UMFPACK Quick Start Guide

Timothy A. Davis

DrTimothyAldenDavis@gmail.com, http://www.suitesparse.com

VERSION 5.7.9, Oct 20, 2019

Abstract

UMFPACK is a set of routines for solving unsymmetric sparse linear systems, $\mathbf{Ax} = \mathbf{b}$, using the Unsymmetric-pattern MultiFrontal method and direct sparse LU factorization. It is written in ANSI/ISO C, with a MATLAB interface. UMFPACK relies on the Level-3 Basic Linear Algebra Subprograms (dense matrix multiply) for its performance. This code works on Windows and many versions of Unix (Sun Solaris, Red Hat Linux, IBM AIX, SGI IRIX, and Compaq Alpha). This is a "quick start" guide for Unix users of the C interface.

1 Overview

UMFPACK is a set of routines for solving systems of linear equations, $\mathbf{Ax} = \mathbf{b}$, when $\mathbf{A}$ is sparse and unsymmetric. The sparse matrix $\mathbf{A}$ can be square or rectangular, singular or non-singular, and real or complex (or any combination). Only square matrices $\mathbf{A}$ can be used to solve $\mathbf{Ax} = \mathbf{b}$ or related systems. Rectangular matrices can only be factorized.

UMFPACK is a built-in routine in MATLAB used by the forward and backslash operator, and the `lu` routine. The following is a short introduction to Unix users of the C interface of UMFPACK.

The C-callable UMFPACK library consists of 32 user-callable routines and one include file. Twenty-eight of the routines come in four versions, with different sizes of integers and for real or complex floating-point numbers. This Quick Start Guide assumes you are working with real matrices (not complex) and with `int`'s as integers (not `long`'s). Refer to the User Guide for information about the complex and long integer versions. The include file `umfpack.h` must be included in any C program that uses UMFPACK.

For more details, see: *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*, Davis, T. A., ACM Trans. Math. Software, vol 30. no 2, 2004, pp. 165-195, and *Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method*, same issue, pp. 196-199.

2 Primary routines, and a simple example

Five primary UMFPACK routines are required to factorize $\mathbf{A}$ or solve $\mathbf{Ax} = \mathbf{b}$. An overview of the primary features of the routines is given in Section 5. Additional routines are available for passing a different column ordering to UMFPACK, changing default parameters, manipulating

sparse matrices, getting the LU factors, save and loading the LU factors from a file, computing the determinant, and reporting results. See the User Guide for more information.

- `umfpack_di_symbolic`:

  Pre-orders the columns of **A** to reduce fill-in and performs a symbolic analysis. Returns an opaque `Symbolic` object as a `void *` pointer. The object contains the symbolic analysis and is needed for the numerical factorization.

- `umfpack_di_numeric`:

  Numerically scales and then factorizes a sparse matrix $\mathbf{PAQ}$, $\mathbf{PRAQ}$, or $\mathbf{PR^{-1}AQ}$ into the product $\mathbf{LU}$, where $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, $\mathbf{R}$ is a diagonal matrix of scale factors, $\mathbf{L}$ is lower triangular with unit diagonal, and $\mathbf{U}$ is upper triangular. Requires the symbolic ordering and analysis computed by `umfpack_di_symbolic`. Returns an opaque `Numeric` object as a `void *` pointer. The object contains the numerical factorization and is used by `umfpack_di_solve`.

- `umfpack_di_solve`:

  Solves a sparse linear system ($\mathbf{Ax} = \mathbf{b}$, $\mathbf{A^T x} = \mathbf{b}$, or systems involving just $\mathbf{L}$ or $\mathbf{U}$), using the numeric factorization computed by `umfpack_di_numeric`.

- `umfpack_di_free_symbolic`:

  Frees the `Symbolic` object created by `umfpack_di_symbolic`.

- `umfpack_di_free_numeric`:

  Frees the `Numeric` object created by `umfpack_di_numeric`.

The matrix **A** is represented in compressed column form, which is identical to the sparse matrix representation used by MATLAB. It consists of three arrays, where the matrix is `m`-by-`n`, with `nz` entries:

```
int Ap [n+1] ;
int Ai [nz] ;
double Ax [nz] ;
```

All nonzeros are entries, but an entry may be numerically zero. The row indices of entries in column `j` are stored in `Ai[Ap[j]] ... Ap[j+1]-1]`. The corresponding numerical values are stored in `Ax[Ap[j]] ... Ap[j+1]-1]`.

No duplicate row indices may be present, and the row indices in any given column must be sorted in ascending order. The first entry `Ap[0]` must be zero. The total number of entries in the matrix is thus `nz = Ap[n]`. Except for the fact that extra zero entries can be included, there is thus a unique compressed column representation of any given matrix **A**.

