HSL_MP82

1 SUMMARY

Given an $M \times N$ matrix A with $N \ll M$, `HSL_MP82` computes its **thin QR factorization** or **economic singular value decomposition** (SVD). Such a matrix A with many more rows than columns is termed a **tall-skinny** matrix. The thin QR factorization of A is given by

$$A = QR,$$

where Q is an $M \times N$ orthonormal matrix and R is an $N \times N$ upper triangular matrix. Three different methods for computing this factorization are offered: CholQR2, shifted-CholQR2 and TSQR. The latter optionally avoids storing Q explicitly. The economic SVD of A is given by

$$A = U\Sigma V^T,$$

where U is an $M \times N$ orthonormal matrix, V is an $N \times N$ orthonormal matrix and Σ is a $N \times N$ diagonal matrix with non negative entries. It is computed using the Gram SVD algorithm.

ATTRIBUTES — Version: 1.0.1 (03 May 2023). **Interfaces:** Fortran. **Types:** Real (single, double). **Original date:** November 2021. **Uses:** BLAS subroutines `_gemm`, `_syrk`, `_trsm`, `_trmm`, and the LAPACK subroutines `_potrf`, `_syev`, `_geqrt`, `_gemqrt`, `_tpqrt`, `_tpmqrt`, `_lacpy`. **Origin:** Hussam Al Daas, Rutherford Appleton Laboratory. **Language:** Fortran 95. **Parallelism:** Uses MPI.

2 HOW TO USE THE PACKAGE

2.1 MPI and data distribution

MPI is used by `HSL_MP82` to provide parallelism for distibuted memory environments. Prior to calling any of the package routines, the user must choose the number of processors, P, to run on and must initialize MPI by calling `MPI_INIT` on each processor. The processors must be inside a communicator `comm` that is passed as an argument to the packages. The ranks of the processors in `comm` must be $0,1,\ldots,P-1$, and the root processor is the one of rank 0.

The user is responsible for distributing the matrix A across the $P \geq 1$ processors before calling the package. It must be distributed in a block-row-panel fashion. That is, each processor must own one or more rows of A with no redundancy, i.e., each row of A must be owned by one and only one processor. Let $\Omega_1,\ldots,\Omega_P$ be a nontrivial disjoint partition of $\{1,\ldots,M\}$ i.e., $\{1,\ldots,M\} = \cup_{i=1}^{P}\Omega_i$, $\Omega_i \neq \phi$, and $\Omega_i \cap \Omega_j = \phi$ for $i \neq j$. A must be distributed such that processor i owns the submatrix $A_i = A(\Omega_i, 1:N)$ of size $M_i \times N$, where $M_i = |\Omega_i|$ (and $M_i \geq N$ if TSQR is the chosen method). Thus there exists a permutation matrix $\mathcal{P}$ of size $M \times M$ such that

$$\mathcal{P}A = \begin{pmatrix} A_1 \\ \vdots \\ A_P \end{pmatrix}.$$

The R factor is stored redundantly on all processors. The computed Q factor has the same distribution pattern as A

$$\mathcal{P}Q = \begin{pmatrix} Q_1 \\ \vdots \\ Q_P \end{pmatrix},$$

where $Q_i = Q(\Omega_i, 1:N)$ is the local matrix owned by processor i. If the option not to hold the Q factor explicitly is selected and Q is applied to a $N \times k$ matrix C, for $k \leq N$, the resulting matrix $B = QC$ is also of the form

$$\mathcal{P}B = \begin{pmatrix} B_1 \\ \vdots \\ B_P \end{pmatrix},$$

where $B_i = B(\Omega_i, 1:k)$ is the local matrix owned by processor i.

If an economic SVD is computed, the matrix containing the left singular vectors has the same distribution pattern as A

$$\mathcal{P}U = \begin{pmatrix} U_1 \\ \vdots \\ U_P \end{pmatrix},$$

where $U_i = U(\Omega_i, 1:N)$ is the local matrix owned by processor i. The singular values (the diagonal values of Σ) and the right singular vectors V are stored redundantly on all processors.

