

LAPACK Working Note 35

Implementation Guide for LAPACK *

Edward Anderson[†], Jack Dongarra, and Susan Ostrouchov
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-1301

August 9, 1991

Abstract

This working note describes how to install, test, and time the third and final test release of LAPACK, a linear algebra package for high-performance computers. Separate instructions are provided for the Unix and non-Unix versions of the test package. Further details are also given on the design of the test and timing programs.

1 Introduction

LAPACK is planned to be a linear algebra library for high-performance computers. The library will include Fortran 77 subroutines for the analysis and solution of systems of simultaneous linear algebraic equations, linear least-squares problems, and matrix eigenvalue problems. Our approach to achieving high efficiency is based on the use of a standard set of Basic Linear Algebra Subprograms (the BLAS), which can be optimized for each computing environment. By confining most of the computational work to the BLAS, the subroutines should be transportable and efficient across a wide range of computers.

This working note describes how to install, test, and time the third and final test release of LAPACK. This release is being made available only to our test sites and is intended only for testing, and not for general distribution. After we receive the results from our test sites and make any necessary corrections, we will make the LAPACK routines available to the public. We do not expect any major changes to the software in this release before the public release, but this software should still be regarded as a preliminary version.

The instructions for installing, testing, and timing are designed for a person whose responsibility is the maintenance of a mathematical software library. We assume the installer has experience in compiling and running Fortran programs and in creating object libraries. The installation process involves reading the tape, creating a set of libraries, and compiling and running the test and timing programs.

*This work was supported by NSF Grant No. ASC-8715728.

[†]Current address: Cray Research Inc., 655F Lone Oak Drive, Eagan, MN55121

This guide combines the instructions for the Unix and non-Unix versions of the LAPACK test package (the non-Unix version is in Appendix F).

Section 2 describes how the files are organized on the tape, and Section 3 gives a general overview of the parts of the test package. Step-by-step instructions appear in Section 4 for the Unix version and in the appendix for the non-Unix version.

For users desiring additional information, Sections 5 and 6 give details of the test and timing programs and their input files. Appendices A and B briefly describe the LAPACK routines and auxiliary routines provided in this release. Appendix C lists the operation counts we have computed for the BLAS and for some of the LAPACK routines. Appendix D entitled "Caveats", is a compendium of the known problems from our own experiences, with suggestions on how to overcome them. Appendix E contains the execution times of the different test and timing runs on two sample machines. Appendix F contains the instructions to install LAPACK on a non-Unix system.

Release 3 of LAPACK includes updates of all of the software from Release 2, with the following additions:

- Driver routines for solving systems of linear equations, for solving least squares problems, for computing some or all eigenvalues and/or eigenvectors of a matrix, and for computing the SVD; test code for all the driver routines is also included with this package.
- New functionality for QR, which now includes the QR, RQ, LQ, and QL factorizations and also QR with pivoting; the matrix to be factored may be m by n with no restrictions on the relative sizes of m and n .
- New functionality for the reduction to specialized forms for eigenvalue computations, including a block algorithm for reduction to bidiagonal form; provision for upper or lower triangular storage of a symmetric matrix in the reduction to tridiagonal form and provision for packed storage of a symmetric matrix.
- A complete set of subroutines to generate an orthogonal matrix from a sequence of elementary transformations or to multiply a matrix C by an orthogonal matrix given as a sequence of elementary transformations, using all the possible storage schemes for the orthogonal matrix Q from the orthogonal factorization and reduction routines.
- Additional techniques for computing eigenvalues, including bisection and inverse iteration for the symmetric eigenvalue problem; a block multi-shift QZ algorithm for the generalized nonsymmetric eigenvalue problem; and a subroutine to find the eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix by performing a Cholesky factorization followed by a high-accuracy method for finding the eigenvalues of the bidiagonal factor.
- Software for the generalized symmetric eigenvalue problem and the generalized nonsymmetric eigenvalue problem.
- Subroutines for solving triangular and tridiagonal linear systems, and for computing and applying the scaling factors to equilibrate a matrix.

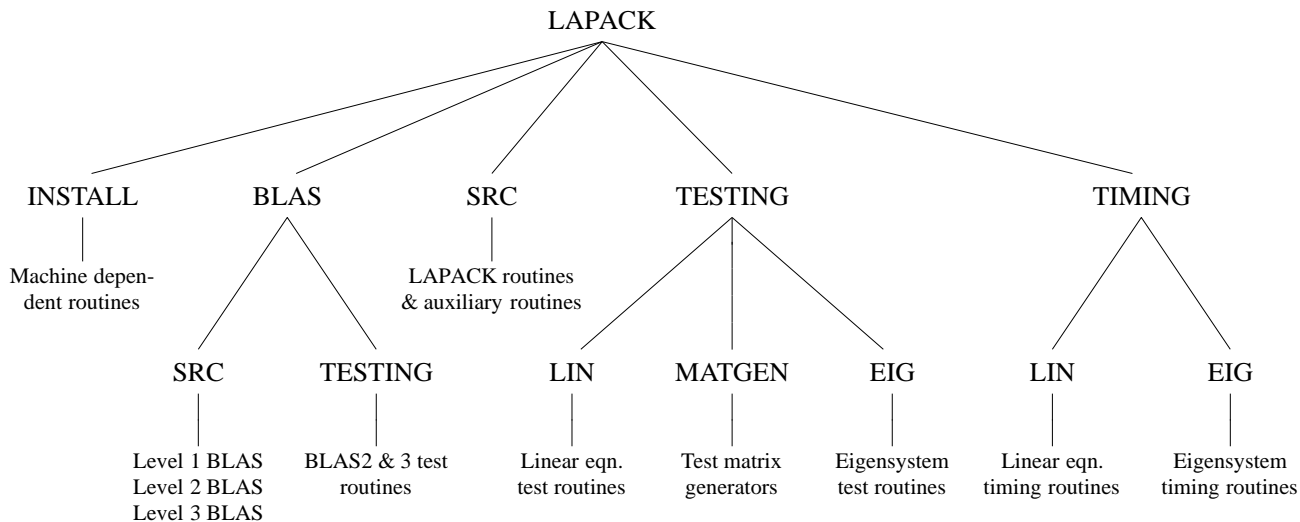


Figure 1: Unix organization of LAPACK

There have also been a number of revisions to correct bugs, improve efficiency, simplify calling sequences, and improve the appearance of output. You should destroy any previous release versions of LAPACK

2 File Format

The software for LAPACK is distributed in the form of a tape or tar file which contains the Fortran source for LAPACK, the Basic Linear Algebra Subprograms (the Level 1, 2, and 3 BLAS) needed by LAPACK, the testing programs, and the timing programs. This section describes the organization of the software for users who have received a Unix tar tape or a tar file via the file-transfer program ftp. Users who have an ASCII or EBCDIC tape should go to appendix F, although the overview in section 3 applies to both the Unix and non-Unix versions.

The software on a tar tape or in a tar file is organized in a number of directories as shown in Figure 1. Each of the lowest level directories in the tree structure contains a makefile to create a library or a set of executable programs for testing and timing. Libraries are created in the LAPACK directory and executable files are created in one of the directories BLAS, TESTING or TIMING. Input files for the test and timing programs are also found in these three directories so that testing may be carried out in the directories LAPACK/BLAS, LAPACK/TESTING and LAPACK/TIMING

0 1 2 3 5 10 16	Values of N (dimension)
5	Number of values of NB, NBMIN, and NX
1 3 3 3 20	Values of NB (blocksize)
2 2 2 2 2	Values of NBMIN (minimum blocksize)
1 0 5 9 1	Values of NX (crossover point)
20.0	Threshold value
T	Put T to test the LAPACK routines
T	Put T to test the driver routines
T	Put T to test the error exits
1	Code to interpret the seed
SEP 15	

The first line of the input file must contain the characters SEP in columns 1-3. Lines 2-12 are read using list-directed input and specify the following values:

line 2:	The number of values of N
line 3:	The values of N, the matrix dimension
line 4:	The number of values of the parameters NB, NBMIN, NX
line 5:	The values of NB, the blocksize
line 6:	The values of NBMIN, the minimum blocksize
line 7:	The values of NX, the crossover point
line 8:	The threshold value for the test ratios
line 9:	TESTCHK flag to test LAPACK routines
line 10:	TESTDRV flag to test driver routines
line 11:	TESTERR flag to test error exits from LAPACK and driver routines
line 12:	An integer code to interpret the random number seed
	=0: Set the seed to a default value before each run
	=1: Initialize the seed to a default value only before the first run
	=2: Like 1, but use the seed values on the next line
line 13:	If line 12 was 2, four integer values for the random number seed

The remaining lines are used to specify the matrix types for one or more sets of tests, as in the nonsymmetric case. The valid 3-character codes are SEP or SST (CST in complex, DST in double precision, and ZST in complex*16).

5.5 Testing the Singular Value Decomposition Routines

The test routine for the LAPACK singular value decomposition (SVD) routines has the following parameters which may be varied:

- the number of rows M and columns N of the test matrix A
- the type of the test matrix A
- the blocksize NB

The test program thus consists of a triply-nested loop, the outer one over NB , the next over pairs (M, N) , and the inner one over matrix types. On each iteration of the innermost loop, a matrix A is generated and used to test the SVD routines.

5.5.1 The Singular Value Decomposition Driver

The driver routine for the singular value decomposition is

`xGESVD` singular value decomposition of A

5.5.2 Test Matrices for the Singular Value Decomposition Routine

Sixteen different types of test matrices may be generated for the singular value decomposition routines. Table 8 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrix types other than the random bidiagonal matrices have $O(1)$ entries.

Type	Singular Value Distribution			
	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3, 6 [†] , 7 [‡]	4	5	
UDV	8, 11 [†] , 12 [‡]	9	10	
Random entries				13, 14 [†] , 15 [‡]
Random bidiagonal				16

[†]—matrix entries are $O(\sqrt{\text{overflow}})$
[‡]—matrix entries are $O(\sqrt{\text{underflow}})$

Table 8: Test matrices for the singular value decomposition

Matrix types identified as “Zero”, “Diagonal”, and “Random entries” should be self-explanatory. The other matrix types have the following meanings:

Identity: $\min(M, N) \times \min(M, N)$ identity matrix with zero rows or columns added to the bottom or right to make it $M \times N$

UDV : Real $M \times N$ diagonal matrix D with $O(1)$ entries multiplied by unitary (or real orthogonal) matrices on the left and right

Random bidiagonal: Upper bidiagonal matrix whose entries are randomly chosen from a logarithmic distribution on $[\varepsilon^{-2}, \varepsilon^{-2}]$

The QR algorithm used in `xBDQR` should compute all singular values, even small ones, to good relative accuracy, even of matrices with entries varying over many orders of magnitude, and the random bidiagonal matrix is intended to test this. Thus, unlike the other matrix types, the random bidiagonal matrix is neither $O(1)$, nor an $O(1)$ matrix scaled to some other magnitude.

The singular value distributions are analogous to the eigenvalue distributions in the nonsymmetric eigenvalue problem (see Section 6.2.1).

5.5.3 Test Matrices for the Singular Value Decomposition Driver

Five different types of test matrices may be generated for the singular value decomposition driver. Table 9 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries.

Type	Eigenvalue Distribution				
	Arithmetic	Geometric	Clustered	Random	Other
Zero					1
Identity					2
UDV	3, 4 [†] , 5 [‡]				

[†]— matrix entries are multiplied by the underflow threshold ε

[‡]— matrix entries are multiplied by the overflow threshold ε^{-1}

Table 9: Test matrices for the singular value decomposition driver

5.5.4 Tests Performed on the Singular Value Decomposition Routine

Finding the singular values and singular vectors of a dense, $m \times n$ matrix A is done in the following stages:

1. A is decomposed as QBP^* , where Q and P are unitary and B is real bidiagonal.
2. B is decomposed as $U\Sigma V$, where U and V are real orthogonal and Σ is a positive real diagonal matrix of singular values. This is done three times to compute
 - (a) $B = U\Sigma_1 V^*$, where Σ_1 is the diagonal matrix of singular values and the columns of the matrices U and V are the left and right singular vectors, respectively, of B .
 - (b) Same as above, but the singular values are stored in Σ_2 and the singular vectors are not computed.
 - (c) $A = (UQ)S(VP)^*$, the SVD of the original matrix A .

For each pair of matrix dimensions (m, n) and each selected matrix type, an m by n matrix A and an m by NRFB matrix X are generated. The problem dimensions are as follows

A	$m \times n$
Q	$m \times \tilde{n}$ (but $m \times m$ if $\text{NRFB} > 0$)
P	$\tilde{n} \times n$
B	$\tilde{n} \times \tilde{n}$
U, V	$\tilde{n} \times \tilde{n}$
$S1, S2$	diagonal, order $\sim n$
X	$m \times \text{NRFB}$

where $\tilde{n} = \min(m, n)$.

To check these calculations, the following test ratios are computed:

$$\begin{aligned}
r_1 &= \frac{\|A - QBP^*\|}{\tilde{n}\varepsilon \|A\|} & r_2 &= \frac{\|I - Q^*Q\|}{m\varepsilon} \\
r_3 &= \frac{\|I - P^*P\|}{n\varepsilon} & r_4 &= \frac{\|B - U\Sigma V^*\|}{\tilde{n}\varepsilon \|B\|} \\
r_5 &= \frac{\|Y - UZ\|}{\max(\tilde{n}, k)\varepsilon \|Y\|}, & \text{where } Y &= Q^*X \text{ and } Z = U^*Y. \\
r_6 &= \frac{\|I - U^*U\|}{\tilde{n}\varepsilon} & r_7 &= \frac{\|I - VV^*\|}{\tilde{n}\varepsilon} \\
r_8 &= \begin{cases} 0 & \text{if } S1 \text{ contains } \tilde{n} \text{ nonnegative values in decreasing order.} \\ \frac{1}{\varepsilon} & \text{otherwise} \end{cases} \\
r_9 &= \begin{cases} 0 & \text{if eigenvalues of } B \text{ are within } THRESH \text{ of those in } S1. \\ 2 * THRESH & \text{otherwise} \end{cases} \\
r_{10} &= \frac{\|S1 - S2\|}{\varepsilon \|S1\|} & r_{11} &= \frac{\|A - (QU)\Sigma(PV)^*\|}{\tilde{n}\varepsilon \|A\|} \\
r_{12} &= \frac{\|X - (QU)Z\|}{\max(m, k)\varepsilon \|X\|} & r_{13} &= \frac{\|I - (QU)^*(QU)\|}{m\varepsilon} \\
r_{14} &= \frac{\|I - (VP)(VP)^*\|}{n\varepsilon}
\end{aligned}$$

where the subscript 1 indicates that U and V were computed at the same time as Σ , and 0 that they were not. (All norms are $\|\cdot\|_1$.) The scalings in the test ratios assure that the ratios will be $O(1)$ (typically less than 10 or 100), independent of $\|A\|$ and ε , and nearly independent of m or n .