Here is a simple main program, `umfpack_simple.c`, that illustrates the basic usage of UMF-PACK.

```
#include <stdio.h>
#include "umfpack.h"
```

```
int     n = 5 ;
int     Ap [ ] = {0, 2, 5, 9, 10, 12} ;
int     Ai [ ] = { 0,   1,   0,    2,   4,   1,   2,   3,    4,   2,   1,   4} ;
double Ax [ ] = {2., 3., 3., -1., 4., 4., -3., 1., 2., 2., 6., 1.} ;
double b [ ] = {8., 45., -3., 3., 19.} ;
double x [5] ;

int main (void)
{
    double *null = (double *) NULL ;
    int i ;
    void *Symbolic, *Numeric ;
    (void) umfpack_di_symbolic (n, n, Ap, Ai, Ax, &Symbolic, null, null) ;
    (void) umfpack_di_numeric (Ap, Ai, Ax, Symbolic, &Numeric, null, null) ;
    umfpack_di_free_symbolic (&Symbolic) ;
    (void) umfpack_di_solve (UMFPACK_A, Ap, Ai, Ax, x, b, Numeric, null, null) ;
    umfpack_di_free_numeric (&Numeric) ;
    for (i = 0 ; i < n ; i++) printf ("x [%d] = %g\n", i, x [i]) ;
    return (0) ;
}
```

The `Ap`, `Ai`, and `Ax` arrays represent the matrix

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 0 & 0 & 0 \\ 3 & 0 & 4 & 0 & 6 \\ 0 & -1 & -3 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 4 & 2 & 0 & 1 \end{bmatrix}.$$

and the solution is $\mathbf{x} = [1\,2\,3\,4\,5]^{\mathsf{T}}$. The program uses default control settings and does not return any statistics about the ordering, factorization, or solution (`Control` and `Info` are both `(double *) NULL`).

For routines to manipulate a simpler "triplet-form" data structure for your sparse matrix $\mathbf{A}$, refer to the UMFPACK User Guide.

3   Synopsis of primary C-callable routines

The matrix $\mathbf{A}$ is `m`-by-`n` with `nz` entries. The optional `umfpack_di_defaults` routine loads the default control parameters into the `Control` array. The settings can then be modified before passing the array to the other routines. Refer to Section 5 for more details.

```
#include "umfpack.h"
int status, sys, n, m, nz, Ap [n+1], Ai [nz] ;
double Control [UMFPACK_CONTROL], Info [UMFPACK_INFO], Ax [nz], X [n], B [n] ;
void *Symbolic, *Numeric ;

umfpack_di_defaults (Control) ;
status = umfpack_di_symbolic (m, n, Ap, Ai, Ax, &Symbolic, Control, Info) ;
status = umfpack_di_numeric (Ap, Ai, Ax, Symbolic, &Numeric, Control, Info) ;
status = umfpack_di_solve (sys, Ap, Ai, Ax, X, B, Numeric, Control, Info) ;
umfpack_di_free_symbolic (&Symbolic) ;
umfpack_di_free_numeric (&Numeric) ;
```

4   Installation

You will need to install both UMFPACK and AMD to use UMFPACK. The `UMFPACK` and `AMD` subdirectories must be placed side-by-side within the same parent directory. AMD is a stand-alone package that is required by UMFPACK. UMFPACK can be compiled without the BLAS but your performance will be much less than what it should be.

UMFPACK also requires CHOLMOD, CCAMD, CCOLAMD, COLAMD, and metis-5.1.0 by default. You can remove this dependency by compiling with `-DNCHOLMOD`. Add this to the `UMFPACK_CONFIG` definition in `SuiteSparse_config/SuiteSparse_config.mk`.

System-dependent configurations are in the `SuiteSparse_config/SuiteSparse_config.mk` file. The default settings will work on most systems, except perhaps for the BLAS definition. Sample configurations are provided for Linux, Mac, Sun Solaris, and IBM AIX. The Makefile file will automatically detect what system you have and will set the compile parameters accordingly.

To compile both packages, go to the `UMFPACK` directory and type `make`. This will compile the static libraries (`AMD/Lib/libamd.*` and `UMFPACK/Lib/libumfpack.*`). A demo of the AMD ordering routine will be compiled and tested in the `AMD/Demo` directory, and five demo programs will then be compiled and tested in the `UMFPACK/Demo` directory. The outputs of these demo programs will then be compared with output files in the distribution. Expect to see a few differences, such as residual norms, compile-time control settings, and perhaps memory usage differences.

To install into /usr/local/lib and /usr/local/include, do `make install`. To uninstall from there, do `make uninstall`. For more options, see the `SuiteSparse/README.txt` file.

Use the MATLAB command `umfpack_make` in the MATLAB directory to compile UMFPACK and AMD for use in MATLAB.

