BASICLU User Guide

Version 2.0

October 15, 2018

Contents

1 Algorithm

BASICLU implements a right-looking *LU* factorization with dynamic Markowitz search and columnwise threshold pivoting. After a column modification to the matrix it applies either a permutation or the Forrest-Tomlin update to maintain a factorized form. It uses the method of Gilbert and Peierls to solve triangular systems with a sparse right-hand side. A more detailed explanation of the method is given in [Technical Report ERGO 17-002, http://www.maths.ed.ac.uk/ERGO/preprints.html].

2 Installation

2.1 Compiling BASICLU

Compiling BASICLU requires GNUmake and a C compiler that (partly) supports the ANSI C99 standard.

To compile the package type `make` in the BASICLU directory. This will create a static and a shared library inside `lib/`. It will also compile a standalone program `maxvolume` in `example/`. You can call the latter with the matrices in `example/data/`.

Compiler and linker flags can be changed in `config.mk` or can be given to `make` on the command line. See the documentation in `config.mk`.

2.2 The integer type

BASICLU integer variables are of type `lu_int`, which is `typedef`'ed in `basiclu.h`. `lu_int` must be a signed integer type. The default is `int64_t`. It can be changed before compiling the package. Note:

- The BASICLU routines do not check for integer overflow. It is in your responsibility to choose a sufficiently large integer type for your problems.
- It is required that all integer values arising in the computation can be stored in `double` variables and converted back to `lu_int` without altering their value.

3 Low level C interface

The low level C interface consists of routines which do not allocate memory. Memory must be provided by the user and reallocated on request. To use the low level C interface, user code must include `basiclu.h`. Defined constants start with `BASICLU_` and function names start with `basiclu_`.

Memory must be provided in form of four `lu_int` arrays and four `double` arrays:

`istore`, `xstore` are arrays whose size depends only on the matrix dimension (see `basiclu_initialize` for their required length). `xstore` is used to input parameters to the routines and to return information to the user. The indices of `xstore` which the user may access have defined names, e. g. `xstore[BASICLU_STATUS]` holds the status code. `istore` need not be accessed by the user.

`Li`, `Lx`, `Ui`, `Ux`, `Wi`, `Wx` are arrays whose required size is not known in advance. Their size must be given by the user as parameters (see below) and BASICLU will request reallocation if the size is insufficient. These arrays need not be accessed by the user.

3.1 basiclu_initialize

```
lu_int basiclu_initialize
(
    lu_int m,
    lu_int istore[],
    double xstore[]
);

Purpose:

    Initialize istore, xstore to a BASICLU instance. Set parameters to defaults
    and reset counters. The initialization fixes the dimension of matrices
    which can be processed by this instance.

    This routine must be called once before passing istore, xstore to any other
    basiclu_ routine.

Return:

    BASICLU_OK

        m, istore, xstore were valid arguments. Only in this case are istore,
        xstore initialized.

    BASICLU_ERROR_argument_missing

        istore or xstore is NULL.

    BASICLU_ERROR_invalid_argument

        m is less than or equal to zero.

Arguments:

    lu_int m

        The dimension of matrices which can be processed. m > 0.

    lu_int istore[]
    double xstore[]

        Fixed size arrays. These must be allocated by the user as follows:

          length of istore: BASICLU_SIZE_ISTORE_1 + BASICLU_SIZE_ISTORE_M * m
          length of xstore: BASICLU_SIZE_XSTORE_1 + BASICLU_SIZE_XSTORE_M * m

Info:

    After initialization, the following entries of xstore are maintained
    throughout by all basiclu_ routines:

    xstore[BASICLU_DIM] Matrix dimension (constant).

    xstore[BASICLU_NUPDATE] Number of updates since last factorization. This is
                            the sum of Forrest-Tomlin updates and permutation
                            updates.

    xstore[BASICLU_NFORREST] Number of Forrest-Tomlin updates since last
                             factorization. The upper limit on Forrest-Tomlin
                             updates before refactorization is m, but that is
                             far too much for performance reasons and numerical
                             stability.

    xstore[BASICLU_NFACTORIZE] Number of factorizations since initialization.
```

```
xstore[BASICLU_NUPDATE_TOTAL] Number of updates since initialization.

xstore[BASICLU_NFORREST_TOTAL] Number of Forrest-Tomlin updates since
                               initialization.

xstore[BASICLU_NSYMPERM_TOTAL] Number of symmetric permutation updates since
                               initialization. A permutation update is
                               "symmetric" if the row and column
                               permutation can be updated symmetrically.

xstore[BASICLU_LNZ] Number of nonzeros in L excluding diagonal elements
                    (not changed by updates).

xstore[BASICLU_UNZ] Number of nonzeros in U excluding diagonal elements
                    (changed by updates).

xstore[BASICLU_RNZ] Number of nonzeros in update ETA vectors excluding
                    diagonal elements (zero after factorization, increased
                    by Forrest-Tomlin updates).

xstore[BASICLU_MIN_PIVOT]
xstore[BASICLU_MAX_PIVOT] After factorization these are the smallest and
                          largest pivot element. xstore[BASICLU_MIN_PIVOT]
                          is replaced when a smaller pivot occurs in an
                          update. xstore[BASICLU_MAX_PIVOT] is replaced when
                          a larger pivot occurs in an update.

xstore[BASICLU_UPDATE_COST] Deterministic measure of solve/update cost
                            compared to cost of last factorization. This
                            value is zero after factorization and
                            monotonically increases with solves/updates.
                            When xstore[BASICLU_UPDATE_COST] > 1.0, then
                            a refactorization is good for performance.

xstore[BASICLU_TIME_FACTORIZE] Wall clock time for last factorization.

xstore[BASICLU_TIME_SOLVE] Wall clock time for all calls to
                           basiclu_solve_sparse and basiclu_solve_for_update
                           since last factorization.

xstore[BASICLU_TIME_UPDATE] Wall clock time for all calls to basiclu_update
                            since last factorization.

xstore[BASICLU_TIME_FACTORIZE_TOTAL]
xstore[BASICLU_TIME_SOLVE_TOTAL]
xstore[BASICLU_TIME_UPDATE_TOTAL] Analogous to above, but summing up all
                                  calls since initialization.

xstore[BASICLU_LFLOPS]
xstore[BASICLU_UFLOPS]
xstore[BASICLU_RFLOPS] Number of flops for operations with L, U and update
                       ETA vectors in calls to basiclu_solve_sparse and
                       basiclu_solve_for_update since last factorization.
```

