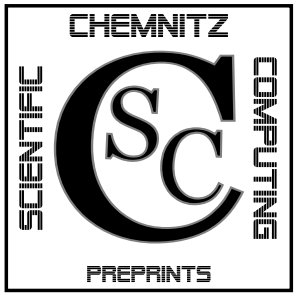
TECHNISCHE UNIVERSITÄT CHEMNITZ

Jens Saak **Stephan Schlömer**

RRQR-MEX

Linux and Windows 32bit MATLAB MEX-Files for the rank revealing QR factorization

CSC/09-09



Chemnitz Scientific Computing
Preprints

Impressum:

Chemnitz Scientific Computing Preprints — ISSN 1864-0087
(1995–2005: Preprintreihe des Chemnitzer SFB393)

Herausgeber:
Professuren für
Numerische und Angewandte Mathematik
an der Fakultät für Mathematik
der Technischen Universität Chemnitz

Postanschrift:
TU Chemnitz, Fakultät für Mathematik
09107 Chemnitz
Sitz:
Reichenhainer Str. 41, 09126 Chemnitz

`http://www.tu-chemnitz.de/mathematik/csc/`

TECHNISCHE UNIVERSITÄT CHEMNITZ

Chemnitz Scientific Computing Preprints

Jens Saak **Stephan Schlömer**

RRQR-MEX

Linux and Windows 32bit MATLAB MEX-Files for the rank revealing QR factorization

CSC/09-09

Abstract

The rank revealing QR decomposition (RRQR) is a special form of the well known QR decomposition of a matrix. It uses specialized pivoting strategies and allows for an easy and efficient numerical rank decision for arbitrary matrices. It is especially valuable when column compression of rectangular matrices needs to be performed. Here we provide documentation and compilation instructions for a MATLAB MEX implementation of the RRQR allowing the easy usage of this decomposition inside the MATLAB environment.

CSC/09-09 ISSN 1864-0087 December 2009

Contents

1 Introduction **1**

2 Contents of the Distribution Archive **1**

2.1 Binary Release 1

2.2 Source Release 1

3 Usage **2**

3.1 Usage of the Gateway 2

4 Compiling the Gateway **3**

4.1 Compiling on MS-Windows (32-bit) 3

4.2 Compiling on Linux (32-bit) 5

4.3 Compiling on Linux (64-bit) 8

5 Measurements and Testing **8**

5.1 Tests on Windows 32bit 10

5.2 Tests on Linux 32bit 13

5.3 Tests on Linux 64bit 16

Author's addresses:

Jens Saak jens.saak@mathematik.tu-chemnitz.de
Stephan Schlömer stephan.schloemer@s2003.tu-chemnitz.de

TU Chemnitz Fakultät für Mathematik D-09107 Chemnitz

http://www.tu-chemnitz.de/mathematik/

Acknowledgments

The author whish to thank Peter Meszmer for the great preparatory work on the gateway file. The current version is mostly only an extension of his work that now covers complex data, as well. Special thanks also go to Vasile Sima (National Institute for Research & Development in Informatics, Bucharest) and Pascal Gahinet (The Mathworks) for many helpful discussions on *mexing* Fortran codes in general.

1 Introduction

We discuss the implementation of the rank revealing QR (rrqr) factorization [1] as MATLAB-MEX-file. We present the documentation of the MATLAB user interface to the rrqr, as well as the compilation of the MEX-files from source. In the numerical testing section we present extensive computations taken out to proof the performance of the codes, even when relying on default LAPACK and BLAS backends.

2 Contents of the Distribution Archive

2.1 Binary Release

The binary release contains the following files:

rrqr.m	the Matlab-Function that calls the MEX-gateway,
rrqrGate.dll	the Windows-MEX-File (*older* MATLAB releases, i.e. before R2008),
rrqeGate.mexw32	the Windows-MEX-File (32bit for MATLAB R2008 and later),
rrqrGate.mexglx	the Linux-MEX-File (32bit),
rrqrGate.mexa64	the Linux-MEX-File (64bit x86_64) and
readme.pdf	this documentation.

2.2 Source Release

The source release contains the following files:

gnumex.exe	Gnumex (MS-Windows only),
rrqr_batch_tool.exe	a small tool for compiling the gateway, (MS-Windows only),
rrqrGate_src.zip	rrqr MEX-Gateway sources and build tools for Linux, (including the required LAPACK and BLAS source files)
readme.pdf	this documentation

3 Usage

If you intend to use the binary distribution, simply extract the archive and add the destination folder to your MATLAB search path. After that the gateway should be accessible via `rrqr` and `help rrqr`. See the next section or the help for usage instructions. Users intending to compile the interface themselves, please refer to Section 4 for detailed instructions.

3.1 Usage of the Gateway

The `rrqr.m` gateway accepts input and supplies output as described in the following.

