LAPACK Working Note 18
Implementation Guide for LAPACK *

Edward Anderson and Jack Dongarra
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-1301

March 26, 1990

Abstract

This working note describes how to install and test the second test release of LAPACK. 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. This document supersedes LAPACK Working Note 10, the implementation guide for the first test release.

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 and test the second test release of LAPACK. LAPACK is still under development and all of the routines presented at this time should be regarded as preliminary versions. This release is being made available only to our test sites and is intended only for testing, and not for general distribution. We expect the testing to reveal weaknesses in the design, and we plan to modify routines to correct any deficiencies.

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 guide combines the instructions for the Unix and non-Unix versions of the LAPACK test package, so most installers will not have to read every section. The following sections

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

describe the installation process and should be considered required reading:

Unix version:	Sections 1, 2, 3.5, and 4
Non-Unix version:	Sections 1, 2, and 5

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 Section 5 for the non-Unix version.

For users desiring additional information, Sections 6 and 7 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.

Release 2 of LAPACK includes updates of all of the software from Release 1, with the following additions:

- Routines for the matrix eigenvalue problem, including the reductions to bidiagonal, tridiagonal, or upper Hessenberg form, some of the routines for finding eigenvalues/eigenvectors and singular values/singular vectors, test programs, and timing programs
- New block factorization algorithms for some of the linear equations routines, including SGBTRF (LU factorization of a general band matrix), SSYTRF (factorization of a symmetric indefinite matrix), and SGELQF (LQ factorization of an $m \times n$ matrix for $m \leq n$)
- Iterative refinement of solutions obtained using each of the factorization routines for linear equations

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 the first release version of LAPACK.

This release contains only some of the routines that will be part of LAPACK; for a complete list of the proposed contents, see [2].

We have planned one more test release of LAPACK before the public release in 1991. The third test release will be in the fall of 1990.

2 Tape Format

The software for LAPACK is distributed in the form of a tape 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.

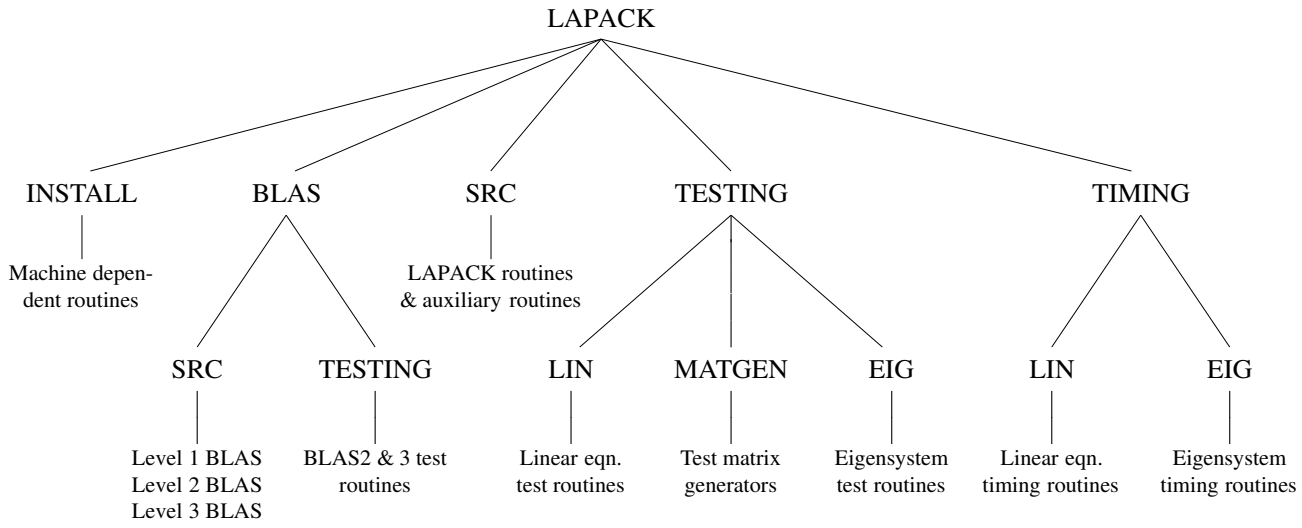

Figure 1: Unix organization of LAPACK

2.1 Unix Version

In the Unix version, the software is distributed on a *tar* tape containing 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 at that directory level.

2.2 Non-Unix version

In the non-Unix version, the software is distributed on an unlabeled ASCII tape containing 161 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 test and 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 programs will use it:

A: all four data types
SC: REAL and COMPLEX
DZ: 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, xLASRCF).

1.	README	List of files as in this section
2.	ALLAUXF	LAPACK auxiliary routines used in all versions
3.	SCLAUXF	LAPACK auxiliary routines used in S and C versions
4.	DZLAUXF	LAPACK auxiliary routines used in D and Z versions
5.	SLASRCF	LAPACK routines and auxiliary routines
6.	CLASRCF	
7.	DLASRCF	
8.	ZLASRCF	
9.	LSAMEF	LSAME: function to compare two characters
10.	TLSAMEF	Test program for LSAME
11.	SLAMCHF	SLAMCH: function to determine machine parameters
12.	TSLAMCHF	Test program for SLAMCH
13.	DLAMCHF	DLAMCH: function to determine machine parameters
14.	TDLAMCHF	Test program for DLAMCH
15.	SECONDF	SECOND: function to return time in seconds
16.	TSECONDF	Test program for SECOND
17.	DSECNDF	DSECND: function to return time in seconds
18.	TDSECNDF	Test program for DSECND
19.	ALLBLASF	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. SBLAT2F Test program for Level 2 BLAS
33. CBLAT2F
34. DBLAT2F
35. ZBLAT2F

36. SBLAT2D Data file for testing Level 2 BLAS
37. CBLAT2D
38. DBLAT2D
39. ZBLAT2D

40. SBLAT3F Test program for Level 3 BLAS
41. CBLAT3F
42. DBLAT3F
43. ZBLAT3F

44. SBLAT3D Data file for testing Level 3 BLAS
45. CBLAT3D
46. DBLAT3D
47. ZBLAT3D

48. SCATGENF Auxiliary routines for the test matrix generators
49. DZATGENF

50. SMATGENF Test matrix generators
51. CMATGENF
52. DMATGENF
53. ZMATGENF

54. ALINTSTF Auxiliary routines for the linear equation test program

55. SLINTSTF Test program for linear equation routines
56. CLINTSTF
57. DLINTSTF
58. ZLINTSTF

59. SLINTSTD Data file 1 for linear equation test program
60. DLINTSTD
61. CLINTSTD
62. ZLINTSTD

63. SLINTS2D Data file 2 for linear equation test program
64. DLINTS2D
65. CLINTS2D
66. ZLINTS2D

67. AEIGTSTF Auxiliary routines for the eigensystem test program
68. SCIGTSTF
69. DZIGTSTF

70. SEIGTSTF Test program for eigensystem routines
71. CEIGTSTF
72. DEIGTSTF
73. ZEIGTSTF

74. NEPTSTD Data file 1 for testing Nonsymmetric Eigenvalue Problem
75. SEPTSTD Data file 1 for testing Symmetric Eigenvalue Problem
76. SVDTSTD Data file 1 for testing Singular Value Decomposition

77. NEPTS2D Data file 2 for testing Nonsymmetric Eigenvalue Problem
78. SEPTS2D Data file 2 for testing Symmetric Eigenvalue Problem
79. SVDTS2D Data file 2 for testing Singular Value Decomposition

80. ALINTIMF Auxiliary routines for the linear system timing program
81. SCINTIMF
82. DZINTIMF

83. SLINTIMF Timing program for linear equations
84. CLINTIMF
85. DLINTIMF
86. ZLINTIMF

87. SLINTIMD Data file 1 for timing dense linear equations
88. DLINTIMD
89. CLINTIMD
90. ZLINTIMD

91. SBNDTIMD Data file 1 for timing banded linear equations
92. DBNDTIMD
93. CBNDTIMD
94. ZBNDTIMD

95. SBLTIMAD Data file 1-a for timing the BLAS
96. DBLTIMAD
97. CBLTIMAD

98. ZBLTIMAD

99. SBLTIMBD Data file 1-b for timing the BLAS
100. DBLTIMBD
101. CBLTIMBD
102. ZBLTIMBD

103. SBLTIMCD Data file 1-c for timing the BLAS
104. DBLTIMCD
105. CBLTIMCD
106. ZBLTIMCD

107. SLINTM2D Data file 2 for timing dense linear equations
108. DLINTM2D
109. CLINTM2D
110. ZLINTM2D

111. SBNDTM2D Data file 2 for timing banded linear equations
112. DBNDTM2D
113. CBNDTM2D
114. ZBNDTM2D

115. SBLTM2AD Data file 2-a for timing the BLAS
116. DBLTM2AD
117. CBLTM2AD
118. ZBLTM2AD

119. SBLTM2BD Data file 2-b for timing the BLAS
120. DBLTM2BD
121. CBLTM2BD
122. ZBLTM2BD

123. SBLTM2CD Data file 2-c for timing the BLAS
124. DBLTM2CD
125. CBLTM2CD
126. ZBLTM2CD

127. AEIGTIMF Auxiliary routines for the eigensystem timing program
128. SCIGTIMF
129. DZIGTIMF

130. SEIGTIMF Timing program for the eigensystem routines
131. CEIGTIMF
132. DEIGTIMF
133. ZEIGTIMF

134. SEIGSRCF Instrumented LAPACK routines and auxiliary routines
135. CEIGSRCF
136. DEIGSRCF
137. ZEIGSRCF

138. SNEPTIMD Data file 1 for timing Nonsymmetric Eigenvalue Problem
139. SSEPTIMD Data file 1 for timing Symmetric Eigenvalue Problem
140. SSVDTIMD Data file 1 for timing Singular Value Decomposition

