

1 SUMMARY

This package solves the $n \times n$ unsymmetric linear system $Ax = b$ using the iterative method **restarted MPMGRES**, otherwise known as **MPGRES (m)**. **MPGRES (m)** is itself a generalization of the standard right-preconditioned generalized minimal residual method with restarts (**GMRES (m)**) that allows the user to employ multiple preconditioners at each iteration.

MPGRES comes in two variants: complete and selective. Complete **MPGRES** finds the approximation which minimizes the 2-norm of the residual over some space, termed the multi-Krylov space, which is defined by the choice of preconditioners [1]. This search space grows exponentially with the iteration number, and so this version is generally to be avoided in practice, but useful in diagnostics. A selective version of **MPGRES** finds an approximation to the solution in some subspace of the full multi-Krylov space, the dimension of which will grow only linearly with the iteration number. The use of both variants is supported in this package.

Reverse communication is used for preconditioning operations and matrix-vector/matrix-matrix products with A .

ATTRIBUTES — **Version:** 1.1.0 (15 April 2015) **Types:** Real, Double. **Calls:** FA14, BLAS: `_copy`, `_nrm2`, `_dot`, `_scal`, `_ger`, `_swap`, `_axpy`, `_rot`, `_rotg`, `_trsv`, `_gemv`, `_gemm`, LAPACK: `_orgqr`, `_geqp3`, `ilanev`. **Original date:** March 2013. **Origin:** T. Rees, Rutherford Appleton Laboratory. **Language:** Fortran 2003 subset (F95+TR155581). **Remark:** None.

2 HOW TO USE THE PACKAGE

2.1 Calling sequences

Access to the package requires a `USE` statement such as

Single precision version

```
USE MI29_single
```

Double precision version

```
USE MI29_double
```

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

The following procedures are available to the user:

- (a) `MI29_solve` takes the matrix A and the right-hand side b , and solves the linear system using **MPGRES (m)**. `MI29_solve` uses reverse communication for preconditioning operations and matrix-vector/matrix-matrix products.
- (b) `MI29_TruncStep` can optionally be called by the user before applying the multi-preconditioning step, and returns a matrix with columns processed ready for multi-preconditioning in a way controlled by the user.
- (c) `MI29_finalize` should be called after all other calls to HSL_MI29 are complete for a particular dimension, number of preconditioners, and choice of controls. `MI29_finalize` deallocates components of the derived types.

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 `MI29_keep`, `MI29_control`, and `MI29_info`. The following pseudocode illustrates this.

```
use HSL_MI29_double
...
type (MI29_keep)      :: keep
type (MI29_control)  :: control
type (MI29_info)     :: info
...
```

The components of `MI29_keep` are private and are used to pass data between the subroutines of the package. The components of the other derived types are explained in Sections 2.3.5 and 2.3.6.

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 or the complex 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 solve phase

MPGMRES (m) may be applied to solve the linear system $Ax = b$ by making a series of calls as follows:

```
call MI29_solve(action,n,t,restart,x,b,W,locY,Z,locZ,keep,control,info) .
```

`action` is a scalar `INTENT(INOUT)` argument of type `INTEGER`. Prior to the first call to `MI29_solve`, `action` must be set by the user to 0. On each exit, `action` indicates the action required by the user. Possible values of `action` and the action required are as follows:

- 1 An error has occurred and the user must terminate the computation (see `info%flag`).
- 1 If `control%test_convergence = .true.`, convergence has been achieved and the user should terminate the computation. If `control%test_convergence = .false.`, the user may test for convergence. If the user does not wish to test for convergence at this iteration, or if convergence has not been achieved, the user must recall `MI29_solve` without changing any of the arguments.
- 2 The user must perform the matrix-matrix product

$$Y = AZ \tag{2.1}$$

and recall `MI29_solve`. The matrix Y is held in the first n entries of columns `locY(1)` to `locY(2)` of array W , i.e., $W(1:n, locY(1):locY(2))$. The matrix Z is held in the first n entries of columns `locZ(1)` to `locZ(2)` of array Z , i.e., $Z(1:n, locZ(1):locZ(2))$.

