1 SUMMARY

GALAHAD_SCU is a suite of Fortran procedures for computing the the **solution to an extended system of $n+m$ sparse real linear equations in $n+m$ unknowns,**

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}. \tag{1.1}$$

in the case where the n by n matrix $\mathbf{A}$ is nonsingular and solutions to the systems

$$\mathbf{A}\mathbf{x} = \mathbf{b} \text{ and } \mathbf{A}^T\mathbf{y} = \mathbf{c}$$

may be obtained from an external source, such as an existing factorization. The subroutine uses reverse communication to obtain the solution to such smaller systems. The method makes use of the Schur complement matrix

$$\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}.$$

The Schur complement is stored and factorized as a dense matrix and the subroutine is thus appropriate only if there is sufficient storage for this matrix. Special advantage is taken of symmetry and definiteness in the coefficient matrices. Provision is made for introducing additional rows and columns to, and removing existing rows and columns from, the extended matrix.

ATTRIBUTES — Versions: GALAHAD_SCU_single, GALAHAD_SCU_double, **Uses:** _ROT, _ROTG. **Date:** October 2001. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

Access to the package requires a USE statement such as

Single precision version

```
USE GALAHAD_SCU_single
```

Double precision version

```
USE GALAHAD_SCU_double
```

If it is required to use both modules at the same time, the derived types SCU_matrix_type, SCU_info_type, and SCU_data_type (Section 2.1), and the subroutines SCU_factorize, SCU_solve, SCU_append, SCU_delete, and SCU_terminate (Section 2.2) must be renamed on one of the USE statements.

2.1 The derived data types

Three derived data types are accessible from the package.

2.1.1 The derived data type for holding matrix information

The derived data type SCU_matrix_type is used to hold data about the extended matrix and its factors. The components of SCU_matrix_type are:

n is a scalar of type default INTEGER, that holds the value of n, the dimension of the matrix $\mathbf{A}$.

m is a scalar of type default INTEGER, that holds the value of m, dimension of the matrix $\mathbf{D}$.

m_max is an scalar of type default INTEGER, that holds the dimension of the largest possible matrix $\mathbf{D}$ to be permitted by the package.

class is a scalar of type default INTEGER, that indicates the type of matrix to be processed. Permitted values are:

1 the extended matrix is unsymmetric,

2 the extended matrix is symmetric,

3 the extended matrix is symmetric and the Schur complement matrix $\mathbf{S}$ is known to be positive definite.

4 the extended matrix is symmetric and the Schur complement matrix $\mathbf{S}$ is known to be negative definite.

BD_val is a rank-one allocatable of type default REAL (double precision in GALAHAD_SCU_double), that holds the values of the entries in the partitioned matrix

$$\begin{pmatrix} \mathbf{B} \\ \mathbf{D}_U \end{pmatrix}, \tag{2.1}$$

where $\mathbf{D}_U$ is the upper triangular part of $\mathbf{D}$, ie, $(\mathbf{D}_U)_{ij} = (\mathbf{D})_{ij}$ if $i \leq j$ and zero otherwise. The entries must be ordered by columns, with the entries in each column contiguous and those of column j preceding those of column $j+1$ $(j = 1, ..., m)$. The ordering within each column is unimportant.

BD_row is a rank-one allocatable of type default INTEGER, that holds the row indices of the corresponding entries in BD_val.

BD_col_start is a rank-one allocatable of type default INTEGER, that must be set so that BD_col_start(j) holds the positions in the arrays BD_val and BD_row of the first entry in column j $(j = 1, ..., m)$. BD_col_start(m+1) must be set to the number of entries in the matrix (2) plus one.

CD_val is a rank-one allocatable of type default REAL (double precision in GALAHAD_SCU_double), that need not be set in the symmetric case. In the unsymmetric case, it holds the values of the entries in the partitioned matrix

$$(\mathbf{C} \ \ \mathbf{D}_L), \tag{2.2}$$

where $\mathbf{D}_L$ is the strict lower triangular part of $\mathbf{D}$, ie, $(\mathbf{D}_L)_{ij} = (\mathbf{D})_{ij}$ if $i > j$ and zero otherwise. The entries must be ordered by rows, with the entries in each row contiguous and those of row i preceding those of row $i+1$ $(i = 1, ..., m)$. The ordering within each row is unimportant.

