Block-Partitioned Algorithms for Solving the Linear Least Squares Problem*

Gregorio Quintana-Ortí † Enrique S. Quintana-Ortí ‡

Antoine Petitet §

LAPACK Working Note # 113.

University of Tennessee at Knoxville Technical Report CS-96-333

Abstract.

The linear least squares problem arises in many areas of sciences and engineerings. When the coefficient matrix has full rank, the solution can be obtained in a fast way by using QR factorization with BLAS-3. In contrast, when the matrix is rank-deficient, or the rank is unknown, other slower methods should be applied: the SVD or the complete orthogonal decompositions. The SVD gives more reliable determination of rank but is computationally more expensive. On the other hand, the complete orthogonal decomposition is faster and in practice works well.

We present several new implementations for solving the linear least squares problem by means of the complete orthogonal decomposition that are faster than the algorithms currently included in LAPACK. Experimental comparison of our methods with the LAPACK implementations on a wide range of platforms (such as IBM RS/6000-370, SUN HyperSPARC, SGI R8000, DEC Alpha/AXP, HP 9000/715, etc.) show considerable performance improvements. Some of the new code has been already included in the latest release of LAPACK (3.0). In addition, for full-rank matrices the performances of the new methods are very close to the performance of the fast method based on QR factorization with BLAS-3, thus providing a valuable general tool for full-rank matrices and rank-deficient matrices, as well as those matrices with unknown rank.

Key words. linear least squares, complete orthogonal factorization, QR factorization with column pivoting, rank-revealing QR factorization, block algorithm.

1 Introduction

The Linear Least Squares (LLS) problem arises in many areas of science and engineering, for example, in geodesy [21], computer-aided design [23], nonlinear

*The two first authors were partially supported by the European ESPRIT Project # 9072–GEPPCOM and by the Spanish CICYT Proyect # TIC96-1062-C03-03.

†Departamento de Informática, Universidad Jaime I, Campus Penyeta Roja, 12071 Castellón, Spain, gquintan@inf.uji.es.

‡Same address as first author; quintana@inf.uji.es.

§Department of Computer Science, University of Tennessee at Knoxville, Tennessee, USA, petitet@cs.utk.edu.

least-squares problems [31], solution of integral equations [18], and calculation of splines [22].

Specifically, the LLS problem consists in finding the vector x that satisfies

$$\min_x \|Ax - b\|_2 \tag{1}$$

where A is an $m \times n$ coefficient matrix and b is a vector of n components.

Basically, depending on the properties of A, the LLS problem has a unique solution or infinite solutions. Consider $m \geq n$, then $Ax = b$ defines an overdetermined linear system (there are more equations than unknowns). If A has full rank ($\text{rank}(A) = n$), then there exists a unique solution to the LLS problem. Otherwise, if A is rank-deficient ($\text{rank}(A) < n$), there exist infinite solutions and the minimum 2-norm solution,

$$\min_x \|x\|_2, \tag{2}$$

is usually the required solution to the LLS problem.

On the other hand, when $m < n$, there exist infinite solutions or no solution to the underdetermined linear system $Ax = b$, but the LLS problem still has a unique minimum norm solution.

A more detailed description of the LLS problem and related issues about existence and uniqueness of solutions, sensitivity, etc., can be consulted in [12, 20,30,37].

Let us focus now on the existing LLS solvers. If A has full rank, a fast method based on the QR factorization with BLAS-3 can be applied to solve the problem. However, if A is rank-deficient or its rank is unknown, alternative methods must be applied. Therefore, no general fast method is available unless the matrix has full rank and it is known *a priori*. Currently, the two widely accepted numerical tools for solving the rank-deficient LLS problem are the Singular Value Decomposition (SVD) and the complete orthogonal decomposition [20]. The former is considered as the most reliable algorithm for solving the LLS problem but it presents an important drawback: its higher computational cost. In contrast, the complete orthogonal decomposition is based on a low-cost algorithm, the QR decomposition with column pivoting, hereafter QRP. It theoretically provides less reliable rank determination, yet it usually performs well in practice. The LAPACK library provides driver subroutines for both approaches [1].

In this paper we present several faster algorithms for solving the LLS problem. We have developed code that has been included in LAPACK release 3.0. The new methods are complete block-partitioned algorithms, that is, the main computations are block oriented and implemented by using BLAS-3.

Our new algorithms can be categorized into two different types. The first type is based on a new BLAS-3 implementation of the QRP developed by G. Quintana-Ortí, X. Sun and C. H. Bischof [34]. The second type relies on a windowed version of the QRP developed by C. H. Bischof and G. Quintana-Ortí [10,11].