- 3 The user must perform an appropriate multi-preconditioning operation on the matrix Y .
 - If a selective version of **MPGMRES** is desired (*highly recommended*), the user can call the subroutine `MI29_TruncStep` to access some preset default truncations (see section 2.3.3). The particular truncation used is set by changing the `control%trunc_method` parameter.

- If `control%trunc_method > 1` (default), then the user must return the matrix

$$\begin{bmatrix} | & & | \\ P_1 Y_{out}(:,1) & \cdots & P_t Y_{out}(:,1) \\ | & & | \end{bmatrix} \quad (2.2)$$

in columns `locZ(1)` to `locZ(2)` of `Z`. Here `Yout` is the matrix output from the call to `MI29_TruncStep` and P_i denotes the application of preconditioner i .

- If `control%trunc_method < 1`, then the user must return

$$Z = \begin{bmatrix} | & & | \\ P_1 Y_{out}(:,1) & \cdots & P_t Y_{out}(:,t) \\ | & & | \end{bmatrix}, \quad (2.3)$$

where, again, `Yout` is the matrix output from the call to `MI29_TruncStep`.

- Alternatively, in the (default) case where `control%storeZ = .true.`, the advanced user can bypass calling `MI29_TruncStep` and return `Z` where columns `locZ(1)` to `locZ(2)` have been set to the result of any multi-preconditioning routine on columns `locY(1)` to `locY(2)` of `W`.
- If complete **MPGMRES** is required, then the user must return the array `Z`, where columns `locZ(1)` to `locZ(2)` have been updated so that

$$Z(:, locZ(1):locZ(2)) = \begin{bmatrix} | & & | \\ P_1 W(:, locY(1):locY(2)) & \cdots & P_t W(:, locY(1):locY(2)) \\ | & & | \end{bmatrix}, \quad (2.4)$$

where P_1, \dots, P_t are the preconditioners. Note that here the row dimension of `Z` will be t times that of `Y`.

Once the multi-preconditioning routine has been executed, the user recalls `MI29_solve`.

Remark: If the user chooses not to store `Z`, so `control%storeZ = .false.`, then the t preconditioners must not change between iterations. If `control%storeZ = .true.` (default), then the preconditioners may change between iterations. In this case a *Flexible* version of **MPGMRES** is executed.

- The user must perform the final multipreconditioning step at the end of each outer **MPGMRES** iteration. This only is required if the user has set `control%storeZ = .false.`, since in this case another multipreconditioning step is needed at the end of every outer iteration to compute the approximate solution. The user must return the array `Z`, where `locZ(1)` to `locZ(2)` have been updated so that

$$Z(:, locZ(1):locZ(2)) = \begin{bmatrix} | & & | \\ P_1 W(:,1) & \cdots & P_t W(:,t) \\ | & & | \end{bmatrix}. \quad (2.5)$$

Here P_1, \dots, P_t denote the preconditioners, which cannot change from the preconditioners used when `action = 5` was returned.

`n` is a scalar `INTENT(IN)` argument of type `INTEGER` that must hold the dimension of `A`. **Restriction:** $n \geq 1$.

`t` is a scalar `INTENT(IN)` argument of type `INTEGER` that must hold the number of preconditioners to be used. **Restriction:** $1 \leq t \leq n$.

`restart` is a scalar `INTENT(IN)` argument of type `INTEGER` that must be set by the user to the maximum number of iterations performed by **MPGMRES(m)** between restarts; i.e. `restart` holds the m in **MPGMRES(m)**. A compromise between a large value of `restart` that reduces the overall number of iterations and a small value

that limits the storage required should be sought. Unfortunately, it is hard to make firm recommendations about a suitable value as a good value is problem dependent.