141. CNEPTIMD
142. CSEPTIMD
143. CSVDTIMD

144. DNEPTIMD
145. DSEPTIMD
146. DSVDTIMD

147. ZNEPTIMD
148. ZSEPTIMD
149. ZSVDTIMD

150. SNEPTM2D Data file 2 for timing Nonsymmetric Eigenvalue Problem
151. SSEPTM2D Data file 2 for timing Symmetric Eigenvalue Problem
152. SSVDTM2D Data file 2 for timing Singular Value Decomposition

153. CNEPTM2D
154. CSEPTM2D
155. CSVDTM2D

156. DNEPTM2D
157. DSEPTM2D
158. DSVDTM2D

159. ZNEPTM2D
160. ZSEPTM2D
161. ZSVDTM2D

3 Overview of Tape Contents

Most routines in LAPACK occur in four versions: REAL, DOUBLE PRECISION, COMPLEX, and COMPLEX*16. The first three versions (REAL, DOUBLE PRECISION, and COMPLEX) are written in standard Fortran 77 and are completely portable; the COMPLEX*16 version is provided for those compilers which allow this data type. For convenience, we often refer to routines by their single precision names; the leading 'S' can be

replaced by a 'D' for double precision, a 'C' for complex, or a 'Z' for complex*16. For LAPACK use and testing you must decide which version(s) of the package you intend to install at your site (for example, REAL and COMPLEX on a Cray computer or DOUBLE PRECISION and COMPLEX*16 on an IBM computer).

3.1 LAPACK Routines and Auxiliary Routines

A slight distinction is made between LAPACK routines and LAPACK auxiliary routines. An LAPACK routine is a subroutine to perform a distinct algorithmic task, such as computing the LU decomposition of an $m \times n$ matrix or finding the eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm. The LAPACK routines are described in [2] and follow the naming conventions given there. An LAPACK auxiliary routine is a subroutine to perform a specific task which is called from one of the LAPACK routines. The tasks performed by the auxiliary routines are usually simpler and may be applicable in more than one context. Most auxiliary routines have the prefix xLA; exceptions are our extensions to the Level 1 and 2 BLAS, which have BLAS-type names, and the special routines LSAME, ENVIR, XENVIR, and XERBLA.

For a complete list of the LAPACK routines in this release, see Appendix A. For a complete list of the LAPACK auxiliary routines, see Appendix B. Further details on the scope of the LAPACK project are available in [2].

3.2 Level 1, 2, and 3 BLAS

The BLAS are a set of Basic Linear Algebra Subprograms that perform vector-vector, matrix-vector, and matrix-matrix operations. LAPACK is designed around the Level 1, 2, and 3 BLAS, and nearly all of the parallelism in the LAPACK routines is contained in the BLAS. Therefore, the key to getting good performance from LAPACK lies in having an efficient version of the BLAS optimized for your particular machine. If you have access to a library containing optimized versions of some or all of the BLAS, you should certainly use it (but be sure to run the BLAS test programs). If an optimized library of the BLAS is not available, Fortran source code for the Level 1, 2, and 3 BLAS is provided on the tape. Users should not expect too much from the Fortran BLAS; these versions were written to define the basic operations and do not employ the standard tricks for optimizing Fortran code.

The formal definitions of the Level 1, 2, and 3 BLAS are in [7], [5], and [3]. Copies of the BLAS Quick Reference card are available from the authors.

3.3 LAPACK Test Routines

This release contains two distinct test programs for LAPACK routines in each data type. One test program tests the routines for solving linear equations and linear least squares problems (as in the first release) and the other tests routines for the matrix eigenvalue problem. The routines for generating test matrices are used by both test programs and are separated from the other test routines.

3.4 LAPACK Timing Routines

This release also contains two distinct timing programs for the LAPACK routines in each data type. One timing program can be used to gather performance data in megaflops on the routines for solving linear equations and linear least squares problems, and also on the BLAS. The operation counts used in computing the megaflop rates are computed from a formula. The other timing program is used with the eigensystem routines and returns the execution time, number of floating point operations, and megaflop rate for each of the requested subroutines. In this program, the number of operations is computed while the code is executing using special instrumented versions of the LAPACK subroutines.

3.5 makefiles for Unix Users

In the Unix version, the libraries and test programs are created using the `makefile` in each directory. Target names are supplied for each of the four data types and are called `single, double, complex,` and `complex16`. To create a library from one of the files called `makefile`, you simply type `make` followed by the data types desired. Here are some examples:

```
make single

make double complex16

make single double complex complex16
```

Alternatively,

```
make
```

without any options creates a library of all four data types. The `make` command can be run more than once to add another data type to the library if necessary.

Similarly, the makefiles for the test routines create separate test programs for each data type. These programs can be created one at a time:

```
make single

make double

. . .
```

or all at once:

```
make single double complex complex16
```

where the last command is equivalent to typing `make` by itself. In the case of the BLAS test programs, where the makefile has a name other than `makefile`, the `-f` option must be added to specify the file name, as in the following example:

```
make -f makeblat2 single
```

The makefiles used to create libraries call `ranlib` after each `ar` command. Some computers (for example, CRAY computers running UNICOS) do not require `ranlib` to be run after creating a library. On these systems, references to `ranlib` should be commented out or removed from the makefiles in `LAPACK/SRC`, `LAPACK/BLAS/SRC`, `LAPACK/TESTING/MATGEN`, and `LAPACK/TIMING/EIG/EIGSRC`.

4 Installing LAPACK on a Unix System

Installing and testing the 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.

4.1 Read the Tape

To unload the tape, type one of the following commands (the device name may be different at your site):

`tar xvf /dev/rst0` (cartridge tape), or

`tar xvf /dev/rmt8` (9-track tape)

This will create a top-level directory called `LAPACK`. You will need about 12 megabytes to read in the complete tape. On a Sun SPARCstation, the libraries used 3.9 MB and the LAPACK executable files used 8.7 MB. In addition, the object files used 6.5 MB, but the object files can be deleted after creating the libraries and executable files. The total space requirements including the object files is approximately 31 MB for all four data types.

4.2 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
SECOND	REAL	Return time in seconds from a fixed starting time
DSECND	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 DSECND, and if you are working only in double precision, you do not need to install SLAMCH and SECOND.

These five subroutines are provided on the tape in `LAPACK/INSTALL`, along with five test programs and a makefile. To compile the five test programs, go to `LAPACK/INSTALL` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine. Then type `make` to create test programs called `testlsame, testslamch, testdlamch, testsecond,` and `testdsecnd`. The expected results of each test program are described below.

4.2.1 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( UPLO, 'U' )
```

is equivalent to

```
( UPLO.EQ.'U' ).OR.( UPLO.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 `lsametst.f` 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.

Run the test program by typing `testlsame`. If LSAME works correctly, the only message you should see is `Tests completed`. Once LSAME is working, copy the file `lsame.f` to both `LAPACK/BLAS/SRC` and `LAPACK/SRC`. The function LSAME is needed by both the BLAS and LAPACK, so it is safer to have it in both libraries as long as this does not cause trouble in the link phase when both libraries are used.

4.2.2 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 `slamchtst.f` 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 SPARCstation is

```
Epsilon                      =     5.96046E-08
Safe minimum                 =     1.17549E-38
Base                         =     2.00000
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
```

Values of 0 or NaN for any of the parameters are obvious indicators that something has gone wrong. Suspect results should be documented and reported to the authors.

Run the test program by typing `testslamch`. If the results from the test program are correct, copy `slamch.f` to `LAPACK/SRC` for inclusion in the LAPACK library. Do the same for DLAMCH and the test program `testdlamch`. If both tests were successful, go to Section 4.2.3.

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:

'U': Underflow threshold

'S': Safe minimum (often same as underflow threshold)

'O': Overflow threshold

'E': Epsilon (relative machine precision)

'B': Base of the machine

Some people may be familiar with R1MACH (D1MACH), 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 R1MACH is on hand, the assignments in SLAMCH can be made to refer to R1MACH using the correspondence

SLAMCH('U') = R1MACH(1)

SLAMCH('O') = R1MACH(2)

SLAMCH('E') = R1MACH(3)

SLAMCH('B') = R1MACH(5)

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

4.2.3 Installing SECOND and DSECND

Both the timing routines and the test routines call SECOND (DSECND), 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 `second.f` calls ETIME, a Fortran library routines available on some computer systems. If ETIME is not available or a better local timing function exists, you will have to provide the correct interface to SECOND and DSECND on your machine.

The test program in `secondtst.f` performs a million operations using 5000 iterations of the SAXPY operation $y := y + \alpha x$ on a vector of length 100. The total time and megaflops 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. Run the test program by typing `testsecond` (or `testdsecnd`). There is no single right answer, but the times in seconds should be positive and the megaflop ratios should be appropriate for your machine. The working versions of SECOND and DSECND should be copied to `LAPACK/SRC` for inclusion in the LAPACK library.