5.5.5 Tests Performed on the Singular Value Decomposition Driver

For the driver routine, the following tests are computed:

$$\begin{aligned}
r_1 &= \frac{\|A - U \text{diag}(S)VT\|}{\|A\| \max(M, N)\varepsilon} \\
r_2 &= \frac{\|I - U^T U\|}{M\varepsilon} \\
r_3 &= \frac{\|I - VT^T V\|}{N\varepsilon}
\end{aligned}$$

$$r_4 = \begin{cases} 0 & \text{if } S \text{ contains } MN \text{ nonnegative values in decreasing order.} \\ \frac{1}{\epsilon} & \text{otherwise} \end{cases}$$

$$r_5 = \frac{\|U - U_p\|}{M\epsilon}, \text{ where } U_p \text{ is a partially computed } U.$$

$$r_6 = \frac{\|VT - VT_p\|}{N\epsilon}, \text{ where } VT_p \text{ is a partially computed } VT.$$

$$r_7 = \frac{\|S - S_p\|}{MNMTN\epsilon \|S\|}, \text{ where } S_p \text{ is the vector of singular values from the partial SVD}$$

5.5.6 Input File for Testing the Singular Value Decomposition Routines

An annotated example of an input file for testing the singular value decomposition routines and driver routine is shown below

```
SVD: Data file for testing Singular Value Decomposition routines
20                               Number of values of M
0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 10 10 16 16  Values of M
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 10 16 10 16  Values of N
5                               Number of parameter values
1 3 3 3 20                      Values of NB (blocksize)
2 2 2 2 2                       Values of NBMIN (minimum blocksize)
1 0 5 9 1                       Values of NX (crossover point)
2 0 2 2 2                       Values of NRHS
20.0                             Threshold value
T                               Put T to test the LAPACK routines
T                               Put T to test the driver routines
T                               Put T to test the error exits
1                               Code to interpret the seed
SVD 16
```

The first line of the input file must contain the characters SVD in columns 1-3. Lines 2-14 are read using list-directed input and specify the following values:

line 2: The number of values of M and N
 line 3: The values of M the matrix row dimension
 line 4: The values of N the matrix column dimension
 line 5: The number of values of the parameters NB , $NBMN$, NX , $NRHS$
 line 6: The values of NB the blocksize
 line 7: The values of $NBMN$ the minimum blocksize
 line 8: The values of NX the crossover point
 line 9: The values of $NRHS$, the number of right hand sides
 line 10: The threshold value for the test ratios
 line 11: $TSCHK$ the flag to test LAPACK routines
 line 12: $TSIDR$ the flag to test driver routines
 line 13: $TSERR$ the flag to test error exits from the LAPACK and driver routines
 line 14: An integer code to interpret the random number seed.
 =0: Set the seed to a default value before each run
 =1: Initialize the seed to a default value only before the first run
 =2: Like 1, but use the seed values on the next line
 line 15: If line 14 was 2, four integer values for the random number seed

The remaining lines are used to specify the matrix types for one or more sets of tests, as in the nonsymmetric case. The valid 3-character codes are SVD or SBD (CBD in complex, DBD in double precision, and ZBD in complex*16).

5.6 Testing the Generalized Nonsymmetric Eigenvalue Routine

The test routine for the LAPACK generalized nonsymmetric eigenvalue routines has the following parameters which may be varied:

- the order N of the pair of test matrices A , B
- the type of the pair of test matrices A , B
- five numerical parameters:
 - the blocksize NB ;
 - the minimum blocksize $NBMN$;
 - the number of shifts NS for the multishift QZ method;
 - the minimum blocksize $MAXB$;
 - $MINBK$ the minimum number of rows/columns to be updated by a block of Householder transformations in order for blocking to be used.

The test program thus consists of a triply-nested loop, the outer one over quintuples $(NB, NBMN, NS, MAXB, MINBK)$, the next over N and the inner one over matrix types. On each iteration of the innermost loop, a pair of matrices A, B is generated and used to test the eigenvalue routines.

5.6.1 The Generalized Nonsymmetric Eigenvalue Drivers

The driver routines for the generalized nonsymmetric eigenvalue problem are

xGEGS factors A and B into generalized Schur form and computes the generalized eigenvalues

xGEGV computes the generalized eigenvalues and the left and right generalized eigenvectors

5.6.2 Test Matrices for the Generalized Nonsymmetric Eigenvalue

Twenty-six different types of test matrix pairs may be generated for the generalized nonsymmetric eigenvalue routines. Tables 10 and 11 show the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries.

Matrix A:	Matrix B:									
	0	I			J^t	$\begin{pmatrix} I & 0 \\ 0 & K \end{pmatrix}$	D_1			D_3
		$\times 1$	$\times \omega$	$\times \frac{1}{\omega}$			$\times 1$	$\times \omega$	$\times \frac{1}{\omega}$	
0	1	3								
I	2	4					8			
$I \times \omega$									12	
$I \times \frac{1}{\omega}$								11		
J^t					5					
$\begin{pmatrix} K & 0 \\ 0 & I \end{pmatrix}$						6				
D_1		7								
$D_1 \times \omega$			14	10						
$D_1 \times \frac{1}{\omega}$			9	13						
D_2										15

Table 10: Sparse test matrices for the generalized nonsymmetric eigenvalue problem

The following symbols and abbreviations are used:

0: The zero matrix.

I: The identity matrix.

ω : Generally, the underflow threshold times the order of the matrix divided by the machine precision. In other words, this is a very small number, useful for testing the sensitivity to underflow and division by small numbers. Its reciprocal tests for overflow problems.

J^t : Transposed Jordan block, i.e., matrix with ones on the first subdiagonal and zeros elsewhere. (Note that the diagonal is zero.)

Distribution of Eigenvalues	Magnitude of A, B				
	$\ A\ \approx 1, \ B\ \approx 1$	$\ A\ \approx \frac{1}{\omega}, \ B\ \approx \omega$	$\ A\ \approx \omega, \ B\ \approx \omega$	$\ A\ \approx \frac{1}{\omega}, \ B\ \approx \frac{1}{\omega}$	$\ A\ \approx \omega, \ B\ \approx \frac{1}{\omega}$
All Ones	16				
(Same as type 15)	17				
Arithmetic	19	22	24	25	23
Geometric	20				
Clustered	18				
Random	21				
Random Entries	26				

Table 11: Dense test matrices for the generalized nonsymmetric eigenvalue problem

K : A $(k+1) \times (k+1)$ transposed Jordan block which is a diagonal block within a $(2k+1) \times (2k+1)$ matrix. Thus, $\begin{pmatrix} K & 0 \\ 0 & I \end{pmatrix}$ has all zero entries except for the last k diagonal entries and the first k entries on the first subdiagonal. (Note that the matrices $\begin{pmatrix} K & 0 \\ 0 & I \end{pmatrix}$ and $\begin{pmatrix} I & 0 \\ 0 & K \end{pmatrix}$ have odd order; if an even order matrix is needed, a zero row and column are added at the end.)

D_1 : A diagonal matrix with the entries $0, 1, 2, \dots, n-1$ on the diagonal, where n is the order of the matrix.

D_2 : A diagonal matrix with the entries $0, 0, 1, 2, \dots, n-3, 0$ on the diagonal, where n is the order of the matrix.

D_3 : A diagonal matrix with the entries $0, n-3, n-4, \dots, 1, 0, 0$ on the diagonal, where n is the order of the matrix.

Except for matrices with random entries, all the matrix pairs include at least one infinite, one zero, and one singular eigenvalue. For arithmetic, geometric, and clustered eigenvalue distributions, the eigenvalues lie between ε (the machine precision) and 1 in absolute value. The eigenvalue distributions have the following meanings:

Arithmetic: Difference between adjacent eigenvalues is a constant.

Geometric: Ratio of adjacent eigenvalues is a constant.

Clustered: One eigenvalue is 1 and the rest are ε in absolute value.

Random: Eigenvalues are logarithmically distributed.

Random entries: Matrix entries are uniformly distributed random numbers.

5.6.3 Test Matrices for the Generalized Nonsymmetric Eigenvalue

The same twenty-six different types of test matrix pairs may be generated for the generalized nonsymmetric eigenvalue drivers. Tables 10 and 11 show the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries.

5.6.4 Tests Performed on the Generalized Nonsymmetric Eigenvalue

Finding the eigenvalues and eigenvectors of a pair of nonsymmetric matrices A , B is done in the following stages:

1. A is decomposed as UHV^* and B as UTV^* , where U and V are unitary, H is upper Hessenberg, T is upper triangular, and U^* is the conjugate transpose of U .
2. H is decomposed as QSZ^* and T as QPZ^* , where Q and Z are unitary, P is upper triangular with non-negative real diagonal entries and S is in Schur form; this also gives the generalized eigenvalues λ_i , which are expressed as pairs (α_i, β_i) , where $\lambda_i = \alpha_i / \beta_i$.
3. The left and right generalized eigenvectors l_i and r_i for the pair S, P are computed, and from them the back-transformed eigenvectors \hat{l}_i and \hat{r}_i for the matrix pair H, T . The eigenvectors are normalized so that their largest element has absolute value 1.¹ (Note that eigenvectors corresponding to singular eigenvalues, i.e., eigenvalues for which $\alpha = \beta = 0$, are not well defined, these are not tested in the eigenvector tests described below)

To check these calculations, the following test ratios are computed:

$$\begin{aligned}
 r_1 &= \frac{\|A - UHV^*\|}{n\varepsilon \|A\|} & r_2 &= \frac{\|B - UTV^*\|}{n\varepsilon \|B\|} \\
 r_3 &= \frac{\|I - UU^*\|}{n\varepsilon} & r_4 &= \frac{\|I - VV^*\|}{n\varepsilon} \\
 r_5 &= \frac{\|H - QSZ^*\|}{n\varepsilon \|H\|} & r_6 &= \frac{\|T - QPZ^*\|}{n\varepsilon \|T\|} \\
 r_7 &= \frac{\|I - QQ^*\|}{n\varepsilon} & r_8 &= \frac{\|I - ZZ^*\|}{n\varepsilon} \\
 r_9 &= \max_i \frac{\|(\beta_i S - \alpha_i P)^T l_i\|}{\varepsilon \max(\|\beta_i S\|, \|\alpha_i P\|)} & r_{10} &= \max_i \frac{\|(\beta_i H - \alpha_i T)^T \hat{l}_i\|}{\varepsilon \max(\|\beta_i H\|, \|\alpha_i T\|)} \\
 r_{11} &= \max_i \frac{\|(\beta_i S - \alpha_i P) r_i\|}{\varepsilon \max(\|\beta_i S\|, \|\alpha_i P\|)} & r_{12} &= \max_i \frac{\|(\beta_i H - \alpha_i T) \hat{r}_i\|}{\varepsilon \max(\|\beta_i H\|, \|\alpha_i T\|)}
 \end{aligned}$$

¹For the purpose of normalization, the “absolute value” of a complex number $z = x + iy$ is computed as $|x| + |y|$.

All norms are $\|\cdot\|_1$. The scalings in the test ratios assure that the ratios will be $O(1)$, independent of $\|A\|$ and ε , and nearly independent of n .

When the test programs run, these test ratios will be compared with a user-specified threshold `THRESH`, and for each test ratio that exceeds `THRESH`, a message is printed specifying the test matrix, the ratio that failed, and its value. An example message is

```
Matrix order= 25, type=18, seed=2548,1429,1713,1411, result 8 is 11.33
```

In this example, the test matrix was of order $n=25$ and of type 18 from Table 11, “seed” is the initial 4-integer seed of the random number generator used to generate A and B , and “result” specifies that test ratio r_8 failed to pass the threshold, and its value was 11.33.

The normalization of the eigenvectors will also be checked. If the absolute value of the largest entry in an eigenvector is not within $\varepsilon \times \text{THRESH}$ of 1, then a message is printed specifying the error. An example message is

```
SCHK51: Right Eigenvectors from STGEVC(JOB=B) incorrectly normalized.
Error/precision=0.103E+05, n= 25, type= 18, seed=2548,1429,1713,1411.
```

5.6.5 Tests Performed on the Generalized Nonsymmetric Eigenvalue

The two driver routines have slightly different tests applied to them. For `SGECS` the following tests are computed:

$$r_1 = \frac{\|A - QSZ^T\|}{\|A\| n\varepsilon} \quad r_2 = \frac{\|B - QTZ^T\|}{\|B\| n\varepsilon}$$

$$r_3 = \frac{\|I - QQ^T\|}{n\varepsilon} \quad r_4 = \frac{\|I - ZZ^T\|}{n\varepsilon}$$

$$r_5 = \max_j D(j) = \begin{cases} \frac{|\alpha(j) - S(j,j)|}{\max(|\alpha(j)|, |S(j,j)|)} + \frac{|\beta(j) - T(j,j)|}{\max(|\beta(j)|, |T(j,j)|)} & \text{if } \alpha(j) \text{ is real} \\ \frac{|\det(sS - wT)|}{\varepsilon \max(|s|, |w|, \|T\|) \|sS - wT\|} & \text{if } \alpha(j) \text{ is complex,} \end{cases}$$

where S and T are the 2×2 diagonal blocks of S and T corresponding to the j th eigenvalue.

For `SGEV` the following tests are computed:

$$r_6 = \max_{\text{left eigenvalue/vector pairs } (\beta/\alpha, l)} \frac{|(\beta A - \alpha B)^T l|}{\varepsilon \max(|\beta A|, |\alpha B|)}$$

$$r_7 = \max_{\text{right eigenvalue/vector pairs } (\beta/\alpha, r)} \frac{|(\beta A - \alpha B) r|}{\varepsilon \max(|\beta A|, |\alpha B|)}$$

5.6.6 Input File for Testing the Generalized Nonsymmetric Eigenvalue Routines and Drivers

An annotated example of an input file for testing the generalized nonsymmetric eigenvalue routines is shown below

```

SGG:  Data file for testing Nonsymmetric Eigenvalue Problem routines
7          Number of values of N
0 1 2 3 5 10 16      Values of N (dimension)
4          Number of parameter values
1   1   2   2        Values of NB (blocksize)
100 100 2   2       Values of NBMIN (minimum blocksize)
2   4   2   4       Values of NS (no. of shifts)
100 100 2   2       Values of MAXB (multishift crossover pt)
100 100 2   2       Values of NBCOL (minimum col. dimension)
20.0      Threshold value
T         Put T to test the LAPACK routines
T         Put T to test the driver routines
T         Put T to test the error exits
1         Code to interpret the seed
SGG  26

```

The first line of the input file must contain the characters SGG in columns 1-3. Lines 2-14 are read using list-directed input and specify the following values:

```

line 2:  The number of values of N
line 3:  The values of N, the matrix dimension
line 4:  Number of values of the parameters NB, NBMIN, NS, MAXB, NBCOL
line 5:  The values for the blocksize NB
line 6:  The values for the minimum blocksize, NBMIN
line 7:  The values for the number of shifts NS
line 8:  The values of MAXB, the multishift crossover point
line 9:  The values of NBCOL, the minimum column dimension for blocks
line 10: The threshold value for the test ratios
line 11: TSICHK flag to test LAPACK routines
line 12: TSIIDV flag to test driver routines
line 13: TSIERR flag to test error exits from LAPACK and driver routines
line 14: An integer code to interpret the random number seed
        =0: Set the seed to a default value before each run
        =1: Initialize the seed to a default value only before the first run
        =2: Like 1, but use the seed values on the next line
line 15: If line 14 was 2, four integer values for the random number seed

```

The remaining lines are used to specify the matrix types for one or more sets of tests, as in the nonsymmetric case. The valid 3-character codes are SGG (CGG in complex, DGG in double precision, and ZGG in complex*16).

6 More About Timing

There are two distinct timing programs for LAPACK routines in each data type, one for the linear equations routines and one for the eigensystem routines. The linear equation

timing program also times the Level 2 and 3 BLAS, variants of the LU and Cholesky factorizations, and the reductions to bidiagonal, tridiagonal, or Hessenberg form for eigenvalue computations. Results from the linear equation timing program are given in megaflops, and the operation counts are computed from a formula (see Appendix C). Results from the eigensystem timing program are given in execution times, operation counts, and megaflops, where the operation counts are calculated during execution using special versions of the LAPACK routines which have been instrumented to count operations. Each program has its own style of input, and the eigensystem timing program accepts four different sets of parameters, for the nonsymmetric eigenvalue problem, the symmetric eigenvalue problem, the singular value decomposition, and the generalized nonsymmetric eigenvalue problem. The following sections describe the different input formats and timing parameters.

Both timing programs, but the linear equation timing program in particular, are intended to be used to collect data to determine optimal values for the block routines. All of the block factorization, inversion, reduction, and orthogonal transformation routines in LAPACK are included in the linear equation timing program. Currently, the block parameters NB and NX, as well as others, are passed to the block routines by the environment inquiry function ILAENV, which in turn receives these values through a common block set in the timing program. Future implementations of ILAENV may be tuned to a specific machine so that users of LAPACK will not have to set the block size. For a brief introduction to ILAENV and guidelines on setting some of the parameters, see the Preliminary LAPACK User's Guide [1].

The main timing procedure for the REAL linear equation routines is found in LAPACK/TIMING/LIN/stimaa.f in the Unix version and is the first program unit in SLIN.TIME in the non-Unix version. The main timing procedure for the REAL eigenvalue routines is found in LAPACK/TIMING/EIG/stimee.f in the Unix version and is the first program unit in SEIGTIME in the non-Unix version.

6.1 The Linear Equation Timing Program

The timing program for the linear equation routines is driven by a data file from which the following parameters may be varied:

- M the matrix row dimension
- N the matrix column dimension
- K the bandwidth for the band routines, or the third dimension for the Level 3 BLAS
- NB the block size for the blocked routines
- NX the crossover point, the point in a block algorithm at which we switch to an unblocked algorithm
- LDA the leading dimension of the dense and banded matrices.