3.2 basiclu_factorize

```
lu_int basiclu_factorize
(
    lu_int istore[],
    double xstore[],
    lu_int Li[],
    double Lx[],
    lu_int Ui[],
    double Ux[],
    lu_int Wi[],
    double Wx[],
    const lu_int Bbegin[],
    const lu_int Bend[],
    const lu_int Bi[],
    const double Bx[],
    lu_int c0ntinue
);

Purpose:

    Factorize the matrix B into its LU factors. Choose pivot elements by a
    Markowitz criterion subject to columnwise threshold pivoting (the pivot may
    not be smaller than a factor of the largest entry in its column).

Return:

    BASICLU_ERROR_invalid_store if istore, xstore do not hold a BASICLU
    instance. In this case xstore[BASICLU_STATUS] is not set.

    Otherwise return the status code. See xstore[BASICLU_STATUS] below.

Arguments:

    lu_int istore[]
    double xstore[]

        BASICLU instance. The instance determines the dimension of matrix B
        (stored in xstore[BASICLU_DIM]).

    lu_int Li[]
    double Lx[]
    lu_int Ui[]
    double Ux[]
    lu_int Wi[]
    double Wx[]

        Arrays used for workspace during the factorization and to store the
        final factors. They must be allocated by the user and their length
        must be provided as parameters:

            xstore[BASICLU_MEMORYL]: length of Li and Lx
            xstore[BASICLU_MEMORYU]: length of Ui and Ux
            xstore[BASICLU_MEMORYW]: length of Wi and Wx

        When the allocated length is insufficient to complete the factorization,
        basiclu_factorize() returns to the caller for reallocation (see
        xstore[BASICLU_STATUS] below). A successful factorization requires at
        least nnz(B) length for each of the arrays.

    const lu_int Bbegin[]
    const lu_int Bend[]
    const lu_int Bi[]
    const double Bx[]

        Matrix B in packed column form. Bi and Bx are arrays of row indices
```

```
    and nonzero values. Column j of matrix B contains elements

        Bi[Bbegin[j] .. Bend[j]-1], Bx[Bbegin[j] .. Bend[j]-1].

    The columns must not contain duplicate row indices. The arrays Bbegin
    and Bend may overlap, so that it is valid to pass Bp, Bp+1 for a matrix
    stored in compressed column form (Bp, Bi, Bx).

lu_int c0ntinue

    zero to start a new factorization; nonzero to continue a factorization
    after reallocation.

Parameters:

xstore[BASICLU_DROP_TOLERANCE]

    Nonzeros which magnitude is less than or equal to the drop tolerance can
    be removed after each pivot step. They are guaranteed removed at the end
    of the factorization. Default: 1e-20

xstore[BASICLU_ABS_PIVOT_TOLERANCE]

    A pivot element must be nonzero and in absolute value must be greater
    than or equal to xstore[BASICLU_ABS_PIVOT_TOLERANCE]. Default: 1e-14

xstore[BASICLU_REL_PIVOT_TOLERANCE]

    A pivot element must be (in absolute value) greater than or equal to
    xstore[BASICLU_REL_PIVOT_TOLERANCE] times the largest entry in its
    column. A value greater than or equal to 1.0 is treated as 1.0 and
    enforces partial pivoting. Default: 0.1

xstore[BASICLU_BIAS_NONZEROS]

    When this value is greater than or equal to zero, the pivot choice
    attempts to keep L sparse, putting entries into U when possible.
    When this value is less than zero, the pivot choice attempts to keep U
    sparse, putting entries into L when possible. Default: 1

xstore[BASICLU_MAXN_SEARCH_PIVOT]

    The Markowitz search is terminated after searching
    xstore[BASICLU_MAXN_SERACH_PIVOT] rows or columns if a numerically
    stable pivot element has been found. Default: 3

xstore[BASICLU_SEARCH_ROWS]

    If xstore[BASICLU_SEARCH_ROWS] is zero, then the Markowitz search only
    scans columns. If nonzero, then both columns and rows are searched in
    increasing order of number of entries. Default: 1

xstore[BASICLU_PAD]
xstore[BASICLU_STRETCH]

    When a row or column cannot be updated by the pivot operation in place,
    it is appended to the end of the workspace. For a row or column with nz
    elements, xstore[BASICLU_PAD] + nz * xstore[BASICLU_STRETCH] elements
    extra space are added for later fill-in.
    Default: xstore[BASICLU_PAD] = 4, xstore[BASICLU_STRETCH] = 0.3

xstore[BASICLU_REMOVE_COLUMNS]

    If this value is nonzero, then a column is removed from the active
    submatrix if its maximum active entry is zero or less than or equal to
```

```
        xstore[BASICLU_ABS_PIVOT_TOLERANCE]. Default: 0

Info:

    xstore[BASICLU_STATUS]: status code.

        BASICLU_OK

            The factorization has successfully completed.

        BASICLU_WARNING_singular_matrix

            The factorization did xstore[BASICLU_RANK] < xstore[BASICLU_DIM]
            pivot steps. The remaining elements in the active submatrix are zero
            or less than xstore[BASICLU_ABS_PIVOT_TOLERANCE]. The factors have
            been augmented by unit columns to form a square matrix. See
            basiclu_get_factors() on how to get the indices of linearly
            dependent columns.

        BASICLU_ERROR_argument_missing

            One or more of the pointer/array arguments are NULL.

        BASICLU_ERROR_invalid_call

            c0ntinue is nonzero, but the factorization was not started before.

        BASICLU_ERROR_invalid_argument

            The matrix is invalid (a column has a negative number of entries,
            a row index is out of range, or a column has duplicate entries).

        BASICLU_REALLOCATE

            Factorization requires more memory in Li,Lx and/or Ui,Ux and/or
            Wi,Wx. The number of additional elements in each of the array pairs
            required for the next pivot operation is given by:

                xstore[BASICLU_ADD_MEMORYL] >= 0
                xstore[BASICLU_ADD_MEMORYU] >= 0
                xstore[BASICLU_ADD_MEMORYW] >= 0

            The user must reallocate the arrays for which additional memory is
            required. It is recommended to reallocate for the requested number
            of additional elements plus some extra space (e.g. 0.5 times the
            current array length). The new array lengths must be provided in

                xstore[BASICLU_MEMORYL]: length of Li and Lx
                xstore[BASICLU_MEMORYU]: length of Ui and Ux
                xstore[BASICLU_MEMORYW]: length of Wi and Wx

            basiclu_factorize() can be called again with c0ntinue not equal to
            zero to continue the factorization.

    xstore[BASICLU_MATRIX_NZ] number of nonzeros in B

    xstore[BASICLU_MATRIX_ONENORM]
    xstore[BASICLU_MATRIX_INFNORM] 1-norm and inf-norm of the input matrix
                                   after replacing dependent columns by unit
                                   columns.

    xstore[BASICLU_RANK] number of pivot steps performed

    xstore[BASICLU_BUMP_SIZE] dimension of matrix after removing singletons
```

```
xstore[BASICLU_BUMP_NZ] # nonzeros in matrix after removing singletons

xstore[BASICLU_NSEARCH_PIVOT] total # columns/rows searched for pivots

xstore[BASICLU_NEXPAND] # columns/rows which had to be appended to the end
                        of the workspace for the rank-1 update

xstore[BASICLU_NGARBAGE] # garbage collections

xstore[BASICLU_FACTOR_FLOPS] # floating point operations performed,
                             counting multiply-add as one flop

xstore[BASICLU_TIME_SINGLETONS] wall clock time for removing the initial
                                triangular factors

xstore[BASICLU_TIME_SEARCH_PIVOT] wall clock time for Markowitz search

xstore[BASIClU_TIME_ELIM_PIVOT] wall clock time for pivot elimination

xstore[BASICLU_RESIDUAL_TEST]

        An estimate for numerical stability of the factorization.
        xstore[BASICLU_RESIDUAL_TEST] is the maximum of the scaled residuals

          ||b-Bx|| / (||b|| + ||B||*||x||)

        and

          ||c-B'y|| / (||c|| + ||B'||*||y||),

        where x=B\b and y=B'\c are computed from the LU factors, b and c
        have components +/-1 that are chosen to make x respectively y large,
        and ||.|| is the 1-norm. Here B is the input matrix after replacing
        dependent columns by unit columns.

        If xstore[BASICLU_RESIDUAL_TEST] > 1e-12, say, the factorization is
        numerically unstable. (This is independent of the condition number
        of B.) In this case tightening the relative pivot tolerance and
        refactorizing is appropriate.

xstore[BASICLU_NORM_L]
xstore[BASICLU_NORM_U] 1-norm of L and U.

xstore[BASICLU_NORMEST_LINV]
xstore[BASICLU_NORMEST_UINV] Estimated 1-norm of L^{-1} and U^{-1},
                             computed by the LINPACK algorithm.

xstore[BASICLU_CONDEST_L]
xstore[BASICLU_CONDEST_U] Estimated 1-norm condition number of L and U.
```