If we ignore linearly dependent vectors, the dimension of the space over which selective (resp. complete) **MPGMRES** minimizes the residual is kt (resp. $(t^{k+1} - t)/(t - 1)$) [1], where k is the (inner) iteration number and t is the number of preconditioners. Let k_s denote the smallest number of iterations such that this value is greater than n , the dimension of the matrix to be solved. The value of k_s is output to the user if `control%diagnostics_level > 0`. In the best case, where there are no linearly search directions, the theory tells us the method will converge to the exact solution after k_s iterations. If the restart parameter supplied is larger than k_s , then MI29 changes `restart` to $k_s + \text{control}\%extra_its$. This value is also chosen if `restart = 0` is supplied. This variable must be preserved by the user between calls to `MI29_solve`. **Restriction: `restart` ≥ 0 .**

- `x` is a rank-1 array `INTENT(INOUT)` of package type that contains the approximate solution computed by the algorithm. If `control%init_guess = .true.`, then on the first call `x` should contain the initial guess. On exit with `action = 1`, `x` contains the solution vector. `x` must be preserved by the user between calls to `MI29_solve`.
- `b` is a rank-1 array `INTENT(IN)` of package type that contains the right hand side vector. `b` must be preserved by the user between calls to `MI29_solve`.
- `W` is a rank-2 array `INTENT(INOUT)` allocatable argument of package type with appropriate dimensions dependent on the problem parameters `n`, `restart` and `t` which are calculated on the first call to `MI29_solve`. `W` contains space to store the orthonormal basis vectors. On the first call to `MI29_solve` for any given column `W` must be passed unallocated. This should be ensured by calling `MI29_finalize` for every change of matrix, right hand side, etc. between initial calls to `MI29_solve`.
- `locY` is a rank-1 array `INTENT(INOUT)` argument of type `INTEGER` and size 2. `locY(1)` describes the location of the first column of `Y` in the array `W`, and `locY(2)` describes the location of the last column. This variable must be preserved by the user between calls to `MI29_solve`.
- `Z` is a rank-2 array `INTENT(INOUT)` allocatable argument of package type with appropriate dimensions dependent on the problem parameters `n`, `restart` and `t` which are calculated on the first call to `MI29_solve`. `Z` is used to store the matrix of search directions. On the first call to `MI29_solve` `Z` must be passed unallocated; any data that is in `Z` will be overwritten. This should be ensured by calling `MI29_finalize` for every change of matrix, right hand side, etc. between initial calls to `MI29_solve`.
- `locZ` is a rank-1 array `INTENT(INOUT)` argument of type `INTEGER` and size 2. `locZ(1)` describes the location of the first column of `Z` in the array `Z`, and `locZ(2)` describes the location of the last column. This variable must be preserved by the user between calls to `MI29_solve`.
- `keep` is a scalar `INTENT(INOUT)` argument of type `MI29_keep`. It is used to hold data about the preconditioner and must be passed unchanged to the other subroutines.
- `control` is a scalar `INTENT(IN)` argument of type `MI29_control` (see section 2.3.5).
- `info` is a scalar `INTENT(OUT)` argument of type `MI29_info` (see section 2.3.6).

2.3.3 Selective multi-preconditioning

If a selective version of **MPGMRES** (`m`) is required, then when `control` is returned to the user with `action = 3` the user can access pre-set possible selection strategies by making a call of the following form:

```
call MI29_TruncStep(W, Yout, locY, n, keep, control, info)
```

`W` is a rank-2 array `INTENT(IN)` argument of package type with extents `n` and `keep%tdW` that contains the array `W` output by `MI29_solve`.

`Yout` is a rank-2 array `INTENT(OUT)` argument of package type with extents `n` and `t` that contains columns of `W` processed ready for the user to apply multi-preconditioning.

`locY` is a rank-1 array `INTENT(IN)` of type `INTEGER` and size 2 that contains the array `locY` output by `MI29_solve`.