2.2 Notation

In the rest of the document **u** stands for the machine precision ($\approx 10^{-8}$ for single precision and $\approx 10^{-16}$ for double precision). The 2-norm condition number of the matrix A is referred to as $\kappa_2(A)$; it is equal to the ratio between the largest and smallest singular values of the matrix A.

2.3 Calling sequences

Access to the package requires a `USE` statement of the form

Single precision version

```
USE HSL_MP82_single
```

Double precision version

```
USE HSL_MP82_double
```

If it is required to use more than one module at the same time, the derived types (Section 2.4) must be renamed in one of the use statements.

The following procedures are available to the user:

(a) `MP82_qr` computes the thin QR factorization.

(b) `MP82_mqr` can only be called if the TSQR method was selected on the call to `MP82_qr`. In this case, it applies the Q factor to one or more vectors of length N.

(a) `MP82_svd` computes the economic SVD factorization.

(c) `MP82_finalise` frees memory that has been allocated by the package.

2.4 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types `MP82_keep`, `MP82_control` and `MP82_info`. The following pseudocode illustrates this.

```
use HSL_MP82_double
...
type (MP82_keep) :: keep
type (MP82_control) :: control
type (MP82_info) :: info

...
```

The components of `MP82_control` and `MP82_info` are explained in Sections 2.5.7, and 2.5.8. The components of `MP82_keep` are private and must be passed unchanged between calls.

2.5 Argument lists and calling sequences

2.5.1 Optional arguments

We use square brackets `[ ]` to indicate `OPTIONAL` arguments. Since we reserve the right to add additional optional arguments in future releases of the code, **we strongly recommend that optional arguments be called by keyword, not by position**.

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

2.5.3 Thin-QR decomposition

Given A as described in Section 2.1, the thin QR decomposition of A can be computed by a call of the form

```
call MP82_qr(comm, job, m, n, a, lda, r, ldr, keep, control, info[, shift])
```

`comm` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the MPI communicator. It must be initialized prior to the first call to a subroutine in the `HSL_MP82` package. It defines the set of processors to be used by the `HSL_MP82` package. Before each subroutine call, the user must have completed all tasks involving `comm` (or involving any other communicator that overlaps `comm`). It must not be altered between calls. Note that the code may be run using a single processor.

`job` is a scalar `INTENT(IN)` argument of type `INTEGER` that specifies the method to be used.

`job` $= -1$ CholQR2

`job` $= 1$ shifted-CholQR2

`job` $= -2$ TSQR with implicit representation of the Q factor

`job` $= 2$ TSQR with explicit representation of the Q factor

`m` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the number of local rows of A ($\texttt{m} = M_i$, the size of Ω_i, see Section 2.1).

Restriction: $\texttt{m} \geq 1$. If $|\texttt{job}| = 2$, $\texttt{m} \geq \texttt{n}$.

`n` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the number N of columns in A. It must be the same on all processors.

Restriction: $\texttt{n} \geq 1$.

a is a rank-2 array INTENT(INOUT) argument of package type with extents at least m, n that on entry holds the local matrix $A(\Omega_i,:)$, see Section 2.1. On exit, a holds the following:

if info%flag $= 0$

if job $\in \{-1,1,2\}$ then a holds the local rows of the Q factor, $Q(\Omega_i,:)$.

if job$= -2$ then a holds the Householder vectors of $A(\Omega_i,:)$ as part of the implicit representation of the Q factor of A.

if info%flag < 0 and info%flag$\neq -21$ then a is unchanged.

if info%flag $= -21$ then a contains the local rows of a matrix Q, $Q(\Omega_i,:)$, that may not be orthonormal but satisfy $\|A-QR\|_2 = O(\mathbf{u})\|A\|_2$, where R is the matrix held in r.

lda is a scalar INTENT(IN) argument of type INTEGER that holds the leading dimension of a.

Restriction: lda $\geq$ m.

r is a rank-2 array INTENT(OUT) argument of package type with extents at least n, n. On exit, r holds the following:

if info%flag $= 0$ then r holds the R factor, see Sections 1 and 2.1.

if info%flag < 0 and info%flag$\neq -21$ then r was not accessed

if info%flag $= -21$ then r contains an upper triangular matrix R that may not be equal to an R factor of the QR decomposition of A but satisfies $\|A-QR\|_2 = O(\mathbf{u})\|A\|_2$, where Q is the matrix held in a.

ldr is a scalar INTENT(IN) argument of type INTEGER that holds the leading dimension of r.