4.3 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 4.4 to make the BLAS test programs. 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.

a) Go to `LAPACK/BLAS/SRC` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine. If you already have some of the BLAS, comment out the lines defining the BLAS you have.

b) Type `make` followed by the data types desired, as in the examples of Section 3.5. The make command can be run more than once to add another data type to the library if necessary.

The BLAS library is created in `LAPACK/blas.a` and not in the current directory.

4.4 Run the BLAS Test Programs

Test programs for the Level 2 and 3 BLAS are in the directory `LAPACK/BLAS/TESTING`. 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.

a) To make the Level 2 BLAS test programs, go to `LAPACK/BLAS/TESTING` and edit the makefile called `makeblat2`. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and desired load options for your machine. If you are not using the Fortran BLAS, define `BLAS` to point to your system's BLAS library, instead of `../../blas.a`.

b) Type `make -f makeblat2` followed by the data types desired, as in the examples of Section 3.5. The executable files are called `xblat2s`, `xblat2d`, `xblat2c`, and `xblat2z` and are created in `LAPACK/BLAS`.

c) Go to `LAPACK/BLAS` and run the Level 2 BLAS tests. For the REAL version, the command is

```
xblat2s < sblat2.in
```

Similar commands should be used for the other test programs, with the leading 's' in the input file name replaced by 'd', 'c', or 'z'. The name of the output file is indicated on the first line of the input file and is currently defined to be `SBLAT2.SUMM` for the REAL version, with similar names for the other data types.

d) To compile and run the Level 3 BLAS test programs, repeat steps a–c using the makefile `makeblat3`. For step c, the executable program in the REAL version is `xblat3s`, the input file is `sblat3.in`, and output is to the file `SBLAT3.SUMM`, with similar names for the other data types.

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 [6] and [4].

4.5 Create the LAPACK Library

a) Go to the directory `LAPACK/SRC` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine.

b) Type `make` followed by the data types desired, as in the examples of Section 3.5. The make command can be run more than once to add another data type to the library if necessary.

The LAPACK library is created in `LAPACK/lapack.a`.

4.6 Create the Test Matrix Generator Library

a) Go to the directory `LAPACK/TESTING/MATGEN` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine.

b) Type `make` followed by the data types desired, as in the examples of Section 3.5. The make command can be run more than once to add another data type to the library if necessary.

The test matrix generator library is created in `LAPACK/tmglib.a`.

4.7 Run the LAPACK Test Programs

There are two distinct test programs for LAPACK routines in each data type, one for the linear equation routines and one for the eigensystem routines. Two sets of input files are provided, a small set for workstation-class computers and a large set for Cray-class computers. A *shar* file is provided in `LAPACK/TESTING` for each of the two sets of input files. Type

`sh lgtst.shar` for the large set of input files, or

`sh smtst.shar` for the small set of input files

Either command creates three input files for the eigensystem routines and one input file in each data type for the linear equation routines.

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

4.7.1 Testing the Linear Equations Routines

a) Go to `LAPACK/TESTING/LIN` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and desired load options for your machine. If you are not using the Fortran BLAS, define `BLAS` to point to your system's BLAS library, instead of `../../blas.a`.

b) Type `make` followed by the data types desired, as in the examples of Section 3.5. The executable files are called `xchks, xchkc, xchkd`, and `xchkz` and are created in `LAPACK/TESTING`.

c) Go to `LAPACK/TESTING` and run the tests for each data type. For the REAL version, the command is

```
xchks < stest.in > stest.out
```

The tests using `xchkd`, `xchkc`, and `xchkz` are similar with the leading 's' in the input and output file names replaced by 'd', 'c', or 'z'.

d) Send the output files to the authors as directed in Section 4.9. 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.

4.7.2 Testing the Eigensystem Routines

a) Go to `LAPACK/TESTING/EIG` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and desired load options for your machine. If you are not using the Fortran BLAS, define `BLAS` to point to your system's BLAS library, instead of `../../blas.a`.

b) Type `make` followed by the data types desired, as in the examples of Section 3.5. The executable files are called `xeigtsts, xeigtstc, xeigtstd`, and `xeigtstz` and are created in `LAPACK/TESTING`.

c) Go to `LAPACK/TESTING` and run the tests for each data type. The tests for the eigensystem routines use three separate input files, for testing the nonsymmetric eigenvalue problem, the symmetric eigenvalue problem, and the singular value decomposition. The tests in single precision are as follows:

```
xeigtsts < nep.in > snep.out
xeigtsts < sep.in > ssep.out
xeigtsts < svd.in > ssvd.out
```

The tests using `xeigtstc`, `xeigtstd`, and `xeigtstz` use the same three input files, but the leading 's' in the output files must be changed to 'c', 'd', or 'z'.

d) Send the output files to the authors as directed in Section 4.9. 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.

4.8 Run the LAPACK Timing Programs

There are two distinct timing programs for LAPACK routines in each data type, one for the linear equation routines and one for the eigensystem routines. The timing program for the linear equation 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 for workstation-class computers and a large set for Cray-class computers. The values of N in the large data set are five times larger than those in the small data set, and the large data set uses five values for the block size NB and two values for the leading array dimension LDA, while the small data set uses only two values for NB and one for LDA. Computers in between should run the large set if possible; suggestions for paring back the large data set are given in the instructions below. A *shar* file is provided in `LAPACK/TIMING` for each of the two sets of input files. Type

`sh lgtim.shar` for the large set of input files, or

`sh smtim.shar` for the small set of input files

Either command creates 8 input files in each data type, two for the linear equation routines, three for the eigensystem routines, and three for the BLAS. Note that the main programs are dimensioned for the large data sets, so the parameters in the main program may have to be reduced; otherwise the compiled program may be too large to run on a small machine.

The minimum time each subroutine will be timed is set to zero in each of these input files and may need to 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 7.

4.8.1 Timing the Linear Equations Routines

Two input files are provided in each data type for timing the linear equation routines, one for full matrices and one for band matrices. The data sets for the REAL version are in `LAPACK/TIMING/stime.in` and `LAPACK/TIMING/sband.in`.

a) To make the linear equation timing programs, go to `LAPACK/TIMING/LIN` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and desired load options for your machine. If you are not using the Fortran BLAS, define `BLAS` to point to your system's BLAS library, instead of `../../blas.a`.

b) Type `make` followed by the data types desired, as in the examples of Section 3.5. The executable files are called `xtims`, `xtimc`, `xtimd`, and `xtimz` and are created in `LAPACK/TIMING`.

c) Go to `LAPACK/TIMING` and 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 blocksize, corresponding changes should be made to the BLAS input files (see Section 4.8.2).

d) Run the programs for each data type you are using. For the REAL version, the commands are

```
xtims < stime.in > stime.out
xtims < sband.in > sband.out
```

Similar commands should be used for the other data types.

e) Send the output files to the authors as directed in Section 4.9. 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.

4.8.2 Timing 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 data files are `sblas.in1`, `sblas.in2`, and `sblas.in3`. 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 `sblas.in1`, M and N are large but K is small, while in `sblas.in2` the small parameter is M, and in `sblas.in3` 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) Go to `LAPACK/TIMING` and 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 4.8.1, 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 blocksizes used in timing the linear equation routines. If necessary, the large data set can be simplified by using only one value of LDA.

b) Run the programs for each data type you are using. For the REAL version, the commands are

```
xtims < sblas.in1 > sblas.out1
xtims < sblas.in2 > sblas.out2
xtims < sblas.in3 > sblas.out3
```

Similar commands should be used for the other data types.

c) Send the output files to the authors as directed in Section 4.9. 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.

4.8.3 Timing the Eigensystem Routines

Three 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, and one for the singular value decomposition. For the REAL version, these data sets are called `sneptim.in`, `sseptim.in`, and `ssvdtim.in`. Each of the three 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) To make a library of the instrumented LAPACK routines, first go to `LAPACK/TIMING/EIG/EIGSRC` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and desired load options for your machine. Then type `make` followed by the data types desired, as in the examples of Section 3.5. The library of instrumented code is created in `LAPACK/TIMING/EIG/eigsrc.a`.

b) To make the eigensystem timing programs, go to `LAPACK/TIMING/EIG` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and desired load options for your machine. If you are not using the Fortran BLAS, define `BLAS` to point to your system's BLAS library, instead of `../../blas.a`.

c) Type `make` followed by the data types desired, as in the examples of Section 3.5. The executable files are called `xeigtims`, `xeigtimc`, `xeigtimd`, and `xeigtimz` and are created in `LAPACK/TIMING`.

d) Go to `LAPACK/TIMING` and 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 4 instead of four matrices of types 1, 3, 4, and 6. See Section 7 for further details.

e) Run the programs for each data type you are using. For the REAL version, the commands are

```
xeigtims < sneptim.in > sneptim.out
xeigtims < sseptim.in > sseptim.out
xeigtims < ssvdtim.in > ssvdtim.out
```

Similar commands should be used for the other data types.

f) Send the output files to the authors as directed in Section 4.9. 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.