3.3 basiclu_get_factors

```
lu_int basiclu_get_factors
(
    lu_int istore[],
    double xstore[],
    lu_int Li[],
    double Lx[],
    lu_int Ui[],
    double Ux[],
    lu_int Wi[],
    double Wx[],
    lu_int rowperm[],
    lu_int colperm[],
    lu_int Lcolptr[],
    lu_int Lrowidx[],
    double Lvalue[],
    lu_int Ucolptr[],
    lu_int Urowidx[],
    double Uvalue[]
);

Purpose:

    Extract the row and column permutation and the LU factors. This routine can
    be used only after basiclu_factorize() has completed and before a call to
    basiclu_update(). At that point the factorized form of matrix B is

        B[rowperm,colperm] = L*U,

    where L is unit lower triangular and U is upper triangular. If the
    factorization was singular (rank < m), then columns colperm[rank..m-1]
    of B have been replaced by unit columns with entry 1 in position
    rowperm[rank..m-1].

    basiclu_get_factors() is intended when the user needs direct access to the
    matrix factors. It is not required to solve linear systems with the factors
    (see basiclu_solve_dense() and basiclu_solve_sparse() instead).

Return:

    BASICLU_ERROR_invalid_store if istore, xstore do not hold a BASICLU
    instance. In this case xstore[BASICLU_STATUS] is not set.

    Otherwise return the status code. See xstore[BASICLU_STATUS] below.

Arguments:

    lu_int istore[]
    double xstore[]
    lu_int Li[]
    double Lx[]
    lu_int Ui[]
    double Ux[]
    lu_int Wi[]
    double Wx[]

        The BASICLU instance after basiclu_factorize() has completed.

    lu_int rowperm[m]

        Returns the row permutation. If the row permutation is not required,
        then NULL can be passed (this is not an error).

    lu_int colperm[m]
```

```
        Returns the column permutation. If the column permutation is not
        required, then NULL can be passed (this is not an error).

    lu_int Lcolptr[m+1]
    lu_int Lrowidx[m+Lnz]
    double Lvalue[m+Lnz], where Lnz = xstore[BASICLU_LNZ]

        If all three arguments are not NULL, then they are filled with L in
        compressed column form. The indices in each column are sorted with the
        unit diagonal element at the front.

        If any of the three arguments is NULL, then L is not returned
        (this is not an error).

    lu_int Ucolptr[m+1]
    lu_int Urowidx[m+Unz]
    double Uvalue[m+Unz], where Unz = xstore[BASICLU_UNZ]

        If all three arguments are not NULL, then they are filled with U in
        compressed column form. The indices in each column are sorted with the
        diagonal element at the end.

        If any of the three arguments is NULL, then U is not returned
        (this is not an error).

Info:

    xstore[BASICLU_STATUS]: status code.

        BASICLU_OK

            The requested quantities have been returned successfully.

        BASICLU_ERROR_argument_missing

            One or more of the mandatory pointer/array arguments are NULL.

        BASICLU_ERROR_invalid_call

            The BASICLU instance does not hold a fresh factorization (either
            basiclu_factorize() has not completed or basiclu_update() has been
            called in the meanwhile).
```