For banded matrices, the values of M are used for the matrix row and column dimensions, and for symmetric or Hermitian matrices that are not banded, the values of N are used for the matrix dimension.

The number and size of the input values are limited by certain program maxima which are defined in PARAMETER statements in the main timing program

Parameter	Description	Value
NMAX	Maximum value of M, N, K and NB for dense matrices	512
IDAMAX	Maximum value of IDA	532
NMNB	Maximum value of M for banded matrices	5000
MAXN	Maximum number of values of M, N, K or NB	12
MNLDA	Maximum number of values of IDA	4

The parameter IDAMAX should be at least NMAX. For the xCB path, we must have $(IDA + K)M \leq 3(IDAMAX - 1)(NMAX - 1)$, where $IDA \geq 3K + 1$, which restricts the value of K. These limits allow K to be as big as 200 for M=1000. For the xPB and xTB paths, the condition is $(2K + 1)M \leq 3(NMAX - 1)(IDAMAX - 1)$.

The input file also specifies a set of LAPACK routine names or LAPACK path names to be timed. The path names are similar to those used for the test program and include the following standard paths:

{S, C, D, Z}	GE	General matrices (LU factorization)
{S, C, D, Z}	GB	General banded matrices
{S, C, D, Z}	PO	Positive definite matrices (Cholesky factorization)
{S, C, D, Z}	PP	Positive definite packed
{S, C, D, Z}	PB	Positive definite banded
{S, C, D, Z}	SY	Symmetric indefinite matrices (Bunch-Kaufman factorization)
{S, C, D, Z}	SP	Symmetric indefinite packed
{C, Z}	HE	Hermitian indefinite matrices (Bunch-Kaufman factorization)
{C, Z}	HP	Hermitian indefinite packed
{S, C, D, Z}	TR	Triangular matrices
{S, C, D, Z}	TP	Triangular packed matrices
{S, C, D, Z}	TB	Triangular band
{S, C, D, Z}	QR	QR decomposition
{S, C, D, Z}	RQ	RQ decomposition
{S, C, D, Z}	LQ	LQ decomposition
{S, C, D, Z}	QL	QL decomposition
{S, C, D, Z}	QP	QR decomposition with column pivoting
{S, C, D, Z}	HR	Reduction to Hessenberg form
{S, C, D, Z}	TD	Reduction to real tridiagonal form
{S, C, D, Z}	BR	Reduction to bidiagonal form
{S, C, D, Z}	LU	Variants of the LU factorization
{S, C, D, Z}	CH	Variants of the Cholesky factorization

For timing the Level 2 and 3 BLAS, two extra paths are provided:

{S, C, D, Z}	B2	Level 2 BLAS
{S, C, D, Z}	B3	Level 3 BLAS

The paths xLU, xCH, xHR, and xTD include timing of the equivalent ITPACK or EISPACK factorizations and reductions for comparison. The xLU path requests timing of three

block variants of the LU factorization (left-looking, Gout, and right-looking) for $N \times N$ matrices, as well as the corresponding unblocked variants on matrices of size $N \times NB$ and the Linpack routine xGEEA. The xCH timing path requests timing of three block variants of the Cholesky factorization and the corresponding Linpack routine xPCEA. The LU and Cholesky variants are not strictly part of LAPACK and will not be included in the public release.

The timing programs have their own matrix generator that supplies random Toeplitz matrices (constant along a diagonal) for many of the timing paths. Toeplitz matrices are used because they can be generated more quickly than dense matrices, and the call to the matrix generator is inside the timing loop. The LAPACKtest matrix generator is used to generate matrices of known condition for the xQR, xRQ, xLQ, xQL, xQP, xHR, xID, and xBR paths.

The user specifies a minimum time for which each routine should run and the computation is repeated if necessary until this time is used. In order to prevent inflated performance due to a matrix remaining in the cache from one iteration to the next, the paths that use random Toeplitz matrices regenerate the matrix before each call to the LAPACK routine in the timing loop. The time for generating the matrix at each iteration is subtracted from the total time.

An annotated example of an input file for timing the REAL linear equation routines that operate on dense square matrices is shown below. The first line of input is printed as the first line of output and can be used to identify different sets of results.

```
LAPACK timing, REAL square matrices
5                               Number of values of M
100 200 300 400 500           Values of M (row dimension)
5                               Number of values of N
100 200 300 400 500           Values of N (column dimension)
2                               Number of values of K
100 400                        Values of K
5                               Number of values of NB
1 16 32 48 64                 Values of NB (blocksize)
0 48 128 128 128              Values of NX (crossover point)
2                               Number of values of LDA
512 513                        Values of LDA (leading dimension)
0.0                            Minimum time in seconds
SGE      T T T
SP0      T T T
SPP      T T T
SSY      T T T
SSP      T T T
STR      T T
STP      T T
SQR      T T F
SLQ      T T F
SQL      T T F
```

```

SRQ      T T F
SQP      T
SHR      T T F F
STD      T T F F
SBR      T F F
SLU      T T T T T T T
SCH      T T T T T T T

```

The first 13 lines of the input file are read using list-directed input and are used to specify the values of M , N , K , NB , NX , LDA , and $TIMIN$ (the minimum time). By default, `xCEM` and `xCEM` are called to sample the BLAS performance on square matrices of order N but this option can be controlled by entering one of the following on line 14:

```

BAND      Time xCEM (instead of xCEM) using matrices of order M and
          bandwidth K and time xCEM using matrices of order K

```

```

NONE      Do not do the sample timing of xCEM and xCEM

```

The timing paths or routine names which follow may be specified in any order.

When timing the band routines it is more interesting to use one large value of the matrix size and vary the bandwidth. An annotated example of an input file for timing the REAL linear equation routines that operate on banded matrices is shown below

LAPACK timing, REAL band matrices

```

1          Number of values of M
1000       Values of M (row dimension)
0          Number of values of N
200       Values of N (column dimension)
5          Number of values of K
25 50 100 150 200  Values of K (bandwidth)
5          Number of values of NB
1 16 32 48 64     Values of NB (blocksize)
0 48 128 128 128  Values of NX (crossover point)
2          Number of values of LDA
601 602       Values of LDA (leading dimension)
0.0         Minimum time in seconds
BAND        Time sample banded BLAS
SGB
SPB
STB

```

Here M specifies the matrix size and K specifies the bandwidth for the test paths `SCB`, `SIB`, and `SIB`. Note that we request timing of the sample BLAS for banded matrices by specifying "BAND" on line 14.

We also provide a separate input file for timing the orthogonal factorization and reduction routines that operate on rectangular matrices. For these routines, the values of M and

N are specified in ordered pairs (M, N) . An annotated example of an input file for timing the REAL linear equation routines that operate on dense rectangular matrices is shown below. The input file is read in the same way as the one for dense square matrices.

LAPACK timing, REAL rectangular matrices

```

7                               Number of values of M
100 200 100 200 400 200 400    Values of M (row dimension)
7                               Number of values of N
100 100 200 200 200 400 400    Values of N (column dimension)
2                               Number of values of K
100 400                         Values of K
5                               Number of values of NB
1 16 32 48 64                 Values of NB (blocksize)
0 48 128 128 128              Values of NX (crossover point)
2                               Number of values of LDA
400 401                        Values of LDA (leading dimension)
0.0                             Minimum time in seconds
none
SQR      T T T
SLQ      T T T
SQL      T T T
SRQ      T T T
SQP      T
SBR      T T F

```

6.2 Timing the Level 2 and 3 BLAS

Timing of the Level 2 and 3 BLAS routines may be requested from one of the linear equation input files, or by using a special BLAS format provided for compatibility with previous releases of LAPACK. The BLAS input format is the same as the linear equation input format, except that values of NX are not read in. The BLAS input format is requested by specifying 'BLAS' on the first line of the file.

Three input files are provided for timing the BLAS with the matrix shapes encountered in the LAPACK routines. In each of these files, one of the parameters M , N , and K for the Level 3 BLAS is on the order of the blocksize while the other two are on the order of the matrix size. The first of these input files also times the Level 2 BLAS, and we include the single precision real version of this data file here for reference:

BLAS timing, REAL data, K small

```

5                               Number of values of M
100 200 300 400 500           Values of M
5                               Number of values of N
100 200 300 400 500           Values of N
5                               Number of values of K
2 16 32 48 64                 Values of K
1                               Number of values of INCX

```

```

1           Values of INCX
2           Number of values of LDA
512 513     Values of LDA
0.0        Minimum time in seconds
none       Do not time the sample BLAS
SB2
SB3

```

Since the Fortran BLAS do not contain any sub-blocking, the block size NB is not required and its value is replaced by that of INCX the increment between successive elements of a vector in the Level 2 BLAS. Note that we have specified “none” on line 13 to suppress timing of the sample BLAS, which are redundant in this case.

6.3 Timing the Nonsymmetric Eigenproblem

A separate input file drives the timing codes for the nonsymmetric eigenproblem. The input file specifies

- N the matrix size
- four-tuples of parameter values (NB, NS, MNB, LDA) specifying the block size NB, the number of shifts NS, the matrix size MNB less than which an unblocked routine is used, and the leading dimension LDA
- the test matrix types
- the routines or sequences of routines from LAPACK or EISPACK to be timed

The parameters NS and MNB apply only to the QR iteration routine xHEQR, and NB is used only by the block algorithms. A goal of this timing code is to determine the values of NB, NS and MNB which maximize the speed of the codes.

The number and size of the input values are limited by certain program maxima which are defined in PARAMETER statements in the main timing program

Parameter	Description	Value
MXN	Maximum value for N, NB, NS, or MNB	400
IDAMX	Maximum value for LDA	420
MXNN	Maximum number of values of N	12
MXPRM	Maximum number of parameter sets (NB, NS, MNB, LDA)	10

The computations that may be timed for the REAL version are

1. SGEHD (LAPACK reduction to upper Hessenberg form)
2. SHSEQRE (LAPACK computation of eigenvalues only of a Hessenberg matrix)
3. SHSEQRS (LAPACK computation of the Schur form of a Hessenberg matrix)

4. `SHEQRV` (LAPACK computation of the Schur form and Schur vectors of a Hessenberg matrix)
5. `SIREM(L)` (LAPACK computation of the left eigenvectors of a matrix in Schur form)
6. `SIREM(R)` (LAPACK computation of the right eigenvectors of a matrix in Schur form)
7. `SHEIN(L)` (LAPACK computation of the left eigenvectors of an upper Hessenberg matrix using inverse iteration)
8. `SHEIN(R)` (LAPACK computation of the right eigenvectors of an upper Hessenberg matrix using inverse iteration)
9. `CRHES` (EISPACK reduction to upper Hessenberg form to be compared to `SGHRD`)
10. `HQR` (EISPACK computation of eigenvalues only of a Hessenberg matrix, to be compared to `SHEQR(E)`)
11. `HQR2` (EISPACK computation of eigenvalues and eigenvectors of a Hessenberg matrix, to be compared to `SHEQR(V)` plus `SIREM(R)`)
12. `INMT` (EISPACK computation of the right eigenvectors of an upper Hessenberg matrix using inverse iteration, to be compared to `SHEIN(R)`).

Eight different matrix types are provided for timing the nonsymmetric eigenvalue routines. A variety of matrix types is allowed because the number of iterations to compute the eigenvalues, and hence the timing, can depend on the type of matrix whose eigendecomposition is desired. The matrices used for timing are of the form XTX^{-1} where X is either orthogonal (for types 1-4) or random with condition number $1/\sqrt{\epsilon}$ (for types 5-8), where ϵ is the machine roundoff error. The matrix T is upper triangular with random $O(1)$ entries in the strict upper triangle and has on its diagonal

- evenly spaced entries from 1 down to ϵ with random signs (matrix types 1 and 5)
- geometrically spaced entries from 1 down to ϵ with random signs (matrix types 2 and 6)
- “clustered” entries $1, \epsilon, \dots, \epsilon$ with random signs (matrix types 3 and 7), or
- real or complex conjugate paired eigenvalues randomly chosen from the interval $(\epsilon, 1)$ (matrix types 4 or 8).

An annotated example of an input file for timing the REAL nonsymmetric eigenproblem routines is shown below

```

NEP: Data file for timing Nonsymmetric Eigenvalue Problem routines
4                                     Number of values of N
50 100 200 300                       Values of N (dimension)

```

```

4                               Number of values of parameters
1  16  32  48                   Values of NB (blocksize)
4   6   8  12                   Values of NS (number of shifts)
40  40  40  40                  Values of MAXB (multishift crossover pt)
301 301 301 301                Values of LDA (leading dimension)
0.0                             Minimum time in seconds
4                               Number of matrix types
1  3  4  6
SHS   T T T T T T T T T T T T

```

The first line of the input file must contain the characters NEP in columns 1-3. Lines 2-10 are read using list-directed input and specify the following values:

- line 2: The number of values of N
- line 3: The values of N, the matrix dimension
- line 4: The number of values of the parameters NB, NS, MAXB, and LDA
- line 5: The values of NB, the blocksize
- line 6: The values of NS, the number of shifts
- line 7: The values of MAXB, the maximum blocksize
- line 8: The values of LDA, the leading dimension
- line 9: The minimum time in seconds that a routine will be timed
- line 10: NYPES, the number of matrix types to be used

If $0 < NYPES < 8$, then line 11 specifies NYPES integer values which are the numbers of the matrix types to be used. The remaining lines specify a path name and the specific computations to be timed. For the nonsymmetric eigenvalue problem the path names for the four data types are SHS, DHS, CHS, and ZHS. A line to request all the routines in the REAL path has the form

```
SHS   T T T T T T T T T T T T
```

where the first 3 characters specify the path name, and up to 12 nonblank characters may appear in columns 4-80. If the k th such character is 'T' or 't', the k th routine will be timed. If at least one but fewer than 12 nonblank characters are specified, the remaining routines will not be timed. If columns 4-80 are blank, all the routines will be timed, so the input line

```
SHS
```

is equivalent to the line above.

The output is in the form of a table which shows the absolute times in seconds, floating point operation counts, and megaflop rates for each routine over all relevant input parameters. For the blocked routines, the table has one line for each different value of NB and for the SHEQR routine, one line for each different combination of NS and MAXB as well.

6.4 Timing the Symmetric Eigenproblem

A separate input file drives the timing codes for the symmetric eigenproblem. The input file specifies

- N the matrix size
- pairs of parameter values (NB LDA) specifying the block size NB and the leading dimension LDA
- the test matrix types
- the routines or sequences of routines from LAPACK or EISPACK to be timed.

Goal of this timing code is to determine the values of NB which maximize the speed of the block algorithms.

The number and size of the input values are limited by certain program maxima which are defined in PARAMETER statements in the main timing program

Parameter	Description	Value
MAXN	Maximum value for N or NB	400
LDAMX	Maximum value for LDA	420
MAXN	Maximum number of values of N	12
MAXPM	Maximum number of pairs of values (NB LDA)	10

The computations that may be timed depend on whether the data is real or complex. For the REAL version the possible computations are

1. SSYTRD (LAPACK reduction to symmetric tridiagonal form)
2. SSIEQR(N) (LAPACK computation of eigenvalues only of a symmetric tridiagonal matrix)
3. SSIEQR(V) (LAPACK computation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix)
4. SSIERF (LAPACK computation of the eigenvalues only of a symmetric tridiagonal matrix using a square-root free algorithm)
5. SPIEQR(COMPZ= N) (LAPACK computation of the eigenvalues of a symmetric positive definite tridiagonal matrix)
6. SPIEQR(COMPZ= V) (LAPACK computation of the eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix)
7. SSIEBZ(RANGE= I') (LAPACK computation of the eigenvalues in a specified interval for a symmetric tridiagonal matrix)
8. SSIEBZ(RANGE= V) (LAPACK computation of the eigenvalues in a half-open interval for a symmetric tridiagonal matrix)
9. SSIEIN (LAPACK computation of the eigenvectors of a symmetric tridiagonal matrix corresponding to specified eigenvalues using inverse iteration)
10. TRED (EISPACK reduction to symmetric tridiagonal form to be compared to SSYTRD)

11. **IMQL1** (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to **SSIEQR(N)**)
12. **IMQL2** (EISPACK computation of eigenvalues and eigenvectors of a symmetric tridiagonal matrix, to be compared to **SSIEQR(V)**)
13. **TQIRAT** (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to **SSIERF**).
14. **TRIDB** (EISPACK computation of the eigenvalues of A) (compare with **SSIEBZ - RANGE=I'**)
15. **HSECT** (EISPACK computation of the eigenvalues of A) (compare with **SSIEBZ - RANGE=V**)
16. **TINMT** (EISPACK computation of the eigenvectors of a triangular matrix using inverse iteration) (compare with **SSIEIN**)

For complex matrices the possible computations are

1. **CHEIRD** (LAPACK reduction of a complex Hermitian matrix to real symmetric tridiagonal form)
2. **CSIEQR(N)** (LAPACK computation of eigenvalues only of a symmetric tridiagonal matrix)
3. **CUNGR+(CSIEQR(V)** (LAPACK computation of the eigenvalues and eigenvectors of a symmetric diagonal matrix)
4. **CPIEQR(VECT= N)** (LAPACK computation of the eigenvalues only of a symmetric positive definite tridiagonal matrix)
5. **CUNGR+(CPIEQR(VECT= V)** (LAPACK computation of the eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix)
6. **SSIEBZ+(SSIEIN+(CUNMR** (LAPACK computation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix)
7. **HIRDI** (EISPACK reduction to symmetric tridiagonal form to be compared to **CHEIRD**)
8. **IMQL1** (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to **CSIEQR(V)**)
9. **IMQL2-HIRBK** (EISPACK computation of eigenvalues and eigenvectors of a complex Hermitian matrix given the reduction to real symmetric tridiagonal form to be compared to **CUNGR+(CSIEQR)**).