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER` that contains the integer `n` input to `MI29_solve`.

`keep` is a scalar `INTENT(INOUT)` argument of type `MI29_keep`. It is used to hold data about the preconditioner and must be passed unchanged to the other subroutines.

`control` is a scalar `INTENT(IN)` argument of type `MI29_control` (see section 2.3.5).

`info` is a scalar `INTENT(OUT)` argument of type `MI29_info` (see section 2.3.6).

2.3.4 The finalization phase

A call of the following form must be made after all other calls are complete for a particular problem (including after an error return that does not allow the computation to continue). This deallocates components of the derived data types.

```
call MI29_finalize(W,Z,keep,control,info)
```

`W` is a rank-2 `INTENT(INOUT)` array of package type that contains the array `W` output by `MI29_solve`.

`Z` is a rank-2 `INTENT(INOUT)` array of package type that contains the array `Z` output by `MI29_solve`.

`keep` is a scalar `INTENT(INOUT)` argument of type `MI29_keep`. It is used to hold data about the preconditioner.

`control` is a scalar `INTENT(IN)` argument of type `MI29_control` (see section 2.3.5).

`info` is a scalar `INTENT(OUT)` argument of type `MI29_info` (see section 2.3.6).

2.3.5 The derived data type for holding control parameters

The derived data type `MI29_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 variable of type default `INTEGER` that is used to control the amount of informational output which is required.

< 0 No informational output will occur.

= 0 Error and warning messages only.

= 1 As 0, plus basic informational messages will be printed.

= 2 As 1, plus the norm of the residual at each inner iteration and the current solution vector at each outer iteration will be printed.

The default is `diagnostics_level = 0`.

`unit_diagnostics` is a scalar variable of type default `INTEGER` that holds the stream number for informational and diagnostic messages. Printing is suppressed if `unit_diagnostics < 0`. The default is `unit_diagnostics = 6`.

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

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

Controls used by MI29_solve

`abs_tol` is a scalar of type `REAL` that sets the absolute convergence tolerance. The default value is 0. See the description of `control%test_convergence` for more details.

`extra_its` is a scalar of type `INTEGER` which sets how many inner iterations more the theoretical amount needed to complete if `restart` is zero or too large (see Section 2.3.2). The default value is 1.

`init_guess` is a scalar of type `LOGICAL` that controls whether the user wishes to supply an initial guess (`.true.`) or use the default starting vector of $(0, \dots, 0)^T$ (`.false.`). The default value is `.false.`.

`max_its` is a scalar of type `INTEGER` that determines the maximum number of (outer) iterations allowed. It has default value -1 and in this case the maximum number of iterations allowed is $2n$. If the user does not want the maximum to be $2n$, `max_its` can be set to the desired value. Values of `max_its` ≤ 0 are treated as if they were the default.

`rel_tol` is a scalar of type `REAL` that sets the relative convergence tolerance. The default value is $\sqrt{\epsilon}$, where ϵ is the relative machine precision as returned by `epsilon()`. See `test_convergence` for more details.

`selective` is a scalar of type `LOGICAL` that controls whether selective or complete **MPGMRES** is used. The default is selective, where the value is set to `.true.`. Complete **MPGMRES** is obtained by setting this value to `.false.`; this option is only recommended for diagnostic purposes, as the storage requirements are significantly higher.

`store_Z` is a scalar of type `LOGICAL` that controls whether or not to store the array `Z`:

`.true.` Stores the array, and this option requires roughly double the storage. Note that if we store `Z`, then the method becomes equivalent to a flexible method and the preconditioners may change at each iteration.

`.false.` Calculates `Z` from the V_i 's when computing the solution x . This option requires another multipreconditioning step.

The default is `.false.`.