CD_col is a rank-one allocatable of type default INTEGER, that need not be set in the symmetric case. In the unsymmetric case, it holds the column indices of the corresponding entries in CD_val.

CD_row_start is a rank-one allocatable of type default INTEGER, that need not be set in the symmetric case. In the unsymmetric case, it must be set so that CD_row_start(i) points to the positions in the arrays CD_val and CD_col of the first entry in row i $(i = 1, ..., m)$. CD_row_start(m+1) must be set to the number of entries in the matrix (2.2) plus one.

2.1.2 The derived data type for holding problem data

The derived data type SCU_data_type is used to hold all the data for a particular problem between calls of SCU procedures. This data should be preserved, untouched, from the initial call to SCU_factorize to the final call to SCU_terminate. All components of SCU_data_type are private.

2.1.3 The derived data type for holding informational parameters

The derived data type SCU_info_type is used to hold parameters that give information about the factorization. The components of SCU_info_type are:

alloc_status is a scalar of type default INTEGER, that contains the return status from the last attempted internal workspace array allocation or deallocation. A non-zero value indicates that the allocation or deallocation was unsuccessful, and corresponds to the STAT= value on the user's system. Consult local compiler documentation for further details.

inertia is a rank-one array of length 3 and type default INTEGER, that holds the inertia of **S** when the extended matrix is symmetric. Specifically, inertia(i), i=1,2,3, give the number of positive, negative and zero eigenvalues of **S** respectively.

2.2 Argument lists and calling sequences

To solve the extended system, the user must first call SCU_factorize to form and factorize the Schur complement, and then call SCU_solve to compute the solution to the extended system. The solution of additional extended systems, with the same coefficient matrix but different right-hand sides, may be found by further calls to SCU_solve.

The solution of further-extended systems of the form

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{c}_1 \\ \mathbf{C} & \mathbf{D} & \mathbf{c}_2 \\ \mathbf{r}_1^T & \mathbf{r}_2^T & d \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ x_{n+m+1} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ b_{n+m+1} \end{pmatrix}.$$

may be found by firstly calling SCU_append to update the existing factorization of **S** (obtained from SCU_factorize) to give that of the Schur complement of **A** in the further-extended coefficient matrix and then by calling SCU_solve. Likewise, the solution of extended systems of the form

$$\begin{pmatrix} \mathbf{A} & \bar{\mathbf{B}} \\ \bar{\mathbf{C}} & \bar{\mathbf{D}} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}.$$

in which a row and column are removed from the coefficient matrix of (1.1), may be found by firstly calling SCU_delete to update the existing factorization of **S** (obtained from SCU_factorize) to give that of the Schur complement of **A** in the new extended coefficient matrix and then once again by calling SCU_solve.

Finally, the user may call SCU_terminate to deallocate any workspace used to hold the factors of **S**.

We use square brackets [ ] to indicate OPTIONALarguments.

2.2.1 The factorization stage

The Schur complement matrix may be factorized as follows:

```
CALL SCU_factorize( matrix, data, VECTOR, status, info )
```

matrix is a scalar INTENT(INOUT) argument of type SCU_matrix_type (see Section 2.1.1). The following components are used by SCU_factorize :

n must be set to n, the dimension of the matrix **A**. It is unchanged by the subroutine. **Restriction:** n ≥ 0.

m_max must be set to the dimension of the largest matrix **D** that will be allowed by the package. Internal workspace of dimension m_max * (m_max+3)/2 (or 3m_max * (m_max+1)/2 if class = 1) will be used by the package. Thus care should be taken not to set m_max larger than is absolutely necessary to account for the sequence of extended matrices likely to be encountered. **Restriction:** m_max ≥ 0.

m must be set to *m*, dimension of the matrix **D**. It is unchanged by the subroutine. **Restriction:** $0 \leq$ m $\leq$ m_max.

class must be set to indicate the type of matrix that will be processed (see Section 2.1.1). If **S** is known to be (positive or negative) definite throughout the sequence of extended matrices to be considered, it is more efficient to set class $= 3$ or 4. Likewise, if the matrix is known to be symmetric throughout the sequence of extended matrices to be considered, it is more efficient to set class > 1. **Restriction:** $1 \leq$ class ≤ 4.