4.9 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

eanderso@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 many parts of it will 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. At the time of the first test release of LAPACK in April, 1989, many people did not yet have optimized versions of the BLAS. If this is still the case at your site, improvements to 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 SGETRF were degrading performance on the IBM 3090 because of the non-unit stride in SSWAP [1]. In response we added the auxiliary routine SLASWP to interchange a block of rows, so that users of the IBM 3090 could easily replace this routine with one in which the row interchanges are applied to one column at a time.

5 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.

5.1 Read the Tape

Read the tape and assign names to the files, preferably as indicated in Section 2. The first file (named README) is a list of the files in the order specified in Section 2. You will need about 12 megabytes to read in the complete tape. On a Sun SPARCstation, the libraries used 3.9 MB and the LAPACK executable files used 8.7 MB. In addition, the object files used 6.5 MB, but the object files can be deleted after creating the libraries and executable files. The total space requirements including the object files is approximately 31 MB for all four data types.

5.2 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
SECOND	REAL	Return time in seconds from a fixed starting time
DSECND	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 DSECND, and if you are working only in double precision, you do not need to install SLAMCH and SECOND. These five subroutines and their test programs are provided in the files LSAMEF and TLSAMEF, SLAMCHF and TSLAMCHF, etc.

5.2.1 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( UPLO, 'U' )
```

is equivalent to

```
( UPLO.EQ.'U' ).OR.( UPLO.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 `Tests completed`. The working version of LSAME should be appended to the file ALLBLASF. This file, which also contains the error handler XERBLA, will be compiled with either the BLAS library in Section 5.3 or the LAPACK library in Section 5.5.

5.2.2 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 SPARCstation is

```
Epsilon                      =     5.96046E-08
Safe minimum                 =     1.17549E-38
Base                         =     2.00000
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
```

Values of 0 or NaN for any of the parameters are obvious indicators that something has gone wrong. Suspect results should be documented and reported to the authors.

Compile SLAMCHF and TSLAMCHF 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 DLAMCHF and TDLAMCHF. If both tests were successful, go to Section 5.2.3.

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:

'U': Underflow threshold

'S': Safe minimum

'O': Overflow threshold

'E': Epsilon (relative machine precision)

'B': Base of the machine

Some people may be familiar with R1MACH (D1MACH), 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 R1MACH is on hand, the assignments in SLAMCH can be made to refer to R1MACH using the correspondence

SLAMCH('U') = R1MACH(1)

SLAMCH('O') = R1MACH(2)

SLAMCH('E') = R1MACH(3)

SLAMCH('B') = R1MACH(5)

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

5.2.3 Installing SECOND and DSECND

Both the timing routines and the test routines call SECOND (DSECND), 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 SECONDF calls ETIME, a Fortran library routine available on some

computer systems. If ETIME is not available or a better local timing function exists, you will have to provide the correct interface to SECOND and DSECND on your machine.

The test program in TSECONDF performs a million operations using 5000 iterations of the SAXPY operation $y := y + \alpha x$ on a vector of length 100. The total time and megaflops 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 SECONDF and TSECONDF and run the test program. There is no single right answer, but the times in seconds should be positive and the megaflop ratios should be appropriate for your machine. Repeat this test for DSECNDF and TDSECNDF and save SECOND and DSECND for inclusion in the LAPACK library in Section 5.5.

5.3 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 5.4 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.

5.4 Run the BLAS Test Programs

Test programs for the Level 2 and 3 BLAS are in the files xBLAT2F and xBLAT3F. 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.

a) Compile the files xBLAT2F and xBLAT3F and link them to your BLAS library or libraries. Note that each program includes a special version of the error-handling routine XERBLA, 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 XERBLA 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 XERBLA from ALLBLASF and recompile the BLAS library.

b) Each BLAS test program has a corresponding data file xBLAT2D or xBLAT3D. 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 SBLAT2.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 [6] and [4].

5.5 Create the LAPACK Library

Compile the files xLASRCF with ALLAUXF and create an object library. If you have compiled either the S or C version, you must also compile and include the files SCLAUXF, SLAMCHF, and SECONDF, and if you have compiled either the D or Z version, you must also compile and include the files DZLAUXF, DLAMCHF, and DSECNDF. If you did not compile the file ALLBLASF and include it in your BLAS library as described in Section 5.3, you must compile it now and include it in your LAPACK library.

5.6 Create the Test Matrix Generator Library

Compile the files xMATGENF and create an object library. If you have compiled either the S or C version, you must also compile and include the file SCATGENF, and if you have compiled either the D or Z version, you must also compile and include the file DZATGENF.

5.7 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. Two sets of input files are provided, a small set for workstation-class computers and a large set for Cray-class computers. The small input files end with the four characters 'TSTD' and the large input files end with the characters 'TS2D'. You need only use one of these sets of files.

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

5.7.1 Testing the Linear Equations Routines

a) Compile the files xLINTSTF 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) There are two sets of data files for the linear equation test program, xLINTSTD for small computer systems and xLINTS2D for large systems. 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., SLINTST.OUT) with this unit number.

d) Run the test programs.

e) Send the output files to the authors as directed in Section 5.9. 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.

5.7.2 Testing the Eigensystem Routines

a) Compile the files xEIGTSTF 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) There are two sets of data files for the linear equation test program, NEPTSTD, SEPTSTD, and SVDTSTD for small computer systems and NEPTS2D, SEPTS2D, and SVDTS2D for large systems. Note that the same three input files are used regardless of the data type of the test program. For each run 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 suitably named files with this unit number (e.g., SNEPTST.OUT, SSEPTST.OUT, and SSVDTST.OUT for the three runs of the REAL program).

d) Run the test programs.

e) Send the output files to the authors as directed in Section 5.9. 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.

5.8 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 for workstation-class computers and a large set for Cray-class computers. The values of N in the large data set are five times larger than those in the small data set, and the large data set uses five values for the block size NB and two values for the leading array dimension LDA, while the small data set uses only two values for NB and one for LDA. Computers in between should run the large set if possible; suggestions for paring back the large data set are given in the instructions below. The small input files end with the four characters 'TIMD' and the large input files end with the characters 'TM2D' (except for the BLAS timing files, see Section 5.8.2). Note that the main programs are dimensioned for the large data sets, so the parameters in the main

program may have to be reduced; otherwise the compiled program may be too large to run on a small machine.

The minimum time each subroutine will be timed is set to zero in each of these input files and may need to 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 7.

5.8.1 Timing the Linear Equations Routines

Two input files are provided in each data type for timing the linear equation routines, one for full matrices and one for band matrices. The small data sets are in xLINTIMD and xBNDTIMD and the large data sets are in xLINTM2D and xBNDTM2D.

a) Compile the files xLATIMF, 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).

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 blocksize, corresponding changes should be made to the BLAS input files (see Section 5.8.2).

   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., SLINTIM.OUT and SBNDTIM.OUT for the REAL version).

e) Run the programs for each data type you are using with the two data sets.

f) Send the output files to the authors as directed in Section 5.9. 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.

5.8.2 Timing 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 SBLTIMAD, SBLTIMBD, and SBLTIMCD and the large data sets are SBLTM2AD, SBLTM2BD, and SBLTM2CD. 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 SBLTIMAD, M and N are large but K is small, while in SBLTIMBD the small parameter is M, and in SBLTIMCD 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 5.8.1, 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 blocksizes 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., SBLTIMA.OUT, SBLTIMB.OUT, and SBLTIMC.OUT 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 5.9. 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.

5.8.3 Timing the Eigensystem Routines

Three 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, and one for the singular value decomposition. For the REAL version, the small data sets are SNEPTIMD, SSEPTIMD, and SSVDTIMD and the large data sets are SNEPTM2D, SSEPTM2D, and SSVDTM2D. Each of the three 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 xEIGSRCF and create an object library. If you have compiled either the S or C version, you must also compile and include the file SCIGSRCF, and if you have compiled either the D or Z version, you must also compile and include the file DZIGSRCF. If you did not compile the file ALLBLASF and include it in your BLAS library as described in Section 5.3, you must compile it now and include it in the instrumented LAPACK library.

b) Compile the files xEIGTIMF with AEIGTIMF and link them to your test matrix generator library, the instrumented LAPACK library created in the previous step, your LAPACK library from Section 5.5, 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).

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 4 instead of four matrices of types 1, 3, 4, and 6. See Section 7 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., SNEPTIM.OUT, SSEPTIM.OUT, and SSVDTIM.OUT for the three runs of the REAL version).

e) Run the programs in each data type you are using with the three data sets.

f) Send the output files to the authors as directed in Section 5.9. 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.

5.9 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

eanderso@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 many parts of it will 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. At the time of the first test release of LAPACK in April, 1989, many people did not yet have optimized versions of the BLAS. If this is still the case at your site, improvements to 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 SGETRF were degrading performance on the IBM 3090 because of the non-unit stride in SSWAP [1]. In response we added the auxiliary routine SLASWP to interchange a block of rows, so that users of the IBM 3090 could easily replace this routine with one in which the row interchanges are applied to one column at a time.

6 More About Testing

There are two distinct test programs for LAPACK routines in each data type, one for the linear equation routines and one for the eigensystem routines. Each program has its own style of input, and the eigensystem test program accepts three different sets of parameters, for the nonsymmetric eigenvalue problem, the symmetric eigenvalue problem, and the singular value decomposition. The following sections describe the different input formats and testing styles.