If you compile UMFPACK and AMD and then later change the `SuiteSparse_config/SuiteSparse_config` file then you should type `make purge` and then `make` to recompile.

Here are a few of the various parameters that you can control in your `SuiteSparse_config/SuiteSparse_co` file. To list them all, do `make config`.

- `CC =`   your C compiler, such as `cc`.

- `RANLIB =`   your system's `ranlib` program, if needed.

- `CFLAGS =`   optimization flags, such as `-O`.

- `UMFPACK_CONFIG =`   configuration settings for the BLAS, memory allocation routines, and timing routines.

- `LDLIBS =`   your libraries, such as `-lm` or `-lblas`.

- `RM =` the command to delete a file.

- `MV =` the command to rename a file.

- `F77 =` the command to compile a Fortran program (optional).

- `F77FLAGS =` the Fortran compiler flags (optional).

- `F77LIB =` the Fortran libraries (optional).

The `UMFPACK_CONFIG` string can include combinations of the following; most deal with how the BLAS are called:

- `-DNBLAS` if you do not have any BLAS at all.

- `-DNSUNPERF` if you are on Solaris but do not have the Sun Performance Library.

- `-DLONGBLAS` if your BLAS takes non-`int` integer arguments.

- `-DBLAS_INT =` the integer used by the BLAS.

- `-DBLAS_NO_UNDERSCORE` for controlling how C calls the Fortran BLAS. This is set automatically for Windows, Sun Solaris, SGI Irix, Red Hat Linux, Compaq Alpha, and AIX (the IBM RS 6000).

- `-DNRECIPROCAL` controls a trade-off between speed and accuracy. This is off by default (speed preferred over accuracy) except when compiling for MATLAB.

When you compile your program that uses the C-callable UMFPACK library, you need to link your program with all libraries: -lumfpack -lamd -lcholmod -lcolamd -lccolamd -lcamd -lmetis. If you don't compile UMFPACK to use METIS, then you can All libraries are placed in `SuiteSparse/lib` and all include files are placed in `SuiteSparse/include`.

You do not need to directly include any AMD include files in your program, unless you directly call AMD routines. You only need the

```
#include "umfpack.h"
```

statement, as described in Section 3.

If you do `make install`, then the compiler will know where to find all of the SuiteSparse libraries. Otherwise, add `-L/home/myself/SuiteSparse/lib` and `-I/home/myself/SuiteSparse/include` to your compiler flags, if your copy of SuiteSparse is located in `/home/myself/SuiteSparse`.

5 The primary UMFPACK routines

5.1 umfpack_di_symbolic

```
int umfpack_di_symbolic
(
    int n_row,
    int n_col,
    const int Ap [ ],
    const int Ai [ ],
    const double Ax [ ],
    void **Symbolic,
    const double Control [UMFPACK_CONTROL],
    double Info [UMFPACK_INFO]
) ;
```

Purpose:

    Given nonzero pattern of a sparse matrix A in column-oriented form,
    umfpack_di_symbolic performs a column pre-ordering to reduce fill-in
    (using COLAMD or AMD) and a symbolic factorization. This is required
    before the matrix can be numerically factorized with umfpack_di_numeric.

    For the following discussion, let S be the submatrix of A obtained after
    eliminating all pivots of zero Markowitz cost. S has dimension
    (n_row-n1-nempty_row) -by- (n_col-n1-nempty_col), where
    n1 = Info [UMFPACK_COL_SINGLETONS] + Info [UMFPACK_ROW_SINGLETONS],
    nempty_row = Info [UMFPACK_NEMPTY_ROW] and
    nempty_col = Info [UMFPACK_NEMPTY_COL].

Returns:

    The status code is returned. See Info [UMFPACK_STATUS], below.

Arguments:

    int n_row ;   Input argument, not modified.
    int n_col ;   Input argument, not modified.

        A is an n_row-by-n_col matrix. Restriction: n_row > 0 and n_col > 0.

    int Ap [n_col+1] ; Input argument, not modified.

        Ap is an integer array of size n_col+1. On input, it holds the
        "pointers" for the column form of the sparse matrix A. Column j of
        the matrix A is held in Ai [(Ap [j]) ... (Ap [j+1]-1)]. The first
        entry, Ap [0], must be zero, and Ap [j] <= Ap [j+1] must hold for all
        j in the range 0 to n_col-1. The value nz = Ap [n_col] is thus the
        total number of entries in the pattern of the matrix A. nz must be
        greater than or equal to zero.

    int Ai [nz] ;   Input argument, not modified, of size nz = Ap [n_col].