BD_val and BD_row must be set as described in Section 2.1.1 to hold the values and row indices, respectively, of the matrix (2.1). Before use, the arrays must be ALLOCATEd to be of sufficient length to hold any matrix of the form (2.1) to be encountered in the sequence of extended matrices to be considered. Any elements in BD_val and BD_row that lie below the diagonal of **D** will be removed by SCU_factorize.

BD_col_start must be set as described in Section 2.1.1 to hold the positions of the start of the columns of (2.1), as well as to the first position past the end of the last column. Before use, this array must be ALLOCATEd to be of length at least m_max+1. The values of this array may be altered if entries that lie below the diagonal of **D** are removed by SCU_factorize.

CD_val and CD_col. When class=1, these must be set as described in Section 2.1.1 to hold the values and column indices, respectively, of the matrix (2.2). Before use, the arrays must be ALLOCATEd to be of sufficient length to hold any matrix of the form (2.2) to be encountered in the sequence of extended matrices to be considered. Any elements in CD_val and CD_col that lie on or to the right of the diagonal of **D** will be removed by SCU_factorize. This component need not be ALLOCATEd or set if class > 1.

CD_row_start . When class=1, this must be set as described in Section 2.1.1 to hold the positions of the start of the rows in (2.2), as well as to the first position past the end of the last row. Before use, this array must be ALLOCATEd to be of length at least m_max+1. The values of this array may be altered if entries that lie on or to the right of the diagonal of **D** are removed by SCU_factorize. This component need not be ALLOCATEd or set if class > 1.

data is a scalar INTENT(INOUT) argument of type SCU_data_type. It must not have been altered since the last call to SCU_factorize, SCU_append or SCU_delete.

VECTOR is an INTENT(INOUT) rank-one array argument of length matrix%n and type REAL (double precision in GALAHAD_SCU_double), that needs not be set by the user on initial (status=1) entry. If status is greater than 1 on exit, a re-entry must be made with VECTOR set appropriately (see Section 2.3).

status is an scalar INTENT(INOUT) argument of type default INTEGER, that must be set by the user on initial input to 1. On output, the value of status is used to request additional information, to signal an error in the input data or to indicate a successful call to the subroutine. A successful call is indicated by the exit value status=0. For other values, see Sections 2.3 and 2.4.

info is a scalar INTENT(INOUT) argument of type SCU_info_type (see Section 2.1.3).

2.2.2 The solution stage

Solve the extended system of equations using the factorization produced by a previous call to SCU_factorize, SCU_append or SCU_delete, as follows:

```
CALL SCU_solve( matrix, data, RHS, X, VECTOR, status )
```

matrix is a scalar INTENT(IN) argument of type SCU_matrix_type (see Section 2.1.1), that must not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete.

data is a scalar INTENT(INOUT) argument of type SCU_data_type. It must not have been altered since the last call to SCU_factorize, SCU_append or SCU_delete.

RHS is an INTENT(IN) rank-one array argument of length matrix%n+matrix%m and type REAL (double precision in GALAHAD_SCU_double), that must be set on entry to the values of the right-hand-side vector

$$\begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$$

of the extended system of equations. RHS is not altered by the subroutine.

X is an INTENT(OUT) rank-one array argument of length matrix%n+matrix%m and type REAL (double precision in GALAHAD_SCU_double). On final (status=0) exit, X contains the values of the solution

$$\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}$$

to the extended system of equations.

VECTOR is an INTENT(INOUT) rank-one array argument of length matrix%n and type REAL (double precision in GALAHAD_SCU_double), that needs not be set by the user on initial (status=1) entry. If status is greater than 1 on exit, a re-entry must be made with VECTOR set appropriately (see Section 2.3).

status is an scalar INTENT(INOUT) argument of type default INTEGER, that must be set by the user on initial input to 1. On output, the value of status is used to request additional information, to signal an error in the input data or to indicate a successful call to the subroutine. A successful call is indicated by the exit value status=0. For other values, see Section 2.3.