3.4 basiclu_solve_dense

```
lu_int basiclu_solve_dense
(
    lu_int istore[],
    double xstore[],
    lu_int Li[],
    double Lx[],
    lu_int Ui[],
    double Ux[],
    lu_int Wi[],
    double Wx[],
    const double rhs[],
    double lhs[],
    char trans
);

Purpose:

    Given the factorization computed by basiclu_factorize() or basiclu_update()
    and the dense right-hand side, rhs, solve a linear system for the solution
    lhs.

Return:

    BASICLU_ERROR_invalid_store if istore, xstore do not hold a BASICLU
    instance. In this case xstore[BASICLU_STATUS] is not set.

    Otherwise return the status code. See xstore[BASICLU_STATUS] below.

Arguments:

    lu_int istore[]
    double xstore[]
    lu_int Li[]
    double Lx[]
    lu_int Ui[]
    double Ux[]
    lu_int Wi[]
    double Wx[]

        Factorization computed by basiclu_factorize() or basiclu_update().

    const double rhs[m]

        The right-hand side vector.

    double lhs[m]

        Uninitialized on entry. On return lhs holds the solution to the linear
        system.

        lhs and rhs are allowed to overlap. To overwrite rhs with the solution
        pass pointers to the same array.

    char trans

        Defines which system to solve. 't' or 'T' for the transposed system, any
        other character for the forward system.

Info:

    xstore[BASICLU_STATUS]: status code.

        BASICLU_OK
```

The linear system has been successfully solved.

BASICLU_ERROR_argument_missing

One or more of the pointer/array arguments are NULL.

BASICLU_ERROR_invalid_call

The factorization is invalid.