Four different matrix types are provided for timing the symmetric eigenvalue routines. The matrices used for timing are of the form XDX^{-1} , where X is orthogonal and D is diagonal with entries

- evenly spaced entries from 1 down to ε with random signs (matrix type 1),
- geometrically spaced entries from 1 down to ε with random signs (matrix type 2),
- “clustered” entries $1, \varepsilon, \dots, \varepsilon$ with random signs (matrix type 3), or
- eigenvalues randomly chosen from the interval $(\varepsilon, 1)$ (matrix type 4).

An annotated example of an input file for timing the REAL symmetric eigenproblem routines is shown below

```

SEP:  Data file for timing Symmetric Eigenvalue Problem routines
5                               Number of values of N
50 100 200 300 400             Values of N (dimension)
5                               Number of values of parameters
1   16  32  48  64             Values of NB (blocksize)
401 401 401 401 401           Values of LDA (leading dimension)
0.0                             Minimum time in seconds
4                               Number of matrix types
SST   T T T T T T T T

```

The first line of the input file must contain the characters SEP in columns 1-3. Lines 2-8 are read using list-directed input and specify the following values:

- line 2: The number of values of N
- line 3: The values of N, the matrix dimension
- line 4: The number of values of the parameters NB and LDA
- line 5: The values of NB, the blocksize
- line 6: The values of LDA, the leading dimension
- line 7: The minimum time in seconds that a routine will be timed
- line 8: NIMPES, the number of matrix types to be used

If $0 < \text{NIMPES} < 4$, then line 9 specifies NIMPES integer values which are the numbers of the matrix types to be used. The remaining lines specify a path name and the specific computations to be timed. For the symmetric eigenvalue problem the path names for the four data types are SST, DST, CST, and ZST. The (optional) characters after the path name indicate the computations to be timed, as in the input file for the nonsymmetric eigenvalue problem

6.5 Timing the Singular Value Decomposition

A separate input file drives the timing codes for the Singular Value Decomposition (SVD). The input file specifies

- pairs of parameter values (M N) specifying the matrix row dimension M and the matrix column dimension N
- pairs of parameter values (NB LDA) specifying the block size NB and the leading dimension LDA

- the test matrix types
- the routines or sequences of routines from LAPACK or LINPACK to be timed.

A goal of this timing code is to determine the values of NB which maximize the speed of the block algorithms.

The number and size of the input values are limited by certain program maxima which are defined in PARAMETER statements in the main timing program

Parameter	Description	Value
MXN	Maximum value for M, N, or NB	400
IDAMX	Maximum value for LDA	420
MXNN	Maximum number of pairs of values (M, N)	12
MXPRM	Maximum number of pairs of values (NB, LDA)	10

The computations that may be timed for the REAL version are

1. SCFBD (LAPACK reduction to bidiagonal form)
2. SDBQR (LAPACK computation of singular values only of a bidiagonal matrix)
3. SDBQR(L) (LAPACK computation of the singular values and left singular vectors of a bidiagonal matrix)
4. SDBQR(R) (LAPACK computation of the singular values and right singular vectors of a bidiagonal matrix)
5. SDBQR(B) (LAPACK computation of the singular values and right and left singular vectors of a bidiagonal matrix)
6. SDBQR(V) (LAPACK computation of the singular values and multiply square matrix of dimension $\min(M, N)$ by transpose of left singular vectors)
7. LAFSD (LAPACK singular values only of a dense matrix, using SCFBD and SDBQR)
8. LAFSD1 (LAPACK singular values and $\min(M, N)$ left singular vectors of a dense matrix, using SCFBD, SCGBR and SDBQR(L))
9. LAFSD(L) (LAPACK singular values and M left singular vectors of a dense matrix, using SCFBD, SCGBR and SDBQR(L))
10. LAFSD(R) (LAPACK singular values and N right singular vectors of a dense matrix, using SCFBD, SCGBR and SDBQR(R))
11. LAFSD(B) (LAPACK singular values, $\min(M, N)$ left singular vectors, and N right singular vectors of a dense matrix, using SCFBD, SCGBR and SDBQR(B))
12. LINSVD (LINPACK singular values only of a dense matrix using SVDG, to be compared to LAFSD)

13. LINSVD(1) (LINPACKsingular values and min(MN) left singular vectors of a dense matrix using SSVDC; to be compared to LAPSD(1))
14. LINSVD(L) (LINPACKsingular values and Mleft singular vectors of a dense matrix using SSVDC; to be compared to LAPSD(L))
15. LINSVD(R) (LINPACKsingular values and Nright singular vectors of a dense matrix using SSVDC; to be compared to LAPSD(R))
16. LINSVD(B) (LINPACKsingular values, min(MN) left singular vectors and Nright singular vectors of a dense matrix using SSVDC; to be compared to LAPSD(B)).

Five different matrix types are provided for timing the singular value decomposition routines. Matrix types 1-3 are of the form UDV , where U and V are orthogonal or unitary, and D is diagonal with entries

- evenly spaced entries from 1 down to ε with random signs (matrix type 1),
- geometrically spaced entries from 1 down to ε with random signs (matrix type 2), or
- "clustered" entries $1, \varepsilon, \dots, \varepsilon$ with random signs (matrix type 3).

Matrix type 4 has in each entry a random number drawn from $[-1, 1]$. Matrix type 5 is a nearly bidiagonal matrix, where the upper bidiagonal entries are $\exp(-2r \log \varepsilon)$ and the nonbidiagonal entries are $r\varepsilon$, where r is a uniform random number drawn from $[0, 1]$ (a different r for each entry).

An annotated example of an input file for timing the REAL singular value decomposition routines is shown below

```

SVD:  Data file for timing Singular Value Decomposition routines
7                                           Number of values of M and N
50  50 100 100 100 200 200                Values of M (row dimension)
50 100  50 100 200 100 200                Values of N (column dimension)
5                                           Number of values of parameters
1   16  32  48  64                        Values of NB (blocksize)
201 201 201 201 201                       Values of LDA (leading dimension)
0.0                                        Minimum time in seconds
4                                           Number of matrix types
1 2 3 4
SBD   T T T T T T T T T T T T T T T T

```

The first line of the input file must contain the characters SVD in columns 1-3. Lines 2-9 are read using list-directed input and specify the following values:

- line 2: The number of values of M and N
- line 3: The values of M the matrix row dimension
- line 3: The values of N the matrix column dimension
- line 4: The number of values of the parameters NB and LDA
- line 5: The values of NB the blocksize
- line 6: The values of LDA the leading dimension
- line 7: The minimum time in seconds that a routine will be timed
- line 8: $NYPES$, the number of matrix types to be used

If $0 < NYPES < 5$, then line 9 specifies $NYPES$ integer values which are the numbers of the matrix types to be used. The remaining lines specify a path name and the specific computations to be timed. For the SVD, the path names for the four data types are SBD, DBD, CBD, and ZBD. The (optional) characters after the path name indicate the computations to be timed, as in the input file for the nonsymmetric eigenvalue problem.

6.6 Timing the Generalized Nonsymmetric Eigenproblem

A separate input file drives the timing codes for the generalized nonsymmetric eigenproblem. The input file specifies

- N the matrix size
- six-tuples of parameter values (NB , NS , MXB , MNB , $MINBK$, LDA) specifying the block size NB , the number of shifts NS , the values of MXB the minimum blocksize MNB , the minimum blocksize $MINBK$ and the leading dimension LDA
- the test matrix types
- the routines or sequences of routines from LAPACK or EISPACK to be timed

The parameters NB , MNB , $MINBK$, NS , and MXB apply only to the QZ iteration routine xHQZ. A goal of this timing code is to determine the values of NB , NS , and MXB (as well as MNB and $MINBK$) which maximize the speed of the codes.

The number and size of the input values are limited by certain program maxima which are defined in PARAMETER statements in the main timing program.

Parameter	Description	Value
MXN	Maximum value for N , NB , NS , MXB , MNB , or $MINBK$	400
$IDMX$	Maximum value for LDA	420
MXN	Maximum number of values of N	12
$MXPM$	Maximum number of parameter sets (NB , NS , MXB , MNB , $MINBK$, LDA)	10

The computations that may be timed for the REAL version are

1. SCQRN (LAPACK reduction to generalized upper Hessenberg form without computing U or V .)

2. `SGHRD(Q)` (LAPACK reduction to generalized upper Hessenberg form, computing U but not V .)
3. `SGHRD(Z)` (LAPACK reduction to generalized upper Hessenberg form, computing V but not U .)
4. `SGHRD(Q, Z)` (LAPACK reduction to generalized upper Hessenberg form, computing U and V .)
5. `SHHQZ(F)` (LAPACK computation of generalized eigenvalues only of a pair of matrices in generalized Hessenberg form)
6. `SHHQZ(S)` (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form)
7. `SHHQZ(Q)` (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Q)
8. `SHHQZ(Z)` (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Z)
9. `SHHQZ(Q, Z)` (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Q and Z)
10. `STGEV(A, L)` (LAPACK computation of the the left generalized eigenvectors of a matrix pair in generalized Schur form)
11. `STGEV(B, L)` (LAPACK computation of the the left generalized eigenvectors of a matrix pair in generalized Schur form back transformed by Q)
12. `STGEV(A, R)` (LAPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form)
13. `STGEV(B, R)` (LAPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form back transformed by Z)
14. `QZHES(F)` (ELSPACK reduction to generalized upper Hessenberg form with `MAIZ=FALSE`, so V is not computed.)
15. `QZHES(T)` (ELSPACK reduction to generalized upper Hessenberg form with `MAIZ=TRUE`, so V is computed.)
16. `QZIT(F)` (QZIT followed by `QZAL` with `MAIZ=FALSE`: ELSPACK computation of generalized eigenvalues only of a pair of matrices in generalized Hessenberg form)
17. `QZIT(T)` (QZIT followed by `QZAL` with `MAIZ=TRUE`: ELSPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Z)
18. `QZMEC` (ELSPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form back transformed by Z)

- line 2: The number of values of N
- line 3: The values of N the matrix dimension
- line 4: Number of values of the parameters
- line 5: The values for NB, the blocksize
- line 6: The values for NS, the number of shifts
- line 7: The values for MNB, the multishift crossover point
- line 8: The values for MINB, determines minimum blocksize
- line 9: The values for MINBK, also determines minimum blocksize
- line 10: The values for the leading dimension LDA
- line 11: The minimum time (in seconds) that a subroutine will be timed. If TIMMIN is zero, each routine should be timed only once.
- line 12: NYPES, the number of matrix types to be used

If NYPES ≥ 4 , all the types are used. If $0 < \text{NYPES} < 4$, then line 13 specifies NYPES integer values, which are the numbers of the matrix types to be used. The remaining lines specify a path name and the specific routines to be timed. For the generalized nonsymmetric eigenvalue problem the path names for the four data types are SHG, CHG, DHG, and ZHG. A line to request all the routines in the REAL path has the form

```
SHG  T T T T T T T T T T T T T T T T T
```

where the first 3 characters specify the path name, and up to MAXYP nonblank characters may appear in columns 4-80. If the k th such character is 'T' or 't', the k th routine will be timed. If at least one but fewer than 18 nonblank characters are specified, the remaining routines will not be timed. If columns 4-80 are blank, all the routines will be timed, so the input line

```
SHG
```

is equivalent to the line above.

The output is in the form of a table which shows the absolute times in seconds, floating point operation counts, and megaflop rates for each routine over all relevant input parameters. For the SHCQZ routine, the table has one line for each different combination of NB, NS, MNB, MINB, and MINBK.

Acknowledgments

Jim Demmel of the University of California-Berkeley, Sven Hammarling of NAG Ltd., and Alan McKenney of the Courant Institute of Mathematical Sciences, New York University, also contributed to this report.

Appendix A: LAPACK Routines

In this appendix, we review the subroutine naming scheme for LAPACK as proposed in [3] and indicate by means of a table which subroutines are included in this release. We also list the driver routines, which are new since release 2.

Each subroutine name in LAPACK is a coded specification of the computation done by the subroutine. All names consist of six characters in the form TXXYY. The first letter, T, indicates the matrix data type as follows:

S	REAL
D	DOUBLE PRECISION
C	COMPLEX
Z	COMPLEX*16 (if available)

The next two letters, XX, indicate the type of matrix. Most of these two-letter codes apply to both real and complex routines; a few apply specifically to one or the other, as indicated below

BD	bi diagonal
GB	general band
GE	general (i.e. unsymmetric, in some cases rectangular)
GG	general matrices, generalized problem (i.e. a pair of general matrices)
GT	general tri diagonal
HB	(complex) Hermitian band
HE	(complex) Hermitian
HG	upper Hessenberg matrix, generalized problem (i.e., a Hessenberg and a triangular matrix)
HP	(complex) Hermitian, packed storage
HS	upper Hessenberg
OR	(real) orthogonal
OP	(real) orthogonal, packed storage
PB	symmetric or Hermitian positive definite band
PO	symmetric or Hermitian positive definite
PP	symmetric or Hermitian positive definite, packed storage
PT	symmetric or Hermitian positive definite tri diagonal
SB	(real) symmetric band
SP	symmetric, packed storage
ST	symmetric tri diagonal
SY	symmetric
TB	triangular band
TG	triangular matrices, generalized problem (i.e., a pair of triangular matrices)
TP	triangular, packed storage
TR	triangular (or in some cases quasi-triangular)
TZ	trapezoidal
UN	(complex) unitary
UP	(complex) unitary, packed storage

The last three characters, *YYY*, indicate the computation done by a particular subroutine. Included in this release are subroutines to perform the following computations:

BAK back transformation of eigenvectors after balancing
BAL permute and/or balance to isolate eigenvalues
BD reduce to bidiagonal form by orthogonal transformations
CON estimate condition number
EBZ compute selected eigenvalues by bisection
FIN compute selected eigenvectors by inverse iteration
EQR compute eigenvalues and/or the Schur form using the QR algorithm
EQU equilibrate a matrix to reduce its condition number
EQZ compute generalized eigenvalues and/or generalized Schur form by QZ method
ERF compute eigenvectors using the Pal-Walker-Kahan variant of the QL or QR algorithm
EXC compute eigenvectors from Schur factorization
EXC swap adjacent diagonal blocks in a quasi-upper triangular matrix
GR generate the orthogonal/unitary matrix from **GBD**
GR generate the orthogonal/unitary matrix from **GBD**
GQ generate the orthogonal/unitary matrix from **GLQF**
QL generate the orthogonal/unitary matrix from **GLQF**
QR generate the orthogonal/unitary matrix from **GLQF**
GQ generate the orthogonal/unitary matrix from **GLQF**
GST reduce a symmetric-definite generalized eigenvalue problem to standard form
GIR generate the orthogonal/unitary matrix from **xxxIBD**
HD reduce to upper Hessenberg form by orthogonal transformations
IQF compute an IQ factorization without pivoting
IQS compute a minimum norm solution using the IQ factorization
MR multiply by the orthogonal/unitary matrix from **GBD**
MR multiply by the orthogonal/unitary matrix from **GBD**
MQ multiply by the orthogonal/unitary matrix from **GLQF**
ML multiply by the orthogonal/unitary matrix from **GLQF**
MR multiply by the orthogonal/unitary matrix from **GLQF**
MQ multiply by the orthogonal/unitary matrix from **GLQF**
MR multiply by the orthogonal/unitary matrix from **xxxIBD**
QIF compute a QL factorization without pivoting
QIS solve a least squares problem using the QL factorization
QIF compute a QR factorization with column pivoting
QIF compute a QR factorization without pivoting
QIS solve a least squares problem using the QR factorization
RES refine initial solution returned by **TR** routines
RQF compute an RQ factorization without pivoting
RQS compute a minimum norm solution using the RQ factorization
SEN compute a basis and/or reciprocal condition number (sensitivity) of an invariant subspace
SNA estimate reciprocal condition numbers of eigenvalue/-vector pairs
SQR compute singular values and/or singular vectors using the QR algorithm

