Algorithm 8xx: SuiteSparseQR, a multifrontal multithreaded sparse QR factorization package

TIMOTHY A. DAVIS
University of Florida

SuiteSparseQR is an implementation of the multifrontal sparse QR factorization method. Parallelism is exploited both in the BLAS and across different frontal matrices using Intel's Threading Building Blocks, a shared-memory programming model for modern multicore architectures. It can obtain a substantial fraction of the theoretical peak performance of a multicore computer. The package is written in C++ with user interfaces for MATLAB, C, and C++.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*linear systems (direct methods), sparse and very large systems*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis, efficiency*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: QR factorization, least-square problems, sparse matrices

---

1. INTRODUCTION

The algorithms used in SuiteSparseQR are discussed in a companion paper, [Davis 2008]. The multifrontal method breaks the factorization of a large sparse matrix into a sequence of small dense *frontal matrices*. These frontal matrices are related to one another in a tree, which provides one of the levels of parallelism exploited by the method. Within each dense frontal matrix, LAPACK [Anderson et al. 1999] and the Level-3 BLAS [Dongarra et al. 1990] provide another layer for parallelism and also allow for efficient use of modern memory hierarchies.

2. USING SUITESPARSEQR IN MATLAB

The simplest way to use SuiteSparseQR is via MATLAB. Its syntax includes every feature of the MATLAB `qr` in version R2008b and earlier [Gilbert et al. 1992], plus additional features not available in MATLAB. It is also a replacement for `x=A\b` for least-squares problems and underdetermined systems. In addition to substantial gains in performance (100x is not uncommon), SuiteSparseQR adds new capabilities that are not present in MATLAB.

Dept. of Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, USA. email: davis@cise.ufl.edu. http://www.cise.ufl.edu/∼davis. Portions of this work were supported by the National Science Foundation, under grants 0203270, 0620286, and 0619080.

2.1 New features for use in MATLAB

(1) The matrix `Q` can be returned as a matrix just like the current MATLAB `qr` syntax, but the matrix form of `Q` can be very costly to store in terms of memory and the time needed to compute it. SuiteSparseQR provides an alternative: a more efficient representation of `Q` as a MATLAB `struct`, with a set of Householder vectors, `Q.H`, coefficients `Q.Tau`, and a permutation `Q.P`. It is often the case that `nnz(Q.H)` is less than `nnz(R)`.

(2) The MATLAB statement `x=A\b` when `A` is underdetermined computes a basic solution. SuiteSparseQR can do this too, but it can also compute a minimum 2-norm solution far more efficiently than MATLAB can. With the MATLAB `qr` this can only be done with `Q` in its matrix form, which is costly.

(3) If `A` can be permuted into the form `A*P=[A11 A12 ; 0 A22]` where `A11` is upper triangular with diagonal entries above a given tolerance, then SuiteSparseQR quickly finds this permutation, leading to a large reduction in time and memory as compared to the MATLAB sparse `qr`. The submatrices `A11` and `A12` become the first rows of `R`, with no numerical work.

(4) MATLAB uses only one fill-reducing ordering within `x=A\b` when `A` is rectangular (COLMMD). SuiteSparseQR provides interfaces to ordering methods that are faster and give better orderings: AMD, COLAMD, and METIS. The default ordering is to use COLAMD when `A22` has twice as many rows as columns, or fewer. Otherwise, AMD is tried. If the fill-in and flop count from AMD is low, the AMD ordering is used. Otherwise, METIS is tried (if installed) and the best of AMD and METIS is used. Other alternatives include trying all three, or just the two minimum degree variants, and taking the best ordering found (the one with the least `nnz(R)`).

(5) For rank-deficient matrices, `R=qr(A)` returns `R` in a "squeezed" form that can be difficult to use in subsequent MATLAB operations. Let `r` be the estimated rank. SuiteSparseQR can return `R` in upper trapezoidal form as `[R11 R12 ; 0 0]` where `R11` is `r`-by-`r` with diagonal elements all greater in magnitude than a given tolerance. For example, `condest(R(1:r,1:r))` can be easily computed if `R` is upper trapezoidal; this cannot be done with the squeezed `R`.

(6) SuiteSparseQR fully supports sparse complex rectangular matrices; the MATLAB `qr` and `x=A\b` do not.

(7) Finally, SuiteSparseQR exploits parallelism, whereas the existing MATLAB methods cannot (neither `qr(A)` nor `x=A\b` when `A` is sparse and rectangular).

2.2 MATLAB Syntax

There are five basic syntaxes available with the function `spqr` that replace the MATLAB `qr`:

(1) `R=spqr(A)` finds the `R` factor, discarding `Q`.

(2) `[Q,R]=spqr(A)` returns both `R` and `Q`, where `Q` can be in matrix form or Householder form.

(3) `[Q,R,P]=spqr(A)` performs the factorization `A*P=Q*R`, where `Q` can be in matrix form, Householder form, or discarded (an empty matrix), and where `P` is a fill-reducing ordering. Note that this is the same syntax as `[Q,R,P]=qr(A)` when

`A` is full, except that for full case the columns are pivoted according to their 2-norm, not for sparsity as is the case for `spqr`.