        The nonzero pattern (row indices) for column j is stored in
        Ai [(Ap [j]) ... (Ap [j+1]-1)]. The row indices in a given column j

must be in ascending order, and no duplicate row indices may be present.
    Row indices must be in the range 0 to n_row-1 (the matrix is 0-based).

double Ax [nz] ;   Optional input argument, not modified.

    The numerical values of the sparse matrix A. The nonzero pattern (row
    indices) for column j is stored in Ai [(Ap [j]) ... (Ap [j+1]-1)], and
    the corresponding numerical values are stored in
    Ax [(Ap [j]) ... (Ap [j+1]-1)]. Used only by the 2-by-2 strategy to
    determine whether entries are "large" or "small". You do not have to
    pass the same numerical values to umfpack_di_numeric. If Ax is not
    present (a (double *) NULL pointer), then any entry in A is assumed to
    be "large".

void **Symbolic ;   Output argument.

    **Symbolic is the address of a (void *) pointer variable in the user's
    calling routine (see Syntax, above). On input, the contents of this
    variable are not defined. On output, this variable holds a (void *)
    pointer to the Symbolic object (if successful), or (void *) NULL if
    a failure occurred.

double Control [UMFPACK_CONTROL] ; Input argument, not modified.

    If a (double *) NULL pointer is passed, then the default control
    settings are used. Only the primary parameters are listed below:

    Control [UMFPACK_STRATEGY]: This is the most important control
        parameter. It determines what kind of ordering and pivoting
        strategy that UMFPACK should use. It is new to Version 4.1
        There are 4 options:

        UMFPACK_STRATEGY_AUTO: This is the default. The input matrix is
            analyzed to determine how symmetric the nonzero pattern is, and
            how many entries there are on the diagonal. It then selects one
            of the following strategies. Refer to the User Guide for a
            description of how the strategy is automatically selected.

        UMFPACK_STRATEGY_UNSYMMETRIC: Use the unsymmetric strategy. COLAMD
            is used to order the columns of A, followed by a postorder of
            the column elimination tree. No attempt is made to perform
            diagonal pivoting. The column ordering is refined during
            factorization. This strategy was the only one provided with
            UMFPACK V4.0.

            In the numerical factorization, the
            Control [UMFPACK_SYM_PIVOT_TOLERANCE] parameter is ignored. A
            pivot is selected if its magnitude is >=
            Control [UMFPACK_PIVOT_TOLERANCE] (default 0.1) times the
            largest entry in its column.

        UMFPACK_STRATEGY_SYMMETRIC: Use the symmetric strategy (new to
            Version 4.1). In this method, the approximate minimum degree
            ordering (AMD) is applied to A+A', followed by a postorder of
            the elimination tree of A+A'. UMFPACK attempts to perform

diagonal pivoting during numerical factorization. No refinement
of the column preordering is performed during factorization.

In the numerical factorization, a nonzero entry on the diagonal
is selected as the pivot if its magnitude is >= Control
[UMFPACK_SYM_PIVOT_TOLERANCE] (default 0.001) times the largest
entry in its column. If this is not acceptable, then an
off-diagonal pivot is selected with magnitude >= Control
[UMFPACK_PIVOT_TOLERANCE] (default 0.1) times the largest entry
in its column.

Control [UMFPACK_SCALE]: This parameter is new to V4.1. See
umfpack_numeric.h for a description. Only affects the 2-by-2
strategy. Default: UMFPACK_SCALE_SUM.

double Info [UMFPACK_INFO] ;   Output argument, not defined on input.

Contains statistics about the symbolic analysis. If a (double *) NULL
pointer is passed, then no statistics are returned in Info (this is not
an error condition). The entire Info array is cleared (all entries set
to -1) and then the following statistics are computed (only the
primary statistics are listed):

Info [UMFPACK_STATUS]: status code. This is also the return value,
whether or not Info is present.

UMFPACK_OK

Each column of the input matrix contained row indices
in increasing order, with no duplicates. Only in this case
does umfpack_di_symbolic compute a valid symbolic factorization.
For the other cases below, no Symbolic object is created
(*Symbolic is (void *) NULL).

UMFPACK_ERROR_n_nonpositive

n is less than or equal to zero.

UMFPACK_ERROR_invalid_matrix

Number of entries in the matrix is negative, Ap [0] is nonzero,
a column has a negative number of entries, a row index is out of
bounds, or the columns of input matrix were jumbled (unsorted
columns or duplicate entries).

UMFPACK_ERROR_out_of_memory

Insufficient memory to perform the symbolic analysis. If the
analysis requires more than 2GB of memory and you are using
the 32-bit ("int") version of UMFPACK, then you are guaranteed
to run out of memory. Try using the 64-bit version of UMFPACK.

UMFPACK_ERROR_argument_missing

One or more required arguments is missing.