`test_convergence` is a scalar of type `LOGICAL` that controls whether the convergence test offered by MI29 is to be used. It has default value `.true.`, and in this case the computed solution x is accepted if

$$\|b - Ax\|_2 \leq \max(\|b - Ax^{(0)}\|_2 \text{rel_tol}, \text{abs_tol}),$$

where $x^{(0)}$ is the initial estimate of the solution. If the user wishes to apply their own convergence test, then `test_convergence` should be set to `.false.`. In this case `control` is returned to the user with `action=1` at the start of every outer iteration for a convergence test to be applied.

Controls used by MI29_TruncStep

`seed` is a scalar of type `INTEGER` that sets a new generator word for the random number generator, FA14. This is used for the options of `control%trunc_method` that have a random element. It must be an integer between 1 and $2^{31} - 1 (= 2147483647)$. The default value is 2013.

`trunc_method` is a scalar of type `INTEGER` which controls the type of selection strategy used upon calling the subroutine `MI29_TruncStep`. If this value is positive, then the user must apply each of the preconditioners to the first column of the $n \times t$ matrix returned. If this value is negative, the user should apply the first preconditioner to column one, the second to column two, etc (see section 2.3.2). Supported values are:

1 Returns `Yout = Y*e`, where `e` denotes the vector of ones.

- 2 Returns $Y_{out} = Y*u$, where u is a vector with entries uniformly distributed in the range $[0, 1]$. The random vector changes at each iteration.
- 1 Returns $Y_{out}(:, i) = Y(:, i)$.
- 2 Returns $Y_{out}(:, i) = Y(:, t-i+1)$.
- 3 Returns $Y_{out}(:, i) = \begin{cases} Y(:, i) & \text{if } i \text{ even} \\ Y(:, t-i+1) & \text{if } i \text{ odd} \end{cases}$
- 4 Returns $Y_{out}(:, i) = Y(:, \pi(i))$, where $\pi(\cdot)$ denotes a random permutation of $[1, \dots, t]$. The perturbation changes at each iteration.

In the above, $Y = W(:, \text{loc}Y(1) : \text{loc}Y(2))$. The default is 1.

2.3.6 The derived data type for holding information

The derived data type `MI29_info` is used to hold parameters that give information about the progress and needs of the algorithm. The components of `MI29_info` are:

`flag` is a scalar of type default `INTEGER` that gives the exit status of the subroutine. See section 2.4 for details.

`inner_iterations` is a scalar of type default `INTEGER` that, after a call to `MI29_solve`, contains the number of inner iterations performed.

`outer_iterations` is a scalar of type default `INTEGER` that, after a call to `MI29_solve`, contains the number of outer iterations performed.

`stat` is a scalar of type default `INTEGER` that is used to hold the Fortran `stat` parameter.

2.4 Warning and error messages

A successful return from a subroutine in the package is indicated by `info%flag` having the value zero. A negative value is associated with an error message that by default will be output on unit `control%unit_error`. Possible negative values are:

- 1 Returned by `MI29_solve` if $n < 1$.
- 2 Returned by `MI29_solve` if $\text{restart} < 0$.
- 3 Returned by `MI29_solve` if $t < 1$ or $t > n$.
- 4 Allocation error. The `stat` parameter is returned in `info%stat`.
- 5 Dellocation error. The `stat` parameter is returned in `info%stat`.
- 6 Returned by `MI29_solve` if the maximum number of iterations determined by the control parameter `max_its` has been exceeded.
- 7 Returned by `MI29_solve` if all search directions computed are linearly dependent.
- 8 Returned by `MI29_TruncStep` if an unsupported choice of `control%trunc_method` is passed.

A positive value is associated with a warning message that by default will be output on unit `control%unit_warning`. Possible positive values are:

- +1 Returned by `MI29_solve` if the user-supplied convergence tolerance `control%rel_tol` lies outside the interval $(u, 1.0)$, where $u = \text{EPSILON}(\text{rel_tol})$, and the control parameter `test_convergence` is set to `.true.`. A tolerance of $\sqrt{\text{EPSILON}(\text{rel_tol})}$ is used in the convergence test.