Restriction: ldr $\geq$ n.

keep is a scalar INTENT(INOUT) argument of type MP82_keep. It must be unchanged by the user between calls.

control is a scalar INTENT(IN) argument of type MP82_control. Its components control the execution of the subroutine, as explained in Section 2.5.7.

info is a scalar INTENT(INOUT) argument of type MP82_info. Its components provide information about the execution of the subroutine and memory allocation statistics, as explained in Section 2.5.8. To get statistics on memory allocation, info must be unchanged by the user between calls to HSL_MP82.

shift is an optional scalar INTENT(IN) argument of package type that holds the shift supplied by the user to be applied in the shifted-CholQR algorithm, see Section 4 for details. If not available or it has a negative value, the algorithm computes an appropriate value to be used.

2.5.4 Apply the Q factor from TSQR

After a call to MP82_qr with job $= -2$, a call of the following form computes the product $B = QC$, where the $N \times k$ matrix C is identical on all processors.

```
call MP82_mqr(comm, m, k, n, a, lda, b, ldb, keep, control, info)
```

comm, m, n, a, lda are INTENT(IN) and must be passed as output from MP82_qr.

k is a scalar INTENT(IN) argument of type INTEGER that specifies the number of columns in C. It must be the same on all processors.

Restriction: $1 \leq$ k $\leq$ n.

`b` is a rank-2 `INTENT(INOUT)` argument of package type with extents at least `m, k`. On entry, `b(1:n,1:k)` must hold the matrix C and, if `control%annihilate_lower_b=.false.` (the default) then `b(n+1:m,1:k)` must be set by the user to 0. If `control%tsqr_bcast_c=.false.` (the default) then `b(1:n,1:k)` must be the same on all processors. Otherwise, it must be set by the user on the root processor only. On output, `b` holds the local rows of the matrix QC. Note that if both `control%annihilate_lower_b` and `control%tsqr_bcast_c` are set to `.false.`, then `b(1:m,1:k)` must be the same on all processors.

`ldb` is a scalar `INTENT(IN)` argument of type `INTEGER` that holds the leading dimension of `b`.

Restriction: `ldb` $\geq$ `m`.

`keep, control, info` are as in Section 2.5.3.

2.5.5 SVD decomposition

Given A as described in Section 2.1, the economic SVD decomposition of A can be computed by a call of the form

```
call MP82_svd(comm, m, n, a, lda, r, ldr, s, keep, control, info)
```

`comm, m, n, lda, ldr, keep, control, info` are as in Section 2.5.3.

`a` is a rank-2 array `INTENT(INOUT)` argument of package type with extents at least `m, n` that on entry holds the local matrix $A(\Omega_i,:)$, see Section 2.1. On exit, `a` holds the following:

if `info%flag` $=0$, then `a` holds the local rows of the left singular vectors, $U(\Omega_i,:)$.

if `info%flag` $=1$, then `a` holds the local rows of a matrix U such that $\|A-U\Sigma V^\top\|_2 = O(\mathbf{u}^{1/2})\|A\|_2$, where V is the matrix stored in `r` and Σ is the diagonal matrix stored in `s`.

otherwise, `a` is unchanged.

`r` is a rank-2 array `INTENT(OUT)` argument of package type with extents at least `n, n`. On exit, `r` holds the following:

if `info%flag` $=0$, then `r` holds the right singular vectors, see Sections 1 and 2.1).

if `info%flag` $=1$, then `r` holds the matrix V such that $\|A-U\Sigma V^\top\|_2 = O(\mathbf{u}^{1/2})\|A\|_2$ (see above).

`s` is an array `INTENT(OUT)` argument of package type with extent at least `n` that holds on output the following:

if `info%flag` $=0$, then `s` holds the singular values of A, see Sections 1 and 2.1).

if `info%flag` $=1$, then `s` holds the diagonal entries of the matrix Σ such that $\|A-U\Sigma V^\top\|_2 = O(u^{1/2})\|A\|_2$ (see above).