- $[Q, R, p, r] = \mathsf{rrqr}(A)$, where A is m-by-n, produces an m-by-n upper triangular matrix R and an m-by-m unitary matrix Q so that

$$A \cdot P = Q \cdot \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},$$

  p is a permutation vector, such that $A(:, p) = QR$ and r is the rank of A.

- $[Q, R, p, r] = \mathsf{rrqr}(A,\text{'s'})$ produces the "economy size" decomposition. If m≤n, R is m-by-n and Q is m-by-m, otherwise R is n-by-n and Q is m-by-n.

- $[Q, R, p, r] = \mathsf{rrqr}(A,tol)$. $\frac{1}{tol}$ specifies an upper bound on the condition number of R_{11}. If $tol = 0$ or tol is unset, tol = "machine precision" is chosen as default. tol must be ≥ 0. The tol parameter can be combined with the 's' parameter.

- $[B, R, p, r] = \mathsf{rrqr}(A, C)$ returns a matrix B so that $B = C \cdot Q$. The tol parameter is accepted as well.

- $[R, p, r] = \mathsf{rrqr}(A)$ is identical to the upper cases but does not compute Q. The tol and 's' parameters are accepted as well.

4 Compiling the Gateway

This part describes how to create the rrqr Gateway from source. For usage only, the binary-release is available. If you don't intend to compile the gateway yourself, you can safely ignore this section completely.

4.1 Compiling on MS-Windows (32-bit)

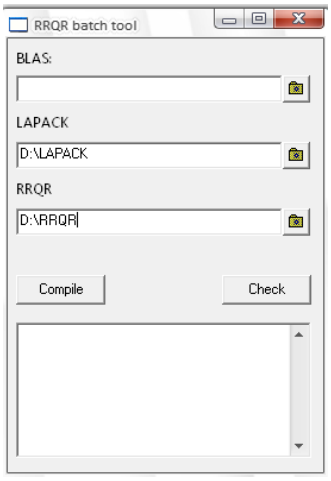
The following tools are highly recommended for an accurate compilation process – although any suggestion for the usage of other compilers and simplified compilation procedures is welcome – under the assumption that free software is required. You should also be able to use the procedure described here on 64bit Windows machines. Note that the tools have only been tested on Windows 2000, XP and Vista in conjunction with MATLAB upto release R2008a.

If you happen to have a commercial Fortran compiler things may be as easy as calling `mex -setup` prior to compiling the gateway. For example Intel®Visual Fortran Compilers integrate easily into MATLAB and in conjunction with the Math kernel library (an implementation of, e.g., the BLAS and LAPACK libraries.) yield very good performance of the produced codes.

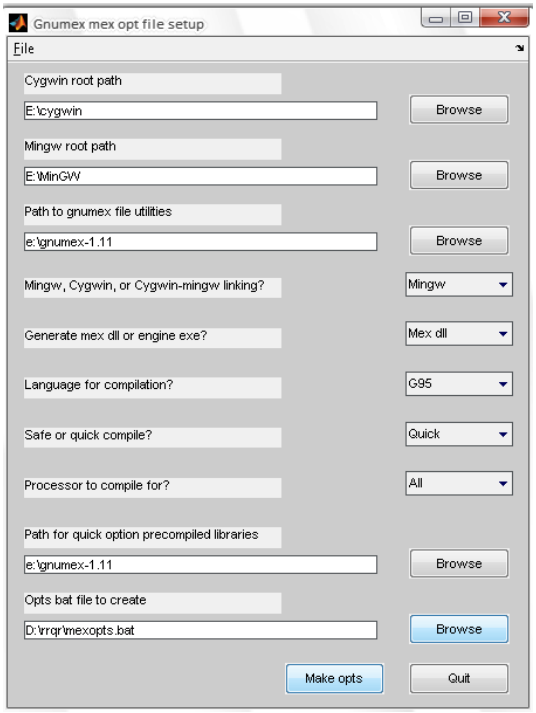
However if you need to use freely available software these are our tools of choice:

gnumex	:	http://sourceforge.net/projects/gnumex/
MinGW	:	www.MinGW.org
g95	:	available from www.g95.org
LAPACK & BLAS	:	sources found on www.netlib.org/lapack/ (optionally one can use the sources provided as part of the download to only build the parts needed by the gateway file.)

Install the required tools and extract the libraries if necessary. Start MATLAB and append the Gnumex folder to MATLAB search path, respectively change to Gnumex folder directly. Call `gnumex` from the MATLAB command line and set the correct MingW root path to the directory, where MinGW is installed.



(a) RRQR batch tool



(b) GnuMex

Figure 1: Screenshots of Windows tools

Choose MinGW as linker and select g95 as language for compilation (also compare Figure 1). Set the correct target processor if necessary. Declare `rrqrGate` as destination path and create the optionsfile by clicking on the *Makeopts* button. You can now quit Gnumex and MATLAB.