3.5 basiclu_solve_sparse

```
lu_int basiclu_solve_sparse
(
    lu_int istore[],
    double xstore[],
    lu_int Li[],
    double Lx[],
    lu_int Ui[],
    double Ux[],
    lu_int Wi[],
    double Wx [],
    lu_int nzrhs,
    const lu_int irhs[],
    const double xrhs[],
    lu_int *p_nzlhs,
    lu_int ilhs[],
    double lhs[],
    char trans
);

Purpose:

    Given the factorization computed by basiclu_factorize() or basiclu_update()
    and the sparse right-hand side, rhs, solve a linear system for the solution
    lhs.

Return:

    BASICLU_ERROR_invalid_store if istore, xstore do not hold a BASICLU
    instance. In this case xstore[BASICLU_STATUS] is not set.

    Otherwise return the status code. See xstore[BASICLU_STATUS] below.

Arguments:

    lu_int istore[]
    double xstore[]
    lu_int Li[]
    double Lx[]
    lu_int Ui[]
    double Ux[]
    lu_int Wi[]
    double Wx[]

        Factorization computed by basiclu_factorize() or basiclu_update().

    lu_int nzrhs
    const lu_int irhs[nzrhs]
    const double xrhs[nzrhs]

        The right-hand side vector in compressed format. irhs[0..nzrhs-1] are
        the indices of nonzeros and xrhs[0..nzrhs-1] the corresponding values.
        irhs must not contain duplicates.

    lu_int *p_nzlhs
    lu_int ilhs[m]
    lu_int lhs[m]

        *p_nzlhs is uninitialized on entry. On return *p_nzlhs holds
        the number of nonzeros in the solution.
        The contents of ilhs is uninitialized on entry. On return
        ilhs[0..*p_nzlhs-1] holds the indices of nonzeros in the solution.
        The contents lhs must be initialized to zero on entry. On return
        the solution is scattered into lhs.
```

```
    char trans

        Defines which system to solve. 't' or 'T' for the transposed system,
        any other character for the forward system.

Parameters:

    xstore[BASICLU_SPARSE_THRESHOLD]

        Defines which method is used for solving a triangular system. A
        triangular solve can be done either by the two phase method of Gilbert
        and Peierls ("sparse solve") or by a sequential pass through the vector
        ("sequential solve").

        Solving B*x=b requires two triangular solves. The first triangular solve
        is done sparse. The second triangular solve is done sparse if its
        right-hand side has not more than m * xstore[BASICLU_SPARSE_THRESHOLD]
        nonzeros. Otherwise the sequential solve is used.

        Default: 0.05

    xstore[BASICLU_DROP_TOLERANCE]

        Nonzeros which magnitude is less than or equal to the drop tolerance
        are removed after each triangular solve. Default: 1e-20

Info:

    xstore[BASICLU_STATUS]: status code.

        BASICLU_OK

            The linear system has been successfully solved.

        BASICLU_ERROR_argument_missing

            One or more of the pointer/array arguments are NULL.

        BASICLU_ERROR_invalid_call

            The factorization is invalid.

        BASICLU_ERROR_invalid_argument

            The right-hand side is invalid (nzrhs < 0 or nzrhs > m or one or
            more indices out of range).
```