2.2.3 The updating stage

Call SCU_append to extend the factorization of the Schur complement when a new row and column are appended to the extended matrix. Subsequent systems of equations with the larger coefficient matrix may then be solved by calls to SCU_solve. Note in particular that the arrays matrix_CD_val, matrix_CD_col, matrix_CD_row_start, matrix_BD_val, matrix_BD_row and matrix_BD_col_start must be sufficiently large to allow for the incoming row and column. The factorization may be extended as follows:

```
CALL SCU_append( matrix, data, VECTOR, status, info )
```

matrix is a scalar INTENT(INOUT) argument of type SCU_matrix_type (see Section 2.1.1). The following components are used by SCU_append :

n , m_max, m and class. These should not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete. On a successful exit (status = 0) from SCU_append, n will be unaltered and m will have been increased by 1 to account for the appended row and column. **Restriction:** $0 \leq \mathtt{m} \leq \mathtt{m\_max} - 1$.

BD_val and BD_row. These must be set as described in Section 2.1.1 to hold the values and row indices, respectively, of the further extended matrix

$$\begin{pmatrix} \mathbf{B} & \mathbf{c}_1 \\ \mathbf{D}_U & \mathbf{c}_2 \\ & d \end{pmatrix}. \tag{2.3}$$

Specifically, the first col_start(m+1)-1 elements of BD_val and BD_row should not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete, while elements col_start(m+1), ... should contain the values and row indices of the appended column

$$\begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ d \end{pmatrix},$$

the ordering within this column being unimportant.

BD_col_start must be set as described in Section 2.1.1 to hold the positions of the start of the columns of (2.3), as well as to the first position past the end of the last column. Specifically, the first m+1 elements of BD_col_start should not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete, while BD_col_start(m+2) should be set to the number of entries in (2.3) plus one.

CD_val and CD_col. When SCU_factorize was last called with matrix%class=1, these must be set as described in Section 2.1.1 to hold the values and column indices respectively of the further extended matrix

$$\begin{pmatrix} \mathbf{C} & \mathbf{D}_L \\ \mathbf{r}_1^T & \mathbf{r}_2^T \end{pmatrix}. \tag{2.4}$$

Specifically, the first row_start(m+1)-1 elements of CD_val and CD_col should not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete, while elements row_start(m+1), ... should contain the values and column indices of the appended row

$$(\mathbf{r}_1 \;\; \mathbf{r}_2^T),$$

the ordering within this row being unimportant. This component need not be ALLOCATEd or set if SCU_factorize was last called with matrix%class > 1.

CD_row_start. When SCU_factorize was last called with matrix%class=1, this must be set as described in Section 2.1.1 to hold the positions of the start of the rows of (6), as well as to the first position past the end of the last row. Specifically, the first m+1 elements of CD_row_start should not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete, while CD_row_start(m+2) should be set to the number of entries in (2.4) plus one. This component need not be ALLOCATEd or set if SCU_factorize was last called with matrix%class > 1.

data is a scalar INTENT(INOUT) argument of type SCU_data_type. It must not have been altered since the last call to SCU_factorize, SCU_append or SCU_delete.

VECTOR is an INTENT(INOUT) rank-one array argument of length matrix%n and type REAL (double precision in GALAHAD_SCU_double), that needs not be set by the user on initial (status=1) entry. If status is greater than 1 on exit, a re-entry must be made with VECTOR set appropriately (see Section 2.3).

status is an scalar INTENT(INOUT) argument of type default INTEGER, that must be set by the user on initial input to 1. On output, the value of status is used to request additional information, to signal an error in the input data or to indicate a successful call to the subroutine. A successful call is indicated by the exit value status=0. For other values, see Section 2.3 and 2.4.

info is a scalar INTENT(INOUT) argument of type SCU_info_type (see Section 2.1.3).

