_norm = 2 will require a greater number of iterations, and each iteration will be more expensive as we perform an extra inner product and two additional vector additions per iteration. The default is 1. **Restriction:** conv_test_norm = 1 or 2.

own_stopping_rule is a scalar variable of type default LOGICAL that is set .TRUE. if the user intends to provide the stopping rule and .FALSE. otherwise. The default is .false..

precondition is a scalar variable of type default LOGICAL that is set .TRUE. if the user intends to provide a preconditioner and .FALSE. otherwise. The default is .true..

stop_relative and stop_absolute are scalar variables of package type that holds the relative and absolute convergence tolerances (see Section 4). If own_stopping_rule is .TRUE., stop_relative and stop_absolute are not accessed by MI32. Otherwise, the computed solution $\mathbf{x}$ is accepted by mi32_minres if $\|\mathbf{Ax}-\mathbf{b}\|_\star$ is less than or equal to $\max(\|\mathbf{Ax}_0-\mathbf{b}\|_\star * \texttt{stop\_relative}, \texttt{stop\_absolute})$, where $\mathbf{x}_0$ is the initial estimate of the solution. $\|\cdot\|_\star$ denotes the norm selected by the control parameter conv_test_norm. The default values are stop_relative = SQRT(EPSILON) and stop_absolute = 0.0.

2.4.2 The derived data type for informational parameters

The derived data type `mi32_info` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `mi32_info` are:

`rnorm` is a scalar variable of package type that holds the two norm of the residual, $||\mathbf{Ax}-\mathbf{b}||_\star$, where $\|\cdot\|_\star$ denotes the norm chosen by the control parameter `control_test_norm`.

`iter` is a scalar variable of type default `INTEGER` that holds the current iteration count.

`st` is a scalar variable of type default `INTEGER` that gives the status of the most recent array allocation.

2.5 Warning and error messages

A negative value of `action` on exit from `mi32_minres` indicates that an error has occurred. No further calls should be made until the problem has been resolved. Possible values are:

−1. The input parameter `n` is not positive.

−2. More than `control%itmax` iterations have been performed without obtaining convergence.

−3. The matrix **A** appears to be singular and the system inconsistent.

−4. An array allocation has failed. A message indicating the offending array is written on unit `control%error` and the returned allocation status is given by
`info%st`.

−5. A value of `control%conv_test_norm` other than `1` or `2` has been supplied.

2.6 Information printed

If `control%out` is positive, information about the progress of the algorithm will be printed on unit `control%out`. A one-line summary of each iteration will be given containing the iteration number, the norm of the residual, the latest diagonal and off-diagonal elements in the Lanczos tridiagonal matrix (see Section 2.4.2) and a flag indicating the pivot type used when factorizing this matrix.

3 GENERAL INFORMATION

Input/output: Output is under control of the arguments `control%error` and `control%out`.

Restrictions: `n` > 0.

4 METHOD

The method is iterative. Starting with the vector $(\mathbf{Ax}_0-\mathbf{b})/\|P^T(\mathbf{Ax}_0-\mathbf{b})\|_2$, a matrix of Lanczos vectors is built one column at a time so that the k-th column is generated during iteration k. The resulting $n\times k$ matrix $\mathbf{Q}_k$ has the property that

$$\mathbf{AQ}_k = \mathbf{Q}_k\mathbf{T}_k + \gamma_{k+1}\begin{bmatrix}\mathbf{0} & \cdots & \mathbf{0} & \mathbf{v}_{k+1}\end{bmatrix},$$

or, equivalently, $\mathbf{Q}_k^T\mathbf{AQ}_k = \mathbf{T}_k$, where T_k is tridiagonal. An approximation to the required solution may then be expressed formally as $\mathbf{x}_{k+1} = \mathbf{x}_0 - \mathbf{Q}_k\mathbf{y}_k$, where

$$\mathbf{y}_k = \arg\min\|\,\|\mathbf{r}_0\|_2\mathbf{e}_1 - \widehat{\mathbf{T}}_{k+1}\|_2.$$

Here $\widehat{\mathbf{T}}_{k+1} \in \mathbb{R}^{(k+1)\times k}$ is formed by appending the row $\begin{bmatrix} 0 & \cdots & 0 & \gamma_{k+1} \end{bmatrix}$ to $\mathbf{T}_k$, and $\mathbf{e}_1$ is the first unit vector. The MINRES algorithm implicitly solves this least squares problem using Givens rotations.