6.1 Testing the Linear Equation Routines

The test 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
- NB, the blocksize for the blocked routines

For symmetric or Hermitian matrices, the values of N are used for the matrix dimension.

The input file also specifies a set of LAPACK path names and the test matrix types to be used in testing the routines in each path. Path names are 3 characters long; the first character indicates the data type, and the next two characters identify a matrix type or problem type. The test paths for the linear equation test program are as follows:

{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}	QR	QR and LQ decompositions

The xQR test path also tests the routines for generating or multiplying by an orthogonal or unitary matrix expressed as a sequence of Householder transformations.

6.1.1 Test Matrices for the Linear Equation Routines

For each LAPACK test path specified in the input file, the test program generates test matrices, calls the LAPACK routines in that path, and computes a number of test ratios to verify that each operation has performed correctly. The test matrices used in each test path are shown in Table 1. In this context, ε is the machine epsilon, *i.e.*, the smallest positive floating-point number such that $1.0 + \varepsilon \neq 1.0$, and κ is the condition number of the matrix A.

Test matrix type	GE, QR	GB	PO, PP	PB	SY, SP	HE, HP
Diagonal	1	1	1	1	1	1
Upper triangular	2	2				
Lower triangular	3	3				
Banded: $kl < m/2, ku < n/2$		4		2		
Banded: $kl < m/2, ku > n/2$		5				
Banded: $kl > m/2, ku < n/2$		6				
Banded: $kl > m/2, ku > n/2$		7		3		
Random, $\kappa = 2$	4	8	2	4	2	2
Random, $\kappa = \sqrt{0.1/\varepsilon}$	5	9	3	5	3	3
Random, $\kappa = 0.1/\varepsilon$	6	10	4	6	4	4
Scaled near underflow	7	11	5	7	5	5
Scaled near overflow	8	12	6	8	6	6
Block diagonal					$7^{\dagger}$	

†– complex test paths only

Table 1: Test matrices for the linear equation test paths

6.1.2 Tests Performed for the Linear Equation Routines

For the LAPACK paths that operate on systems of linear equations, each test matrix is subjected to the following tests:

- Factor the matrix using xxxTRF, and compute the ratio

  $$||LU - A||/(n||A||\varepsilon)$$

- Invert the matrix A using xxxTRI, and compute the ratio

  $$||I - AA^{-1}||/(n||A||\,||A^{-1}||\varepsilon)$$

  For banded matrices, inversion routines are not available because the inverse would be dense.

- Solve the system $Ax = b$ using xxxTRS, and compute the ratios

  $$||b - Ax||/(||A||\,||x||\varepsilon) \text{ and } ||x - x^*||/(||x^*||\kappa\varepsilon)$$

  where x^* is the exact solution and κ is the condition number of A.

- Use iterative refinement (xxxRFS) to improve the solution, and compute the ratios

  $||x - x^*||/(||x^*||\kappa\varepsilon)$
  (backward error) $/\varepsilon$
  $||x - x^*||/(||x^*||$ (error bound))

- Compute the condition number using xxxCON, and form the product RCOND $* \kappa$.

The solve and iterative refinement steps are also tested with A replaced by A^T or A^H where applicable. The test ratios computed for the real linear equation test paths (except SQR) are listed in Table 2. The complex test ratios are the same except for the CGE and CGB paths; there A^T is replaced by A^H, and two more tests are inserted to test the solution (without iterative refinement) of $A^T x = b$.

Test ratio	SGE	SGB	SPO, SPP	SPB	SSY, SSP
$\|\|LU - A\|\|/(n\|\|A\|\|\varepsilon)$	1	1	1	1	1
$\|\|I - AA^{-1}\|\|/(n\|\|A\|\|\|\|A^{-1}\|\|\varepsilon)$	2		2		2
$\|\|b - Ax\|\|/(\|\|A\|\|\|\|x\|\|\varepsilon)$	3, 8^{T}	2, 7^{T}	3	2	3
$\|\|x - x^*\|\|/(\|\|x^*\|\|\kappa\varepsilon)$	4, 9^{T}	3, 8^{T}	4	3	4
$\|\|x - x^*\|\|/(\|\|x^*\|\|\kappa\varepsilon)$, refined	5, 10^{T}	4, 9^{T}	5	4	5
(backward error)$/\varepsilon$	6, 11^{T}	5, 10^{T}	6	5	6
$\|\|x - x^*\|\|/(\|\|x^*\|\|$(errorbound))	7, 12^{T}	6, 11^{T}	7	6	7
RCOND $* \kappa$	13	12	8	7	8

T – solve $A^T x = b$

Table 2: Tests performed for the REAL linear equation test paths

In the SQR test path, routines are tested for computing the QR decomposition (SGEQRF), computing the LQ decomposition (SGELQF), generating an orthogonal matrix expressed as a sequence of Householder transformations (SORGEN), and multiplying by an orthogonal matrix expressed as a sequence of Householder transformations (SORMUL). In the complex case, SORGEN is called CUNGEN, SORMUL is called CUNMUL, and Q is unitary instead of orthogonal. Tests 1–7 in the list below are performed if the $m \times n$ test matrix satisfies $m \geq n$, and tests 8–14 are performed if $m \leq n$.

- Compute the QR factorization using SGEQRF, generate the orthogonal matrix Q from the Householder vectors using SORGEN, and compute the ratio

  1. $||A - QR||/(m||A||\varepsilon)$

- Test the orthogonality of the computed matrix Q by computing the ratio

  2. $||I - Q^H Q||/(m\varepsilon)$

- Generate a random matrix C and multiply it by Q or Q^H using SORMUL with UPLO = 'L', and compare the result to the product of C and Q (or Q^H) using the explicit matrix Q generated by SORGEN. The different options for SORMUL are tested by computing the 4 ratios

3. $||QC - QC||/(m||C||\varepsilon)$
4. $||CQ - CQ||/(m||C||\varepsilon)$
5. $||Q^HC - Q^HC||/(m||C||\varepsilon)$
6. $||CQ^H - CQ^H||/(m||C||\varepsilon)$

where the first product is computed using SORMUL and the second using the explicit matrix Q.

- Compute the least-squares solution to a system of equations $Ax = b$ using SGEQRS, and compute the ratio

7. $||b - Ax||/(||A||\,||x||\varepsilon)$

- Compute the LQ factorization using SGELQF, and compute the ratio

8. $||A - LQ||/(n||A||\varepsilon)$

- Test the orthogonality of the computed matrix Q by computing the ratio

9. $||I - Q^HQ||/(n\varepsilon)$

- Generate a random matrix C and multiply it by Q or Q^H using SORMUL with UPLO = 'U', and compare the result to the product of C and Q (or Q^H) using the explicit matrix Q generated by SORGEN. The different options for SORMUL are tested by computing the 4 ratios

10. $||QC - QC||/(n||C||\varepsilon)$
11. $||CQ - CQ||/(n||C||\varepsilon)$
12. $||Q^HC - Q^HC||/(n||C||\varepsilon)$
13. $||CQ^H - CQ^H||/(n||C||\varepsilon)$

- Compute the minimum-norm solution to a system of equations $Ax = b$ using SGELQS, and compute the ratio

14. $||b - Ax||/(||A||\,||x||\varepsilon)$

When the tests are run, each test ratio that is greater than or equal to the threshold value causes a line of information to be printed to the output file. The first such line is preceded by a header that lists the matrix types used and the tests performed for the current path. A sample line for a test from the SGE path that did not pass when the threshold was set to 1.0 is

```
M =    4, N =    4, NB =   1, type  2, test 13, ratio =   1.14270
```

To get this information for every test, set the threshold to zero. After all the unsuccessful tests have been listed, a summary line is printed of the form

```
SGE:     11 out of   1960 tests failed to pass the threshold
```

If all the tests pass the threshold, only one line is printed for each path:

```
All tests for SGE passed the threshold (  1960 tests run)
```

6.1.3 Input File for Testing the Linear Equation Routines

From the test program's input file, one can control the size of the test matrices, the block size for the blocked routines, the paths to be tested, and the matrix types used in testing. We have set the options in the input files to run through all of the test paths. An annotated example of an input file for the REAL test program is shown below.

```
Data file for testing REAL LAPACK linear equation routines
8                       Number of values of M
0 1 2 3 5 10 20 70      Values of M (row dimension)
8                       Number of values of N
0 1 2 3 5 10 20 70      Values of N (column dimension)
3                       Number of values of NB
1 3 20                  Values of NB (the blocksize)
2                       Number of right hand sides
20.0                    Threshold value of test ratio.
SGE    8                List types on next line if 0 < NTYPES < 8
SGB   12                List types on next line if 0 < NTYPES < 12
SPO    6                List types on next line if 0 < NTYPES < 6
SPP    6                List types on next line if 0 < NTYPES < 6
SPB    8                List types on next line if 0 < NTYPES < 8
SSY    6                List types on next line if 0 < NTYPES < 6
SSP    6                List types on next line if 0 < NTYPES < 6
SQR    8                List types on next line if 0 < NTYPES < 8
```

The first 9 lines of the input file are read using list-directed input and are used to specify the values of M, N, NB, and THRESH (the threshold value). The remaining lines occur in sets of 1 or 2 and allow the user to specify the matrix types. Each line contains a 3-character path name in columns 1-3 and the number of test matrix types in columns 5-10. If the number of matrix types is at least 1 but is less than the maximum number of possible types, a second line will be read to get the numbers of the matrix types to be used. For example, the input line

```
SGE    8
```

requests all of the matrix types for path SGE, while

```
SGE    3
 4 5 6
```

requests only matrices of type 4, 5, and 6.