2.2.4 The deletion stage

Call SCU_delete to extend the factorization of the Schur complement when a row and column of the existing extended matrix are to be removed. Subsequent systems of equations with the smaller coefficient matrix may then be solved by calls to SCU_solve. The data structures for holding **B**, **C** and **D** will be automatically updated to account for the row and column deletions. Compute the factorization of the extended matrix following a row and column removal as follows:

```
CALL SCU_delete( matrix, data, VECTOR, status, info, col_del [, row_del ] )
```

matrix is a scalar INTENT(INOUT) argument of type SCU_matrix_type (see Section 2.1.1). The following components are used by SCU_delete :

n , m_max, m, class, BD_val, BD_row, BD_col_start, CD_val, CD_col and CD_row_start. These should not have been changed since the last call to SCU_factorize, SCU_append or SCU_delete. On a successful exit (status = 0) from SCU_delete, n will be unchanged, m will have been decreased by 1 to account for the deleted row and column, and the contents of the arrays may have been changed.

data is a scalar INTENT(INOUT) argument of type SCU_data_type. It must not have been altered since the last call to SCU_factorize, SCU_append or SCU_delete.

VECTOR is an INTENT(OUT) rank-one array argument of length matrix%n and type REAL (double precision in GALAHAD_SCU_double), that is used as workspace.

status is an scalar INTENT(OUT) argument of type default INTEGER. On output, the value of status is used to signal an error in the input data or to indicate a successful call to the subroutine. A successful call is indicated by the exit value status=0. For other values, see Section 2.4.

info is a scalar INTENT(INOUT) argument of type SCU_info_type (see Section 2.1.3).

col_del is an scalar INTENT(IN) argument of type default INTEGER. On input, this must be set by the user to the index of the column of (2.1) that is to be removed. **Restriction:** $1 \leq$ col_del $\leq$ m.

row_del is an optional scalar INTENT(IN) argument of type default INTEGER. If SCU_factorize was last called with matrix%class=1, this can be set by the user to the index of the row of (2.2) that is to be removed. If the argument is absent, or if matrix%class > 1, row col_del will be removed. **Restriction:** $1 \leq$ row_del $\leq$ m.

2.2.5 The final stage

Deallocate the space required to hold the factors of the (sequence of) Schur complement(s) as follows:

```
CALL SCU_terminate( data, status, info )
```

data is a scalar INTENT(INOUT) argument of type SCU_data_type. It must not have been altered since the last call to SCU_factorize, SCU_append or SCU_delete.

status is an scalar INTENT(OUT) argument of type default INTEGER, that contains the exit status following a call to SCU_terminate. A successful call is indicated by the exit value status=0. For other values, see Section 2.4.

info is a scalar INTENT(OUT) argument of type SCU_info_type (see Section 2.1.3).

2.3 Reverse communication

If one of the subroutines SCU_factorize, SCU_solve or SCU_append returns with a strictly possible value of status, the subroutine in question requires the user to manipluate the vector VECTOR, and to re-enter with status and all other arguments unchanged. Possible values of status and their consequences are as follows:

2 The user must obtain the solution to the system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. The particular vector $\mathbf{b}$ is returned in the array VECTOR; the user must calculate $\mathbf{x}$ and pass this vector back in VECTOR.

3 The user must obtain the solution to the system of equations $\mathbf{A}^T\mathbf{y} = \mathbf{c}$. The particular vector $\mathbf{c}$ is returned in the array VECTOR; the user must calculate $\mathbf{y}$ and pass this vector back in VECTOR.

2.4 Warning and error messages

A negative value of `status` on exit from any of the preceding subroutines indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1 One or more of the stated restrictions on the components $1 \leq$ `matrix%class` ≤ 4, `matrix%n` ≥ 0, $0 \leq$ `matrix%m` $\leq$ `matrix%m_max`, ($0 \leq$ `matrix%m` $\leq$ `matrix%m_max` $- 1$ in `SCU_append`) $1 \leq$ `col_del` $\leq$ `matrix%m` and $1 \leq$ `row_del` $\leq$ `matrix%m` has been violated.

-2 The subroutine has been called with an initial value `status` ≤ 0.