Execute `rrqr_batch_tool` from outside Matlab and build the gateway by clicking on the compile button. Make sure that you have already selected accurate RRQRand LAPACKdirectories (and BLASfolder, if necessary). Without these, a faultless compile process is not possible.

Note: The BLAS source is part of the LAPACK package. If no BLAS root is declared, the BLAS root will be expand from LAPACK root folder automatically. It may be desirable to use threaded BLAS implementations, e.g, GotoBlas[1], on MulitCore processors, though.

Note: By clicking the check button, rrqr batch tool tests availability for all needed sources.

Note that it is essential that the three files `makeopts.bat`, `rrqrGate.f` and `rrqr_batch_tool.exe` are in the same working directory.

Note further that when building the LAPACK and BLAS routines need by the RRQR from the sources provided with the download, it is essential that the file `dlamch.f` is compiled without any compiler optimization flags. Compare also the behavior of `make mexfromsrc` in the following section.

4.2 Compiling on Linux (32-bit)

The following tools are required for compiling the rrqr Gateway.

g95/gfortran	:	found on `www.g95.org` or as part of recent `gcc` suites on `gcc.gnu.org` respectively. (Any other Fortran 95 compliant compiler may work as well.)
LAPACK & BLAS	:	found on `www.netlib.org/lapack/` (optional, one may as well compile against the shared object libraries coming as part of MATLAB. However building a standalone version can help avoiding many hour of debugging.)

Extract `rrqr_acm.tar.gz`. Compile the RRQR library. For more details, see the corresponding `README`. Compile the required libraries BLAS , LAPACK ,

[1] `http://www.cs.utexas.edu/users/flame/goto/`

if necessary. Run MATLAB and change directory to `rrqrGate`. Compile the Gateway by prompting the following line.

```
mex -fortran rrqrGate.f -L/rrqrGate/rrqr_acm/ -lrrqr
```

This will build the MEXfile for your architecture in the working directory and link it against the shared object versions of LAPACK and BLAS provided by MATLAB. This should be fine on recent MATLAB versions. So if it works you are done. Alternatively you can compile using `make` and the `Makefile` provided in the package. You will need to supply some information in the `make.inc`, though. A sample include file looks like this:

```
###########################################################################
#   select the mex-compiler binary and options                            #
###########################################################################
FC      = $(PATH_TO_MATLAB)/bin/mex

FFLAGS = -fPIC
#FFLAGS = -v  #let's MEX show you what it uses
#FFLAGS = -n  #let's MEX show you which commands it would run

OFLAGS = -O
# Note that OFLAGS = -O is the only option all other optimization flags
# have to go to the mexopts.sh file. This only tell mex to use the
# OFLAGS given in that file.

###########################################################################
#   where do we find the required libraries and files?                    #
###########################################################################
MEXOPTS =
#MEXOPTS = -f ./mexopts.sh
# use empty MEXOPTS to use the default mexopts.sh in the current working
# directory, your .matlab directory or the globally installed one if
# both the others do not exist.


###########################################################################
#   and now specify the tools needed to build the library containing      #
#   the required parts of LAPACK and BLAS from source                     #
###########################################################################
FORTRAN   = g95
OPTS      = -funroll-all-loops -O3 -fPIC
NOOPT     = -fPIC

LOADER     = g95
LOADOPTS   = -O -fPIC
```

```
ARCHIVER = ar
ARCHIVERFLAGS = cr
RANLIB = ranlib

########################################################################
#  These are the libraries involved in the build                       #
########################################################################
RRQRLIB   = $(YOUR_WORK_DIR)/rrqr_acm/rrqr.a
LAPACKLIB = $(PATH_TO_LAPACK)/lapack.a
BLASLIB   = $(PATH_TO_LAPACK)/blas.a
TMGLIB    = $(PATH_TO_LAPACK)/tmglib.a
```

So mainly you have to specify the path to your `mex` binary, the Fortran compiler to use and the Paths to your RRQR, LAPACK, and BLAS libraries. The `-fPIC` flag comes in handy when switching between 32 and 64bit processors frequently. You may want to add more processor specific optimization flags to the `OPTS` variable if you are building for a special machine only. As an example, `gfortran` from the GCC suite starting in version 4.2 supports the `-march=native` flag that tries to figure out optimal settings for your processor automatically.

Note that the compiler specified in the `FORTRAN` variable needs to be binary compatible to the one specified in your `mexopts.sh`. To play it save you may want to use the same compiler and flags in both of them. Note further that these settings are only relevant in the `mexfromsrc` case (see below).