3.6 basiclu_solve_for_update

```
lu_int basiclu_solve_for_update
(
    lu_int istore[],
    double xstore[],
    lu_int Li[],
    double Lx[],
    lu_int Ui[],
    double Ux[],
    lu_int Wi[],
    double Wx[],
    lu_int nzrhs,
    const lu_int irhs[],
    const double xrhs[],
    lu_int *p_nzlhs,
    lu_int ilhs[],
    double lhs[],
    char trans
);

Purpose:

    Given the factorization computed by basiclu_factorize() or basiclu_update(),
    solve a linear system in preparation to update the factorization.

    When the forward system is solved, then the right-hand side is the column
    to be inserted into the factorized matrix. When the transposed system is
    solved, then the right-hand side is a unit vector with entry 1 in position
    of the column to be replaced in the factorized matrix.

    For BASICLU to prepare the update, it is sufficient to compute only a
    partial solution. If the left-hand side is not requested by the user (see
    below), then only one triangular solve is done. If the left-hand side is
    requested, then a second triangular solve is required.

Return:

    BASICLU_ERROR_invalid_store if istore, xstore do not hold a BASICLU
    instance. In this case xstore[BASICLU_STATUS] is not set.

    Otherwise return the status code. See xstore[BASICLU_STATUS] below.

Arguments:

    lu_int istore[]
    double xstore[]
    lu_int Li[]
    double Lx[]
    lu_int Ui[]
    double Ux[]
    lu_int Wi[]
    double Wx[]

        Factorization computed by basiclu_factorize() or basiclu_update().

    lu_int nzrhs
    const lu_int irhs[nzrhs]
    const double xrhs[nzrhs]

        The right-hand side vector in compressed format.

        When the forward system is solved, irhs[0..nzrhs-1] are the indices of
        nonzeros and xrhs[0..nzrhs-1] the corresponding values. irhs must not
        contain duplicates.
```

```
        When the transposed system is solved, the right-hand side is a unit
        vector with entry 1 in position irhs[0]. nzrhs, xrhs and elements of
        irhs other than irhs[0] are not accessed. xrhs can be NULL.

    lu_int *p_nzlhs
    lu_int ilhs[m]
    lu_int lhs[m]

        If any of p_nzlhs, ilhs or lhs is NULL, then the solution to the linear
        system is not requested. In this case only the update is prepared.

        Otherwise:

        *p_nzlhs is uninitialized on entry. On return *p_nzlhs holds
        the number of nonzeros in the solution.
        The contents of ilhs is uninitialized on entry. On return
        ilhs[0..*p_nzlhs-1] holds the indices of nonzeros in the solution.
        The contents of lhs must be initialized to zero on entry. On return
        the solution is  scattered into lhs.

    char trans

        Defines which system to solve. 't' or 'T' for the transposed system,
        any other character for the forward system.

Parameters:

    xstore[BASICLU_MEMORYL]: length of Li and Lx
    xstore[BASICLU_MEMORYU]: length of Ui and Ux
    xstore[BASICLU_MEMORYW]: length of Wi and Wx

    xstore[BASICLU_SPARSE_THRESHOLD]

        Defines which method is used for solving a triangular system. A
        triangular solve can be done either by the two phase method of Gilbert
        and Peierls ("sparse solve") or by a sequential pass through the vector
        ("sequential solve").

        When the solution to the linear system is requested, then two triangular
        systems are solved. The first triangular solve is done sparse. The
        second triangular solve is done sparse if its right-hand side has not
        more than m * xstore[BASICLU_SPARSE_THRESHOLD] nonzeros. Otherwise the
        sequential solve is used.

        When the solution to the linear system is not requested, then this
        parameter has no effect.

        Default: 0.05

    xstore[BASICLU_DROP_TOLERANCE]

        Nonzeros which magnitude is less than or equal to the drop tolerance
        are removed after each triangular solve. Default: 1e-20

Info:

    xstore[BASICLU_STATUS]: status code.

        BASICLU_OK

            The updated has been successfully prepared and, if requested, the
            solution to the linear system has been computed.

        BASICLU_ERROR_argument_missing
```

```
    One or more of the mandatory pointer/array arguments are NULL.

BASICLU_ERROR_invalid_call

    The factorization is invalid.

BASICLU_ERROR_maximum_updates

    There have already been m Forrest-Tomlin updates, see
    xstore[BASICLU_NFORREST]. The factorization cannot be updated any
    more and must be recomputed by basiclu_factorize().
    The solution to the linear system has not been computed.

BASICLU_ERROR_invalid_argument

    The right-hand side is invalid (forward system: nzrhs < 0 or
    nzrhs > m or one or more indices out of range; backward system:
    irhs[0] out of range).

BASICLU_REALLOCATE

    The solve was aborted because of insufficient memory in Li,Lx or
    Ui,Ux to store data for basiclu_update(). The number of additional
    elements required is given by

        xstore[BASICLU_ADD_MEMORYL] >= 0
        xstore[BASICLU_ADD_MEMORYU] >= 0

    The user must reallocate the arrays for which additional memory is
    required. It is recommended to reallocate for the requested number
    of additional elements plus some extra space for further updates
    (e.g. 0.5 times the current array length). The new array lengths
    must be provided in

        xstore[BASICLU_MEMORYL]: length of Li and Lx
        xstore[BASICLU_MEMORYU]: length of Ui and Ux

    basiclu_solve_for_update() will start from scratch in the next call.
```

