Overview of SuiteSparseQR, a multifrontal multithreaded sparse QR factorization package

Timothy A. Davis*

May 20, 2011

1 The SuiteSparseQR subset

The SuiteSparseQR meta-package is a subset of the SuiteSparse meta-package, and includes following packages:

- SPQR: SuiteSparseQR itself
- CHOLMOD: which SPQR relies on for its symbolic QR analysis
- AMD: approximate minimum degre ordering
- CAMD: constrained approximate minimum degre ordering, for use with METIS
- COLAMD: column approximate minimum degre ordering
- CAMD: constrained column approximate minimum degre ordering, for use with METIS
- UFconfig: configuration parameters for all SuiteSparse packages

An overview of SuiteSparseQR (SPQR) is given below. Details of each package are described in full user manuals in the `Doc` directories. In particular, refer to `SuiteSparseQR/SPQR/Doc` for more details on SuiteSparseQR.

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

2 Using SuiteSparseQR in MATLAB

The simplest way to use SuiteSparseQR is via MATLAB. To compile for MATLAB, use these commands in the MATLAB Command Window:

```
cd SuiteSparseQR/SPQR/MATLAB
spqr_install
spqr_demo
```

MATLAB R2009a (and later versions) include SuiteSparseQR as the built-in sparse QR method. The following features are new to the sparse `qr` in R2009a:

1. The prior MATLAB `qr` does not exploit singletons. It uses COLMMD, which is slower than COLAMD and provides lower quality orderings.

2. 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 also return `R` in upper trapezoidal form as `[R11 R12 ; 0 0]` where `R11` is `r`-by-`r` upper triangular, via `[Q,R,E]=qr(A)`. Computing `condest(R(1:r,1:r))` is simple with the upper trapezoidal `R`.

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

4. SuiteSparseQR exploits parallelism. The prior MATLAB sparse `qr` does not.

MATLAB R2009a does not expose all of the new features of SuiteSparseQR. These features are available only if SuiteSparseQR is installed by the end user:

1. 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`. This takes much less space than representing `Q` as a sparse matrix. It is often the case that `nnz(Q.H)` is less than `nnz(R)`.

2. The MATLAB statement `x=A\b` when `A` is under-determined 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. Default parameters can be modified, which can:

   - change the rank-detection tolerance τ,
   - 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`).
   - change the default ordering (COLAMD, AMD, METIS, and strategies where multiple orderings are tried and the one with the least `nnz(R)` is chosen),
   - and control the degree of parallelism exploited by TBB and the BLAS.

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.

For Linux/Unix/Mac users who want to use the C++ callable library:

- To compile the C++ library and run a short demo, just type `make` in the Unix shell, in the top-level directory.
- To compile the SuiteSparseQR C++ library, in the Unix shell, do: `cd SPQR/Lib ; make`
- To compile and test an exhaustive test, edit the Tcov/Makefile to select the LAPACK and BLAS libraries, and then do (in the Unix shell): `cd SPQR/Tcov ; make`
- Compilation options in `UFconfig/UFconfig.mk`, `SPQR/*/Makefile`, or `SPQR/MATLAB/spqr_make.m`:
  - `-DNPARTITION` to compile without METIS (default is to use METIS)
  - `-DNEXPERT` to compile without the min 2-norm solution option (default is to include the Expert routines)
  - `-DHAVE_TBB` to compile with Intel's Threading Building Blocks (default is to not use Intel TBB)
  - `-DTIMING` to compile with timing and exact flop counts enabled (default is to not compile with timing and flop counts)

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`:

```
#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 in Matrix Market format solves it, and prints the norm of the residual and the estimated rank of `A`.

```
#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 ;
    cc = &Common ;                                      // start CHOLMOD
    cholmod_l_start (cc) ;
    A = (cholmod_sparse *) cholmod_l_read_matrix (stdin, 1, &mtype, cc) ;
    B = cholmod_l_ones (A->nrow, 1, A->xtype, cc) ;     // B = ones (size (A,1),1)
    X = SuiteSparseQR <double> (A, B, cc) ;             // X = A\B

    Residual = cholmod_l_copy_dense (B, cc) ;           // rnorm = norm (B-A*X)
    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]) ;

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

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. 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 `qr` and `x=A\b` in MATLAB.

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

3. `SuiteSparseQR_symbolic`: performs the symbolic factorization and returns a QR factorization object to be passed to `SuiteSparseQR_numeric`. To permit the reuse of this object, singletons are not exploited.

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`: computes `Q*x`, `Q'*x`, `x*Q`, or `x*Q'` using the Householder representation of `Q`.

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

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

4 License