2.5.6 Finalization:

After all calls are complete, the user should make a call of the following form to deallocate memory used by the package.

```
call MP82_finalize(keep, info)
```

`keep` is an `INTENT(INOUT)` scalar of type `MP82_keep` that must be passed unchanged from the earlier calls. On exit, allocatable components will be deallocated.

`info` is an `INTENT(IN)` scalar of type `MP82_info`. Only the components `flag` and `stat` are accessed (see Section 2.5.8).

Note that, given a sequence of matrices $A_1,\ldots,A_f$ having the same number of rows and columns that are distributed in the same way across the processors, it is not nevessary to call `MP82_finalize` until after the factorization of the final matrix A_f. This avoids repeated memory allocations and deallocations and can be useful, for example, within a block Arnoldi iteration.

2.5.7 The derived data type for controlling the execution

The derived data type `MP82_control` is used to control the execution. The components are automatically given default values in the definition of the type.

Components that control printing

`unit_error` is a scalar of type `INTEGER` with default value `6` that is used as the output stream for error messages. If it is negative, these messages will be suppressed.

`unit_warning` is a scalar of type `INTEGER` with default value `6` that is used as the output stream for warning messages. If it is negative, these messages will be suppressed.

Components that control the checking of the input data.

`check_input` is scalar of type `LOGICAL` that controls whether the user-supplied data is checked for errors. The default value is `.true.`.

`tsqr_bcast_c` is a scalar of type `LOGICAL` that controls whether the leading `n`× `k` matrix stored in `b` on the root processor is broadcast by the subroutine `MP82_mqr`. The default value is `.false.`, that is, the user needs to make sure that the leading `n`× `k` matrix stored in `b` is the same on all processors.

`annihilate_lower_b` is a scalar of type `LOGICAL` that controls whether the lower `m-n` × `k` matrix stored in `b` is annihilated by the subroutine `MP82_mqr`. The default value is `.false.`, that is, the user needs to make sure that the lower `m-n` × `k` matrix stored in `b` is 0 on all processors. Note that if both `tsqr_bcast_c` and `annihilate_lower_b` are set to `.false.` then `b(1:m,1:k)` must be the same on all processors.

2.5.8 The derived data type for holding information

The components of the derived data type `MP82_info` are used to hold information.

`flag` is a scalar of type `INTEGER` that gives the exit status of the algorithm (details in Section 2.6).

`stat` is a scalar of type `INTEGER` that, in the event of an allocation error, holds the Fortran `stat` parameter (and is set to 0 otherwise).

`proc` is a scalar of type `INTEGER` that, in the case of an error, holds the rank of one of the processors on which the error occurred.

`amem` is a scalar of type `INTEGER` that holds the current amount of memory used by the package.

`pmem` is a scalar of type `INTEGER` that holds the maximum amount of memory used by the package.

`shift` is scalar of package type that holds the shift used in shifted-CholQR.

2.6 Warning and error messages

A successful return is indicated by `info%flag` having the value zero. A negative value is associated with error as follows:

-1 Error in memory allocation.

-2 Error in the number of columns of the local matrix $A(\Omega_i,:)$, $i =$`info%proc`.

-3 Error in the number of rows of the local matrix $A(\Omega_i,:)$, $i =$`info%proc`.

-4 Error in the number of rows of the rank-2 array `a`.

-5 Error in the number of columns of the rank-2 array `a`.

-6 Error in the leading dimension of the rank-2 array `a`.

-7 Error in the number of rows of the rank-2 array `r`.

-8 Error in the number of columns of the rank-2 array `r`.

-9 Error in the leading dimension of the rank-2 array `r`.

-10 Unknown value of the parameter `job`.

-11 `job` is not the same on all processors.

-12 The number of columns of the matrix A is not the same on all processors.

-13 Error in the number of rows in the rank-2 array `b`

-14 Error in the number of columns of the rank-2 array `b`.

-15 Error in the leading dimension of the rank-2 array `b`

-16 The number of columns of the matrix B is not the same on all processors.