Our new LLS methods are faster than the existing methods on rank-deficient matrices and, when applied to full-rank matrices, they perform very close to the LLS solver for full-rank matrices. Therefore, these methods can be successfully applied to all types of matrices and achieve good performance.

The organization of the paper is as follows. In section 2 we briefly describe the LLS solvers currently included in LAPACK. In section 3 we introduce the basic tools for our algorithms (BLAS-3 QRP and windowed QRP); then, we present the new LLS solvers and outline their advantages. In section 4 the results of the experimental study are reported. Finally, section 5 contains the concluding remarks.

2 LLS solvers available in LAPACK

Currently, the LAPACK library provides three different driver subroutines for solving the LLS problem: `xGELS`, `xGELSX`, and `xGELSS`.

Subroutine `xGELS` solves (1) when $\mathrm{rank}(A) = \min(m, n)$, that is, when A has full rank. If $m > n$, it computes the unique solution of the over-determined linear system. On the other hand, if $m < n$, then it computes the minimum 2-norm solution of the undetermined problem. `xGELS` is based on the QR factorization with BLAS-3, thus providing a high-speed computational tool.

When $\mathrm{rank}(A) < \min(m, n)$, subroutines `xGELSX` and `xGELSS` compute the solution that minimizes both $\|Ax - b\|_2$ and $\|x\|_2$. The first subroutine is based on the complete orthogonal factorization that uses the QR factorization with column pivoting, whereas the second one is based on the SVD. `xGELSX` is usually much faster than `xGELSS`, though it is completely based on BLAS-2.

All the above mentioned subroutines allow the computation of several solutions $x_1, x_2, \ldots, x_k$ to different problems defined by $b_1, b_2, \ldots, b_k$ with the same coefficient matrix in just one call. It should be noted that this process is different from minimizing $\|AX - B\|_2$, where $X = [x_1, x_2, \ldots, x_k]$ and $B = [b_1, b_2, \ldots, b_k]$.

Since our algorithms also compute a complete orthogonal factorization, next we will describe subroutine `xGELSX`.

2.1 Subroutine `xGELSX`

The computational tasks carried out by this subroutine are the following:

1. Matrices A and B are scaled (if necessary) to avoid overflow and underflow.

2. The QR factorization with column pivoting of the matrix A is computed: $AP = QR$.

3. From the triangular matrix R, the numerical rank r of A is obtained using the incremental condition estimator ICE [5, 6]. This value defines the following partition of R:

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

   where R_{11} is an $r \times r$ well-conditioned upper triangular matrix.

4. R_{12} is zeroed applying unitary transformations from the right defined by Y: $(R_{11}, R_{12}) = (T_{11}, 0)Y$.

5. The unitary transformations of stage 2 are also applied to B from the left: $Q^H B$.

6. $T_{11}^{-1}(Q^H B)$ is computed. Specifically, this stage consists in solving an upper triangular linear system with a possibly multiple right-hand side matrix.

7. Y^H is applied to the previous result: $Y^H(T_{11}^{-1}Q^H B)$.

8. The solution X is obtained by applying the permutation matrix P to the result of the previous stage: $X = P(Y^H T_{11}^{-1} Q^H B)$.

9. A and X are unscaled (if necessary).

3 New algorithms for the LLS problem

In this section we first describe some key tools of our LLS solvers: a BLAS-3 version of the QRP, and two windowed pivoting versions of the QRP. Then we outline the main advantages of our LLS solvers.

3.1 BLAS-3 QR factorization with column pivoting (`xGEQP3`)

This algorithm, developed by G. Quintana-Ortí, X. Sun and C. H. Bischof [34] and called **`xGEQP3`**, is a BLAS-3 version of QRP with considerable performance improvements over LINPACK subroutine **`xQRDC`** (based on BLAS-1) and LAPACK subroutine **`xGEQPF`** (based on BLAS-2), while maintaining the same numerical behavior as those implementations.

For each column, both LINPACK and LAPACK implementations select one column, permute it, compute the reflector that zeroes its components under the matrix diagonal, and apply it to the rest of the matrix. LINPACK code performs the update by means of BLAS-1, whereas LAPACK code performs the update by means of BLAS-2. In contrast, for each stage the new method only updates one column and one row of the rest of the matrix (since that is the only information needed for the next pivoting phase). Thus, the update of the rest of the matrix is delayed until nb columns have been processed and therefore nb reflectors are available. This delay allows the use of BLAS-3 kernels, thus obtaining a faster execution speed.