3.7 basiclu_update

```
lu_int basiclu_update
(
    lu_int istore[],
    double xstore[],
    lu_int Li[],
    double Lx[],
    lu_int Ui[],
    double Ux[],
    lu_int Wi[],
    double Wx[],
    double xtbl
);

Purpose:

    Update the factorization to replace one column of the factorized matrix.
    A call to basiclu_update() must be preceded by calls to
    basiclu_solve_for_update() to provide the column to be inserted and the
    index of the column to be replaced.

    The column to be inserted is defined as the right-hand side in the last call
    to basiclu_solve_for_update() in which the forward system was solved.

    The index of the column to be replaced is defined by the unit vector in the
    last call to basiclu_solve_for_update() in which the transposed system was
    solved.

Return:

    BASICLU_ERROR_invalid_store if istore, xstore do not hold a BASICLU
    instance. In this case xstore[BASICLU_STATUS] is not set.

    Otherwise return the status code. See xstore[BASICLU_STATUS] below.

Arguments:

    lu_int istore[]
    double xstore[]
    lu_int Li[]
    double Lx[]
    lu_int Ui[]
    double Ux[]
    lu_int Wi[]
    double Wx[]

        Factorization computed by basiclu_factorize() or basiclu_update().

    double xtbl

        This is an optional argument to monitor numerical stability. xtbl can be
        either of

        (a) element j0 of the solution to the forward system computed by
            basiclu_solve_for_update(), where j0 is the column to be replaced;

        (b) the dot product of the incoming column and the solution to the
            transposed system computed by basiclu_solve_for_update().

        In either case xstore[BASICLU_PIVOT_ERROR] (see below) has a defined
        value. If monitoring stability is not desired, xtbl can be any value.

Parameters:

    xstore[BASICLU_MEMORYL]: length of Li and Lx
```

```
    xstore[BASICLU_MEMORYU]: length of Ui and Ux
    xstore[BASICLU_MEMORYW]: length of Wi and Wx

    xstore[BASICLU_DROP_TOLERANCE]

        Nonzeros which magnitude is less than or equal to the drop tolerance
        are removed from the row eta matrix. Default: 1e-20

Info:

    xstore[BASICLU_STATUS]: status code.

        BASICLU_OK

            The update has successfully completed.

        BASICLU_ERROR_argument_missing

            One or more of the pointer/array arguments are NULL.

        BASICLU_ERROR_invalid_call

            The factorization is invalid or the update was not prepared by two
            calls to basiclu_solve_for_update().

        BASICLU_REALLOCATE

            Insufficient memory in Wi,Wx. The number of additional elements
            required is given by

                xstore[BASICLU_ADD_MEMORYW] > 0

            The user must reallocate Wi,Wx. It is recommended to reallocate for
            the requested number of additional elements plus some extra space
            for further updates (e.g. 0.5 times the current array length). The
            new array length must be provided in

                xstore[BASICLU_MEMORYW]: length of Wi and Wx

            basiclu_update will start from scratch in the next call.

        BASICLU_ERROR_singular_update

            The updated factorization would be (numerically) singular. No update
            has been computed and the old factorization is still valid.

    xstore[BASICLU_PIVOT_ERROR]

        When xtbl was given (see above), then xstore[BASICLU_PIVOT_ERROR] is a
        measure for numerical stability. It is the difference between two
        computations of the new pivot element relative to the new pivot element.
        A value larger than 1e-10 indicates numerical instability and suggests
        refactorization (and possibly tightening the pivot tolerance).

    xstore[BASICLU_MAX_ETA]

        The maximum entry (in absolute value) in the eta vectors from the
        Forrest-Tomlin update. A large value, say > 1e6, indicates that pivoting
        on diagonal element was unstable and refactorization might be necessary.
```

4 High level C interface

The high level C interface consists of wrapper functions around the low level interface which do memory allocation. They maintain the arrays used by the low level interface inside a `struct basiclu_object`. To use the high level C interface, user code must include `basiclu.h`. Defined constants start with `BASICLU_` and function names start with `basiclu_obj_`. See `example/maxvolume.c` for an application using the high level C interface.

4.1 basiclu_object

```
struct basiclu_object
{
    lu_int *istore;
    double *xstore;
    lu_int *Li, *Ui, *Wi;
    double *Lx, *Ux, *Wx;
    double *lhs;
    lu_int *ilhs;
    lu_int nzlhs;
    double realloc_factor;
};

A variable of type struct basiclu_object must be defined in user code. Its
members are set and maintained by basiclu_obj_* routines. User code should only
access the following members:

    xstore (read/write)

        set parameters and get info values

    lhs, ilhs, nzlhs (read only)

        holds solution after solve_sparse() and solve_for_update()

    realloc_factor (read/write)

        Arrays are reallocated for max(realloc_factor, 1.0) times the
        required size. Default: 1.5
```

4.2 basiclu_obj_initialize

```
lu_int basiclu_obj_initialize
(
    struct basiclu_object *obj,
    lu_int m
);

Purpose:

    Initialize a BASICLU object. When m is positive, then *obj is initialized to
    process matrices of dimension m. When m is zero, then *obj is initialized to
    a "null" object, which cannot be used for factorization, but can be passed
    to basiclu_obj_free().

    This routine must be called once before passing obj to any other
    basiclu_obj_ routine. When obj is initialized to a null object, then the
    routine can be called again to reinitialize obj.

Return:

    BASICLU_OK

        *obj successfully initialized.

    BASICLU_ERROR_argument_missing

        obj is NULL.

    BASICLU_ERROR_invalid_argument

        m is negative.

    BASICLU_ERROR_out_of_memory

        insufficient memory to initialize object.

Arguments:

    struct basiclu_object *obj

        Pointer to the object to be initialized.

    lu_int m

        The dimension of matrices which can be processed, or 0.
```

4.3 basiclu_obj_factorize

```
lu_int basiclu_obj_factorize
(
    struct basiclu_object *obj,
    const lu_int *Bbegin,
    const lu_int *Bend,
    const lu_int *Bi,
    const double *Bx
);

Purpose:

    Call basiclu_factorize() on a BASICLU object.

Return:

    BASICLU_ERROR_invalid_object

        obj is NULL or initialized to a null object.

    BASICLU_ERROR_out_of_memory

        reallocation failed because of insufficient memory.

    Other return codes are passed through from basiclu_factorize().

Arguments:

    struct basiclu_object *obj

        Pointer to an initialized BASICLU object.

    The other arguments are passed through to basiclu_factorize().
```