-17 For $|$`job`$| = 2$, the number of local rows in $A(\Omega_i,:)$, $i =$`info%proc`, must be larger than the number of columns.

-18 The size of `s` must be larger than `n`.

-19 `job`$= -1$ (CholQR2) failed. The condition number of A is larger than $\mathbf{u}^{-1/2}$.

-20 `job`$= 1$ (shifted-CholQR2) failed. The condition number of A is larger than $\mathbf{u}^{-1}$.

-21 LAPACK routine `_syev` did not converge.

A positive value is associated with a warning as follows:

1 The condition number of A is larger than $\mathbf{u}^{-1/2}$. The SVD decomposition obtained is only $\mathbf{u}^{1/2}$ accurate.

2 The shift supplied by the user was not used. Instead the method computed an appropriate shift, see `info%shift` in section 2.5.8.

3 Shifted CholQR2 was requested as a method `job` $= 1$, however, CholQR2 was sufficient to provide an accurate QR decomposition, thus no shift was employed.

3 GENERAL INFORMATION

Other routines called directly: BLAS subroutines `_gemm`, `_syrk`, `_trsm`, `_trsm`, and the LAPACK subroutines `_potrf`, `_syev`, `_geqrt`, `_gemqrt`, `_tpqrt`, `_tpmqrt`,`_lacpy`.

Restrictions: `m` ≥ 1; `n` ≥ 1; `k` ≥ 1. If $|$`job`$| = 2$, `m` $\geq$ `n`.

4 METHOD

When one of the subroutines `MP82_qr` `MP82_mqr` `MP82_svd` is called and the control option `control%check_input` is set to `.true.`, the package performs the checks on the user-supplied data. If the check fails, an error is returned by the package in `info%flag`. `info%proc` holds the rank of one of the processors on which the error occured. If the check passes and memory was not allocated in a previous call to the package, memory is allocated.

4.1 QR Algorithms

Algorithm 1 computes the QR factorization of a matrix $A \in \mathbb{R}^{M\times N}$, with $M \geq N$. The QR decomposition of A is $A = QR$ with $Q \in \mathbb{R}^{M\times N}$ is an orthonormal matrix, i.e., $Q^\top Q = I_N$, and $R \in \mathbb{R}^{N\times N}$ is an upper triangular matrix. The algorithm is based on the fact that $C = A^\top A = R^\top R$. If A has full column rank, applying a Cholesky decomposition of C results in computing the R factor. To retrieve the Q factor, backward substitution can be applied. If $\kappa_2(A)$, the spectral condition number of A, is smaller but close to $\mathbf{u}^{-1/2}$, then C can become badly conditioned. Indeed, $\kappa_2(C) = \kappa_2(A)^2$. Hence, repeating the procedure, see Algorithm 1, results in an accurate QR factorization. However, the algorithm will fail if the Cholesky decomposition in the first step fails. This occurs if the condition number of A is of the order of $\mathbf{u}^{-1/2}$ or higher.

If $\kappa_2(A)$ is of the order of $\mathbf{u}^{-1}$, a preliminary step can be performed to obtain a numerically full rank auxiliary triangular matrix $\widetilde{R}$ such that $A\widetilde{R}^{-1}$ has condition number of the order of $\mathbf{u}^{-1/2}$. Then, Algorithm 1 can be applied to compute the QR factorization of $A\widetilde{R}^{-1}$. The QR factorization of A can be obtained easily. This is summarized in Algorithm 2.

The unconditionally stable QR factorization TSQR is based on a sequence of orthogonal transformations. The basic idea of TSQR can be illustrated on the matrix A that has the form

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}.$$

By performing a QR decomposition of $A_1 = Q_{11}R_{11}$ and $A_2 = Q_{21}R_{21}$, where Q_{i1} is an orthogonal matrix and R_{i1} is an upper triangular matrix, we have

$$\begin{aligned} A &= \begin{pmatrix} Q_{11}R_{11} \\ Q_{21}R_{21} \end{pmatrix}, \\ &= \begin{pmatrix} Q_{11} & \\ & Q_{21} \end{pmatrix} \begin{pmatrix} R_{11} \\ R_{21} \end{pmatrix}. \end{aligned}$$