2.5 Information printed

If `control%diagnostics_level` is positive, information about the progress of the algorithm will be printed on unit `control%unit_diagnostics`.

3 GENERAL INFORMATION

Workspace: Provided automatically by the module.

Other routines called directly: HSL_MI29 calls the BLAS kernels `_copy`, `_nrm2`, `_dot`, `_scal`, `_ger`, `_swap`, `_axpy`, `_rot`, `_rotg`, `_trsv`, `_gemv`, `_gemm` and the LAPACK kernels `_orgqr`, `_geqp3`, `ilanev` (& deps).

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: $n \geq 1$, $restart \geq 0$, $n \geq t \geq 1$.

4 METHOD

The Multi-Preconditioned Generalized Minimal Residual method is due to Greif, Rees and Szyld [1]. The method forms a block-Arnoldi-style decomposition, orthogonalizing vectors chosen from a multi-Krylov subspace, and then minimizes the residual over the space of search vectors by solving a least squares problem with Givens rotations. The algorithm used by `MI29_solve` takes the following form:

Check the input data for errors. Set $k = 0$

do $j = 0, 1, 2, \dots, \text{max_its}$

Return to the user with `control%action = 2` to obtain $Ax^{(j)}$.

Compute the residual $r^{(j)} = b - Ax^{(j)}$.

if `control%test_convergence = .true.` **then,**

if $\|r^{(j)}\|_2 \leq \max(\|r^{(0)}\|_2 * \text{control\%rel_tol}, \text{control\%abs_tol})$, convergence has been achieved:

Return with `control%action = 1`.

else

Return with `control%action = 1` to allow the user to check for convergence.

end if

Compute $V_1 = r^{(j)} / \|r^{(j)}\|_2$

Set $s_1 = r^{(j)} / \|r^{(j)}\|_2$

do $i = 1, 2, \dots, \text{restart}$

Return to the user with `control%action=4` to obtain $Z_i = \text{multi-precondition}(V_i)$

Return to the user with `control%action=2` to obtain $W = AZ_i$

Orthogonalize columns of W against those of $[V_1 \dots V_i]$ using the modified block Gram-Schmidt process

Check for columns which are linearly dependent

Compute 'skinny' QR factorization of $W = V_{i+1}H_{i+1,i}$.

Form a trapezoidal matrix H that is a basis for the Krylov subspace spanned by V_1, \dots, V_{i+1}

Use H to calculate the residual of the vector $x^{(k)} + y_i$, where y_i lies in the

multi-Krylov subspace and is selected to minimize $\|b - A(x^{(j)} + y)\|_2$.

If this residual is small, exit the j -loop

end do

if `control%store_Z = .false.` **then**


```

TYPE (mi29_keep)      :: keep
TYPE (mi29_control)  :: control
TYPE (mi29_info)     :: info

! selective MPMRES
control%selective = .true.

action = 0      ! let mi29 know it's the first call
t = 2          ! solve with two preconditioners
restart = 7     ! restart parameter for the inner iteration

b(1) = 3.0
b(2:n-1) = 2.0
b(n) = 1.0

! set the relative tolerance
control%rel_tol = 1e-4

! Set up an n x t array for Yout
allocate(Yout(n,t),stat = info%stat)
! At first iteration we pass W, Z to mi29_solve unallocated, and the
! subroutine allocates for us

do
  call mi29_solve(action,n,t,restart,x,b,W,locY,Z,locZ,keep,control,info)
  select case (action)

    case(-1)  !!! Error !!!
      write(outputStr,'(a,i2)') 'Error.  Flag = ', info%flag
      exit

    case (1)  !!! convergence !!!
      ! method has converged or reached max number of iterations
      if (info%flag .ne. -6) then
        write(outputStr,'(a,i2,a,i2)') 'Method converged in ', &
          info%inner_iterations,' inner iterations and ', &
          info%outer_iterations,' outer iterations.'
        write(outputStr,'(a)') 'x:'
        do i = 1,n
          write(outputStr,'(3f10.2)') x(i)
        end do
      else
        write(outputStr,'(a)') 'Method failed to converge'
      end if
      exit

    case (2)  !!! Perform M-M product !!!
      call ApplyMatMatMult(W(1:n,locY(1):locY(2)),Z(1:n,locZ(1):locZ(2)),n)
  end do