4.4 basiclu_obj_get_factors

```
lu_int basiclu_obj_get_factors
(
    struct basiclu_object *obj,
    lu_int rowperm[],
    lu_int colperm[],
    lu_int Lcolptr[],
    lu_int Lrowidx[],
    double Lvalue[],
    lu_int Ucolptr[],
    lu_int Urowidx[],
    double Uvalue[]
);

Purpose:

    Call basiclu_get_factors() on a BASICLU object.

Return:

    BASICLU_ERROR_invalid_object

        obj is NULL or initialized to a null object.

    Other return codes are passed through from basiclu_get_factors().

Arguments:

    struct basiclu_object *obj

        Pointer to an initialized BASICLU object.

    The other arguments are passed through to basiclu_get_factors().
```

4.5 basiclu_obj_solve_dense

```
lu_int basiclu_obj_solve_dense
(
    struct basiclu_object *obj,
    const double rhs[],
    double lhs[],
    char trans
);

Purpose:

    Call basiclu_solve_dense() on a BASICLU object.

Return:

    BASICLU_ERROR_invalid_object

        obj is NULL or initialized to a null object.

    Other return codes are passed through from basiclu_solve_dense().

Arguments:

    struct basiclu_object *obj

        Pointer to an initialized BASICLU object.

    The other arguments are passed through to basiclu_solve_dense().
```

4.6 basiclu_obj_solve_sparse

```
lu_int basiclu_obj_solve_sparse
(
    struct basiclu_object *obj,
    lu_int nzrhs,
    const lu_int irhs[],
    const double xrhs[],
    char trans
);

Purpose:

    Call basiclu_solve_sparse() on a BASICLU object. On success, the solution
    is provided in obj->lhs and the nonzero pattern is stored in
    obj->ilhs[0..obj->nzlhs-1].

Return:

    BASICLU_ERROR_invalid_object

        obj is NULL or initialized to a null object.

    Other return codes are passed through from basiclu_solve_sparse().

Arguments:

    struct basiclu_object *obj

        Pointer to an initialized BASICLU object.

    The other arguments are passed through to basiclu_solve_sparse().
```

4.7 basiclu_obj_solve_for_update

```
lu_int basiclu_obj_solve_for_update
(
    struct basiclu_object *obj,
    lu_int nzrhs,
    const lu_int irhs[],
    const double xrhs[],
    char trans,
    lu_int want_solution
);

Purpose:

    Call basiclu_solve_for_update() on a BASICLU object. On success, if the
    solution was requested, it is provided in obj->lhs and the nonzero pattern
    is stored in obj->ilhs[0..obj->nzlhs-1].

Return:

    BASICLU_ERROR_invalid_object

        obj is NULL or initialized to a null object.

    BASICLU_ERROR_out_of_memory

        reallocation failed because of insufficient memory.

    Other return codes are passed through from basiclu_solve_for_update().

Arguments:

    struct basiclu_object *obj

        Pointer to an initialized BASICLU object.

    lu_int want_solution

        Nonzero to compute the solution to the linear system,
        zero to only prepare the update.

    The other arguments are passed through to basiclu_solve_for_update().
```

4.8 basiclu_obj_update

```
lu_int basiclu_obj_update
(
    struct basiclu_object *obj,
    double xtbl
);

Purpose:

    Call basiclu_update() on a BASICLU object.

Return:

    BASICLU_ERROR_invalid_object

        obj is NULL or initialized to a null object.

    BASICLU_ERROR_out_of_memory

        reallocation failed because of insufficient memory.

    Other return codes are passed through from basiclu_update().

Arguments:

    struct basiclu_object *obj

        Pointer to an initialized BASICLU object.

    The other arguments are passed through to basiclu_update().
```

4.9 basiclu_obj_free

```
void basiclu_obj_free
(
    struct basiclu_object *obj
);

Purpose:

    Free memory allocated from a BASICLU object. The object must have been
    initialized before by basiclu_obj_initialize(). Subsequent calls to
    basiclu_obj_free() will do nothing.

Arguments:

    struct basiclu_object *obj

        Pointer to the object which memory is to be freed. When obj is NULL,
        then the routine does nothing.
```

5 Julia interface

BASICLU can be used from the Julia programming language. To do so, the package must be compiled and the `lib/` directory must be added to the shared library load path of your system. Then run Julia and include the `basiclu` module by

```
include("path/to/BASICLU/Julia/basiclu.jl")
```

The following is an example for a Julia program using BASICLU. See also the documentation of the module functions and `Julia/test.jl`.

```
include("BASICLU/Julia/basiclu.jl")
m = 1000
this = basiclu.initialize(m)
B = sprand(m,m,5e-3) + speye(m)             # get a sparse matrix
err = basiclu.factorize(this, B)
if err != basiclu.BASICLU_OK
  error("factorization failed or singular")
end
rhs = randn(m)                              # get a right-hand side
lhs = basiclu.solve(this, rhs, 'N')
res = norm(B*lhs-rhs,Inf)                   # compute residual
col = speye(m)[:,1]                         # unit vector to be inserted into B
lhs = basiclu.solve4update(this, col, true)
(vmax,j) = findmax(abs(lhs))
xtbl = lhs[j]
basiclu.solve4update(this, j)               # prepare to replace column j of B
piverr = basiclu.update(this, xtbl)
lhs = basiclu.solve(this, rhs, 'N')
B[:,j] = col
res = norm(B*lhs-rhs,Inf)
```