Now we compute the QR decomposition of $\begin{pmatrix} R_{11} \\ R_{21} \end{pmatrix} = Q_{12}R_{12}$ and we have

$$A = \begin{pmatrix} Q_{11} & \\ & Q_{21} \end{pmatrix} Q_{21}R_{21}.$$

Since R_{21} is upper triangular and $\begin{pmatrix} Q_{11} & \\ & Q_{21} \end{pmatrix} Q_{21}$ is an orthogonal matrix – a product of orthogonal matrices – we have the QR decomposition of A. Note that the Q factor can be assembled by successively applying the orthogonal matrices to the identity matrix I_N. Further details are given in [2,3].

4.1.1 Which method to use

The accuracy of the thin QR factorization depends on the method and the conditioning of the matrix A. The TSQR method is unconditionaly stable (and thus the safest option), shifted-CholQR2 is less stable, and CholQR2 is the least stable but most efficient approach. If A is known to be only slightly ill-conditioned ($\kappa_2(A) < \mathbf{u}^{-1}$), shifted-CholQR2 (`job` $= 1$) is generally a good choice, and if $\kappa_2(A) < \mathbf{u}^{-1/2}$, CholQR2 (`job` $= -1$) is appropriate. If `job` $= 1$ is selected, the package checks whether CholQR2 is sufficient to form an accurate factorization and, if so, proceeds without a shift; a warning flag is set in that case.

Algorithm 1 CholQR2

```
Input: $M \times N$ matrix $A$.
Output: $A = QR$, where $Q^T Q = I$, and $R$ is upper triangular.
1: $C = A^T A$
2: $R = \text{chol}(C)$
3: $Q = AR^{-1}$
4: $\widetilde{R} = \text{chol}(Q^T Q)$, $Q = Q\widetilde{R}^{-1}$; $R = \widetilde{R}R$
```

Algorithm 2 shifted-CholQR2

```
Input: $M \times N$ matrix $A$.
Output: $A = QR$, where $Q^T Q = I$, and $R$ is upper triangular.
1: $C = A^T A$
2: $s = 11(MN + N(N+1))\mathbf{u}\|A\|_2^2$
3: $R = \text{chol}(C + sI)$
4: $Q = AR^{-1}$
5: $\widetilde{R} = \text{chol}(Q^T Q)$, $Q = Q\widetilde{R}^{-1}$; $R = \widetilde{R}R$
6: $\widetilde{R} = \text{chol}(Q^T Q)$, $Q = Q\widetilde{R}^{-1}$; $R = \widetilde{R}R$
```

4.2 Gram-SVD Algorithm

Algorithm 3 Gram-SVD

```
Input: $M \times N$ matrix $A$.
Output: $A = U\Sigma V^T$, where $U^T U = V^T V = I$, and $\Sigma$ is diagonal with non negative entries.
1: $C = A^T A$
2: $C = V^T \Sigma^2 V$
3: $U = AV\Sigma^{-1}$
```

The Gram-SVD algorithm is based on the fact that if $A = U\Sigma V^\top$, then $C = V\Sigma^2 V^\top$ is an eigen decomposition of $C = A^\top A$. Once V and Σ are available, we have $U = AV\Sigma^{-1}$. Again, numerical issues would appear with computations based on C if A is not well conditioned. However, failure can only occur if the eigen decomposition does not succeed. As a result, the Gram-SVD algorithm computes an accurate ecnonomic SVD factorization if $\kappa_2(A) < \mathbf{u}^{-1/2}$. Otherwise, the computed factorization satisfies $\|A - U\Sigma V^\top\|_2 = O(\mathbf{u})\|A\|_2$.

References:

[1] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, and Y. Yanagisawa (2020). Shifted Cholesky QR for Computing the QR Factorization of Ill-Conditioned Matrices. SIAM J. Sci. Comput., 42(1), A477–A503.

[2] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations: theory and practice. arXiv:0806.2159; 2008.

[3] H. Al Daas, G. Ballard, and P. Benner. Parallel Algorithms for Tensor Train Arithmetic. arxiv:2011.06532; 2021.