```

```

case (3) !!! Multi-precondition !!!
  ! send to mi29_TruncStep to process the vectors for
  ! multi-preconditioning
  call mi29_TruncStep(W,Yout,locY,n,keep,control,info)
  ! Preconditioner 1 -- Diag(A)
  call ApplyP1(Z(1:n,locZ(1)),Yout(1:n,1),n)
  ! Preconditioner 2 -- LT(A)
  call ApplyP2(Z(1:n,locZ(1)+1),Yout(1:n,1),n)
  ! send back to mi29_Solve

case(4) !!! Final multiprecondition !!!
  ! since we compute, not store, Z...
  ! Preconditioner 1 -- Diag(A)
  call ApplyP1(Z(1:n,locZ(1)),W(1:n,1),n)
  ! Preconditioner 2 -- LT(A)
  call ApplyP2(Z(1:n,locZ(2)),W(1:n,2),n)
  ! Send back to mi29_Solve
end select
end do

! clean up
deallocate(Yout,stat = info%stat)
! ensure mi29_finalize is called before calling mi29_solve again
! for a different problem
call mi29_finalize(W,Z,keep,control,info)

```

CONTAINS

```

SUBROUTINE ApplyMatMatMult(W,Z,n)
  !! Apply matrix
  !! (/ 1 2 0 ... ;
  !!    1 4 1 ... ;
  !!    0 1 4 1 ...;
  !!
  !!    ...      2 4 /)
  !! to a matrix Z
  !! giving W = A*Z
  REAL(wp), DIMENSION(:, :), INTENT(IN)    :: Z
  REAL(wp), DIMENSION(:, :), INTENT(INOUT) :: W
  INTEGER, INTENT(IN)                       :: n
  INTEGER                                    :: i

  W(1,:) = 1.0*Z(1,:) + 2.0*Z(2,:)
  do i=2,n-1
    W(i,:) = 1.0*Z(i-1,:) + 4.0*Z(i,:) + 1.0*Z(i+1,:)
  end do
  W(n,:) = 2.0*Z(n-1,:) + 4.0*Z(n,:)

END SUBROUTINE ApplyMatMatMult

```

```
SUBROUTINE ApplyP1(Z,Y,n)
  !! Apply diagonal as a prec
  REAL(wp), DIMENSION(:), INTENT(INOUT)  :: Z
  REAL(wp), DIMENSION(:), INTENT(IN)     :: Y
  INTEGER, INTENT(IN)                    :: n

  Z(1) = Y( 1 )
  Z(2:n) = 0.25*( Y( 2:n ) )

END SUBROUTINE ApplyP1

SUBROUTINE ApplyP2(Z,Y,n)
  !! Apply lower triangular as a prec
  REAL(wp), DIMENSION(:), INTENT(IN )    :: Y
  REAL(wp), DIMENSION(:), INTENT(INOUT)  :: Z
  INTEGER, INTENT(IN)                    :: n
  INTEGER                                 :: i

  Z(1) = ( Y( 1 ) )
  pre2: do i = 2,n-1
    Z(i) = 0.25*( Y( i ) - Z(i-1) )
  end do pre2
  Z(i) = 0.25*( Y( i ) - 2.0*Z(i-1) )

END SUBROUTINE ApplyP2

END PROGRAM hsl_mi29ds
```

This produces the following output:

Method converged in 6 inner iterations and 0
outer iterations.

```
x:
  4.64
 -0.82
  0.64
  0.25
  0.36
  0.33
  0.34
  0.31
  0.41
  0.04
```