The number and size of the input values are limited by certain program maximums which are defined in PARAMETER statements in the main test program:

Parameter	Description	Value
NMAX	Maximum value for M, N, or NB	132
MAXIN	Maximum number of values of M, N, or NB	12
MAXRHS	Maximum number of right hand sides	10

The main test procedure for the REAL linear equation routines is in `LAPACK/TESTING/LIN/schkaa.f` in the Unix version and is the first program unit in `SLINTSTF` in the non-Unix version.

6.2 Testing the Nonsymmetric Eigenvalue Routines

The test routine for the LAPACK nonsymmetric eigenvalue routines, like the test program for the routines which solve linear systems, generates a number of different test matrices and computes measures of the error. The parameters which may be varied are:

- the order N of the test matrix A
- the type of the test matrix A
- three numerical parameters: the blocksize NB, the number of shifts NS for the multishift QR method, and the (sub)matrix size MAXB below or equal to which an unblocked, EISPACK-style method will be used

The test program thus consists of a triply-nested loop, the outer one over triples (NB, NS, MAXB), the next over 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 eigenvalue routines.

6.2.1 Test Matrices for the Nonsymmetric Eigenvalue Routines

Twenty-one different types of test matrices may be generated for the nonsymmetric eigenvalue routines. Table 3 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries.

	Eigenvalue Distribution				
Type	Arithmetic	Geometric	Clustered	Random	Other
Zero					1
Identity					2
(Jordan Block)^T					3
Diagonal	4, 7†, 8‡	5	6		
UTU^{-1}	9	10	11	12	
XTX^{-1}	13	14	15	16, 17†, 18‡	
Random entries					19, 20†, 21‡

$\dagger$– matrix entries are $O(\sqrt{\text{overflow}})$
$\ddagger$– matrix entries are $O(\sqrt{\text{underflow}})$

Table 3: Test matrices for the nonsymmetric eigenvalue problem

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

(Jordan Block)^T**:** Matrix with ones on the diagonal and the first subdiagonal, and zeros elsewhere

UTU^{-1}: Schur-form matrix T with $O(1)$ entries conjugated by a unitary (or real orthogonal) matrix U

XTX^{-1}: Schur-form matrix T with $O(1)$ entries conjugated by an ill-conditioned matrix X

For eigenvalue distributions other than "Other", 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

6.2.2 Tests Performed on the Nonsymmetric Eigenvalue Routines

Finding the eigenvalues and eigenvectors of a nonsymmetric matrix A is done in the following stages:

1. A is decomposed as UHU^*, where U is unitary, H is upper Hessenberg, and U^* is the conjugate transpose of U.
2. H is decomposed as ZTZ^*, where Z is unitary and T is in Schur form; this also gives the eigenvalues λ_i, which may be considered to form a diagonal matrix Λ.
3. The left and right eigenvector matrices L and R of the Schur matrix T are computed.
4. Inverse iteration is used to obtain the left and right eigenvector matrices Y and X of the matrix H.

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

$$r_1 = \frac{||A-UHU^*||}{n\varepsilon||A||} \qquad r_2 = \frac{||I-UU^*||}{n\varepsilon}$$

$$r_3 = \frac{||H-ZTZ^*||}{n\varepsilon||H||} \qquad r_4 = \frac{||I-ZZ^*||}{n\varepsilon}$$

$$r_5 = \frac{||A-(UZ)T(UZ)^*||}{n\varepsilon||A||} \qquad r_6 = \frac{||I-(UZ)(UZ)^*||}{n\varepsilon}$$

$$r_7 = \frac{||T_1-T_0||}{\varepsilon||T||} \qquad r_8 = \frac{||\Lambda_1-\Lambda_0||}{\varepsilon||\Lambda||}$$

$$r_9 = \frac{||TR-R\Lambda||}{\varepsilon||T||||R||} \qquad r_{10} = \frac{||LT-\Lambda L||}{\varepsilon||T||||L||}$$

$$r_{11} = \frac{||HX-X\Lambda||}{n\varepsilon||H||||X||} \qquad r_{12} = \frac{||YH-\Lambda Y||}{n\varepsilon||H||||Y||}$$

where the subscript 1 indicates that the eigenvalues and eigenvectors were computed at the same time, and 0 that they were computed in separate steps. (All norms are $\|.\|_1$.) 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 program is 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. A sample message is

```
Matrix order=   25, type=11, seed=2548,1429,1713,1411, result  8 is   11.33
```

In this example, the test matrix was of order $n = 25$ and of type 11 from Table 3, "seed" is the initial 4-integer seed of the random number generator used to generate A, and "result" specifies that test ratio r_8 failed to pass the threshold, and its value was 11.33.

6.2.3 Input File for Testing the Nonsymmetric Eigenvalue Routines

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

```
NEP:  Data file for testing the Nonsymmetric Eigenvalue Problem
8                                   Number of values of N
0 1 2 3 5 10 20 70                  Values of N (dimension)
3                                   Number of values of NB
1  3 20                             Values of NB (blocksize)
1  6  6                             Values of NSHIFT (no. of shifts)
2 10 10                             Values of MAXB (min. blocksize)
20.0                                Threshold value
1                                   Code to interpret the seed
NEP  21
```

The first line of the input file must contain the characters `NEP` 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 N
- line 3: The values of N, the matrix dimension
- line 4: The number of values of the parameters NB, NS, and MAXB
- 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 minimum blocksize
- line 8: The threshold value for the test ratios
- line 9: 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 10: If line 9 was 2, four integer values for the random number seed

The remaining lines occur in sets of 1 or 2 and allow the user to specify the matrix types. Each line contains a 3-character identification in columns 1–3, which must be either `NEP` or `SHS` (`CHS` in complex, `DHS` in double precision, and `ZHS` in complex*16), and the number of matrix types must be the first nonblank item in columns 4–80. If the number of matrix types is at least 1 but is less than the maximum number of possible types, a second line will be read to get the numbers of the matrix types to be used. For example,

```
NEP 21
```

requests all of the matrix types for the nonsymmetric eigenvalue problem, while

```
NEP  4
 9 10 11 12
```

requests only matrices of type 9, 10, 11, and 12.

The number and size of the input values are limited by certain program maximums which are defined in PARAMETER statements in the main test program:

Parameter	Description	Value
NMAX	Maximum value for N, NB, NS, and MAXB	132
MAXIN	Maximum number of values of the parameters	20

For the nonsymmetric eigenvalue input file, MAXIN is both the maximum number of values of N and the maximum number of 3-tuples (NB, NS, MAXB). The main test procedure for the REAL eigenvalue routines is in `LAPACK/TESTING/EIG/schkee.f` in the Unix version and is the first program unit in `SEIGTSTF` in the non-Unix version.

6.3 Testing the Symmetric Eigenvalue Routines

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

- the order N of the test matrix A
- the type of the test matrix A
- the blocksize NB

The testing program thus consists of a triply-nested loop, the outer one over NB, the next over 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 eigenvalue routines.

6.3.1 Test Matrices for the Symmetric Eigenvalue Routines

Fifteen different types of test matrices may be generated for the symmetric eigenvalue routines. Table 4 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries. The expression UDU^{-1} means a real diagonal matrix D with $O(1)$ entries conjugated by a unitary (or real orthogonal) matrix U. The eigenvalue distributions have the same meanings as in the nonsymmetric case (see Section 6.2.1).

	Eigenvalue Distribution			
Type	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3, $6^{\dagger}$, $7^{\ddagger}$	4	5	
UDU^{-1}	8, $11^{\dagger}$, $12^{\ddagger}$	9	10	
Random entries				13, $14^{\dagger}$, $15^{\ddagger}$

$\dagger$– matrix entries are $O(\sqrt{\text{overflow}})$
$\ddagger$– matrix entries are $O(\sqrt{\text{underflow}})$

Table 4: Test matrices for the symmetric eigenvalue problem

6.3.2 Tests Performed on the Symmetric Eigenvalue Routines

Finding the eigenvalues and eigenvectors of a symmetric matrix A is done in the following stages:

1. A is decomposed as USU^*, where U is unitary, S is real symmetric tridiagonal, and U^* is the conjugate transpose of U.
2. S is decomposed as $Z\Lambda Z^*$, where Z is real orthogonal and Λ is a real diagonal matrix of eigenvalues.
3. The "PWK" method is used to compute Λ using a square-root-free method which does not compute Z.