8

UMFPACK_ERROR_internal_error

    Something very serious went wrong. This is a bug.
    Please contact the author (DrTimothyAldenDavis@gmail.com).

Info [UMFPACK_SIZE_OF_UNIT]: the number of bytes in a Unit,
    for memory usage statistics below.

Info [UMFPACK_SYMBOLIC_PEAK_MEMORY]: the amount of memory (in Units)
    required for umfpack_di_symbolic to complete. This count includes
    the size of the Symbolic object itself, which is also reported in
    Info [UMFPACK_SYMBOLIC_SIZE].

Info [UMFPACK_NUMERIC_SIZE_ESTIMATE]: an estimate of the final size (in
    Units) of the entire Numeric object (both fixed-size and variable-
    sized parts), which holds the LU factorization (including the L, U,
    P and Q matrices).

Info [UMFPACK_PEAK_MEMORY_ESTIMATE]: an estimate of the total amount of
    memory (in Units) required by umfpack_di_symbolic and
    umfpack_di_numeric to perform both the symbolic and numeric
    factorization. This is the larger of the amount of memory needed
    in umfpack_di_numeric itself, and the amount of memory needed in
    umfpack_di_symbolic (Info [UMFPACK_SYMBOLIC_PEAK_MEMORY]). The
    count includes the size of both the Symbolic and Numeric objects
    themselves. It can be a very loose upper bound, particularly when
    the symmetric or 2-by-2 strategies are used.

Info [UMFPACK_FLOPS_ESTIMATE]: an estimate of the total floating-point
    operations required to factorize the matrix. This is a "true"
    theoretical estimate of the number of flops that would be performed
    by a flop-parsimonious sparse LU algorithm. It assumes that no
    extra flops are performed except for what is strictly required to
    compute the LU factorization. It ignores, for example, the flops
    performed by umfpack_di_numeric to add contribution blocks of
    frontal matrices together. If L and U are the upper bound on the
    pattern of the factors, then this flop count estimate can be
    represented in MATLAB (for real matrices, not complex) as:

```
Lnz = full (sum (spones (L))) - 1 ;     % nz in each col of L
Unz = full (sum (spones (U')))' - 1 ;   % nz in each row of U
flops = 2*Lnz*Unz + sum (Lnz) ;
```

    The actual "true flop" count found by umfpack_di_numeric will be
    less than this estimate.

Info [UMFPACK_LNZ_ESTIMATE]: an estimate of the number of nonzeros in
    L, including the diagonal. Since L is unit-diagonal, the diagonal
    of L is not stored. This estimate is a strict upper bound on the
    actual nonzeros in L to be computed by umfpack_di_numeric.

Info [UMFPACK_UNZ_ESTIMATE]: an estimate of the number of nonzeros in
    U, including the diagonal. This estimate is a strict upper bound on
    the actual nonzeros in U to be computed by umfpack_di_numeric.

```
Info [UMFPACK_SYMBOLIC_TIME]:   The CPU time taken, in seconds.

Info [UMFPACK_STRATEGY_USED]: The ordering strategy used:
    UMFPACK_STRATEGY_SYMMETRIC or UMFPACK_STRATEGY_UNSYMMETRIC
```

5.2   umfpack_di_numeric

```
int umfpack_di_numeric
(
    const int Ap [ ],
    const int Ai [ ],
    const double Ax [ ],
    void *Symbolic,
    void **Numeric,
    const double Control [UMFPACK_CONTROL],
    double Info [UMFPACK_INFO]
) ;
```

Purpose:

    Given a sparse matrix A in column-oriented form, and a symbolic analysis
    computed by umfpack_di_symbolic, the umfpack_di_numeric routine performs the
    numerical factorization, PAQ=LU, PRAQ=LU, or P(R\A)Q=LU, where P and Q are
    permutation matrices (represented as permutation vectors), R is the row
    scaling, L is unit-lower triangular, and U is upper triangular. This is
    required before the system Ax=b (or other related linear systems) can be
    solved. umfpack_di_numeric can be called multiple times for each call to
    umfpack_di_symbolic, to factorize a sequence of matrices with identical
    nonzero pattern. Simply compute the Symbolic object once, with
    umfpack_di_symbolic, and reuse it for subsequent matrices.
    umfpack_di_numeric safely detects if the pattern changes, and sets an
    appropriate error code.

Returns:

    The status code is returned. See Info [UMFPACK_STATUS], below.

Arguments:

    int Ap [n_col+1] ; Input argument, not modified.

        This must be identical to the Ap array passed to umfpack_di_symbolic.
        The value of n_col is what was passed to umfpack_di_symbolic (this is
        held in the Symbolic object).

    int Ai [nz] ;   Input argument, not modified, of size nz = Ap [n_col].