At the kth iteration MINRES finds the vector $\mathbf{x}_k \in P\left(\mathbf{x}_0 + \operatorname{span}\left\{AM\mathbf{r}_0, (AM)^2\mathbf{r}_0, \ldots, (AM)^{k-1}\mathbf{r}_0\right\}\right)$ that minimizes $\|A\mathbf{x}_k - \mathbf{b}\|_M$. If MINRES is used without a preconditioner we therefore have that the solution in the Krylov subspace with the minimal residual in the two-norm is found.

The aim of preconditioning is to accelerate the convergence of the method by clustering the eigenvalues of the preconditioned matrix $\bar{\mathbf{A}}$ around a small number of distinct values. If $\mathbf{A}$ is positive definite, this is often achieved by choosing $\mathbf{P}\mathbf{P}^T \approx \mathbf{A}^{-1}$. When $\mathbf{A}$ is indefinite, such a choice will not be possible, and the best that can be hoped for is that the eigenvalues of $\bar{\mathbf{A}}$ cluster around one positive and one negative value.

References

Paige, C. C., & Saunders, M. A. (1975). Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, **12** (4), 617-629.

5 EXAMPLE OF USE

Suppose we wish to solve the linear system

$$\begin{bmatrix} 1 & & & & & 1 & & & & \\ & 2 & & & & & 1 & & & \\ & & 3 & & & & & 1 & & \\ & & & 4 & & & & & 1 & \\ & & & & 5 & & & & & 1 \\ 1 & & & & & & & & & \\ & 1 & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 1 & & & & & & \\ & & & & 1 & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

The coefficient matrix is indefinite, and we choose to precondition with the positive definite matrix

$$\begin{bmatrix} 1 & & & & & & & & & \\ & 1/2 & & & & & & & & \\ & & 1/3 & & & & & & & \\ & & & 1/4 & & & & & & \\ & & & & 1/5 & & & & & \\ & & & & & 1 & & & & \\ & & & & & & 1 & & & \\ & & & & & & & 1 & & \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{bmatrix}.$$

We start from $\mathbf{x} = \mathbf{0}$. We may use the following code:

```
PROGRAM HSL_MI32_EXAMPLE
   USE HSL_MI32_DOUBLE
   IMPLICIT NONE

   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
   INTEGER, PARAMETER :: n = 10
```

```
   REAL(wp), DIMENSION(n) :: X
   REAL(wp), DIMENSION(n) :: V_in
   REAL(wp), POINTER, DIMENSION(:) :: V_out
   TYPE(MI32_KEEP) :: keep
   TYPE(MI32_CONTROL) :: control
   TYPE(MI32_INFO) :: info
   INTEGER :: i, action

   X = 0.0_wp                              ! Set the initial point
   DO i = 1, 5                             ! Set the initial residual
      V_in( i ) = - i - 1
   END DO
   V_in( 6 : n ) = -1
   action = 1
   DO                                      !  Solve the system
      CALL MI32_MINRES( action, n, X, V_in, V_out, keep, control, info )
      SELECT CASE( action )
      CASE( 2 )                            !  Use the preconditioner
         DO i = 1, 5
            V_in( i ) = V_out( i ) / i
         END DO
         V_in(  6 : n ) = V_out( 6 : n )
      CASE( 3 )                            !  Form the matrix-vector product
         DO i = 1, 5
            V_in( i ) = i * V_out( i ) + V_out( i + 5 )
         END DO
         V_in( 6 : n ) = V_out( : 5 )
      CASE DEFAULT
         EXIT
      END SELECT
   END DO
   DO i = 1, 5                             !  Compute the final residual
      V_in( i ) = i * X( i ) + X( i + 5 ) - i - 1
   END DO
   V_in( 6 : n ) = X( : 5 ) - 1
   WRITE( 6, "( /, ' Output status = ', I6,                            &
          &       ' norm of final residual = ', ES9.1 )" )             &
          action, SQRT( DOT_PRODUCT( V_in, V_in ) )
   WRITE( 6, "( /, ' final x = ', //, ( 5ES12.4 ) )" ) X
   CALL MI32_FINALIZE( keep )  !  Deallocate internal arrays
END PROGRAM HSL_MI32_EXAMPLE
```

This produces the following output:

```
 Output status =      0 norm of final residual =   1.3E-14

 final x =

  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
```

```
  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00  1.0000E+00
```