After providing the required information you can compile the MEXfile by simply calling

```
make
```

This will build the MEXfile, linking it against the specified LAPACK and BLAS libraries. The Makefile also provides a target `mexfile_so` that allows you to build a MEXfile linked against the LAPACK and BLAS libraries provided by MATLAB, as in the example from within MATLAB above. It is used by calling

```
make mexfile_so
```

A third target will compile only the required parts of LAPACK and BLAS (see files in `ffiles` subdirectory) from source and link them into the MEXfile. Note that these are not updated frequently and therefore may not include the most recent features of latest LAPACK and BLAS versions, so you may whish

to replace them by the latest files from http://www.netlib.org/lapack/ Run

```
make mexfromsrc
```

to use this compilation approach.

4.3 Compiling on Linux (64-bit)

Compiling on 64bit Linux essentially works as above. We just want to note, that the g95 compiler has shown much more reliable than the gfortran compiler. Especially the handling of integer types is more than tricky in gfortran. To get real 64bit integers one needs to specify the `-fdefault-integer-8` flag. This however breaks the compatibility with the `mwSize` datatype. Also note that the `rrqrGate` MEX-function does not meet the API description (we use `integer` instead of `integer*4`) due to problems with 64bit MEXcompilation.

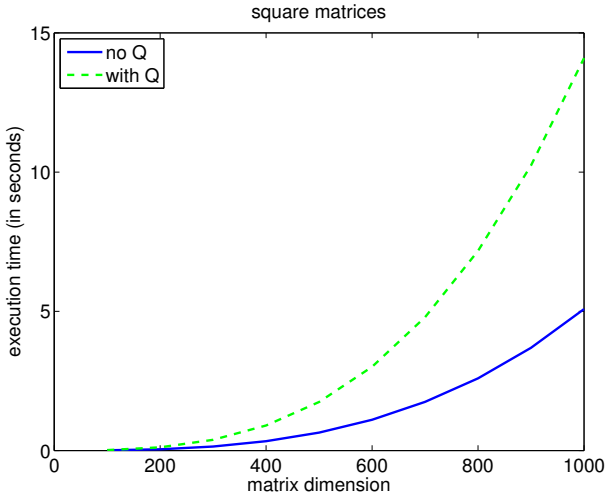
5 Measurements and Testing

The performance of the MEXfile has been measured in four test sequences on three different platforms. The test sequences show the scaling of the runtimes of the codes corresponding to rising dimension in four typical appearances. The first sequence tests square $n \times n$ matrices. The dimension here is increased from 100 to 1 000 in steps of 100. In the second sequence we tested tall and thin matrices, i.e., the number of rows m is larger than the number of columns n. In this case it is much more efficient with regard to memory consumption to compute the *economy size* QR decomposition, which has been performed throughout this test. Here we fixed $n = 100$ and varied m from 500 to 5 000 in steps of 250. As third test case we used the transposed situation, where $n \gg m$. These tests were carried out with $m = 100$ fixed and n varying from 1 000 to 50 000 in steps of 500. Since the last version is especially important in column compression techniques performed in iterative methods computing low rank solution factors of large scale matrix equations (see, e.g.[2]) a fourth sequence treats the scaling with increasing m in the case $n \gg m$. Here we fixed the number of columns to $n = 10\,000$ and m grew from 100 to 500 in steps of 25

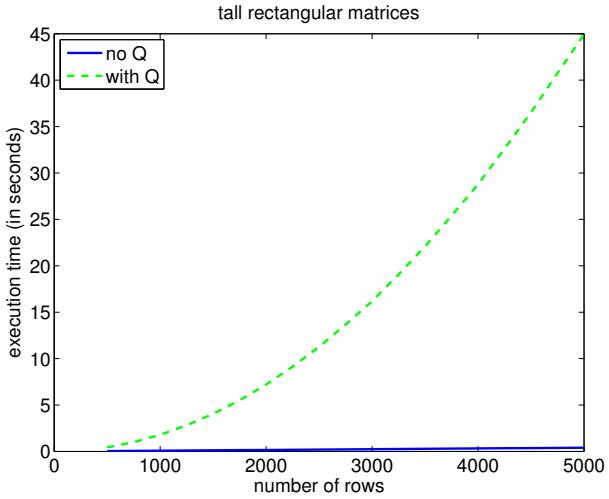
All test have been performed with the default truncation tolerance and repeated 100 times per chosen dimension. The test matrices were generated as

random matrices with an expected rank deficiency of 10 by generating them lacking the last 10 columns (tests 1 and 2) or rows (tests 3 and 4). The missing rows/columns have then been added as random linear combinations of the first 10 rows or columns, respectively. A new matrix of the corresponding size has been generated in each of the 100 repetitions per size. We can nicely observe the expected rates corresponding to the complexity of the algorithm. Only in the cases of the wide rectangular matrices we observe strange peaks (see Figures 3a, 6a and 9a). However judging from Figures 3b, 6b and 9b each of these seems to be produced by a single outlier, such that the standard behavior is again as theoretically expected.