(4) `[C,R]=spqr(A,B)` is the same as `R=spqr(A)` but also returns `C=Q'*B`, discarding `Q`.
(5) `[C,R,P]=spqr(A,B)` factorizes `A*P=Q*R` and returns both `R` and `C=Q'*B`, discarding `Q`.

The function `x=spqr_solve(A,b)` is a replacement for and extension of `x=A\b` when `A` is sparse and rectangular. The function `spqr_qmult(Q,x)` takes the Householder form of `Q` and applies it to a sparse or full matrix `x` (either `Q*x`, `Q'*x`, `x*Q`, or `x*Q'`).

Each of these syntaxes allow for the use of a final optional input parameter `opts`. If `opts` is a scalar equal to 0, the economy QR is computed, just like the existing MATLAB `qr`. Otherwise, it is a `struct` that can be used to set non-default parameters. If not present, defaults are used. With `opts`, the user can:

(1) change the rank-detection tolerance `tol`,
(2) request the return of the complete QR, the "economy QR," (where `R` has `min(m,n)` rows and `Q` has `min(m,n)` columns) or the "rank-sized QR" (where `R` has `r` rows and `Q` has `r` columns, with `r` being the estimated rank of `A`).
(3) change the default ordering (COLAMD, AMD, METIS, and meta-strategies where one or more orderings are tried and the one with the least `nnz(R)` is chosen),
(4) request `P` as a sparse permutation matrix or as a permutation vector,
(5) control the number of parallel threads used and the granularity of the parallel tasks,
(6) and request `spqr_solve` to compute either the basic solution or minimum 2-norm solution to an underdetermined system.

3. USING SUITESPARSEQR IN C AND C++

SuiteSparseQR relies on CHOLMOD for its basic sparse matrix data structure: a compressed sparse column format. CHOLMOD provides interfaces to the AMD, COLAMD, and METIS ordering methods, supernodal symbolic Cholesky factorization (namely, `symbfact` in MATLAB), functions for converting between different data structures, and for basic operations such as transpose, matrix multiply, reading a matrix from a file, writing a matrix to a file, and many other functions.

3.1 C/C++ Example

The C++ interface is written using templates for handling both real and complex matrices. The simplest function computes the MATLAB equivalent of `x=A\b` and is almost as simple:

```
#include "SuiteSparseQR.hpp"
X = SuiteSparseQR <double> (A, B, cc) ;
```

The C version of this function is almost identical:

```
#include "SuiteSparseQR_C.h"
X = SuiteSparseQR_C_backslash_default (A, B, cc) ;
```

Below is a simple C++ program that illustrates the use of SuiteSparseQR. The program reads in a least-squares problem from `stdin` in MatrixMarket format [Boisvert et al. 1997], solves it, and prints the norm of the residual and the estimated rank of `A`. The comments reflect the MATLAB equivalent statements. The C version of this program is identical except for the `#include` statement and call to SuiteSparseQR which are replaced with the C version of the statement above, and C-style comments.

```
#include "SuiteSparseQR.hpp"
int main (int argc, char **argv)
{
    cholmod_common Common, *cc ;
    cholmod_sparse *A ;
    cholmod_dense *X, *B, *Residual ;
    double rnorm, one [2] = {1,0}, minusone [2] = {-1,0} ;
    int mtype ;

    // start CHOLMOD
    cc = &Common ;
    cholmod_l_start (cc) ;

    // load A
    A = (cholmod_sparse *)
        cholmod_l_read_matrix (stdin, 1, &mtype, cc) ;

    // B = ones (size (A,1),1)
    B = cholmod_l_ones (A->nrow, 1, A->xtype, cc) ;

    // X = A\B
    X = SuiteSparseQR <double> (A, B, cc) ;

    // rnorm = norm (B-A*X)
    Residual = cholmod_l_copy_dense (B, cc) ;
    cholmod_l_sdmult (A, 0, minusone, one, X, Residual, cc) ;
    rnorm = cholmod_l_norm_dense (Residual, 2, cc) ;
    printf ("2-norm of residual: %8.1e\n", rnorm) ;
    printf ("rank %ld\n", cc->SPQR_istat [4]) ;

    // free everything and finish CHOLMOD
    cholmod_l_free_dense (&Residual, cc) ;
    cholmod_l_free_sparse (&A, cc) ;
    cholmod_l_free_dense (&X, cc) ;
    cholmod_l_free_dense (&B, cc) ;
    cholmod_l_finish (cc) ;
    return (0) ;
}
```

3.2 C++ Syntax

All features available to the MATLAB user are also available to both the C and C++ interfaces using a syntax that is not much more complicated than the MATLAB syntax. Additional features not available via the MATLAB interface include the ability to compute the symbolic and numeric factorizations separately (for multiple matrices with the same nonzero pattern but different numerical values). The following is a list of user-callable C++ functions and what they can do:

(1) `SuiteSparseQR`: an overloaded function that provides functions equivalent to `spqr` and `spqr_solve` in the SuiteSparseQR MATLAB interface.