3.2 New block-algorithms for computing rank-revealing QR (RRQR) factorizations `xGEQPX` and `xGEQPY`

These two new methods, developed by C. H. Bischof and G. Quintana-Ortí [10,11], are based on a faster approach.

Both algorithms consist of two stages: preprocessing and postprocessing. The first stage is an efficient block-oriented algorithm for computing an approximate RRQR factorization. Basically, it is a windowed version of the QRP, based on BLAS-3 and monitorized by Bischof's incremental condition estimation (ICE) [5, 6]. The second stage is an efficient implementation of RRQR algorithms well-suited for triangular matrices. Subroutine **`xGEQPX`** includes a variant of S. Chandrasekaran and I. Ipsen's algorithm [14] with improvements with respect to condition estimation, termination criteria and Givens update. A theoretical study as well as the description of this postprocessing algorithm can be found in [35]. Subroutine **`xGEQPY`** includes a variant of C. T. Pan and P. T. P. Tang's algorithm [32] with similar improvements.

The experimental study in [10] shows that the performances of the these two new algorithms are usually within 15% of the performance of QR factorization (LAPACK **`xGEQRF`**) but 2 to 3 times faster than the QR factorization with column pivoting (LAPACK **`xGEQPF`**).

3.3 New LLS solvers

Using the above mentioned algorithms we have developed the following three LLS solvers:

`xGELSY`: One of the main differences of this solver is that it performs stage 2 (the QR factorization with column pivoting of the coefficient matrix) by means of subroutine **`xGEQP3`**.

`xGELSA`: This algorithm performs stage 2 by using subroutine **`xGEQPX`**.

`xGELSB`: This algorithm performs stage 2 by using subroutine **`xGEQPY`**.

The advantages of our new methods over the current implementations are the following:

Faster QRP decomposition: (Step 2 in algorithm **`xGELSX`**). All the new solvers use BLAS-3 to compute the QRP factorization of the coefficient matrix. Experience of previous experimental studies shows that **`xGEQP3`**, **`xGEQPX`**, and **`xGEQPY`** are much faster than LINPACK and LAPACK QRP [10,34].

Faster annihilation of R_{12}: (Step 4). LAPACK subroutine `xGELSX` nullifies R_{12} from the right by using BLAS-2 code, while the three new drivers use BLAS-3.

Faster update of B: (Steps 5 and 7). LAPACK subroutine `xGELSX` updates matrix B by means of BLAS-2 code (subroutine `xORM2R` for stage 5 and `xLATZM` for stage 7), whereas the new drivers use BLAS-3 code in both cases. Solver `xGELSY` uses subroutine `xORMQR` for stage 5 and `xORMRZ` for stage 7). On the other hand, solvers `xGELSA` and `xGELSB` do not use subroutine `xORMQR` since that update is carried out while matrix A is being triangularized and, therefore, the same block reflectors are used in both tasks.

Faster permutation: (Step 8). The original code in LAPACK uses a floating point vector as workspace to control the components that have been permuted. Our new algorithms use this vector in a faster and more simple way: each column to be permuted is copied to this vector and then it is moved to its proper position. The new method requires the same workspace and is faster since no comparisons between floating point numbers are required.

All these advantages provide faster LLS solvers. The important improvements in the execution speed of our new algorithms are due to the fact that all stages in the process have been redesigned to use BLAS-3 and are block-oriented. Thus, we expect that the improvement of the new drivers will be similar to that obtained when migrating an application from BLAS-2 to BLAS-3.

Single, double, complex, and double complex code for `xGELSY`, `xGELSA`, and `xGELSB` have been developed.

4 Experimental Results

We report in this section the experimental results comparing the double precision codes `DGELS`, `DGELSX`, and `DGELSS` from LAPACK, and our new solvers `DGELSY`, `DGELSA` and `DGELSB`. The tests included a wide range of platforms: IBM RS/6000-370, SUN HyperSPARC @ 150MHz, SGI MIPS R8000 @ 90MHz, DEC Alpha/AXP, and HP 9000/715. In each case, we used the vendor-supplied BLAS (ESSL, Performance Library, SGIMATH, DXML, and Blaslib, respectively).

We only present the results on the IBM and the SUN since similar results were obtained on the other platforms.

We generated 18 different matrix types to evaluate the algorithms, with various singular value distributions and numerical rank ranging from 3 to full rank. Details of the test matrix generation are beyond the scope of this paper, and we give only a brief synopsis here.