5.1 Tests on Windows 32bit

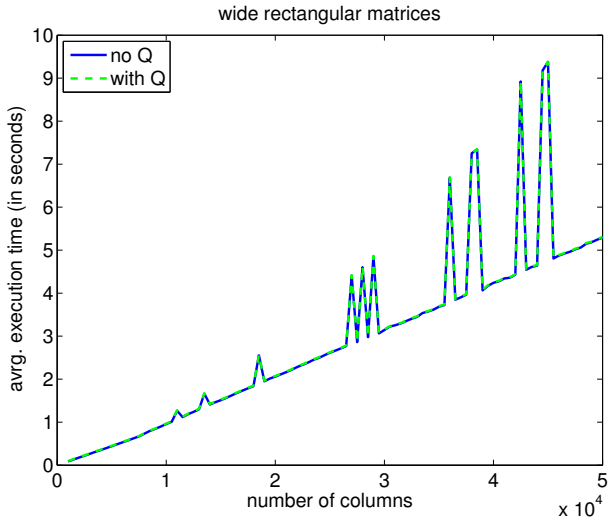


(a) Square matrices.

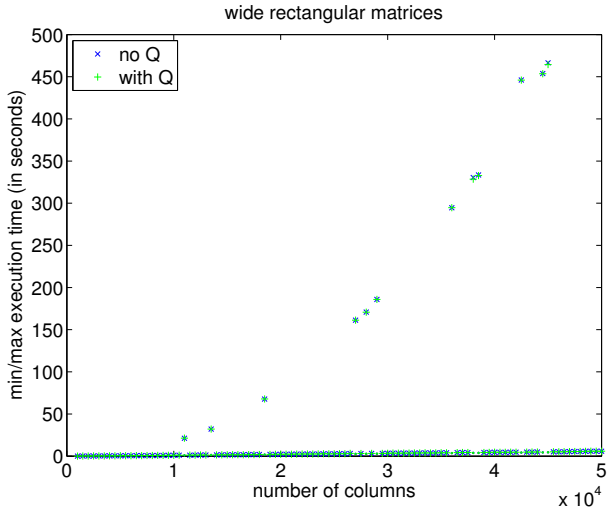


(b) Tall rectangular matrices.

Figure 2: 32bit Windows average execution time comparison for rrqr with and without accumulation of Q.

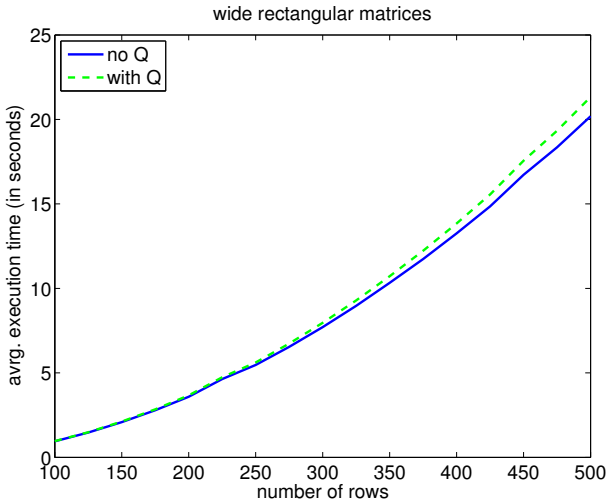


(a) Average times.

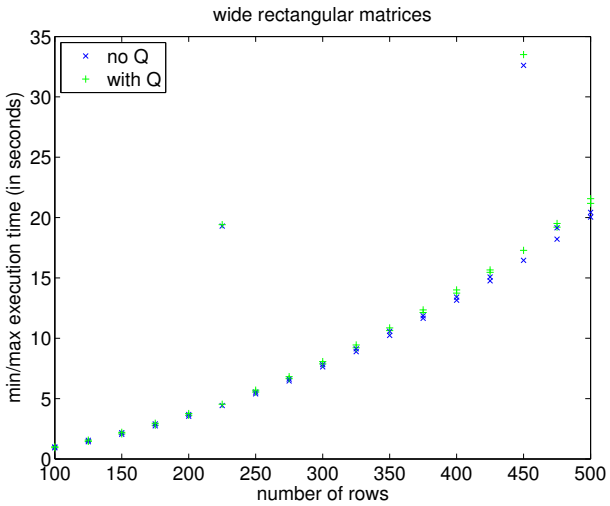


(b) Minimum and maximum times.

Figure 3: 32bit Windows execution time comparison for rrqr with and without accumulation of Q on wide rectangular matrices with growing number of columns.