- SYL solve the Sylvester matrix equation
- TRD reduce a symmetric matrix to real symmetric tridiagonal form
- TRF compute a triangular factorization (LU, Cholesky, etc.)
- TRI compute inverse (based on triangular factorization)
- TRS solve systems of linear equations (based on triangular factorization)

Given these definitions, the following table indicates the LAPACK subroutines for the solution of systems of linear equations:

	GE	GB	GF	PO	PP	PB	PT	HE SY	HP SP	TR	TP	TB	UN CR
TRF	×	×	×	×	×	×	×	×	×				
TRS	×	×	×	×	×	×	×	×	×	×	×	×	
RES	×	×	×	×	×	×	×	×	×	×	×	×	
TRI	×			×	×			×	×	×	×		
CON	×	×	×	×	×	×	×	×	×	×	×	×	
EQU	×	×		×	×	×							
QPF	×												
QRF †	×												
QRS †	×												
QQR †													×
MQR †													×

†—also RQ, QL, and IQ

The following table indicates the LAPACK subroutines for finding eigenvalues and eigenvectors or singular values and singular vectors:

	GE	GG	HB	HG	TR	TG	HE SY	HP SP	HB SB	ST	PT	BD
HD	×	×										
TRD							×	×	×			
HD	×											
EQR			×							×	×	
HQZ				×								
EN			×							×		
ENC					×	×						
HZ										×		
HF										×		
SQR												×
SEN					×							
SNA					×							
SYL					×							
EXC					×							
BAL	×	×										
BAK	×	×										
GST							×	×				

Orthogonal/unitary transformation routines have also been provided for the reductions that use elementary transformations.

	UN	UP
	QR	QP
GR	×	
GR	×	×
GR	×	
MR	×	
MR	×	×
MR	×	

In addition, a number of driver routines are provided with this release. The naming convention for the driver routines is the same as for the LAPACK routines, but the last 3 characters **YYY** have the following meanings (note an 'X' in the last character position indicates a more expert driver):

- SV factor the matrix and solve a system of equations
- SVX equilibrate, factor, solve, compute error bounds and do iterative refinement, and estimate the condition number
- IS solve over- or underdetermined linear system using orthogonal factorizations
- ISX compute a minimum norm solution using a complete orthogonal factorization (using QR with column pivoting)
- ISS solve least squares problem using the SVD
- EV compute all eigenvalues and/or eigenvectors
- EXX compute selected eigenvalues and eigenvectors
- ES compute all eigenvalues, Schur form and/or Schur vectors
- ESX compute all eigenvalues, Schur form and/or Schur vectors and the conditioning of selected eigenvalues or eigenvectors
- GV compute generalized eigenvalues and/or generalized eigenvectors
- GS compute generalized eigenvalues, Schur form and/or Schur vectors
- SVD compute the SVD and/or singular vectors

The driver routines provided in LAPACK are indicated by the following table:

	GE	GB	GT	FO	FP	FB	PT	HE	HP	HB	ST
								SY	SP	SB	
SV	×	×	×	×	×	×	×	×	×		
SVX	×	×	×	×	×	×	×	×	×		
IS	×										
ISX	×										
ISS	×										
EV	×							×	×	×	×
EXX	×							×	×	×	×
ES	×										
ESX	×										
GV	×							×	×		
GS	×										
SVD	×										

Appendix B: LAPACK Auxiliary Routines

This appendix lists all of the auxiliary routines (except for the BLAS) that are called from the LAPACK routines. These routines are found in the directory LAPACK/SRC in the Unix version and in the files xxLAPACK and xLAPACK in the non-Unix version. Routines specified with an underscore as the first character are available in all four data types (S, D, C, and Z), except those marked (real), for which the first character may be 'S' or 'D', and those marked (complex), for which the first character may be 'C' or 'Z'.

Special subroutines:

XERBLA Error handler for the BLAS and LAPACK routines

Special functions:

ILAEN	INTEGER	Return block size and other parameters
LSAME	LOGICAL	Return .TRUE. if two characters are the same regardless of case
LSAMN	LOGICAL	Return .TRUE. if two character strings are the same regardless of case
SLAMH	REAL	Return single precision machine parameters
DLAMH	DOUBLE PRECISION	Return double precision machine parameters

Functions for computing norms:

_LANCB	General band matrix
_LANGE	General matrix
_LANGT	General tridiagonal matrix
_LANHB	(complex) Hermitian band matrix
_LANHE	(complex) Hermitian matrix
_LANHP	(complex) Hermitian packed matrix
_LANHS	Upper Hessenberg matrix
_LANSB	Symmetric band matrix
_LANSP	Symmetric packed matrix
_LANST	Symmetric tridiagonal matrix
_LANSY	Symmetric matrix
_LANIB	Triangular band matrix
_LANIP	Triangular packed matrix
_LANIR	Trapezoidal matrix

Extensions to the Level 1 and 2 BLAS:

CROT	Apply a plane rotation to a pair of complex vectors, where the cos is real and the sin is complex
CSROT	Apply a real plane rotation to a pair of complex vectors
ZDROT	Double precision version of CSROT
_SSYM	(complex) Symmetric matrix times vector
_SSPM	(complex) Symmetric packed matrix times vector

_SBMV (complex) Symmetric band matrix times vector
 _SYR (complex) Symmetric rank-1 update
 _SPR (complex) Symmetric rank-1 update of a packed matrix
 ICMAX Find the index of element whose real part has max. abs. value
 IZMAX Find the index of element whose real part has max. abs. value
 SCSUM Sum absolute values of a complex vector
 DCSUM Double precision version of SCSUM
 _RSCL (real) Scale a vector by the reciprocal of a constant
 CRSCL Scale a complex vector by the reciprocal of a real constant
 ZRSCL Double precision version of CRSCL

Level 2 BLAS versions of the block routines:

_GBF2 compute the LU factorization of a general band matrix
 _GBD2 reduce a general matrix to bidiagonal form
 _GHD2 reduce a square matrix to upper Hessenberg form
 _GEQ2 compute an LQ factorization without pivoting
 _GEQ2 compute a QL factorization without pivoting
 _GEQ2 compute a QR factorization without pivoting
 _GEQ2 compute an RQ factorization without pivoting
 _GEF2 compute the LU factorization of a general matrix
 _HGS2 (complex) reduce a Hermitian-definite generalized eigenvalue problem to standard form
 _HEID2 (complex) reduce a Hermitian matrix to real tridiagonal form
 _HEIF2 (complex) compute diagonal pivoting factorization of a Hermitian matrix
 _ORG2L (real) generate the orthogonal matrix fromxGEQLF
 _ORG2R (real) generate the orthogonal matrix fromxGEQRF
 _ORC2L (real) generate the orthogonal matrix fromxGEQLF
 _ORC2R (real) generate the orthogonal matrix fromxGEQRF
 _ORM2L (real) multiply by the orthogonal matrix fromxGEQLF
 _ORM2R (real) multiply by the orthogonal matrix fromxGEQRF
 _ORM2L (real) multiply by the orthogonal matrix fromxGEQLF
 _ORM2R (real) multiply by the orthogonal matrix fromxGEQRF
 _ORM2L (real) multiply by the orthogonal matrix fromxGEQLF
 _ORM2R (real) multiply by the orthogonal matrix fromxGEQRF
 _PBF2 compute the Cholesky factorization of a positive definite band matrix
 _POF2 compute the Cholesky factorization of a positive definite matrix
 _SYGS2 (real) reduce a symmetric-definite generalized eigenvalue problem to standard form
 _SYID2 (real) reduce a symmetric matrix to tridiagonal form
 _SYIF2 compute the diagonal pivoting factorization of a symmetric matrix
 _TRIF2 compute the inverse of a triangular matrix
 _UNG2L (complex) generate the unitary matrix fromxGEQLF
 _UNG2R (complex) generate the unitary matrix fromxGEQRF
 _UNC2L (complex) generate the unitary matrix fromxGEQLF
 _UNC2R (complex) generate the unitary matrix fromxGEQRF
 _UNM2L (complex) multiply by the unitary matrix fromxGEQLF
 _UNM2R (complex) multiply by the unitary matrix fromxGEQRF

`_UNM2` (complex) multiply by the unitary matrix from `CHQF`
`_UMR2` (complex) multiply by the unitary matrix from `CHQF`

Other LAPACK auxiliary routines:

`_LABD` (real) returns square root of underflow and overflow if exponent range is large
`_LABD` reduce `NB` rows or columns of a matrix to upper or lower bidiagonal form
`_LACN` estimate the norm of a matrix for use in condition estimation
`_LAPY` copy a matrix to another matrix
`_LAIV` perform complex division in real arithmetic
`_LAE2` compute eigenvalues of a 2 x 2 real symmetric or complex Hermitian matrix
`_LAEB` chases a bulge down an upper Hessenberg block
`_LAEBZ` compute and use the count of eigenvalues of a symmetric tridiagonal matrix
`_LAEIN` Use inverse iteration to find a specified right and/or left eigenvector of an upper Hessenberg matrix
`_LAEQU` perform an orthogonal similarity transformation to standardize a 2 by 2 diagonal block of a quasi-triangular matrix
`_LAEQZ` unblocked single-/double-shift version of QZ method
`_LAESY` (complex) Compute eigenvalues and eigenvectors of a complex symmetric 2 x 2 matrix
`_LAE2` Compute eigenvalues and eigenvectors of a 2 x 2 real symmetric or complex Hermitian matrix
`_LAEX` swap adjacent diagonal blocks in a quasi-upper triangular matrix
`_LAQ2` compute the eigenvalues of a 2 by 2 generalized eigenvalue problem with scaling to avoid over-/underflow
`_LACB` bulge-chasing for the multishift QZ method
`_LAGIF` factorizes the matrix $(T - \lambda I)$
`_LAGIM` matrix-vector product where the matrix is tridiagonal
`_LAGIS` solves a system of equations $(T - \lambda I)x = y$ where T is a tridiagonal matrix
`_LAHEF` (complex) compute part of the diagonal pivoting factorization of a Hermitian matrix
`_LAQRR` Find the Schur factorization of a Hessenberg matrix (modified version of HQR from EISPACK)
`_LAHD` reduce `NB` columns of a general matrix to Hessenberg form
`_LACI` apply one step of incremental condition estimation
`_LAN2` (real) Solve a 1 x 1 or 2 x 2 linear system
`_LARD` sort the elements of a vector in increasing or decreasing order
`_LAPIM` multiply a matrix by a symmetric tridiagonal matrix
`_LAP2` Compute square root of $X^{*2} + Y^{*2}$
`_LAP3` (real) Compute square root of $X^{*2} + Y^{*2} + Z^{*2}$
`_LAQCB` equilibrate a general band matrix
`_LAQCE` equilibrate a general matrix
`_LAQSB` equilibrate a symmetric band matrix
`_LAQSP` equilibrate a symmetric packed matrix

_LAQSY equilibrate a symmetric matrix
 _LAQTR solve a real or complex quasi-triangular system
 _LAR2V apply real plane rotations from both sides to a sequence
 of 2 by 2 real symmetric matrices
 _LARF apply (multiply by) an elementary reflector
 _LARFB apply (multiply by) a block reflector
 _LARFG generate an elementary reflector
 _LARFT form the triangular factor of a block reflector
 _LAREX unrolled version of xLARF
 _LARCV generate a vector of plane rotations
 _LARIG generate a plane rotation
 _LARIV apply a vector of plane rotations to a pair of vectors
 _LAS2 (real) Compute singular values of a 2 x 2 triangular matrix
 _LASCL scale a matrix by C1/CFROM
 _LASET initializes a matrix to BETA on the diagonal and ALPHA on
 the off-diagonals
 _LASR Apply a sequence of plane rotations to a rectangular matrix
 _LASSQ Compute a scaled sum of squares of the elements of a vector
 _LASV2 (real) Compute singular values and singular vectors of a 2 x 2 triangular
 matrix
 _LASW Perform a series of row interchanges
 _LASZ2 solve for a matrix X that satisfies the equation
 $TL * X + ISGN * X * TR = SCALE * B$
 _LASZF compute part of the diagonal pivoting factorization of a symmetric matrix
 _LAIBS solve a triangular band system with scaling to prevent overflow
 _LAIPS solve a packed triangular system with scaling to prevent overflow
 _LAIRD reduce NB rows and columns of a real symmetric or complex Hermitian
 matrix to tridiagonal form
 _LAIBS solve a triangular system with scaling to prevent overflow
 _LAIZM apply a Householder matrix generated by xTRQF to a matrix
 _LAU2 Unblocked version of _LAUM
 _LAUM Compute the product U*U or L*L (blocked version)
 _LAPY Add a multiple of a matrix to another matrix
 _LARO Initialize a rectangular matrix (usually to zero)

Appendix C: Operation Counts for the BLAS and LAPACK

In this appendix we reproduce in tabular form the formulas we have used to compute operation counts for the BLAS and LAPACK routines. In single precision, the functions `SCHE2`, `SCHE3`, `SCPAK` and `SCHL` return the operation counts for the Level 2 BLAS, Level 3 BLAS, LAPACK auxiliary routines, and LAPACK routines, respectively. All four functions are found in the directory `LAPACK/TIMING/LIN` in the Unix version and in `SCINTSTF` in the non-Unix version.

In the tables below we give operation counts for the single precision real dense and banded routines (the counts for the symmetric packed routines are the same as for the dense routines). Separate counts are given for multiplies (including divisions) and additions, and the total is the sum of these expressions. For the complex analogues of these routines, each multiplication would count as 6 operations and each addition as 2 operations, so the total would be different. For the double precision routines, we use the same operation counts as for the single precision real or complex routines.

Operation Counts for the Level 2 BLAS

The four parameters used in counting operations for the Level 2 BLAS are the matrix dimensions m and n and the upper and lower bandwidths k_u and k_l for the band routines (k if symmetric or triangular). An exact count also depends slightly on the values of the scaling factors α and β , since some common special cases (such as $\alpha = 1$ and $\beta = 0$) can be treated separately.

The count for `SGMV` from the Level 2 BLAS is as follows:

<code>SGMV</code>	multiplies:	$mn - (m - k_l - 1)(m - k_l)/2 - (n - k_u - 1)(n - k_u)/2$
	additions:	$mn - (m - k_l - 1)(m - k_l)/2 - (n - k_u - 1)(n - k_u)/2$
	total flops:	$2mn - (m - k_l - 1)(m - k_l) - (n - k_u - 1)(n - k_u)$

plus m multiplies if $\alpha \neq \pm 1$ and another m multiplies if $\beta \neq \pm 1$ or 0. The other Level 2 BLAS operation counts are shown in Table 12.

Operation Counts for the Level 3 BLAS

Three parameters are used to count operations for the Level 3 BLAS: the matrix dimensions m , n , and k . In some cases we also must know whether the matrix is multiplied on the left or right. An exact count depends slightly on the values of the scaling factors α and β , but in Table 13 we assume these parameters are always ± 1 or 0, since that is how they are used in the LAPACK routines.

Operation Counts for the LAPACK Routines

The parameters used in counting operations for the LAPACK routines are the matrix dimensions m and n , the upper and lower bandwidths k_u and k_l for the band routines (k if symmetric or triangular), and `NRHS`, the number of right hand sides in the solution phase. The operation counts for the LAPACK routines not listed here are not computed by a formula. In particular, the operation counts for the eigenvalue routines are problem dependent and are computed during execution of the timing program.