(2) `SuiteSparseQR_factorize`: performs both the symbolic and numeric factorizations and returns a QR factorization object such that `A*P=Q*R`. It always exploits singletons.

(3) `SuiteSparseQR_symbolic`: performs the symbolic factorization and returns a QR factorization object to be passed to `SuiteSparseQR_numeric`. It does not exploit singletons.

(4) `SuiteSparseQR_numeric`: performs the numeric factorization on a QR factorization object, either one constructed by `SuiteSparseQR_symbolic`, or reusing one from a prior call to `SuiteSparseQR_numeric` for a matrix `A` with the same pattern as the first one, but with different numerical values.

(5) `SuiteSparseQR_solve`: solves a linear system `x=R\b`, `x=P*R\b`, `x=R'\b`, or `x=R'\(P'*b)`, using the object returned by `SuiteSparseQR_factorize` or `SuiteSparseQR_numeric`.

(6) `SuiteSparseQR_qmult`: provides the same function as `spqr_qmult` in the MATLAB interface, computing `Q*x`, `Q'*x`, `x*Q`, or `x*Q'`. It uses either a QR factorization in conventional sparse matrix format, or the QR factorization object returned by `SuiteSparseQR_factorize` or `SuiteSparseQR_numeric`.

(7) `SuiteSparseQR_min2norm`: finds the minimum 2-norm solution to an underdetermined linear system.

(8) `SuiteSparseQR_free`: frees the QR factorization object.

4. REQUIREMENTS AND AVAILABILITY

SuiteSparseQR requires four prior Collected Algorithms of the ACM: CHOLMOD [Chen et al. 2009; Davis and Hager 2009] (version 1.7 or later), AMD [Amestoy et al. 1996; 2004], and COLAMD [Davis et al. 2004a; 2004b] for its ordering/analysis phase and for its basic sparse matrix data structure, and the BLAS [Dongarra et al. 1990] for dense matrix computations on its frontal matrices; also required is LAPACK [Anderson et al. 1999] for its Householder reflections. An efficient implementation of the BLAS is strongly recommended, either vendor-provided (such as the Intel MKL, the AMD ACML, or the Sun Performance Library) or other high-performance BLAS such as those of [Goto and van de Geijn 2008].

The use of Intel's Threading Building Blocks is optional [Reinders 2007], but without it, only parallelism within the BLAS can be exploited (if available). SuiteSparseQR can optionally use METIS 4.0.1 [Karypis and Kumar 1998] and two constrained minimum degree ordering algorithms, CCOLAMD and CAMD [Chen et al. 2009], for its fill-reducing ordering options. SuiteSparseQR can be compiled without these ordering methods.

In addition to appearing as Collected Algorithm 8xx of the ACM, SuiteSparseQR is available at http://www.cise.ufl.edu/research/sparse and at MATLAB Central (http://www.mathworks.com/matlabcentral) in the user-contributed File Exchange. SuiteSparseQR is licensed under the GNU GPL.

5. ACKNOWLEDGMENTS

I would like to thank Pat Quillen for illuminating me on how well C++ templates can be used in a mathematical code that can handle both real and complex matrices, and for getting me started with Intel's Threading Building Blocks library.

REFERENCES

Amestoy, P. R., Davis, T. A., and Duff, I. S. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl. 17,* 4, 886–905.

Amestoy, P. R., Davis, T. A., and Duff, I. S. 2004. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. 30,* 3, 381–388.

Anderson, E., Bai, Z., Bischof, C. H., Blackford, S., Demmel, J. W., Dongarra, J. J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. C. 1999. *LAPACK Users' Guide*, 3rd ed. SIAM, Philadelphia, PA.

Boisvert, R. F., Pozo, R., Remington, K., Barrett, R., and Dongarra, J. J. 1997. The Matrix Market: A web resource for test matrix collections. In *Quality of Numerical Software, Assessment and Enhancement*, R. F. Boisvert, Ed. Chapman & Hall, London, 125–137. (`http://math.nist.gov/MatrixMarket`).

Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. 2009. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw. 35,* 3.

Davis, T. A. 2008. Multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Softw.*. under submission.

Davis, T. A., Gilbert, J. R., Larimore, S. I., and Ng, E. G. 2004a. Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. 30,* 3, 377–380.

Davis, T. A., Gilbert, J. R., Larimore, S. I., and Ng, E. G. 2004b. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. 30,* 3, 353–376.

Davis, T. A. and Hager, W. W. 2009. Dynamic supernodes in sparse Cholesky update/downdate and triangular solves. *ACM Trans. Math. Softw. 35,* 4.

Dongarra, J. J., Du Croz, J. J., Duff, I. S., and Hammarling, S. 1990. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw. 16,* 1–17.

Gilbert, J. R., Moler, C., and Schreiber, R. 1992. Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Appl. 13,* 1, 333–356.

Goto, K. and van de Geijn, R. 2008. High performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw. 35,* 1 (July), 4. Article 4, 14 pages.

Karypis, G. and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 359–392.

Reinders, J. 2007. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism.* O'Reilly Media, Sebastopol, CA.

Received Month Year; revised Month Year; accepted Month Year