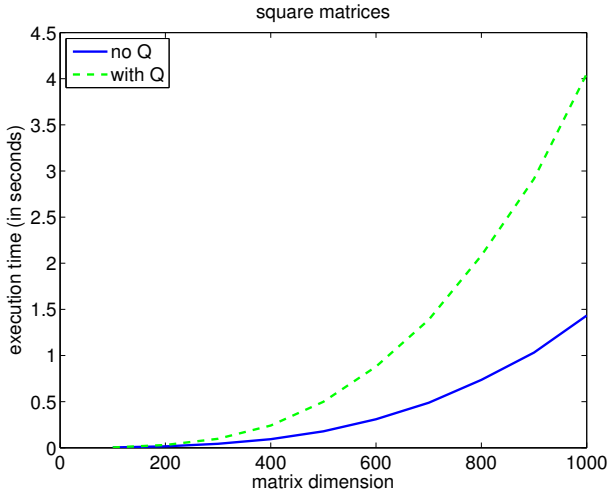
(a) Average times.



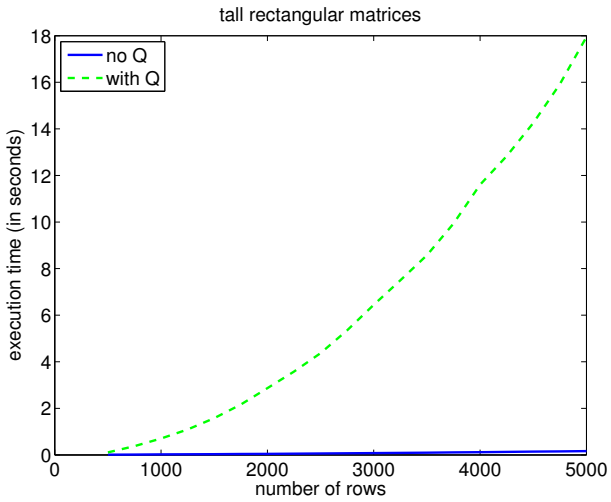
(b) Minimum and maximum times.

Figure 4: 32bit Windows execution time comparison for rrqr with and without accumulation of Q on wide rectangular matrices with growing number of rows.

5.2 Tests on Linux 32bit

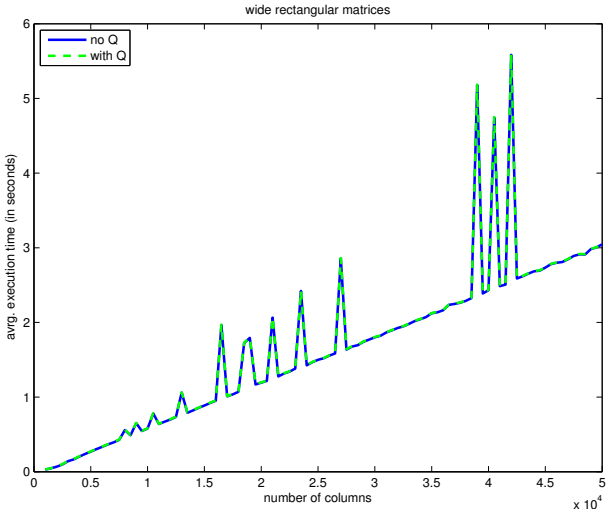


(a) Square matrices.

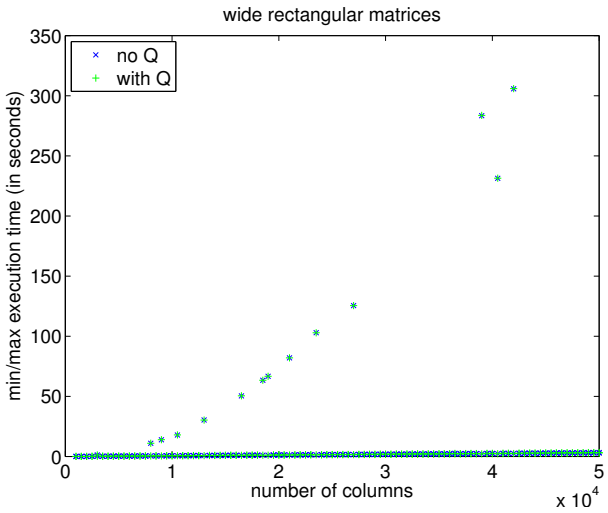


(b) Tall rectangular matrices.

Figure 5: 32bit Linux average execution time comparison for rrqr with and without accumulation of Q.