5 EXAMPLE OF USE

Suppose we want to compute the thin QR factorization $A = QR$ using two processors, where

$$A = \begin{pmatrix} 0.5377 & -0.4336 \\ 1.8339 & 0.3426 \\ -2.2588 & 3.5784 \\ 1.4090 & 0.4889 \\ 1.4172 & 1.0347 \\ 0.6715 & 0.7269 \end{pmatrix}.$$

the first 3 rows of A, $A(1:3,:)$, are stored on the processor with rank 0 and the remaining rows, $A(4:6,:)$, are stored on the processor with rank 1.

We may use the following code:

```
program MP82_spec_single
  use mpi
  use HSL_MP82_single
  implicit none
  integer, parameter  :: wp = kind(0.0)
  type(MP82_control) :: control
  type(MP82_info) :: info
  type(MP82_keep) :: keep
  real(wp), allocatable :: a(:,:)     ! local matrix
  real(wp), allocatable :: r(:,:)     ! r factor (QR) / V factor (SVD)
  integer :: m, n                     ! size of local matrix
  integer :: job                      ! qr method
  integer :: mpiierr                  ! info variables
  integer :: comm,comm_rank,comm_size ! communicator,rank,and size
  integer :: i,j,k                    ! indeces
  integer :: stat                     ! stat for allocation
  call MPI_INIT(mpiierr)
  comm = MPI_COMM_WORLD
  call MPI_COMM_RANK(comm, comm_rank, mpiierr)
  call MPI_COMM_SIZE(comm, comm_size, mpiierr)
  ! local problem size
  m = 3
  n = 2
  ! allocate memory for arrays
  allocate(a(m,n), stat=stat)
  allocate(r(n,n), stat=stat)
  if(comm_rank.eq.0) then
    a(1,1) =  0.5377
    a(2,1) =  1.8339
    a(3,1) = -2.2588
    a(1,2) = -0.4336
    a(2,2) =  0.3426
    a(3,2) =  3.5784
  else
    a(1,1) =  1.4090
    a(2,1) =  1.4172
    a(3,1) =  0.6715
    a(1,2) =  0.4889
    a(2,2) =  1.0347
    a(3,2) =  0.7269
  end if
  job = -1
  call MP82_QR(comm,job,m,n,a,m,r,n,keep,control,info)
  if(comm_rank.eq.0) then
    write(*,"(a)") "-------------------------"
    write(*,"(a)") "The tirangular factor R: "
    write(*,"(a)") "-------------------------"
    do j=1,n
      do k=1,n
        write(*,"(es16.5)",advance="no") r(j,k)
      end do
      write(*,"(a)") " "
    end do
    write(*,"(a)") "-------------------------"
    write(*,"(a)") "The orthonormal factor Q:"
    write(*,"(a)") "-------------------------"
  end if
  do i =0,comm_size-1
    if(comm_rank.eq.i) then
      write(*,"(a)") "-------------------------"
      write(*,"(a,i1)") "On processor ", i
      write(*,"(a)") "-------------------------"
      do j=1,m
        do k=1,n
          write(*,"(es16.5)",advance="no") a(j,k)
        end do
        write(*,"(a)") " "
      end do
      write(*,"(a)") "-------------------------"
    end if
    call mpi_barrier(comm,mpiierr)
  end do
  call MP82_finalise(keep, info)
  deallocate(a,stat=stat)
  deallocate(r,stat=stat)
  call mpi_finalize(mpiierr)
end program MP82_spec_single
```

this produces the output:

```
-------------------------
The triangular factor R:
-------------------------
     3.63306E+00    -1.38847E+00
     0.00000E+00     3.60839E+00
```

```
-------------------------
The orthonormal factor Q:
-------------------------
-------------------------
On processor 0
-------------------------
     1.48002E-01    -6.32149E-02
     5.04781E-01     2.89179E-01
    -6.21735E-01     7.52452E-01
-------------------------
-------------------------
On processor 1
-------------------------
     3.87827E-01     2.84721E-01
     3.90084E-01     4.36848E-01
     1.84830E-01     2.72568E-01
-------------------------
```