Level 2 BLAS	multipliations	additions	total flops
SGEMV 1,2	mn	mn	$2mn$
SSYMV 3,4	n^2	n^2	$2n^2$
SSEMV 3,4	$n(2k+1) - k(k+1)$	$n(2k+1) - k(k+1)$	$n(4k+2) - 2k(k+1)$
SIRMV 3,4,5	$n(n+1)/2$	$(n-1)n/2$	n^2
SIBMV 3,4,5	$n(k+1) - k(k+1)/2$	$nk - k(k+1)/2$	$n(2k+1) - k(k+1)$
SIRSV 5	$n(n+1)/2$	$(n-1)n/2$	n^2
SIBSV 5	$n(k+1) - k(k+1)/2$	$nk - k(k+1)/2$	$n(2k+1) - k(k+1)$
SGER 1	mn	mn	$2mn$
SSYR 3	$n(n+1)/2$	$n(n+1)/2$	$n(n+1)$
SSYR2 3	$n(n+1)$	n^2	$2n^2 + n$

- 1 - Uses m multiplies if $\alpha \neq \pm 1$
- 2 - Uses m multiplies if $\beta \neq \pm 1$ or 0
- 3 - Uses n multiplies if $\alpha \neq \pm 1$
- 4 - Uses n multiplies if $\beta \neq \pm 1$ or 0
- 5 - Less n multiplies if matrix is unit triangular

Table 12: Operation counts for the Level 2 BLAS

Level 3 BLAS	multipliations	additions	total flops
SGEMM	mkn	mkn	$2mkn$
SSYMM(SIDE='L')	m^2n	m^2n	$2m^2n$
SSYMM(SIDE='R')	mn^2	mn^2	$2mn^2$
SSYRK	$kn(n+1)/2$	$kn(n+1)/2$	$kn(n+1)$
SSYRK	kn^2	$kn^2 + n$	$2kn^2 + n$
SIRMM(SIDE='L')	$nm(m+1)/2$	$nm(m-1)/2$	nm^2
SIRMM(SIDE='R')	$mn(n+1)/2$	$mn(n-1)/2$	mn^2
SIRSM(SIDE='L')	$nm(m+1)/2$	$nm(m-1)/2$	nm^2
SIRSM(SIDE='R')	$mn(n+1)/2$	$mn(n-1)/2$	mn^2

Table 13: Operation counts for the Level 3 BLAS

LAPACK routines:

$$\begin{array}{l}
 \text{SCEIRF} \quad \text{mul ti pli cati ons:} \quad 1/2mn^2 - 1/6n^3 + 1/2mn - 1/2n^2 + 2/3n \\
 \text{addi ti ons:} \quad 1/2mn^2 - 1/6n^3 - 1/2mn + 1/6n \\
 \hline
 \text{total flops:} \quad mn^2 - 1/3n^3 - 1/2n^2 + 5/6n
 \end{array}$$

$$\begin{array}{l}
 \text{SCEIR} \quad \text{mul ti pli cati ons:} \quad 2/3n^3 + 1/2n^2 + 5/6n \\
 \text{addi ti ons:} \quad 2/3n^3 - 3/2n^2 + 5/6n \\
 \hline
 \text{total flops:} \quad 4/3n^3 - n^2 + 5/3n
 \end{array}$$

$$\begin{array}{l}
 \text{SCEIRS} \quad \text{mul ti pli cati ons:} \quad \mathbf{NBS} [n^2] \\
 \text{addi ti ons:} \quad \mathbf{NBS} [n^2 - n] \\
 \hline
 \text{total flops:} \quad \mathbf{NBS} [2n^2 - n]
 \end{array}$$

$$\begin{array}{l}
 \text{SFOIRF} \quad \text{mul ti pli cati ons:} \quad 1/6n^3 + 1/2n^2 + 1/3n \\
 \text{addi ti ons:} \quad 1/6n^3 - 1/6n \\
 \hline
 \text{total flops:} \quad 1/3n^3 + 1/2n^2 + 1/6n
 \end{array}$$

$$\begin{array}{l}
 \text{SFOIR} \quad \text{mul ti pli cati ons:} \quad 1/3n^3 + n^2 + 2/3n \\
 \text{addi ti ons:} \quad 1/3n^3 - 1/2n^2 + 1/6n \\
 \hline
 \text{total flops:} \quad 2/3n^3 + 1/2n^2 + 5/6n
 \end{array}$$

$$\begin{array}{l}
 \text{SFOIRS} \quad \text{mul ti pli cati ons:} \quad \mathbf{NBS} [n^2 + n] \\
 \text{addi ti ons:} \quad \mathbf{NBS} [n^2 - n] \\
 \hline
 \text{total flops:} \quad \mathbf{NBS} [2n^2]
 \end{array}$$

$$\begin{array}{l}
 \text{SFBIRF} \quad \text{mul ti pli cati ons:} \quad n(1/2k^2 + 3/2k + 1) - 1/3k^3 - k^2 - 2/3k \\
 \text{addi ti ons:} \quad n(1/2k^2 + 1/2k) - 1/3k^3 - 1/2k^2 - 1/6k \\
 \hline
 \text{total flops:} \quad n(k^2 + 2k + 1) - 2/3k^3 - 3/2k^2 - 5/6k
 \end{array}$$

$$\begin{array}{l}
 \text{SFBIRS} \quad \text{mul ti pli cati ons:} \quad \mathbf{NBS} [2nk + 2n - k^2 - k] \\
 \text{addi ti ons:} \quad \mathbf{NBS} [2nk - k^2 - k] \\
 \hline
 \text{total flops:} \quad \mathbf{NBS} [4nk + 2n - 2k^2 - 2k]
 \end{array}$$

$$\begin{array}{l}
\text{SSYIR} \quad \text{mul ti pli cati ons:} \quad 1/6n^3 + 1/2n^2 + 10/3n \\
\text{addi ti ons:} \quad 1/6n^3 - 1/6n \\
\hline
\text{total flops:} \quad 1/3n^3 + 1/2n^2 + 19/6n
\end{array}$$

$$\begin{array}{l}
\text{SSYIR} \quad \text{mul ti pli cati ons:} \quad 1/3n^3 + 2/3n \\
\text{addi ti ons:} \quad 1/3n^3 - 1/3n \\
\hline
\text{total flops:} \quad 2/3n^3 + 1/3n
\end{array}$$

$$\begin{array}{l}
\text{SSYIR} \quad \text{mul ti pli cati ons:} \quad \mathbb{N} \mathbb{B} [n^2 + n] \\
\text{addi ti ons:} \quad \mathbb{N} \mathbb{B} [n^2 - n] \\
\hline
\text{total flops:} \quad \mathbb{N} \mathbb{B} [2n^2]
\end{array}$$

$$\begin{array}{l}
\text{SCQRF or SCQLF} (m \geq n) \\
\text{mul ti pli cati ons:} \quad mn^2 - 1/3n^3 + mn + 1/2n^2 + 23/6n \\
\text{addi ti ons:} \quad mn^2 - 1/3n^3 + 1/2n^2 + 5/6n \\
\hline
\text{total flops:} \quad 2mn^2 - 2/3n^3 + mn + n^2 + 14/3n
\end{array}$$

$$\begin{array}{l}
\text{SCQRF or SCQLF} (m \leq n) \\
\text{mul ti pli cati ons:} \quad nm^2 - 1/3m^3 + 2nm - 1/2m^2 + 23/6m \\
\text{addi ti ons:} \quad nm^2 - 1/3m^3 + nm - 1/2m^2 + 5/6m \\
\hline
\text{total flops:} \quad 2nm^2 - 2/3m^3 + 3nm - m^2 + 14/3n
\end{array}$$

$$\begin{array}{l}
\text{SCRF or SCQLF} (m \geq n) \\
\text{mul ti pli cati ons:} \quad mn^2 - 1/3n^3 + mn + 1/2n^2 + 29/6n \\
\text{addi ti ons:} \quad mn^2 - 1/3n^3 + mn - 1/2n^2 + 5/6n \\
\hline
\text{total flops:} \quad 2mn^2 - 2/3n^3 + 2mn + 17/3n
\end{array}$$

$$\begin{array}{l}
\text{SCRF or SCQLF} (m \leq n) \\
\text{mul ti pli cati ons:} \quad nm^2 - 1/3m^3 + 2nm - 1/2m^2 + 29/6m \\
\text{addi ti ons:} \quad nm^2 - 1/3m^3 + 1/2m^2 + 5/6m \\
\hline
\text{total flops:} \quad 2nm^2 - 2/3m^3 + 2nm + 17/3n
\end{array}$$

$$\begin{array}{l}
\text{SRQR or SRQL} \\
\text{mul ti pli cati ons:} \quad 2mnk - (m+n)k^2 + 2/3k^3 + 2nk - k^2 - 5/3k \\
\text{addi ti ons:} \quad 2mnk - (m+n)k^2 + 2/3k^3 + nk - mk + 1/3k \\
\hline
\text{total flops:} \quad 4mnk - 2(m+n)k^2 + 4/3k^3 + 3nk - mk - k^2 - 4/3k
\end{array}$$

SCRLQ or SCRLQ

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 2mnk - (m+n)k^2 + 2/3k^3 + mk + nk - k^2 - 2/3k \\
 \text{addi ti ons:} \quad 2mnk - (m+n)k^2 + 2/3k^3 + mk - nk + 1/3k \\
 \hline
 \text{total flops:} \quad 4mnk - 2(m+n)k^2 + 4/3k^3 + 2mk - k^2 - 1/3k
 \end{array}$$

SCQLS

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad \mathbf{NHS} [2mn - 1/2n^2 + 5/2n] \\
 \text{addi ti ons:} \quad \mathbf{NHS} [2mn - 1/2n^2 + 1/2n] \\
 \hline
 \text{total flops:} \quad \mathbf{NHS} [4mn - n^2 + 3n]
 \end{array}$$

SCMLR, SCMLQ, SCMLL or SCMLQ (SIDE='L')

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 2nmk - nk^2 + 2nk \\
 \text{addi ti ons:} \quad 2nmk - nk^2 + nk \\
 \hline
 \text{total flops:} \quad 4nmk - 2nk^2 + 3nk
 \end{array}$$

SCMLR, SCMLQ, SCMLL or SCMLQ (SIDE='R')

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 2nmk - mk^2 + mk + nk - 1/2k^2 + 1/2k \\
 \text{addi ti ons:} \quad 2nmk - mk^2 + mk \\
 \hline
 \text{total flops:} \quad 4nmk - 2mk^2 + 2mk + nk - 1/2k^2 + 1/2k
 \end{array}$$

STRIR

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 1/6n^3 + 1/2n^2 + 1/3n \\
 \text{addi ti ons:} \quad 1/6n^3 - 1/2n^2 + 1/3n \\
 \hline
 \text{total flops:} \quad 1/3n^3 + 2/3n
 \end{array}$$

SCHRD

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 5/3n^3 + 1/2n^2 - 7/6n - 13 \\
 \text{addi ti ons:} \quad 5/3n^3 - n^2 - 2/3n - 8 \\
 \hline
 \text{total flops:} \quad 10/3n^3 - 1/2n^2 - 11/6n - 21
 \end{array}$$

SSYIRD

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 2/3n^3 + 5/2n^2 - 1/6n - 15 \\
 \text{addi ti ons:} \quad 2/3n^3 + n^2 - 8/3n - 4 \\
 \hline
 \text{total flops:} \quad 4/3n^3 + 3n^2 - 17/6n - 19
 \end{array}$$

SCBRD($m \geq n$)

$$\begin{array}{l}
 \text{mul ti pli ca ti ons:} \quad 2mn^2 - 2/3n^3 + 2n^2 + 20/3n \\
 \text{addi ti ons:} \quad 2mn^2 - 2/3n^3 + n^2 - mn + 5/3n \\
 \hline
 \text{total flops:} \quad 4mn^2 - 4/3n^3 + 3n^2 - mn + 25/3n
 \end{array}$$

SCBRD($m < n$)

exchange m and n in above

Appendix D: Caveats

In this appendix we list the machine-specific difficulties we have encountered in our own experience with LAPACK. We assume the user has installed the machine-specific routines correctly and that the Level 2 and 3 BLAS test programs have run successfully, so we do not list any warnings associated with those routines.

LAPACK is written in Fortran 77. Prospective users with only a Fortran 66 compiler will not be able to use this package.

Some IBM compilers do not recognize DBLE as a generic function as used in LAPACK. The software tools we use to convert from single precision to double precision convert REAL(C) and AIMG(C), where C is COMPLEX to DBL(Z) and DIMG(Z), where Z is COMPLEX*16, but IBM compilers use DREAL(Z) and DIMG(Z) to take the real and imaginary parts of a double complex number. IBM users can fix this problem by changing DBLE to DREAL when the argument of DBLE is COMPLEX*16.

IBM compilers do not permit the data type COMPLEX*16 in a FUNCTION subprogram definition. The data type on the first line of the function subprogram must be changed from COMPLEX*16 to DOUBLE COMPLEX for the following functions:

ZBEG from the Level 2 BLAS test program
ZBEG from the Level 3 BLAS test program
ZLAIV from the LAPACK library
ZLRND from the test matrix generator library
ZLAIM from the test matrix generator library
ZLAIM from the test matrix generator library

The functions ZDZIC and ZDZIU from the Level 1 BLAS are already declared DOUBLE COMPLEX. If that doesn't work, try the declaration COMPLEX FUNCTION*16.

If compiling on a SUN you may run out of space in /tmp (especially when compiling in the LAPACK/SRC directory). Thus, you will need to have your system administrator increase the size of your tmp partition.

We have not included test programs for the Level 1 BLAS. Users should therefore be aware of a common problem in machine-specific implementations of xNRM, the function to compute the 2-norm of a vector. The Fortran version of xNRM avoids underflow or overflow by scaling intermediate results, but some library versions of xNRM are not so careful about scaling. If xNRM is implemented without scaling intermediate results, some of the LAPACK test ratios may be unusually high, or a floating point exception may occur in the problems scaled near underflow or overflow. The solution to these problems is to link the Fortran version of xNRM with the test program.

Some of our test matrices are scaled near overflow or underflow but on the Crays, problems with the arithmetic near overflow and underflow forced us to scale by only the square root of overflow and underflow. The LAPACK auxiliary routine SLABAD (or DLABAD) is called to take the square root of underflow and overflow in cases where it could cause difficulties. We assume we are on a Cray if $\log_{10}(\text{overflow})$ is greater than 2000 and take the square root of underflow and overflow in this case. The test in SLABAD is as follows:

```
IF( LOG10( LARGE ) .GT. 2000. ) THEN
```

```

        SMALL = SQRT( SMALL )
        LARGE = SQRT( LARGE )
    END IF

```

Users of other machines with similar restrictions on the effective range of usable numbers may have to modify this test so that the square roots are done on their machine as well. In the Unix version, SLABAD is found in LAPACK/SRC and in the non-Unix version it is in SCLAUXF.

In the eigensystem timing program calls are made to the IINPACK and EISPACK equivalents of the LAPACK routines to allow a direct comparison of performance measures. In some cases we have increased the minimum number of iterations in the IINPACK and EISPACK routines to allow them to converge for our test problems, but even this may not be enough. One goal of the LAPACK project is to improve the convergence properties of these routines, so error messages in the output file indicating that a IINPACK or EISPACK routine did not converge should not be regarded with alarm.

In the eigensystem timing program we have equivalenced some work arrays and then passed them to a subroutine, where both arrays are modified. This is a violation of the Fortran 77 standard, which says "if a subprogram reference causes a dummy argument in the referenced subprogram to become associated with another dummy argument in the referenced subprogram neither dummy argument may become defined during execution of the subprogram"² If this causes any difficulties, the equivalence can be commented out as explained in the comments for the main eigensystem timing program.

We have added a lot of new software since the second release of LAPACK. Expect a few bugs. We will try to correct them before the public release.

²ANSI X3.9-1978, sec. 15.9.3.6

Appendix E: Estimated Time

In this appendix we list the execution times (in seconds) for the test and timing runs on a Sun SPARCstation and on one processor of a Gray VM. For timing, the small data sets were used for the SPARCstation and the large data sets for the Gray VM. The minimum time was set to 0.05 seconds for the Sun and 0.0 seconds for the Gray. The Fortran BLAS were used on the Sun and the Gray BLAS were used on the Gray, except for the complex tests and the timings with the input files CBAND.in and CIIME.in, for which the Fortran BLAS were used on the Gray as well. These times (particularly for the Gray) were obtained on a loaded machine and should be considered rough approximations.