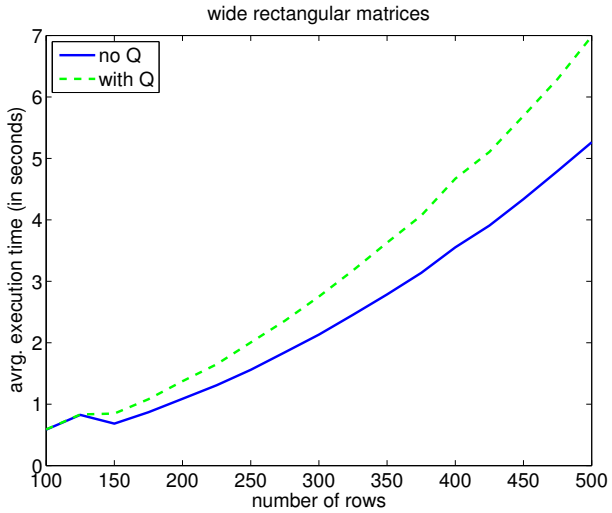


(a) Average times.

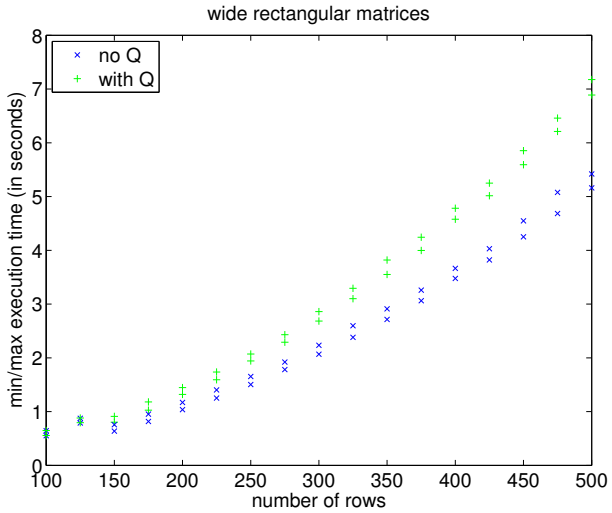


(b) Minimum and maximum times.

Figure 6: 32bit Linux execution time comparison for rrqr with and without accumulation of Q on wide rectangular matrices with growing number of columns.



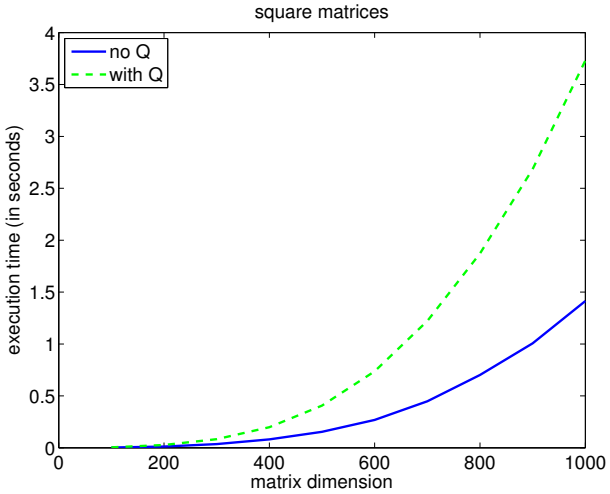
(a) Average times.



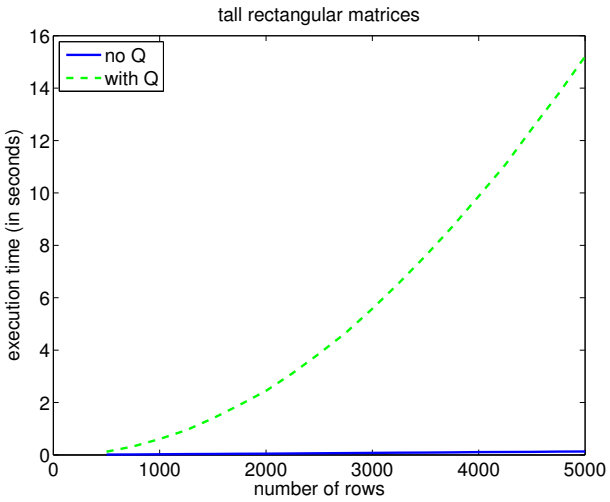
(b) Minimum and maximum times.

Figure 7: 32bit Linux execution time comparison for rrqr with and without accumulation of Q on wide rectangular matrices with growing number of rows.

5.3 Tests on Linux 64bit

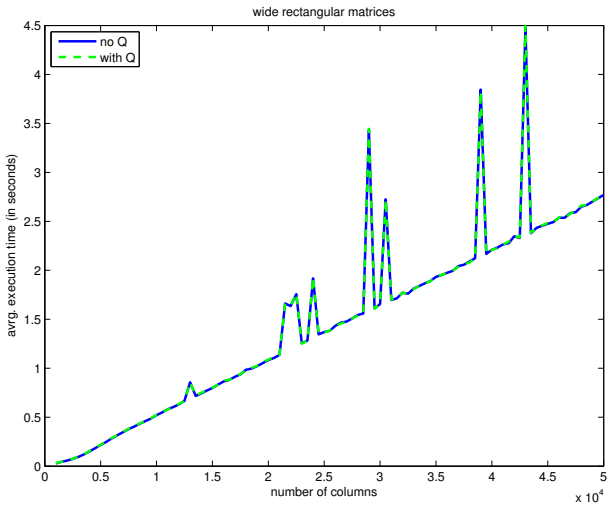


(a) Square matrices.