        This must be identical to the Ai array passed to umfpack_di_symbolic.

    double Ax [nz] ;   Input argument, not modified, of size nz = Ap [n_col].

        The numerical values of the sparse matrix A. The nonzero pattern (row
        indices) for column j is stored in Ai [(Ap [j]) ... (Ap [j+1]-1)], and
        the corresponding numerical values are stored in
        Ax [(Ap [j]) ... (Ap [j+1]-1)].

    void *Symbolic ;   Input argument, not modified.

        The Symbolic object, which holds the symbolic factorization computed by
        umfpack_di_symbolic. The Symbolic object is not modified by
        umfpack_di_numeric.

```
    void **Numeric ;    Output argument.

        **Numeric is the address of a (void *) pointer variable in the user's
        calling routine (see Syntax, above).  On input, the contents of this
        variable are not defined.  On output, this variable holds a (void *)
        pointer to the Numeric object (if successful), or (void *) NULL if
        a failure occurred.

double Control [UMFPACK_CONTROL] ;   Input argument, not modified.

        If a (double *) NULL pointer is passed, then the default control
        settings are used.  Only the primary parameters are listed below:

        Control [UMFPACK_PIVOT_TOLERANCE]:  relative pivot tolerance for
            threshold partial pivoting with row interchanges.  In any given
            column, an entry is numerically acceptable if its absolute value is
            greater than or equal to Control [UMFPACK_PIVOT_TOLERANCE] times
            the largest absolute value in the column.  A value of 1.0 gives true
            partial pivoting.  If less than or equal to zero, then any nonzero
            entry is numerically acceptable as a pivot (this is changed from
            Version 4.0).  Default: 0.1.

            Smaller values tend to lead to sparser LU factors, but the solution
            to the linear system can become inaccurate.  Larger values can lead
            to a more accurate solution (but not always), and usually an
            increase in the total work.

        Control [UMFPACK_SYM_PIVOT_TOLERANCE]:  This parameter is new to V4.1.
            If diagonal pivoting is attempted (the symmetric
            strategy is used) then this parameter is used to control when the
            diagonal entry is selected in a given pivot column.  The absolute
            value of the entry must be >= Control [UMFPACK_SYM_PIVOT_TOLERANCE]
            times the largest absolute value in the column.  A value of zero
            will ensure that no off-diagonal pivoting is performed, except that
            zero diagonal entries are not selected if there are any off-diagonal
            nonzero entries.

            If an off-diagonal pivot is selected, an attempt is made to restore
            symmetry later on.  Suppose A (i,j) is selected, where i != j.
            If column i has not yet been selected as a pivot column, then
            the entry A (j,i) is redefined as a "diagonal" entry, except that
            the tighter tolerance (Control [UMFPACK_PIVOT_TOLERANCE]) is
            applied.  This strategy has an effect similar to 2-by-2 pivoting
            for symmetric indefinite matrices.  If a 2-by-2 block pivot with
            nonzero structure

                        i j
                    i: 0 x
                    j: x 0

            is selected in a symmetric indefinite factorization method, the
            2-by-2 block is inverted and a rank-2 update is applied.  In
            UMFPACK, this 2-by-2 block would be reordered as
```

```
                    j i
               i: x 0
               j: 0 x
```

        In both cases, the symmetry of the Schur complement is preserved.

    Control [UMFPACK_SCALE]: This parameter is new to V4.1. Version 4.0
        did not scale the matrix. Note that the user's input matrix is
        never modified, only an internal copy is scaled.

        There are three valid settings for this parameter. If any other
        value is provided, the default is used.

        UMFPACK_SCALE_NONE: no scaling is performed.

        UMFPACK_SCALE_SUM: each row of the input matrix A is divided by
            the sum of the absolute values of the entries in that row.
            The scaled matrix has an infinity norm of 1.

        UMFPACK_SCALE_MAX: each row of the input matrix A is divided by
            the maximum the absolute values of the entries in that row.
            In the scaled matrix the largest entry in each row has
            a magnitude exactly equal to 1.

        Scaling is very important for the "symmetric" strategy when
        diagonal pivoting is attempted. It also improves the performance
        of the "unsymmetric" strategy.

        Default: UMFPACK_SCALE_SUM.

double Info [UMFPACK_INFO] ;   Output argument.

    Contains statistics about the numeric factorization. If a
    (double *) NULL pointer is passed, then no statistics are returned in
    Info (this is not an error condition). The following statistics are
    computed in umfpack_di_numeric (only the primary statistics are listed):

    Info [UMFPACK_STATUS]: status code. This is also the return value,
        whether or not Info is present.

        UMFPACK_OK

            Numeric factorization was successful. umfpack_di_numeric
            computed a valid numeric factorization.