On the Sun, the Fortran files were compiled with `f77 -O`. Tests were done using two versions of the Sun-4 compiler, with an older version, `ctrft2.f`, `chetrs.f`, `cchol.f`, `zchol.f`, and `dstein.f` (from the LAPACK libraries) had to be compiled without optimization. On the Gray, the Fortran files were compiled with `cf77 -Zp` using `cf77 5.0` and `UNICOS 7.0`.

Test/timing run	Data set	S	C	D	Z
Linear eqn testing	_test.in	439	2046	516	1921
Eigensystemtesting	rep.in	43	282	69	314
	sep.in	147	647	267	726
	svd.in	210	1224	347	1341
	_ec.in	340	40	434	44
	_ed.in	154	509	238	621
	_gg.in	91	555	152	610
	_sg.in	55	355	87	340
	_sb.in	5	36	6	34
	_bal.in	<1	<1	<1	<1
_bak.in	<1	<1	<1	<1	
Linear eqn timing	_tim.in	116	941	151	786
	_tim2.in	136	1180	188	983
	_band.in	38	322	56	266
BLAS timing	_blas.in1	120	925	139	779
	_blas.in2	35	182	36	158
	_blas.in3	35	178	36	158
Eigensystemtiming	_neptim.in	64	544	113	596
	_septim.in	41	887	80	907
	_svdtim.in	42	259	60	261
	_geptim.in	163	4239	322	1784

Table 14: Sun SPARCstation execution times (in seconds)

Test/timing run	Data set	S	C
Linear eqn testing	_test.in	44	101
Eigensystemtesting	rep.in	7	10
	sep.in	30	
	svd.in	41	56
	_ec.in	73	12
	_ed.in	42	
	_gg.in	18	24
	_sg.in	9	13
	_sb.in	1	1
	_bal.in	< 1	< 1
	_bak.in	< 1	< 1
Linear eqn timing	_TIME.in	475	3171
	_TIME2.in	374	1661
	_BAND.in	44	293
BLAS timing	_BLAS.in1	246	1445
	_BLAS.in2	41	283
	_BLAS.in3	45	319
Eigensystemtiming	_NEP.M.in	230	836
	_SEP.M.in	60	757
	_SVD.M.in	73	155
	_CEP.M.in	390	1001

Table 15: Gray **MP**– 1 processor, execution times (in seconds)

Appendix F: Implementation Guide for Non-Unix Systems

In the non-Unix version, the software is distributed on an unlabeled ASCII tape containing 200 files. All files consist of 80-character fixed-length records, with a maximum block size of 8000.

In the installation instructions, each file will be identified by the name given below and we recommend that you assign these names to the files when the tape is read. Files with names ending in 'F' contain Fortran source code; those with names ending in 'D' contain data for input to the test and timing programs. There are two sets of data for each timing run; data file 1 for small, non-vector computers, such as workstations, and data file 2 for large computers, particularly Cray-class supercomputers. All file names have at most eight characters.

The leading one or two characters of the file name generally indicates which of the different versions of the library or test program will use it:

A all four data types
SC REAL and COMPLEX
DE DOUBLE PRECISION and COMPLEX*16
S: REAL
D DOUBLE PRECISION
C COMPLEX
Z COMPLEX*16

Many of the files occur in groups of four, corresponding to the four different Fortran floating-point data types, and we will frequently refer to these files generically, using 'x' in place of the first letter (for example, xLASRF).

1. **README** List of files as in this section
2. **ALANF** LAPACK auxiliary routines used in all versions
3. **SCLANF** LAPACK auxiliary routines used in S and C versions
4. **DZLANF** LAPACK auxiliary routines used in D and Z versions
5. **SLASRF** LAPACK routines and auxiliary routines
6. **CLASRF**
7. **DLASRF**
8. **ZLASRF**
9. **ISAMEF** ISAME function to compare two characters
10. **TISAMEF** Test program for ISAME
11. **SLAMCH** SLAMCH function to determine machine parameters
12. **TSLAMCH** Test program for SLAMCH
13. **DLAMCH** DLAMCH function to determine machine parameters
14. **TDLAMCH** Test program for DLAMCH
15. **SECONF** SECONF function to return time in seconds
16. **TSECONF** Test program for SECONF

- 17. DSECNF DSECNF function to return time in seconds
- 18. TIDSECNF Test program for DSECNF

- 19. AILBLASF Auxiliary routines for the BLAS (and LAPACK)

- 20. SBLAS1F Level 1 BLAS
- 21. CBLAS1F
- 22. DBLAS1F
- 23. ZBLAS1F

- 24. SBLAS2F Level 2 BLAS
- 25. CBLAS2F
- 26. DBLAS2F
- 27. ZBLAS2F

- 28. SBLAS3F Level 3 BLAS
- 29. CBLAS3F
- 30. DBLAS3F
- 31. ZBLAS3F

- 32. SBLA2F Test program for Level 2 BLAS
- 33. CBLA2F
- 34. DBLA2F
- 35. ZBLA2F

- 36. SBLA2D Data file for testing Level 2 BLAS
- 37. CBLA2D
- 38. DBLA2D
- 39. ZBLA2D

- 40. SBLA3F Test program for Level 3 BLAS
- 41. CBLA3F
- 42. DBLA3F
- 43. ZBLA3F

- 44. SBLA3D Data file for testing Level 3 BLAS
- 45. CBLA3D
- 46. DBLA3D
- 47. ZBLA3D

- 48. SCAIGEN Auxiliary routines for the test matrix generators
- 49. DZAI GEN

- 50. SMIGEN Test matrix generators
- 51. CMIGEN

- 52. ~~DMGENF~~
- 53. ~~ZMGENF~~

- 54. ~~AMNISIF~~ Auxiliary routines for the linear equation test program

- 55. ~~SLINISIF~~ Test program for linear equation routines
- 56. ~~CLINISIF~~
- 57. ~~DLINISIF~~
- 58. ~~ZLINISIF~~

- 59. ~~SLINISID~~ Data file 1 for linear equation test program
- 60. ~~DLINISID~~
- 61. ~~CLINISID~~
- 62. ~~ZLINISID~~

- 63. ~~SBAKISID~~ Data file for testing ~~SCBAK~~
- 64. ~~DBAKISID~~ Data file for testing ~~DCBAK~~
- 65. ~~CBAKISID~~ Data file for testing ~~CCBAK~~
- 66. ~~ZBAKISID~~ Data file for testing ~~ZCBAK~~

- 67. ~~SBALISID~~ Data file for testing ~~SCBAL~~
- 68. ~~DBALISID~~ Data file for testing ~~DCBAL~~
- 69. ~~CBALISID~~ Data file for testing ~~CCBAL~~
- 70. ~~ZBALISID~~ Data file for testing ~~ZCBAL~~

- 71. ~~SECSID~~ Data file for testing eigencondition routines
- 72. ~~DECSID~~
- 73. ~~CECSID~~
- 74. ~~ZECSID~~

- 75. ~~SEDISID~~ Data file for testing nonsymmetric eigenvalue driver routines
- 76. ~~DEDISID~~
- 77. ~~CEDISID~~
- 78. ~~ZEDISID~~

- 79. ~~SCGISID~~ Data file for testing nonsymmetric generalized eigenvalue routines
- 80. ~~DCGISID~~
- 81. ~~CCGISID~~
- 82. ~~ZCGISID~~

- 83. ~~SSBISID~~ Data file for testing ~~SSBIRD~~
- 84. ~~DSBISID~~ Data file for testing ~~DSBIRD~~
- 85. ~~CSBISID~~ Data file for testing ~~CSBIRD~~
- 86. ~~ZSBISID~~ Data file for testing ~~ZSBIRD~~

- 87. **SSGSID** Data file for testing symmetric generalized eigenvalue routines
- 88. **BSGSID**
- 89. **CGSID**
- 90. **ZSGSID**

- 91. **AEIGSIF** Auxiliary routines for the eigensystemtest program
- 92. **SEIGSIF**
- 93. **DEIGSIF**

- 94. **SEIGSIF** Test program for eigensystem routines
- 95. **CEIGSIF**
- 96. **DEIGSIF**
- 97. **ZEIGSIF**

- 98. **NEPISID** Data file for testing Nonsymmetric Eigenvalue Problem
- 99. **SEPISID** Data file for testing Symmetric Eigenvalue Problem
- 100. **SMDSID** Data file for testing Singular Value Decomposition

- 101. **ALINIMF** Auxiliary routines for the linear system timing program
- 102. **SLINIMF**
- 103. **DLINIMF**

- 104. **SLINIMF** Timing program for linear equations
- 105. **CLINIMF**
- 106. **DLINIMF**
- 107. **ZLINIMF**

- 108. **SLINI1D** Data file 1 for timing dense square linear equations
- 109. **DLINI1D**
- 110. **CLINI1D**
- 111. **ZLINI1D**

- 112. **SRECI1D** Data file 1 for timing dense rectangular linear equations
- 113. **DRECI1D**
- 114. **CRECI1D**
- 115. **ZRECI1D**

- 116. **SENDI1D** Data file 1 for timing banded linear equations
- 117. **DENDI1D**
- 118. **CENDI1D**
- 119. **ZENDI1D**

- 120. **SBEI1MD** Data file 1-a for timing the BLAS
- 121. **DBEI1MD**
- 122. **CBEI1MD**

- 123. ZBEIMD

- 124. SBEIMD Data file 1-b for timing the BLAS
- 125. DBEIMD
- 126. CBEIMD
- 127. ZBEIMD

- 128. SBEIMD Data file 1-c for timing the BLAS
- 129. DBEIMD
- 130. CBEIMD
- 131. ZBEIMD

- 132. SLINMD Data file 2 for timing dense square linear equations
- 133. DLINMD
- 134. CLINMD
- 135. ZLINMD

- 136. SRECMD Data file 2 for timing dense rectangular linear equations
- 137. DRECMD
- 138. CRECMD
- 139. ZRECMD

- 140. SBNDMD Data file 2 for timing banded linear equations
- 141. DBNDMD
- 142. CBNDMD
- 143. ZBNDMD

- 144. SBEIMD Data file 2-a for timing the BLAS
- 145. DBEIMD
- 146. CBEIMD
- 147. ZBEIMD

- 148. SBEIMD Data file 2-b for timing the BLAS
- 149. DBEIMD
- 150. CBEIMD
- 151. ZBEIMD

- 152. SBEIMD Data file 2-c for timing the BLAS
- 153. DBEIMD
- 154. CBEIMD
- 155. ZBEIMD

- 156. AEIGMF Auxiliary routines for the eigensystem timing program
- 157. SCIGMF
- 158. DIIGMF

- 159. SEIGIME Timing program for the eigensystem routines
- 160. CEIGIME
- 161. DEIGIME
- 162. ZEIGIME

- 163. SEISRCF Instrumented LAPACK routines
- 164. CEISRCF
- 165. DEISRCF
- 166. ZEISRCF

- 167. SCISRCF Instrumented auxiliary routines used in S and C versions
- 168. DCISRCF Instrumented auxiliary routines used in D and Z versions

- 169. SCEPIMD Data file 1 for timing Generalized Nonsymmetric Eigenvalue Problem
- 170. SNEPIMD Data file 1 for timing Nonsymmetric Eigenvalue Problem
- 171. SSEPIMD Data file 1 for timing Symmetric Eigenvalue Problem
- 172. SVDIMD Data file 1 for timing Singular Value Decomposition

- 173. CEPIMD
- 174. CNEPIMD
- 175. SEPIMD
- 176. SVDIMD

- 177. DEPIMD
- 178. DNEPIMD
- 179. DSEPIMD
- 180. DSDIMD

- 181. ZEPIMD
- 182. ZNEPIMD
- 183. ZSEPIMD
- 184. ZSDIMD

- 185. SCEPIMD Data file 2 for timing Generalized Nonsymmetric Eigenvalue Problem
- 186. SNEPIMD Data file 2 for timing Nonsymmetric Eigenvalue Problem
- 187. SSEPIMD Data file 2 for timing Symmetric Eigenvalue Problem
- 188. SVDIMD Data file 2 for timing Singular Value Decomposition

- 189. CEPIMD
- 190. CNEPIMD
- 191. SEPIMD
- 192. SVDIMD

- 193. DEPIMD

- 194. DNEPMD
- 195. DSEPMMD
- 196. DSDMMD

- 197. ZCEPMD
- 198. ZNEPMD
- 199. ZSEPMMD
- 200. ZSDMMD

A Installing LAPACK on a non-Unix System

Installing and testing the non-Unix version of LAPACK involves the following steps:

1. Read the tape.
2. Test and install the machine-dependent routines.
3. Create the BLAS library, if necessary.
4. Run the Level 2 and 3 BLAS test programs.
5. Create the LAPACK library.
6. Create the library of test matrix generators.
7. Run the LAPACK test programs.
8. Run the LAPACK timing programs.
9. Send the results from steps 7 and 8 to the authors at the University of Tennessee.

A. 1 Read the Tape

Read the tape and assign names to the files, preferably as indicated in the beginning of this appendix. The first file (named `README`) is a list of the files in the order specified in the beginning of this appendix. You will need about 28 megabytes to read in the complete tape. On a Sun SPARC station, the libraries used 14 MB and the LAPACK executable files used 20 MB. In addition, the object files used 18 MB, but the object files can be deleted after creating the libraries and executable files. Your actual space requirements will be less if you do not use all four data types. The total space requirements including the object files is approximately 70 MB for all four data types.

A. 2 Test and Install the Machine-Dependent Routines.

There are five machine-dependent functions in the test and timing package, at least three of which must be installed. They are

LSAME	LOGICAL	Test if two characters are the same regardless of case
SLAMCH	REAL	Determine machine-dependent parameters
DLAMCH	DOUBLE PRECISION	Determine machine-dependent parameters
SECON	REAL	Return time in seconds from a fixed starting time
DSECON	DOUBLE PRECISION	Return time in seconds from a fixed starting time

If you are working only in single precision, you do not need to install DLAMCH and DSECON and if you are working only in double precision, you do not need to install SLAMCH and SECON. These five subroutines and their test programs are provided in the files LSAMEF and TLSAMEF, SLAMCHF and TSLAMCHF, etc.

A. 2. Installing LSAME

LSAME is a logical function with two character parameters, A and B. It returns .TRUE. if A and B are the same regardless of case, or .FALSE. if they are different. For example, the expression

```
LSAME( UPL0, 'U' )
```

is equivalent to

```
( UPL0.EQ.'U' ).OR.( UPL0.EQ.'u' )
```

The supplied version works correctly on all systems that use the ASCII code for internal representations of characters. For systems that use the EBCDIC code, one constant must be changed. For CDC systems with 6-12 bit representation, alternative code is provided in the comments. The test program in TLSAMEF tests all combinations of the same character in upper and lower case for A and B and two cases where A and B are different characters.

Compile LSAMEF and TLSAMEF and run the test program. If LSAME works correctly, the only message you should see is

```
ASCII character set
Tests completed
```

The working version of LSAME should be appended to the file ALIBLASE. This file, which also contains the error handler XERBLA, will be compiled with either the BLAS library in Section A3 or the LAPACK library in Section A5.

A. 2. Installing SLAMCH and DLAMCH

SLAMCH and DLAMCH are real functions with a single character parameter that indicates the machine parameter to be returned. The test program in TSLAMCHF simply prints out the different values computed by SLAMCH so you need to know something about what the values should be. For example, the output of the test program for SLAMCH on a Sun SPARC station is

```

Epsilon                =      5.96046E-08
Safe minimum           =      1.17549E-38
Base                   =      2.00000
Precision              =      1.19209E-07
Number of digits in mantissa =      24.0000
Rounding mode          =      1.00000
Minimum exponent       =     -125.000
Underflow threshold    =      1.17549E-38
Largest exponent       =      128.000
Overflow threshold     =      3.40282E+38
Reciprocal of safe minimum =      8.50706E+37

```

On a Gray machine, the safe minimum underflows its output representation and the overflow threshold overflows its output representation, so the safe minimum is printed as 0.00000 and overflow is printed as R. This is normal. If you would prefer to print a representable number, you can modify the test program to print SEMN*100. and RMAX/100. for the safe minimum and overflow thresholds.