-3 The factors of **S** have not yet been formed in `data`. This indicates that either `SCU_factorize` has not yet been called, or that the last call to `SCU_factorize`, `SCU_append` or `SCU_delete` ended in a failure.

-4 One or more of the arrays `matrix%BD_val`, `matrix%BD_row` and `matrix%BD_col_start` has not been allocated.

-5 When the extended matrix is unsymmetric, one or more of the arrays `matrix%CD_val`, `matrix%CD_col` and `matrix%CD_row_start` has not been allocated.

-6 One or more of the arrays `matrix%BD_val`, `matrix%BD_row` and `matrix%BD_col_start` is not large enough. Check that the dimension of `matrix%BD_col_start` is no smaller than `matrix%m+1` (`matrix%m+2` for `SCU_append`), and that those of `matrix%BD_val` and `matrix%BD_row` are no smaller than `matrix%BD_col_start(matrix%m+1)-1`, and re-enter. (`matrix%BD_col_start(matrix%m+2)-1` for `SCU_append` and `matrix%BD_col_start(matrix%m+1)+ |matrix%col_del-matrix%row_del|-1` for `SCU_delete`).

-7 When the extended matrix is unsymmetric, one or more of the arrays `matrix%CD_val`, `matrix%CD_col` and `matrix%CD_row_start` is not large enough. Check that the dimension of `matrix%CD_row_start` is no smaller than `matrix%m+1` (`matrix%m+2` for `SCU_append`), and that those of `matrix%CD_val` and `matrix%CD_col` are no smaller than `matrix%CD_row_start(matrix%m+1)-1` (`matrix%CD_row_start(matrix%m+2)-1` for `SCU_append` and `matrix%CD_row_start(matrix%m+1)+ |matrix%col_del-matrix%row_del|-1` for `SCU_delete`).

-8 The value recorded in `matrix_m` does not correspond to the dimension of **D**.

-9 The Schur complement matrix is singular; this has been detected during the **QR** factorization of **S**.

-10 The Schur complement matrix is expected to be positive definite, but this has been found not to be the case during the Cholesky factorization of **S**.

-11 The Schur complement matrix is expected to be negative definite, but this has been found not to be the case during the Cholesky factorization of $-$**S**.

-12 An internal array allocation or deallocation failed. See `info%alloc_status` for further details.

2.5 Multiple updates

Once `SCU_append` or `SCU_delete` has been used to update the factorization of the Schur complement matrix, it is as if the enlarged or reduced matrix were that originally factorized by `SCU_factorize`. Consequently *sequences* of row and column additions and removals may be performed so long as sufficient storage is available.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: the basic linear algebra subprograms (BLAS) SROT and SROTG (DROT and DROTG in GALAHAD_SCU_DOUBLE) are called.

Other modules used directly: None.

Input/output: None.

Restrictions: matrix%n ≥ 0, $0 \leq$ matrix%m $\leq$ matrix%m_max and $1 \leq$ matrix%class ≤ 4. Also $1 \leq$ row_del $\leq$ matrix%m and $1 \leq$ col_del $\leq$ matrix%m for SCU_delete.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

The subroutine SCU_factorize forms the Schur complement $\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$ of $\mathbf{A}$ in the extended matrix by repeated reverse communication to obtain the columns of $\mathbf{A}^{-1}\mathbf{B}$. The Schur complement or its negative is then factorized into its **QR** or, if possible, Cholesky factors.

The subroutine SCU_solve solves the extended system using the following well-known scheme:

(i) Compute the solution to $\mathbf{A}\mathbf{u} = \mathbf{b}_1$;

(ii) Compute $\mathbf{x}_2$ from $\mathbf{S}\mathbf{x}_2 = \mathbf{b}_2 - \mathbf{C}\mathbf{u}$;

(iii) Compute the solution to $\mathbf{A}\mathbf{v} = \mathbf{B}\mathbf{x}_2$; and

(iv) Compute $\mathbf{x}_1 = \mathbf{u} - \mathbf{v}$.

The subroutines SCU_append and SCU_delete compute the factorization of the Schur complement after a row and column have been appended to, and removed from, the extended matrix, respectively. The existing factorization is updated to obtain the new one; this is normally more efficient than forming the factorization from scratch.