Test matrices 1 through 5 were designed to exercise column pivoting. Matrix 6 was designed to test the behavior of the condition estimation in the presence

Table 1: Test Matrix Types ($p = \min(m, n)$).

N#	Description	r
1	Matrix with rank $p/2 - 1$	$p/2 - 1$
2	$A(:, 2:p)$ has full rank, $\mathcal{R}(A) = \mathcal{R}(A(:, 2:p))$	$p - 1$
3	Full rank	p
4	$A(:, 1:3)$ small in norm, $A(:, 4:n)$ of full rank	$p - 3$
5	$A(:, 1:3)$ small in norm, $\mathcal{R}(A) = \mathcal{R}(A(:, 1:3))$	3
6	5 smallest singular values clustered	p
7	Break1 distribution	$p/2 + 1$
8	Reversed break1 distribution	$p/2 + 1$
9	Geometric distribution	$p/2 + 1$
10	Reversed geometric distribution	$p/2 + 1$
11	Arithmetic distribution	$p/2 + 1$
12	Reversed arithmetic distribution	$p/2 + 1$
13	Break1 distribution	$p - 1$
14	Reversed break1 distribution	$p - 1$
15	Geometric distribution	$3p/4 + 1$
16	Reversed geometric distribution	$3p/4 + 1$
17	Arithmetic distribution	$p - 1$
18	Reversed arithmetic distribution	$p - 1$

of clusters for the smallest singular value. For the other cases, we employed the LAPACK matrix generator `xLATMS`, which generates random symmetric matrices by multiplying a diagonal matrix with prescribed singular values by random orthogonal matrices from the left and right. For the break1 distribution, all singular values are 1.0 except for one. In the arithmetic and geometric distributions, they decay from 1.0 to a specified smallest singular value in an arithmetic and geometric fashion, respectively. In the "reversed" distributions, the order of the diagonal entries was reversed. For test cases 7 though 12, we used `xLATMS` to generate a matrix of order $\frac{n}{2} + 1$ with smallest singular value 5.0e-4, and then interspersed random linear combinations of these "full-rank" columns to pad the matrix to order n. For test cases 13 through 18, we used `xLATMS` to generate matrices of order n with the smallest singular value being 2.0e-7. We believe this set to be representative of matrices that can be encountered in practice.

4.1 Computing performance

In all the figures, we employ the solid line for the performance of `DGELSX`, the dotted line with symbol "+" for `DGELSY`, the dotted line with symbol "x"

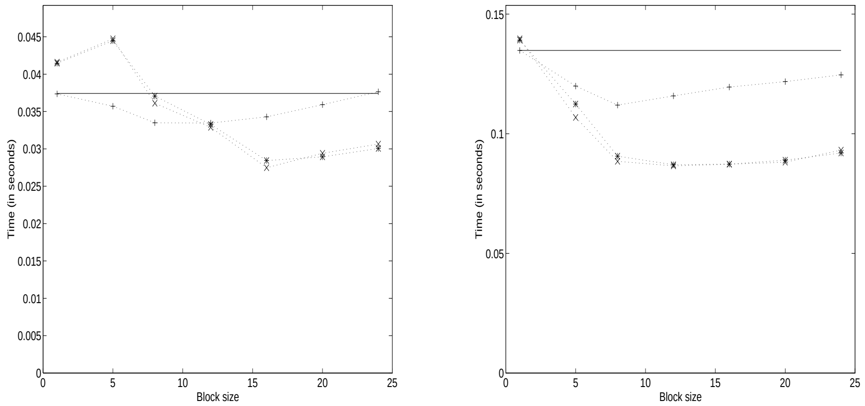

Figure 1: Average Performance (in seconds) versus Block Size on the IBM RS/6000-370 for 150×150 (left) and 300×150 matrices (right).

for **DGELSA**, and the dotted line with symbol "*" for **DGELSB**.

In figures 1 through 4 (IBM), and figures 5 through 8 (SUN), we present the results on square and rectangular matrices of size 150, 250, 300, 500, and 1000 using a block size (nb) of 1, 5, 8, 12, 16, 20, and 24. These figures show the average performances of algorithms **DGELSX**, **DGELSY**, **DGELSA**, and **DGELSB** on the 18 matrix types versus the block size.