        UMFPACK_WARNING_singular_matrix

            Numeric factorization was successful, but the matrix is
            singular. umfpack_di_numeric computed a valid numeric
            factorization, but you will get a divide by zero in
            umfpack_di_solve. For the other cases below, no Numeric object
            is created (*Numeric is (void *) NULL).

        UMFPACK_ERROR_out_of_memory
```

Insufficient memory to complete the numeric factorization.

UMFPACK_ERROR_argument_missing

One or more required arguments are missing.

UMFPACK_ERROR_invalid_Symbolic_object

Symbolic object provided as input is invalid.

UMFPACK_ERROR_different_pattern

The pattern (Ap and/or Ai) has changed since the call to
umfpack_di_symbolic which produced the Symbolic object.

Info [UMFPACK_NUMERIC_SIZE]:  the actual final size (in Units) of the
entire Numeric object, including the final size of the variable
part of the object.  Info [UMFPACK_NUMERIC_SIZE_ESTIMATE],
an estimate, was computed by umfpack_di_symbolic.  The estimate is
normally an upper bound on the actual final size, but this is not
guaranteed.

Info [UMFPACK_PEAK_MEMORY]:  the actual peak memory usage (in Units) of
both umfpack_di_symbolic and umfpack_di_numeric.  An estimate,
Info [UMFPACK_PEAK_MEMORY_ESTIMATE], was computed by
umfpack_di_symbolic.  The estimate is normally an upper bound on the
actual peak usage, but this is not guaranteed.  With testing on
hundreds of matrix arising in real applications, I have never
observed a matrix where this estimate or the Numeric size estimate
was less than the actual result, but this is theoretically possible.
Please send me one if you find such a matrix.

Info [UMFPACK_FLOPS]:  the actual count of the (useful) floating-point
operations performed.  An estimate, Info [UMFPACK_FLOPS_ESTIMATE],
was computed by umfpack_di_symbolic.  The estimate is guaranteed to
be an upper bound on this flop count.  The flop count excludes
"useless" flops on zero values, flops performed during the pivot
search (for tentative updates and assembly of candidate columns),
and flops performed to add frontal matrices together.

Info [UMFPACK_LNZ]: the actual nonzero entries in final factor L,
including the diagonal.  This excludes any zero entries in L,
although some of these are stored in the Numeric object.  The
Info [UMFPACK_LU_ENTRIES] statistic does account for all
explicitly stored zeros, however.  Info [UMFPACK_LNZ_ESTIMATE],
an estimate, was computed by umfpack_di_symbolic.  The estimate is
guaranteed to be an upper bound on Info [UMFPACK_LNZ].

Info [UMFPACK_UNZ]: the actual nonzero entries in final factor U,
including the diagonal.  This excludes any zero entries in U,
although some of these are stored in the Numeric object.  The
Info [UMFPACK_LU_ENTRIES] statistic does account for all
explicitly stored zeros, however.  Info [UMFPACK_UNZ_ESTIMATE],
an estimate, was computed by umfpack_di_symbolic.  The estimate is
guaranteed to be an upper bound on Info [UMFPACK_UNZ].

Info [UMFPACK_NUMERIC_TIME]:  The CPU time taken, in seconds.

## 5.3   umfpack_di_solve