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

$$r_1 = \frac{||A-USU^*||}{n\varepsilon||A||} \qquad r_2 = \frac{||I-UU^*||}{n\varepsilon}$$

$$r_3 = \frac{||S-Z\Lambda Z^*||}{n\varepsilon||S||} \qquad r_4 = \frac{||I-ZZ^*||}{n\varepsilon}$$

$$r_5 = \frac{||A-(UZ)\Lambda(UZ)^*||}{n\varepsilon||A||} \qquad r_6 = \frac{||I-(UZ)(UZ)^*||}{n\varepsilon}$$

$$r_7 = \frac{||\Lambda_1-\Lambda_0||}{\varepsilon||\Lambda||} \qquad r_8 = \frac{||\Lambda_1-\Lambda_{\mathrm{PWK}}||}{\varepsilon||\Lambda||}$$

$$r_9 = \frac{w}{\varepsilon||\Lambda||} \quad w \text{ from Sturm sequence test}$$

where the subscript 1 indicates that the eigenvalues and eigenvectors were computed at the same time, and 0 that they were computed in separate steps. (All norms are $\|.\|_1$.) 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 n.

The "Sturm sequence test" is a test of how much the eigenvalues in Λ differ from the eigenvalues of S. Sturm sequences are used to test whether an eigenvalue of S lies within an

interval $(\lambda - w, \lambda + w)$, where λ is a diagonal entry of Λ. Increasingly larger values of w are tried until one is found such that all the diagonal entries λ lie within w of an eigenvalue of S. The first (smallest) such w, divided by ε times the absolute value of the largest eigenvalue, is then r_9.

As in the nonsymmetric case, the test ratios for each test matrix are compared to a user-specified threshold THRESH, and a message is printed for each test that exceeds this threshold.

6.3.3 Input File for Testing the Symmetric Eigenvalue Routines

An annotated example of an input file for testing the symmetric eigenvalue routines is shown below.

```
SEP:  Data file for testing the Symmetric Eigenvalue Problem
8                                   Number of values of N
0 1 2 3 5 10 20 70                  Values of N (dimension)
3                                   Number of values of NB
1 3 20                              Values of NB (blocksize)
20.0                                Threshold value
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 2–7 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 parameter NB
line 5: The values of NB, the blocksize
line 6: The threshold value for the test ratios
line 7: 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 8: If line 7 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).

The number and size of the input values are limited by certain program maximums which are defined in PARAMETER statements in the main test program:

Parameter	Description	Value
NMAX	Maximum value for N and NB	132
MAXIN	Maximum number of values of N and NB	20

The main test procedure for the single precision real eigenvalue routines is in `LAPACK/TESTING/EIG/schkee.f` in the Unix version and is the first program unit in `SEIGTSTF` in the non-Unix version.

6.4 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.

6.4.1 Test Matrices for the Singular Value Decomposition Routines

Sixteen different types of test matrices may be generated for the singular value decomposition routines. Table 5 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.

	Singular Value Distribution			
Type	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3, $6^{\dagger}$, $7^{\ddagger}$	4	5	
UDV	8, $11^{\dagger}$, $12^{\ddagger}$	9	10	
Random entries				13, $14^{\dagger}$, $15^{\ddagger}$
Random bidiagonal				16

$\dagger$– matrix entries are $O(\sqrt{\text{overflow}})$
$\ddagger$– matrix entries are $O(\sqrt{\text{underflow}})$

Table 5: 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: A $\min(M, N) \times \min(M, N)$ 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 xBDSQR 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).

6.4.2 Tests Performed on the Singular Value Decomposition Routines

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.

In addition, the LAPACK routines xBDSQR can apply the transformations that form U to an arbitrary $\tilde{n} = \min(m, n) \times k$ matrix C; the resulting matrix we call D. The test routines therefore start with a random $m \times k$ matrix R with $O(1)$ entries, as well as A, and apply all transformations to R which are applied to A from the left.

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

$$r_1 = \frac{||A-QBP||}{\max(m,n)\varepsilon||A||} \qquad r_2 = \frac{||R-QC||}{\max(m,k)\varepsilon||R||}$$

$$r_3 = \frac{||I-Q^*Q||}{m\varepsilon} \qquad r_4 = \frac{||I-PP^*||}{n\varepsilon}$$

$$r_5 = \frac{||B-U\Sigma V||}{\tilde{n}\varepsilon||B||} \qquad r_6 = \frac{||C-UD||}{\max(\tilde{n},k)\varepsilon||C||}$$

$$r_7 = \frac{||I-U^*U||}{\tilde{n}\varepsilon} \qquad r_8 = \frac{||I-VV^*||}{\tilde{n}\varepsilon}$$

$$r_9 = \frac{||A-(UZ)\Sigma(UZ)^*||}{\max(m,n)\varepsilon||A||} \qquad r_{10} = \frac{||R-(QU)D||}{\max(m,k)\varepsilon||R||}$$

$$r_{11} = \frac{||I-(QU)^*(QU)||}{m\varepsilon} \qquad r_{12} = \frac{||I-(VP)(VP)^*||}{n\varepsilon}$$

$$r_{13} = \frac{||\Sigma_1-\Sigma_0||}{\varepsilon||\Sigma||} \qquad r_{14} = \frac{s}{\varepsilon} \ s \text{ from Sturm sequence test}$$

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 $||.||_1$.) 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.

The "Sturm sequence test" is a test of how much the singular values in Σ differ from the singular values of B: Sturm sequences are used to test whether a singular value of B lies within an interval $((1-s)\sigma,(1+s)\sigma)$, where σ is a diagonal entry of Σ. Increasingly larger values of s are tried until one is found such that all the diagonal entries σ lie within s of a singular value of B. The first (smallest) such s, divided by ε, is then r_{14}.

When the test program is run, these test ratios will be compared with a user-specified threshold THRESH: if a test ratio exceeds THRESH, a message such as

```
Matrix order=   25, type=10, seed=2548,1429,1713,1411, result  8 is   11.33
```

if A is square, or

```
   3 x    5 matrix, type=14, seed=1904,2941,2246,1241, result  5 is   14.57
```

will be printed out. This specifies the test matrix, the ratio that failed, and its value. In the second example, m is 3, n is 5, the type number (see Table 5) is 14, which means a "random bidiagonal" matrix, the test ratio which failed was r_5, and the value of r_5 was 14.57. Given the seed, the size, and the type, it is possible to reconstruct the test matrix exactly, subject only to possible differing numerical properties on different machines, and thus reproduce any problem with a routine.

6.4.3 Input File for Testing the Singular Value Decomposition Routines

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

```
SVD:  Data file for testing the Singular Value Decomposition
20                                  Number of values of M
0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 10  10  70  70    Values of M
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 10  70  10  70    Values of N
1                                   Number of values of NB
1                                   Values of NB (blocksize)
2                                   Values of NRHS
20.0                                Threshold value
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 2–9 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 4: The values of N, the matrix column dimension
line 5: The number of values of the parameters NB and NRHS
line 6: The values of NB, the blocksize
line 7: The values of NRHS, the number of right hand sides
line 8: The threshold value for the test ratios
line 9: 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 10: If line 9 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).

The number and size of the input values are limited by certain program maximums which are defined in PARAMETER statements in the main test program:

Parameter	Description	Value
NMAX	Maximum value for M, N, NB, and NRHS	132
MAXIN	Maximum number of values of the parameters	20

For the singular value decomposition input file, MAXIN is both the maximum number of pairs (M, N) and the maximum number of pairs (NB, NRHS). The main test procedure for the REAL eigenvalue routines is in `LAPACK/TESTING/EIG/schkee.f` in the Unix version and is the first program unit in `SEIGTSTF` in the non-Unix version.

7 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. Results from the linear equation timing program are given in megaflops, and the operation counts are generally computed as some function of the problem size. 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 three different sets of parameters, for the nonsymmetric eigenvalue problem, the symmetric eigenvalue problem, and the singular value decomposition. The following sections describe the different input formats and timing parameters.

7.1 Timing the Linear Equation Routines

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 banded routines, or the third size parameter for the Level 3 BLAS
- NB, the blocksize for the blocked routines
- 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 maximums 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
LDAMAX	Maximum value of LDA	532
NMAXB	Maximum value of M for banded matrices	5000
MAXIN	Maximum number of values of M, N, K, or NB	12
MXNLDA	Maximum number of values of LDA	4

The parameter LDAMAX should be at least NMAX. For the xGB path, we must have $(4K + 1)M \leq 2(\text{NMAX})(\text{LDAMAX})$, which restricts the value of K. The above limits allow K to be as big as 200 for M = 1000. For the xPB path, the condition is $(2K + 1)M \leq 3(\text{NMAX})(\text{LDAMAX})$. The main timing program 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 `SLINTIMF` in the non-Unix version.

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}	QR	QR decomposition
{S, C, D, Z}	TR	Triangular matrices
{S, C, D, Z}	TP	Triangular packed matrices

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

In addition, this release contains some experimental routines which can be accessed using the following temporary path names:

{S, D}	LU	Variants of the LU decomposition
{S, D}	CH	Variants of the Cholesky decomposition
{S, C, D, Z}	HR	Reduction to Hessenberg form
{S, C, D, Z}	TD	Reduction to real tridiagonal form

The xLU path requests timing of three block variants of the LU factorization (left-looking, Crout, and right-looking) for $N \times N$ matrices, as well as the corresponding unblocked variants on matrices of size $N \times NB$. Timings for the Linpack routine xGEFA are included for comparison. The xCH timing path requests timing of three block variants of the Cholesky factorization and the corresponding Linpack routine xPOFA. Timing of the blocked and unblocked reductions to Hessenberg and tridiagonal form are requested by the paths xHR and xTD, and times for the Eispack routines ORTHES and TRED1 are included for comparision.