5 EXAMPLE OF USE

As a simple example, suppose we wish to solve the system of equations

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 5 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 0 & 1 & 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 7 \\ 8 \\ 10 \end{pmatrix}.$$

Notice that the leading 5 by 5 coefficient matrix is diagonal and hence easily invertible. So we might choose $n = 5$, $m = 2$, and use SCU_factorize/SCU_solve to find the required solution. As the matrix is unsymmetric, we must set matrix%class=1.

Now suppose that we have solved this system as described and are now confronted with the further system

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 \\ 1 & 0 & 1 & 0 & 1 & 3 & 4 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 4 \\ 5 \\ 7 \\ 12 \\ 12 \\ 4 \end{pmatrix}.$$

Rather than applying `SCU_factorize` /`SCU_solve` with $n = 5$, $m = 3$, we note that the new coefficient matrix differs from the old one merely in having an extra row and column. Thus, we can use `SCU_append` with $n = 5$, $m = 2$ to update the existing factorization and then call `SCU_solve` with $n = 5$, $m = 3$ to calculate the desired solution.

Finally, suppose that we have solved this second system and now wish to solve

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 5 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 3 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ 4 \\ 5 \\ 6 \\ 6 \\ 2 \end{pmatrix}.$$

Again, rather than applying `SCU_factorize` /`SCU_solve` with $n = 5$, $m = 2$, we note that this third coefficient matrix is the second one with its $n+1$-st row and $n+2$-nd column removed. Thus, we can use `SCU_delete` with $n = 5$, $m = 3$ to update the existing factorization and then call `SCU_solve` with $n = 5$, $m = 2$ to calculate the desired solution.