```
int umfpack_di_solve
(
    int sys,
    const int Ap [ ],
    const int Ai [ ],
    const double Ax [ ],
    double X [ ],
    const double B [ ],
    void *Numeric,
    const double Control [UMFPACK_CONTROL],
    double Info [UMFPACK_INFO]
) ;
```

Purpose:

   Given LU factors computed by umfpack_di_numeric (PAQ=LU, PRAQ=LU, or
   P(R\A)Q=LU) and the right-hand-side, B, solve a linear system for the
   solution X. Iterative refinement is optionally performed. Only square
   systems are handled. Singular matrices result in a divide-by-zero for all
   systems except those involving just the matrix L. Iterative refinement is
   not performed for singular matrices.

   In the discussion below, n is equal to n_row and n_col, because only
   square systems are handled.

Returns:

   The status code is returned. See Info [UMFPACK_STATUS], below.

Arguments:

   int sys ;   Input argument, not modified.

      Defines which system to solve. (') is the linear algebraic transpose.

            sys value   system solved

            UMFPACK_A   Ax=b
            UMFPACK_At   A'x=b
            UMFPACK_Pt_L   P'Lx=b
            UMFPACK_L   Lx=b
            UMFPACK_Lt_P   L'Px=b
            UMFPACK_Lt   L'x=b
            UMFPACK_U_Qt   UQ'x=b
            UMFPACK_U   Ux=b
            UMFPACK_Q_Ut   QU'x=b
            UMFPACK_Ut   U'x=b

      Iterative refinement can be optionally performed when sys is any of
      the following:

            UMFPACK_A   Ax=b
            UMFPACK_At   A'x=b
```

```
    For the other values of the sys argument, iterative refinement is not
    performed (Control [UMFPACK_IRSTEP], Ap, Ai, and Ax are ignored).

int Ap [n+1] ;       Input argument, not modified.
int Ai [nz] ;        Input argument, not modified.
double Ax [nz] ;     Input argument, not modified.

    If iterative refinement is requested (Control [UMFPACK_IRSTEP] >= 1,
    Ax=b or A'x=b is being solved, and A is nonsingular), then
    these arrays must be identical to the same ones passed to
    umfpack_di_numeric.  The umfpack_di_solve routine does not check the
    contents of these arguments, so the results are undefined if Ap, Ai, Ax,
    are modified between the calls the umfpack_di_numeric and
    umfpack_di_solve.  These three arrays do not need to be present (NULL
    pointers can be passed) if Control [UMFPACK_IRSTEP] is zero, or if a
    system other than Ax=b or A'x=b is being solved, or if A is
    singular, since in each of these cases A is not accessed.

double X [n] ;       Output argument.

    The solution to the linear system, where n = n_row = n_col is the
    dimension of the matrices A, L, and U.

double B [n] ;       Input argument, not modified.

    The right-hand side vector, b, stored as a conventional array of size n
    (or two arrays of size n for complex versions).  This routine does not
    solve for multiple right-hand-sides, nor does it allow b to be stored in
    a sparse-column form.

void *Numeric ;                 Input argument, not modified.

    Numeric must point to a valid Numeric object, computed by
    umfpack_di_numeric.

double Control [UMFPACK_CONTROL] ;  Input argument, not modified.

    If a (double *) NULL pointer is passed, then the default control
    settings are used.

    Control [UMFPACK_IRSTEP]:  The maximum number of iterative refinement
        steps to attempt.  A value less than zero is treated as zero.  If
        less than 1, or if Ax=b or A'x=b is not being solved, or
        if A is singular, then the Ap, Ai, and Ax arguments are not
        accessed.  Default: 2.

double Info [UMFPACK_INFO] ;        Output argument.

    Contains statistics about the solution factorization.  If a
    (double *) NULL pointer is passed, then no statistics are returned in
    Info (this is not an error condition).  The following statistics are
    computed in umfpack_di_solve (only the primary statistics are listed):

    Info [UMFPACK_STATUS]: status code.  This is also the return value,
        whether or not Info is present.
```

```
UMFPACK_OK

    The linear system was successfully solved.

UMFPACK_WARNING_singular_matrix

    A divide-by-zero occurred.  Your solution will contain Inf's
    and/or NaN's.  Some parts of the solution may be valid.  For
    example, solving Ax=b with

    A = [2 0]  b = [ 1 ]  returns x = [ 0.5 ]
        [0 0]     [ 0 ]              [ Inf ]

UMFPACK_ERROR_out_of_memory

    Insufficient memory to solve the linear system.

UMFPACK_ERROR_argument_missing

    One or more required arguments are missing.  The B and X
    arguments are always required.  Info and Control are not
    required.  Ap, Ai and Ax are required if Ax=b or
    A'x=b is to be solved, the (default) iterative
    refinement is requested, and the matrix A is nonsingular.

UMFPACK_ERROR_invalid_system

    The sys argument is not valid, or the matrix A is not square.

UMFPACK_ERROR_invalid_Numeric_object

    The Numeric object is not valid.
```

Info [UMFPACK_SOLVE_FLOPS]: the number of floating point operations
    performed to solve the linear system. This includes the work
    taken for all iterative refinement steps, including the backtrack
    (if any).

Info [UMFPACK_SOLVE_TIME]: The time taken, in seconds.

5.4   umfpack di free symbolic

```
void umfpack_di_free_symbolic
(
    void **Symbolic
) ;
```

Purpose:

　　Deallocates the Symbolic object and sets the Symbolic handle to NULL.

Arguments:

```
    void **Symbolic ;            Input argument, deallocated and Symbolic is
                                 set to (void *) NULL on output.
```

5.5   umfpack di free numeric

```
void umfpack_di_free_numeric
(
    void **Numeric
) ;
```

Purpose:

　　Deallocates the Numeric object and sets the Numeric handle to NULL.

Arguments:

```
    void **Numeric ;            Input argument, deallocated and Numeric is
                                set to (void *) NULL on output.
```

5.6   umfpack di defaults

```
void umfpack_di_defaults
(
    double Control [UMFPACK_CONTROL]
) ;
```

Purpose:

　　Sets the default control parameter settings.

Arguments:

```
    double Control [UMFPACK_CONTROL] ;  Output argument.

        Control is set to the default control parameter settings.
```