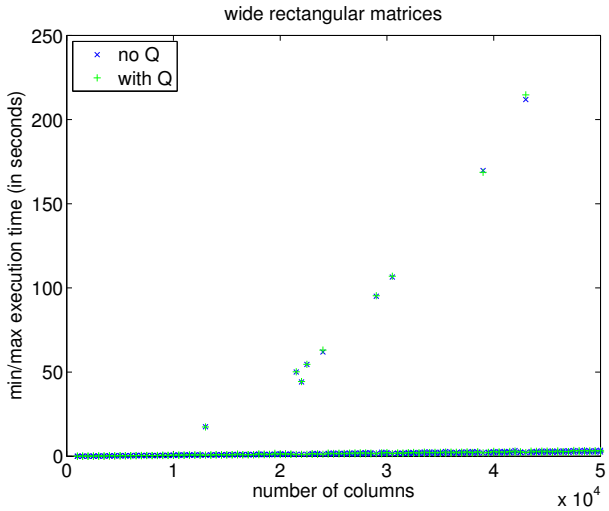


(b) Tall rectangular matrices.

Figure 8: 64bit Linux average execution time comparison for rrqr with and without accumulation of Q.

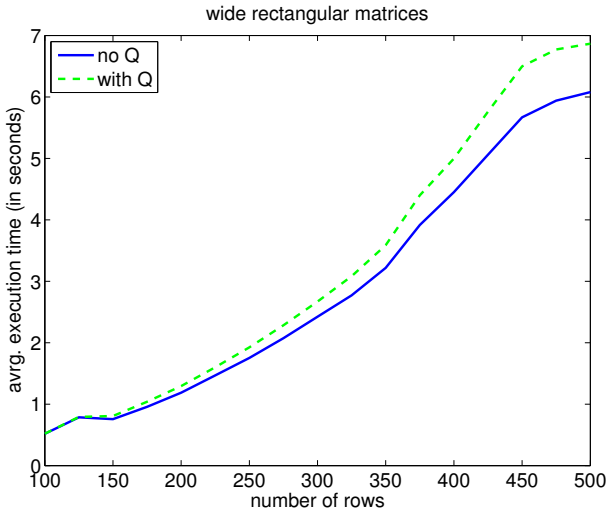


(a) Average times.

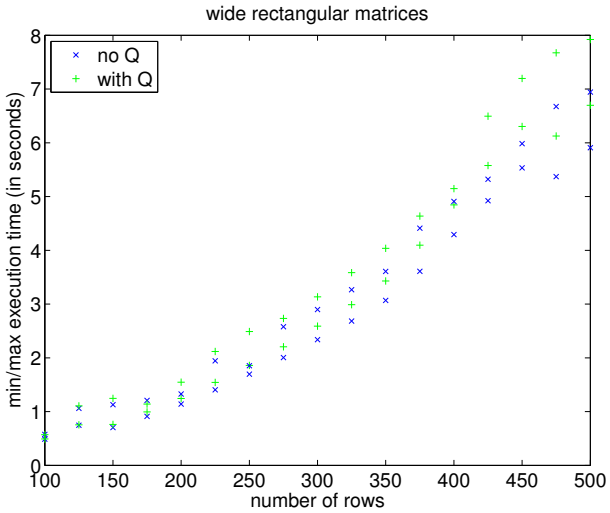


(b) Minimum and maximum times.

Figure 9: 64bit Linux execution time comparison for rrqr with and without accumulation of Q on wide rectangular matrices with growing number of columns.



(a) Average times.



(b) Minimum and maximum times.

Figure 10: 64bit Linux execution time comparison for rrqr with and without accumulation of Q on wide rectangular matrices with growing number of rows.

References

[1] C. H. Bischof and G. Quintana-Ortí, *Algorithm 782: Codes for rank-revealing QR factorizations of dense matrices.*, ACM Trans. Math. Softw., 24 (1998), pp. 254–257. 1

[2] J. Saak, *Efficient Numerical Solution of Large Scale Algebraic Matrix Equations in PDE Control and Model Order Reduction*, PhD thesis, TU Chemnitz, July 2009. available from http://www.tu-chemnitz.de/~saak/Data/Diss-web.pdf. 8

Chemnitz Scientific Computing Preprints – ISSN 1864-0087