Let us focus now on the difference between our new LLS solvers and **DGELSX**. The latter solver is based on BLAS-2 and it is not block-oriented. Therefore, its execution time, as shown in all the figures, is not affected by the block size at all. The behavior of our LLS solvers is very different. Since all of them are mainly based on BLAS-3 subroutines, the new solvers perform better than **DGELSX** in all cases except for very small matrices. **DGELSA** and **DGELSB** obtain very similar execution times since they differ only in the post-processing stage and it has very little influence on the overall execution time of the algorithm. Besides, both solvers achieve better results than **DGELSY**.

Figure 9 does not show the average performance, but the exact behavior of the solvers on every one of the 18 matrix types for block size 20. We only present the required seconds for 1000×500 on the SUN. The plot shows that in some cases the new solvers are up to 3 times faster.

Table 2 compares all the solvers in LAPACK and our new solvers. In order to compare **DGELS**, the tested matrices have full rank. The solver based on the SVD, **DGELSS**, is much slower than the others. The new solvers **DGELSY**, **DGELSA**, and **DGELSB** obtain performances much higher than the LAPACK solver **DGELSX**

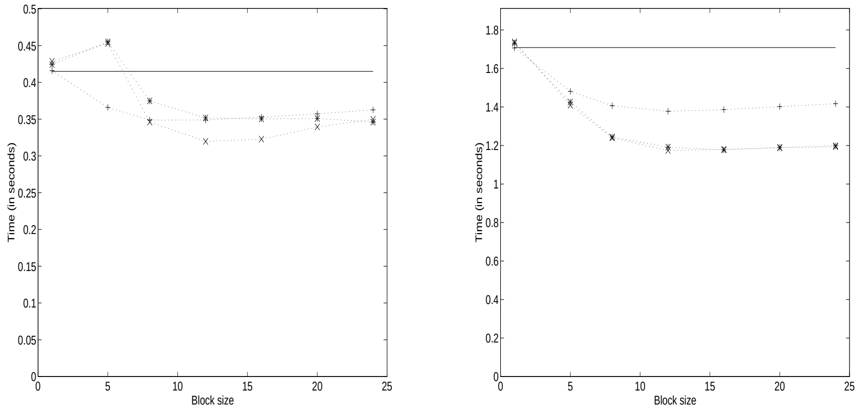

Figure 2: Average Performance (in seconds) versus Block Size on the IBM RS/6000-370 for 250×250 (left) and 500×250 matrices (right).

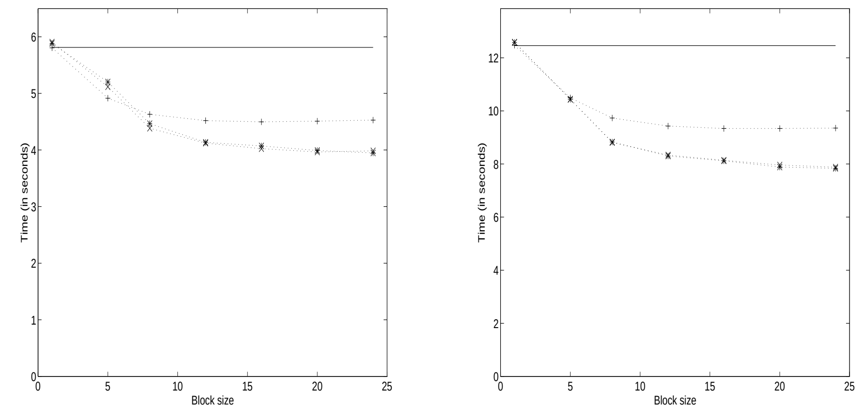

Figure 3: Average Performance (in seconds) versus Block Size on the IBM RS/6000-370 for 500×500 (left) and 1000×500 matrices (right).

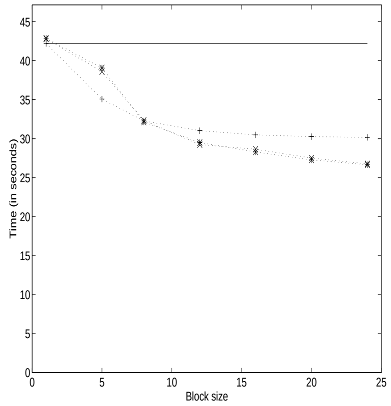

Figure 4: Average Performance (in seconds) versus Block Size on the IBM RS/6000-370 for 1000×1000 matrices.