Compile SLAMCH and TSLAMCH and run the test program. If the results from the test program are correct, save SLAMCH for inclusion in the LAPACK library. Repeat these steps with DLAMCH and TDLAMCH. If both tests were successful, go to Section A2.3.

If SLAMCH (or DLAMCH) returns an invalid value, you will have to create your own version of this function. The following options are used in LAPACK and must be set:

- 'B': Base of the machine
- 'E': Epsilon (relative machine precision)
- 'O': Overflow threshold
- 'P': Precision = Epsilon*Base
- 'S': Safe minimum (often same as underflow threshold)
- 'U': Underflow threshold

Some people may be familiar with RUMCH (DUMCH), a primitive routine for setting machine parameters in which the user must comment out the appropriate assignment statements for the target machine. If a version of RUMCH is on hand, the assignments in SLAMCH can be made to refer to RUMCH using the correspondence

```

SLAMCH('U') = RUMCH(1)
SLAMCH('O') = RUMCH(2)
SLAMCH('E') = RUMCH(3)
SLAMCH('B') = RUMCH(5)

```

The safe minimum returned by SLAMCH('S') is initially set to the underflow value, but if $1/(\text{overflow}) \geq (\text{underflow})$ it is recomputed as $(1/(\text{overflow})) * (1 + \epsilon)$, where ϵ is the machine precision.

A. 2. Installing SECOND and DSECOND

Both the timing routines and the test routines call `SECOND(DSECOND)`, a real function with no arguments that returns the time in seconds from some fixed starting time. Our version of this routine returns only “user time”, and not “user time + system time”. The version of `second` in `SECONF` calls `EIIME`, a Fortran library routine available on some computer systems. If `EIIME` is not available or a better local timing function exists, you will have to provide the correct interface to `SECOND` and `DSECOND` on your machine.

The test program in `TSECONF` performs a million operations using 5000 iterations of the `SANPY` operation $y := y + \alpha x$ on a vector of length 100. The total time and `mgaflops` for this test is reported, then the operation is repeated including a call to `SECOND` on each of the 5000 iterations to determine the overhead due to calling `SECOND`. Compile `SECONF` and `TSECONF` and run the test program. There is no single right answer, but the times in seconds should be positive and the `mgaflop` ratios should be appropriate for your machine. Repeat this test for `DSECONF` and `TDSECONF` and save `SECOND` and `DSECOND` for inclusion in the `LAPACK` library in Section A.5.

A. 3 Create the BLAS Library

Ideally, a highly optimized version of the `BLAS` library already exists on your machine. In this case you can go directly to Section A.4 to make the `BLAS` test programs. Otherwise, you must create a library using the files `xBLAS1F`, `xBLAS2F`, `xBLAS3F`, and `ALLBLASF`. You may already have a library containing some of the `BLAS`, but not all (Level 1 and 2, but not Level 3, for example). If so, you should use your local version of the `BLAS` wherever possible and, if necessary, delete the `BLAS` you already have from the provided files. The file `ALLBLASF` must be included if any part of `xBLAS2F` or `xBLAS3F` is used. Compile these files and create an object library.

A. 4 Run the BLAS Test Programs

Test programs for the Level 2 and 3 `BLAS` are in the files `xBLA2F` and `xBLA3F`. A test program for the Level 1 `BLAS` is not included, in part because only a subset of the original set of Level 1 `BLAS` is actually used in `LAPACK` and the old test program was designed to test the full set of Level 1 `BLAS`. The original Level 1 `BLAS` test program is available from netlib as `TOMS` algorithm 539.

- a) Compile the files `xBLA2F` and `xBLA3F` and link them to your `BLAS` library or libraries. Note that each program includes a special version of the error-handling routine `XERRLA`, which tests the error-exits from the Level 2 and 3 `BLAS`. On most systems this will take precedence at link time over the standard version of `XERRLA` in the `BLAS` library. If this is not the case (the symptom will be that the program stops as soon as it tries to test an error-exit), you must temporarily delete `XERRLA` from `ALLBLASF` and recompile the `BLAS` library.
- b) Each `BLAS` test program has a corresponding data file `xBLA2D` or `xBLA3D`. Associate this file with Fortran unit number 5.

- c) The name of the output file is indicated on the first line of each input file and is currently defined to be `SHLAE2.SUMM` for the REAL Level 2 BLAS, with similar names for the other files. If necessary, edit the name of the output file to ensure that it is valid on your system.
- d) Run the Level 2 and 3 BLAS test programs.

If the tests using the supplied data files were completed successfully, consider whether the tests were sufficiently thorough. For example, on a machine with vector registers, at least one value of N greater than the length of the vector registers should be used; otherwise, important parts of the compiled code may not be exercised by the tests. If the tests were not successful, either because the program did not finish or the test ratios did not pass the threshold, you will probably have to find and correct the problem before continuing. If you have been testing a system-specific BLAS library, try using the Fortran BLAS for the routines that did not pass the tests. For more details on the BLAS test programs, see [8 and [6].]

A. 5 Create the LAPACK Library

Compile the files `xLASRF` with `ALLANF` and create an object library. If you have compiled either the S or C version, you must also compile and include the files `SCLANF`, `SLAMCH`, and `SECNMF`, and if you have compiled either the D or Z version, you must also compile and include the files `DZLANF`, `DLAMCH`, and `DSECNF`. If you did not compile the file `ALLBLASF` and include it in your BLAS library as described in Section A.3, you must compile it now and include it in your LAPACK library.

A. 6 Create the Test Matrix Generator Library

Compile the files `xMGENF` and create an object library. If you have compiled either the S or C version, you must also compile and include the file `SCAIGENF`, and if you have compiled either the D or Z version, you must also compile and include the file `DZAIGENF`.

A. 7 Run the LAPACK Test Programs

There are two distinct test programs for LAPACK routines in each data type, one for the linear equations routines and one for the eigensystem routines. In each data type, there is one input file for testing the linear equation routines and ten input files for testing the eigenvalue routines. For more information on the test programs and how to modify the input files, see Section 5.

A. 7.1 Testing the Linear Equation Routines

- a) Compile the files `xLINISIF` and link them to your matrix generator library, your LAPACK library, and your BLAS library or libraries in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order).

- b) The data files for the linear equation test program are called xLINISID. For each of the test programs, associate the appropriate data file with Fortran unit number 5.
- c) The output file is written to Fortran unit number 6. Associate a suitably named file (e.g., SLINIST.OUT) with this unit number.
- d) Run the test program.
- e) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A7.2 Testing the Eigensystem Routines

- a) Compile the files xEIGSIF and link them to your matrix generator library, your LAPACK library, and your BLAS library or libraries in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order). If you have compiled either the S or C version, you must also compile and include the file SCIGSIF, and if you have compiled either the D or Z version, you must also compile and include the file ZIGSIF.
- b) There are ten sets of data files for the eigensystem test program xBAKISID, xBAFISID, xFCISID, xEDISID, xCGISID, xSBISID, xSGISID, NEPISID, SEPIISID, and SVDISID. Note that three of the input files (NEPISID, SEPIISID, and SVDISID) are used regardless of the data type of the test program. For each run of the test program, associate the appropriate data file with Fortran unit number 5.
- c) The output file is written to Fortran unit number 6. Associate suitably named files with this unit number (e.g., SNEPIST.OUT, SBAKIST.OUT, etc.).
- d) Run the test program.
- e) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A. 8 Run the LAPACK Timing Programs

There are two distinct timing programs for LAPACK routines in each data type, one for the linear equations routines and one for the eigensystem routines. The timing program for the linear equations routines is also used to time the BLAS. We encourage you to conduct these timing experiments in REAL and COMPLEX or in DOUBLE PRECISION and COMPLEX*16; it is not necessary to send timing results in all four data types.

Two sets of input files are provided, a small set and a large set. The small data sets are appropriate for a standard workstation or other non-vector machine. The large data sets are appropriate for supercomputers, vector computers, and high-performance workstations. We are mainly interested in results from the large data sets, and it is not necessary to run both the large and small sets. The values of N in the large data sets are about five times

larger than those in the small data set, and the large data sets use additional values for parameters such as the block size `NB` and the leading array dimension `LDA`. The small input files end with the four characters `'TIMD` and the large input files end with the characters `'TMD` (except for the `BLAS` timing files, see Section A.8.2).

We encourage you to obtain timing results with the large data sets, as this allows us to compare different machines. If this would take too much time, suggestions for paring back the large data sets are given in the instructions below. We also encourage you to experiment with these timing programs and send us any interesting results, such as results for larger problems or for a wider range of block sizes. The main programs are dimensioned for the large data sets, so the parameters in the main program may have to be reduced in order to run the small data sets on a small machine, or increased to run experiments with larger problems.

The minimum time each subroutine will be timed is set to 0.0 in the large data files and to 0.05 in the small data files, and on many machines this value should be increased. If the timing interval is not long enough, the time for the subroutine after subtracting the overhead may be very small or zero, resulting in megaflop rates that are very large or zero. (To avoid division by zero, the megaflop rate is set to zero if the time is less than or equal to zero.) The minimum time that should be used depends on the machine and the resolution of the clock.

For more information on the timing programs and how to modify the input files, see Section 6.

A8.1 Timing the Linear Equations Routines

Three input files are provided in each data type for timing the linear equation routines, one for square matrices, one for band matrices, and one for rectangular matrices. The small data sets are in `xLINIMD`, `xBNDIMD`, and `xRECTMD`, and the large data sets are in `xLINIMD`, `xBNDIMD`, and `xRECTMD`.

- a) Compile the files `xLINIME`, and link them to your LAPACK library and your BLAS library or libraries in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order). If you have compiled either the S or C version, you must also compile and include the file `SCNLSIE`, and if you have compiled either the D or Z version, you must also compile and include the file `DZLNLSIE`.
- b) Make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value, or to restrict the size of the tests if you are using a computer with performance in between that of a workstation and that of a supercomputer. The computational requirements can be cut in half by using only one value of `LDA`. If it is necessary to also reduce the matrix sizes or the values of the block size, corresponding changes should be made to the `BLAS` input files (see Section A.8.2).

Associate the appropriate input file with Fortran unit number 5.

- c) The output file is written to Fortran unit number 6. Associate a suitably named file with this unit number (e.g., SLIIMOUT, SBLIIMOUT, and SRECIIMOUT for the REAL version).
- e) Run the timing programs in each data type you are using for each of the three input files.
- f) Send the output files to the authors as directed in section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A8.2 Timing the BLAS

The linear equation timing programs are also used to time the BLAS. Three input files are provided in each data type for timing the Level 2 and 3 BLAS. These input files time the BLAS using the matrix shapes encountered in the LAPACK routines, and we will use the results to analyze the performance of the LAPACK routines. For the REAL version, the small data sets are SBLIIMD, SBLIIMB, and SBLIIMC and the large data sets are SBLIIMAD, SBLIIMBD, and SBLIIMCD. There are three sets of inputs because there are three parameters in the Level 3 BLAS, M , N , and K , and in most applications one of these parameters is small (on the order of the blocksize) while the other two are large (on the order of the matrix size). In SBLIIMD, M and N are large but K is small, while in SBLIIMB the small parameter is M and in SBLIIMC the small parameter is N . The Level 2 BLAS are timed only in the first data set, where K is also used as the bandwidth for the banded routines.

- a) Make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value. If you modified the values of N or NB in Section A.8.1, set M , N , and K accordingly. The large parameters among M , N , and K should be the same as the matrix sizes used in timing the linear equation routines, and the small parameter should be the same as the block sizes used in timing the linear equations routines. If necessary, the large data set can be simplified by using only one value of LDA.

Associate the appropriate input file with Fortran unit number 5.

- b) The output file is written to Fortran unit number 6. Associate a suitably named file with this unit number (e.g., SBLIIMACUT, SBLIIMBOUT, and SBLIIMCOUT for the three runs of the REAL version).
- c) Run the timing programs in each data type you are using for each of the three input files.
- d) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A.8.3 Timing the Eigensystem Routines

Four input files are provided in each data type for timing the eigensystem routines, one for the nonsymmetric eigenvalue problem, one for the symmetric eigenvalue problem, one for the singular value decomposition, and one for the generalized nonsymmetric eigenvalue problem. For the REAL version, the small data sets are SCEPIMD, SNEPIMD, SSEPIMD, and SSVDIMD and the large data sets are SCPEIMD, SNEPEIMD, SSEPEIMD, and SSVDEIMD. Each of the four input files reads a different set of parameters and the format of the input is indicated by a 3-character code on the first line.

The timing program for eigenvalue/singular value routines accumulates the operation count as the routines are executing using special instrumented versions of the LAPACK routines. The first step in compiling the timing program is therefore to make a library of the instrumented routines.

- a) Compile the files xEIGSRC and create an object library. If you have compiled either the S or C version, you must also compile and include the file SCISRC, and if you have compiled either the D or Z version, you must also compile and include the file DZISRC. If you did not compile the file ALBLASF and include it in your BLAS library as described in Section A.3, you must compile it now and include it in the instrumented LAPACK library.
- b) Compile the files xEIGIME with AEIGIME and link them to your test matrix generator library, the instrumented LAPACK library created in the previous step, your LAPACK library from Section A.5, and your BLAS library in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order). If you have compiled either the S or C version, you must also compile and include the file SCIGIME, and if you have compiled either the D or Z version, you must also compile and include the file DZIGIME.
- c) Make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value, or to restrict the number of tests if you are using a computer with performance in between that of a workstation and that of a supercomputer. Instead of decreasing the matrix dimensions to reduce the time, it would be better to reduce the number of matrix types to be timed, since the performance varies more with the matrix size than with the type. For example, for the nonsymmetric eigenvalue routines, you could use only one matrix of type 4i instead of four matrices of types 1, 3, 4, and 6. See Section 6 for further details.
Associate the appropriate input file with Fortran unit number 5.
- d) The output file is written to Fortran unit number 6. Associate a suitably named file with this unit number (e.g., SCPEIMOUT, SNEPEIMOUT, SSEPEIMOUT, and SSVDEIMOUT for the four runs of the REAL version).
- e) Run the programs in each data type you are using with the four data sets.
- f) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A.9 Send the Results to Tennessee

Congratulations! You have now finished installing and testing LAPACK. Your participation is greatly appreciated. If possible, results and comments should be sent by electronic mail to

sost@cs.utk.edu

Otherwise, results may be submitted either by sending the authors a hard copy of the output files or by returning the distribution tape with the output files stored on it.

We encourage you to make the LAPACK library available to your users and provide us with feedback from their experiences. You should make it clear that this software is still under development, and parts of it may be changed before the project is completed. The changes may affect the calling sequences of some routines, so the public release of LAPACK is not guaranteed to be compatible with this version.

If you would like to do more, please contact us so that we may coordinate your efforts with the development of the final test release of LAPACK. One option is to look at ways to improve the performance of LAPACK on your machine. If you do not have optimized BLAS, tuning the BLAS would likely have a dramatic effect on performance. Other suggestions on fine-tuning specific algorithms are also welcome. For example, one of our test sites noticed that the row interchanges in the LU factorization routine SGEHRF were degrading performance on the IBM3090 because of the non-unit stride in SSWP [2]. In response we added the auxiliary routine SLASWP to interchange a block of rows, so that users of the IBM3090 could easily replace this routine with one in which the row interchanges are applied to one column at a time.

References

- [1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *Preliminary LAPACK User's Guide*, University of Tennessee, July 1991.
- [2] E. Anderson and J. Dongarra, *LAPACK Working Note 16: Results from the Initial Release of LAPACK* University of Tennessee, CS-89-89, November 1989.
- [3] C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, *LAPACK Working Note #5: Provisional Contents*, Argonne National Laboratory, AN-88-38, September 1988.
- [4] Z. Bai, J. Demmel, and A. McKenney, *LAPACK Working Note #3: On the Conditioning of the Nonsymmetric Eigenvalue Problem: Theory and Software*, University of Tennessee, CS-89-86, October 1989.
- [5] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, 16, 1:1-17, March 1990.
- [6] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Program," *ACM Trans. Math. Soft.*, 16, 1:18-28, March 1990.

- [7] J. Dongarra, J. Di Croz, S. Hammarling, and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, 14, 1:1-17, March 1988.
- [8] J. Dongarra, J. Di Croz, S. Hammarling, and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms: Model Implementation and Test Programs," *ACM Trans. Math. Soft.*, 14, 1:18-32, March 1988.
- [9] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Trans. Math. Soft.*, 5, 3:308-323, September 1979.