To carry out these calculations, we might use the following piece of code. Notice how the extra row and column for the second problem are simply introduced at the end of the existing data structures.

```
PROGRAM GALAHAD_SCU_EXAMPLE
USE GALAHAD_SCU_DOUBLE                     ! double precision version
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
INTEGER :: i, row_del, col_del, status
INTEGER, PARAMETER :: n = 5, m = 2, mmax = m + 1
TYPE ( SCU_matrix_type ) :: mat
TYPE ( SCU_data_type ) :: data
TYPE ( SCU_info_type ) :: info
REAL ( KIND = wp ) :: X1( n + m ), RHS1( n + m )
REAL ( KIND = wp ) :: X2( n + m + 1 ), RHS2( n + m + 1 )
REAL ( KIND = wp ) :: X3( n + m ), RHS3( n + m ), VECTOR( n )
mat%m_max = mmax ; mat%class = 1
mat%n = n ; mat%m = m
ALLOCATE ( mat%BD_val( 15 ), mat%BD_row( 15 ), mat%BD_col_start( mmax+1 ),&
           mat%CD_val( 13 ), mat%CD_col( 13 ), mat%CD_row_start( mmax+1 ) )
mat%BD_col_start( : 3 ) = (/1, 7, 10/)
mat%CD_row_start( : 3 ) = (/1, 6, 10/)
mat%BD_row( : 9 ) = (/1, 2, 3, 4, 5, 6, 5, 6, 7/)
mat%BD_val( : 9 ) = (/1.0_wp, 1.0_wp, 1.0_wp, 1.0_wp, &
                  1.0_wp, 1.0_wp, 1.0_wp, 2.0_wp, 4.0_wp/)
mat%CD_col( : 9 ) = (/1, 2, 3, 4, 5, 1, 3, 5, 6/)
```

```
   mat%CD_val( : 9 ) = (/1.0_wp, 1.0_wp, 1.0_wp, 1.0_wp, &
                         1.0_wp, 1.0_wp, 1.0_wp, 1.0_wp, 3.0_wp/)
   RHS1 = (/2.0_wp, 3.0_wp, 4.0_wp, 5.0_wp, 7.0_wp, 8.0_wp, 10.0_wp/)
   RHS2 = (/5.0_wp, 5.0_wp, 4.0_wp, 5.0_wp, 7.0_wp, 12.0_wp, 12.0_wp, 4.0_wp/)
   RHS3 = (/3.0_wp, 5.0_wp, 4.0_wp, 5.0_wp, 6.0_wp, 6.0_wp, 2.0_wp/)
!  First system
   status = 1
   DO
     CALL SCU_factorize( mat, data, VECTOR, status, info )
     IF ( status <= 0 ) EXIT
     DO i = 1, n
        VECTOR( i ) = VECTOR( i ) / DBLE( FLOAT( i ) )
     END DO
   END DO
   WRITE( 6, "( /, ' On exit from SCU_factorize,  status = ', I3 )" ) status
   IF ( status < 0 ) STOP
   status = 1
   DO
     CALL SCU_solve( mat, data, RHS1, X1, VECTOR, status )
     IF ( status <= 0 ) EXIT
     DO i = 1, n        !  multiply by the inverse of A
        VECTOR( i ) = VECTOR( i ) / DBLE( FLOAT( i ) )
     END DO
   END DO
   WRITE( 6, "( ' On exit from SCU_solve,      status = ', I3 )" ) status
   IF ( status < 0 ) STOP
   WRITE( 6, "( /, ' Solution (first system)', /, ( 8ES9.2 ) )" ) X1( : )
!  Second system
   mat%BD_row( 10 : 12 ) = (/ 1, 6, 8 /)
   mat%BD_val( 10 : 12 ) = (/ 1.0_wp, 1.0_wp, 1.0_wp /)
   mat%BD_col_start( 4 ) = 13
   mat%CD_col( 10 ) = 1
   mat%CD_val( 10 ) = 1.0_wp
   mat%CD_row_start( 4 ) = 11
   status = 1
   DO
     CALL SCU_append( mat, data, VECTOR, status, info )
     IF ( status <= 0 ) EXIT
     DO i = 1, n
        VECTOR( i ) = VECTOR( i ) / DBLE( FLOAT( i ) )
     END DO
   END DO
   WRITE( 6, "( /, ' On exit from SCU_append,     status = ', I3 )" ) status
   IF ( status < 0 ) STOP
   status = 1
   DO
     CALL SCU_solve( mat, data, RHS2, X2, VECTOR, status )
     IF ( status <= 0 ) EXIT
     DO i = 1, n
        VECTOR( i ) = VECTOR( i ) / DBLE( FLOAT( i ) )
     END DO
```

```
   END DO
   WRITE( 6, "( ' On exit from SCU_solve,      status = ', I3 )" ) status
   IF ( status < 0 ) STOP
   WRITE( 6, "( /, ' Solution (second system)', /, ( 8ES9.2 ) )" ) X2
!  Third system
   row_del = 1
   col_del = 2
   status = 1
   CALL SCU_delete( mat, data, VECTOR, status, info, col_del, &
                    row_del = row_del )
   WRITE( 6, "( /, ' On exit from SCU_delete,     status = ', I3 )" ) status
   IF ( status < 0 ) STOP
   status = 1
   DO
     CALL SCU_solve( mat, data, RHS3, X3, VECTOR, status )
     IF ( status <= 0 ) EXIT
       DO i = 1, n
        VECTOR( i ) = VECTOR( i ) / DBLE( FLOAT( i ) )
     END DO
   END DO
   WRITE( 6, "( ' On exit from SCU_solve,      status = ', I3 )" ) status
   IF ( status < 0 ) STOP
   WRITE( 6, "( /, ' Solution (third system)', /, ( 8ES9.2 ) )" ) X3
   CALL SCU_terminate( data, status, info )
   WRITE( 6, "( /, ' On exit from SCU_terminate,  status = ', I3 )" ) status
   END PROGRAM GALAHAD_SCU_EXAMPLE
```

This produces the following output:

```
 On exit from SCU_factorize,  status =   0
 On exit from SCU_solve,      status =   0

 Solution (first system)
 1.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00

 On exit from SCU_append,     status =   0
 On exit from SCU_solve,      status =   0

 Solution (second system)
 3.00E+00 2.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00

 On exit from SCU_delete,     status =   0
 On exit from SCU_solve,      status =   0

 Solution (third system)
 1.00E+00 2.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00 1.00E+00

 On exit from SCU_terminate,  status =   0
```