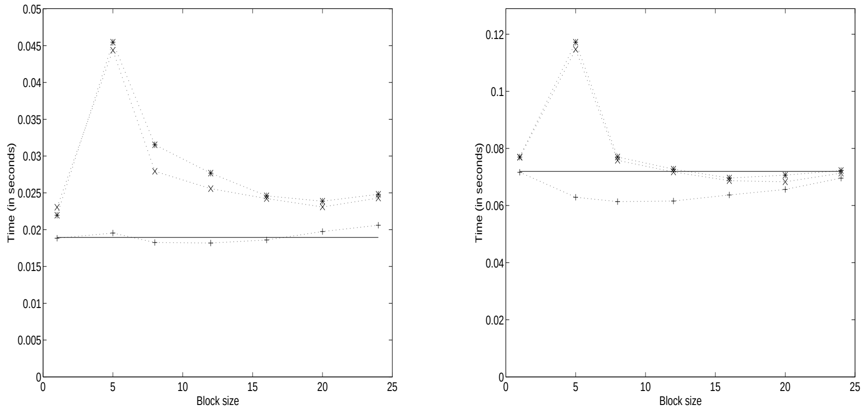

Figure 5: Average Performance (in seconds) versus Block Size on the SUN Hypersparc for 150×150 (left) and 300×150 matrices (right).

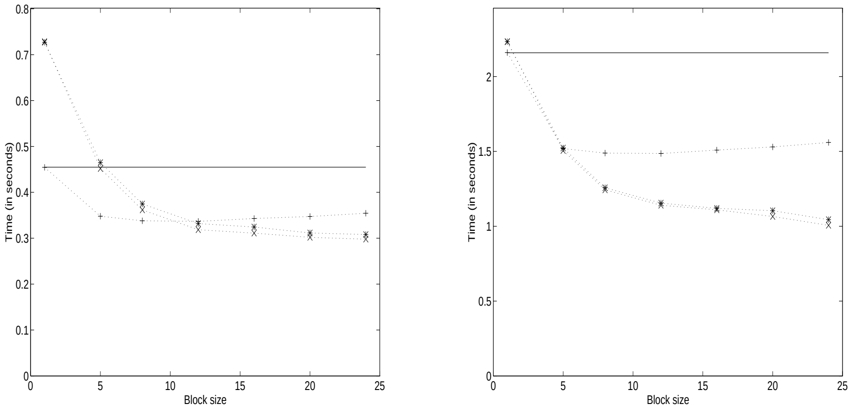

Figure 6: Average Performance (in seconds) versus Block Size on the SUN Hypersparc for 250×250 (left) and 500×250 matrices (right).

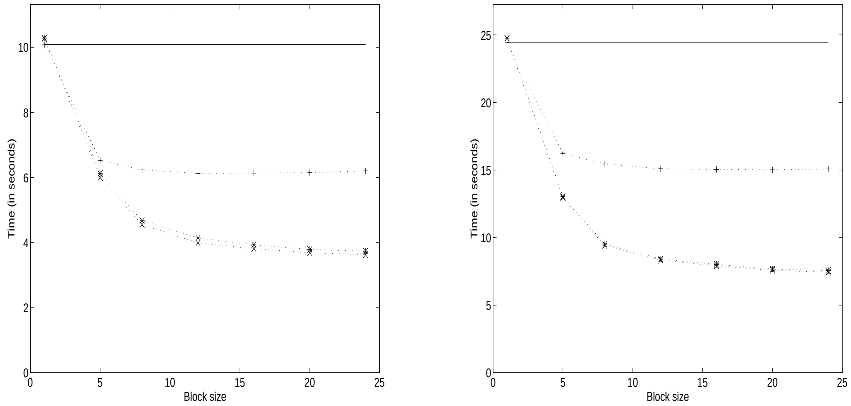

Figure 7: Average Performance (in seconds) versus Block Size on the SUN Hypersparc for 500×500 (left) and 1000×500 matrices (right).

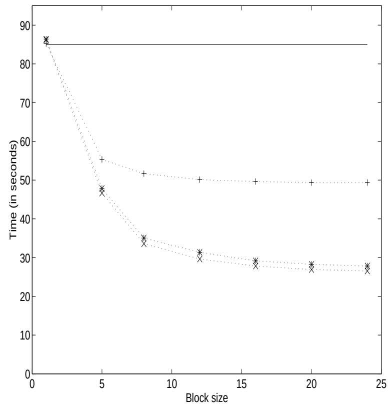

Figure 8: Average Performance (in seconds) versus Block Size on the SUN Hypersparc for 1000×1000 matrices.

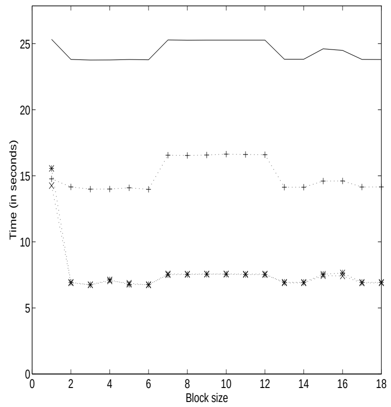

Figure 9: Performance (in seconds) versus Matrix Type on the SUN Hypersparc for 1000×500 matrices.

Table 2: Performance (in seconds) on several full-rank matrix sizes on the SUN Hypersparc with block size 20.

Matrix Size	DGELSS	DGELSX	DGELSY	DGELSA	DGELSB	DGELS
150×150	1.30	0.016	0.013	0.013	0.013	0.011
300×150	1.60	0.068	0.048	0.048	0.048	0.040
250×250	6.80	0.43	0.31	0.15	0.15	0.13
500×250	8.64	2.10	1.44	0.97	0.97	0.41
500×500	57.13	9.42	5.63	2.94	2.92	2.53
1000×500	74.48	23.76	13.99	6.75	6.75	5.88
1000×1000	456.30	77.75	44.93	21.14	21.07	18.38

and very close to those of the solver for full-rank matrices based on BLAS-3, `DGELS`.

4.2 Numerical accuracy

We have conducted several experiments to evaluate the accuracy of the new algorithms on different matrix types and sizes. Some of the tests were obtained from the LAPACK Test Suite. We have computed the following residuals and tests:

- $\|B - AX\|/(\max(n,m)\|A\|\|X\|\epsilon)$, where ϵ is the precision machine.
- $\|R^T A\|/(\|A\|\|B\|\max(m,n,k)\epsilon)$, where k is the number of right hand sides.
- The norm of the trailing block of the triangular factor of the QR factorization with column pivoting of matrix $[A, X]$.
- The norm of the subtraction of the singular values of R and the singular values of A.

The new algorithms obtained results very similar to those of **`DGELSX`**. The solver **`DGELSB`** gave a higher residual in a few cases because it did not reveal the numerical rank, though it revealed a well conditioned R_{11}.

5 Concluding Remarks

We have developed three new solvers based on BLAS-3 and block oriented for solving the linear least squares problem. The new subroutines perform much faster than the LAPACK code for rank-deficient matrices and very close to the LAPACK code for full-rank matrices. Some of the drivers have already been included in the latest release of LAPACK.

While maintaining the numerical behavior of the subroutine `xGELSX`, new subroutine `xGELSY` consistently outperforms it. Subroutines `xGELSA` and `xGELSA` are based on a different approach that allows an even higher data locality. These subroutines are usually even faster than the above mentioned `xGELSY` and LAPACK `xGELSX`.

Although not reported, similar performance results were obtained on DEC Alpha/AXP and HP 9000/715 platforms.

Acknowledgments

We express our gratitude to Christian H. Bischof, Xiaobai Sun, Susan Blackford, and Sven Hammarling for their interesting suggestions, ideas, and discussions.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide Release 2.0*, SIAM, Philadelphia, 1994.

[2] Z. Bai and J. Demmel, *Design of a parallel nonsymmetric eigenroutine toolbox, Part I*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. F. S. *et al*, ed., SIAM, 1993, pp. 391–398.

[3] C. H. Bischof, *A block QR factorization algorithm using restricted pivoting*, in Proceedings SUPERCOMPUTING '89, Baltimore, Md., 1989, ACM Press, pp. 248–256.

[4] ——, *A parallel QR factorization algorithm with controlled local pivoting*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 36–57.

[5] ——, *Incremental Condition Estimator*, SIAM Journal on Matrix Analysis and Applications, vol. 11 no. 2 (1990), pp. 312–322.

[6] C. H. Bischof and P. T. P. Tang, *A robust incremental condition scheme*, Argonne preprint MCS-P225-0391, Mathematics and Computer Science Division, Argonne National Laboratory, 1991.

[7] C. H. Bischof and P. C. Hansen, *Structure-preserving and rank-revealing QR factorizations*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 1332–1350.

[8] ——, *A block algorithm for computing rank-revealing QR factorizations*, Numerical Algorithms, 2 (1992), pp. 371–392.

[9] C. H. Bischof and G. M. Shroff, *On updating signal subspaces*, IEEE Transactions on Signal Processing, 40 (1992), pp. 96–105.

[10] C. H. Bischof and G. Quintana-Ortí, *Computing rank-revealing QR factorizations of dense matrices*, Argonne preprint MCS-P559-0196, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[11] ——, *Codes for rank-revealing QR factorizations of dense matrices*, Argonne preprint MCS-P560-0196, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[12] Å. Björk, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, PA, USA, 1996, ISBN: 0-89871-360-9.

[13] T. F. Chan, *Rank revealing QR factorizations*, Linear Algebra and Its Applications, 88/89 (1987), pp. 67–82.

[14] S. Chandrasekaran and I. Ipsen, *On rank-revealing QR factorizations*, SIAM Journal on Matrix Analysis and Applications, 15 (1994).

[15] J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[16] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM Press, Philadelphia, 1979.

[17] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson, *An extended set of Fortran basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 14 (1988), pp. 1–17.

[18] L. Eldén and R. Schreiber, *An application of systolic arrays to linear discrete ill-posed problems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 892–903.

[19] G. H. Golub, *Numerical methods for solving linear least squares problems*, Numerische Mathematik, 7, pp. 206–216, 1965.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 2nd ed., 1989.

[21] G. H. Golub, P. Manneback, and P. L. Toint, *A comparison between some direct and iterative methods for certain large scale geodetic least-squares problem*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 799–816.

[22] T. A. GRANDINE, *An iterative method for computing multivariate C^1 piecewise polynomial interpolants*, Computer Aided Geometric Design, 4 (1987), pp. 307–319.

[23] ——, *Rank deficient interpolation and optimal design: An example*, Tech. Report SCA–TR–113, Boeing Computer Services, Engineering and Scientific Services Division, February 1989.

[24] M. GU AND S. EISENSTAT, *An efficient algorithm for computing a strong rank-revealing factorization*, Tech. Report YALEU/DCS/RR-967, Yale University, Department of Computer Science, 1994.

[25] P. C. HANSEN, *Truncated SVD solutions to discrete ill-posed problems with ill-determined numerical rank*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 503–518.

[26] P. C. HANSEN, S. TAKISHI, AND S. HIROMOTO, *The Modified Truncated SVD-Method for Regularization in General Form*, SIAM Journal on Scientific and Statistical Computing, 13 (1991), pp. 1142–1150.

[27] Y. P. HONG AND C.-T. PAN, *The rank-revealing QR decomposition and SVD*, Mathematics of Computation, 58 (1992), pp. 213–232.

[28] S. F. HSIEH, K. J. R. LIU, AND K. YAO, *Comparisons of truncated QR and SVD methods for AR spectral estimations*, in SVD and Signal Processing II, R. J. Vaccaro, ed., Elsevier Science Publishers, 1991, pp. 403–418.

[29] C. L. LAWSON, R. J. HANSON, R. J. KINCAID, AND F. T. KROGH, *Basic linear algebra subprograms for Fortran usage*, ACM Transactions on Mathematical Software, 5 (1979), pp. 308–323.

[30] C. L. LAWSON, R. J. HANSON, *Solving Least Squares Problems*, Classics in Applied Mathematics, 15. SIAM, Philadelphia, PA, USA, 1995, ISBN: 0-89871-356-0.

[31] J. MORÉ, *The Levenberg-Marquardt algorithm: Implementation and theory*, in Proceedings of the Dundee Conference on Numerical Analysis, G. A. Watson, ed., Berlin, 1978, Springer-Verlag.

[32] C.-T. PAN AND P. T. P. TANG, *Bounds on singular values revealed by QR factorizaton*, Tech. Report MCS-P332-1092, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[33] D. PIERCE AND J. G. LEWIS, *Sparse rank-revealing QR factorization*, Tech. Report MEA-TR-193, Boeing Computer Services, Engineering and Scientific Services Division, 1992.

[34] G. Quintana-Ortí, X. Sun, and C. H. Bischof, *A BLAS-3 version of the QR factorizaton with column pivoting*, Tech. Report MCS-P551-1295, Mathematics and Computer Science Division, Argonne National Laboratory, 1995.

[35] G. Quintana-Ortí and E. S. Quintana-Ortí, *Guaranteeing termination of Chandrasekaran & Ipsen's algorithm for computing rank-revealing QR factorizations*, Tech. Report MCS-P564-0196, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[36] R. Schreiber and C. F. Van Loan, *A storage efficient WY representation for products of Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 53–57.

[37] G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[38] B. Waldén, *Using a Fast Signal Processor to Solve the Inverse Kinematic Problem with Special Emphasis on the Singularity Problem*, Ph. D. thesis, Linköping University, Dept. of Mathematics, 1991.