The timing programs have their own matrix generator that supplies computed, rather than random, matrices for timing. Computed matrices are used because they can be generated more quickly than random matrices, and the call to the matrix generator is inside the timing loop. 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, we 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 matrices is shown below.

```
Data file for timing REAL LAPACK linear equation routines
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)
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)
2                         Number of values of LDA
```

```
512 513                   Values of LDA (leading dimension)
0.0                       Minimum time in seconds
SGE   T T T T             Put T to time:  -TRF -TRI -TRS -CON
SPO   T T T T                             -TRF -TRI -TRS -CON
SPP   T T T T                             -TRF -TRI -TRS -CON
SSY   T T T T                             -TRF -TRI -TRS -CON
SSP   T T T T                             -TRF -TRI -TRS -CON
SQR   T T                                 -GEQRF  -GEQRS
STR   T                                   -TRI
STP   T                                   -TRI
SLU
SCH
SHR
STD
```

The first 12 lines of the input file are read using list-directed input and are used to specify the values of M, N, K, NB, LDA, and TIMMIN (the minimum time). By default, xGEMV and xGEMM 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 13:

BAND Time xGBMV (instead of xGEMV) using matrices of order M and bandwidth K, and time xGEMM using matrices of order K.

NONE Do not do the sample timing of xGEMV and xGEMM.

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

When timing the banded 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.

```
Data file for timing REAL LAPACK banded routines
1                         Number of values of M
1000                      Values of M (row dimension)
0                         Number of values of N
1000                      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)
1                         Number of values of LDA
601                       Values of LDA (leading dimension)
0.0                       Minimum time in seconds
BAND                      Time sample banded BLAS
SGB   T T T               Put T to time:  -TRF -TRS -CON
SPB   T T T                               -TRF -TRS -CON
```

Here M specifies the matrix size and K specifies the bandwidth for the test paths SGB and SPB. Note that we request timing of the sample BLAS for banded matrices by specifying "BAND" on line 13.

7.2 Timing the Level 2 and 3 BLAS

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:

```
Data file 1 for timing the REAL BLAS routines
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.

7.3 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, MAXB, LDA) specifying the block size NB, the number of shifts NS, the matrix size MAXB 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 MAXB apply only to the QR iteration routine xHSEQR, and NB is used only by the block algorithms. A goal of this timing code is to determine the values of NB, NS and MAXB which maximize the speed of the codes.

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

Parameter	Description	Value
MAXN	Maximum value for N, NB, NS, or MAXB	400
LDAMAX	Maximum value for LDA	420
MAXIN	Maximum number of values of N	12
MAXPRM	Maximum number of parameter sets (NB, NS, MAXB, LDA)	10

The main timing program for the REAL routines is found in `LAPACK/TIMING/EIG/stimee.f` in the Unix version and is the first program unit in `SEIGTIMF` in the non-Unix version.

The computations that may be timed for the REAL version are

1. SGEHRD (LAPACK reduction to upper Hessenberg form)
2. SHSEQR(E) (LAPACK computation of eigenvalues only of a Hessenberg matrix)
3. SHSEQR(S) (LAPACK computation of the Schur form of a Hessenberg matrix)
4. SHSEQR(I) (LAPACK computation of the Schur form and Schur vectors of a Hessenberg matrix)
5. STREVC(L) (LAPACK computation of the the left eigenvectors of a matrix in Schur form)
6. STREVC(R) (LAPACK computation of the the right eigenvectors of a matrix in Schur form)
7. SHSEIN(L) (LAPACK computation of the the left eigenvectors of an upper Hessenberg matrix using inverse iteration)
8. SHSEIN(R) (LAPACK computation of the the right eigenvectors of an upper Hessenberg matrix using inverse iteration)
9. ORTHES (EISPACK reduction to upper Hessenberg form, to be compared to SGEHRD)
10. HQR (EISPACK computation of eigenvalues only of a Hessenberg matrix, to be compared to SHSEQR(E))
11. HQR2 (EISPACK computation of eigenvalues and eigenvectors of a Hessenberg matrix, to be compared to SHSEQR(I) plus STREVC(R))
12. INVIT (EISPACK computation of the right eigenvectors of an upper Hessenberg matrix using inverse iteration, to be compared to SHSEIN).

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 $1/\sqrt{\varepsilon}$ (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 $1, \varepsilon, \ldots, \varepsilon$ with random signs (matrix types 3 and 7), or
- real or complex conjugate paired eigenvalues randomly chosen from the interval $(\varepsilon, 1)$ (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
100 200 300 400                     Values of N (dimension)
7                                   Number of values of parameters
1   1   1   12  10  8   6           Values of NB (blocksize)
6   12  18  6   12  16  18          Values of NS (number of shifts)
20  50  100 50  100 200 300         Values of MAXB (max. blocksize)
401 401 401 401 401 401 401         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
```

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:	NTYPES, the number of matrix types to be used

If 0 < NTYPES < 8, then line 11 specifies NTYPES 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
```

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 SHSEQR routine, one line for each different combination of NS and MAXB as well.

7.4 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.

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 maximums which are defined in PARAMETER statements in the main timing program:

Parameter	Description	Value
MAXN	Maximum value for N or NB	400
LDAMAX	Maximum value for LDA	420
MAXIN	Maximum number of values of N	12
MAXPRM	Maximum number of pairs of values (NB, LDA)	10

The main timing program for the REAL routines is found in `LAPACK/TIMING/EIG/stimee.f` in the Unix version and is the first program unit in `SEIGTIMF` in the non-Unix version.

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. SSTEQR(N) (LAPACK computation of eigenvalues only of a symmetric tridiagonal matrix)

3. SSTEQR(V) (LAPACK computation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix)

4. SSTERF (LAPACK computation of the eigenvalues only of a symmetric tridiagonal matrix using a square-root free algorithm)

5. TRED1 (EISPACK reduction to symmetric tridiagonal form, to be compared to SSYTRD)

6. IMTQL1 (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to SSTEQR(N))

7. IMTQL2 (EISPACK computation of eigenvalues and eigenvectors of a symmetric tridiagonal matrix, to be compared to SSTEQR(V))

8. TQLRAT (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to SSTERF).

For complex matrices the possible computations are

1. CHETRD (LAPACK reduction of a complex Hermitian matrix to real symmetric tridiagonal form)

2. CSTEQR(N) (LAPACK computation of eigenvalues only of a real symmetric tridiagonal matrix)

3. CSTEQR(V) (LAPACK computation of the eigenvalues and eigenvectors of a real symmetric tridiagonal matrix (accumulating them into a complex matrix))

4. CUNGEN+CSTEQR (LAPACK computation of the eigenvalues and eigenvectors of a Hermitian matrix given the reduction to real symmetric tridiagonal form)

5. HTRIDI (EISPACK reduction to symmetric tridiagonal form, to be compared to CHETRD)

6. IMTQL1 (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to CSTEQR(V))

7. IMTQL2+HTRIBK (EISPACK computation of eigenvalues and eigenvectors of a complex Hermitian matrix given the reduction to real symmetric tridiagonal form, to be compared to CUNGEN+CSTEQR).

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 $1, \varepsilon, \ldots, \varepsilon$ 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
4                                   Number of values of N
100 200 300 400                     Values of N (dimension)
10                                  Number of values of parameters
1   16  32  48  64  1   16  32  48  64   Values of NB (blocksize)
400 400 400 400 400 401 401 401 401 401  Values of LDA (leading dim.)
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: NTYPES, the number of matrix types to be used

If $0 <$ NTYPES < 4, then line 9 specifies NTYPES 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 (see Section 7.3).

7.5 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 maximums which are defined in PARAMETER statements in the main timing program:

Parameter	Description	Value
MAXN	Maximum value for M, N, or NB	400
LDAMAX	Maximum value for LDA	420
MAXIN	Maximum number of pairs of values (M, N)	12
MAXPRM	Maximum number of pairs of values (NB, LDA)	10

The main timing program for the REAL routines is found in `LAPACK/TIMING/EIG/stimee.f` in the Unix version and is the first program unit in `SEIGTIMF` in the non-Unix version.

The computations that may be timed for the REAL version are

1. SGEBRD (LAPACK reduction to bidiagonal form)
2. SBDSQR (LAPACK computation of singular values only of a bidiagonal matrix)
3. SBDSQR(L) (LAPACK computation of the singular values and left singular vectors of a bidiagonal matrix)
4. SBDSQR(R) (LAPACK computation of the singular values and right singular vectors of a bidiagonal matrix)
5. SBDSQR(B) (LAPACK computation of the singular values and right and left singular vectors of a bidiagonal matrix)
6. SBDSQR(V) (LAPACK computation of the singular values and multiply square matrix of dimension min(M,N) by transpose of left singular vectors)
7. LAPSVD (LAPACK singular values only of a dense matrix, using SGEBRD and SBDSQR)
8. LAPSVD(l) (LAPACK singular values and min(M,N) left singular vectors of a dense matrix, using SGEBRD, SORGEN and SBDSQR(L))
9. LAPSVD(L) (LAPACK singular values and M left singular vectors of a dense matrix, using SGEBRD, SORGEN and SBDSQR(L))
10. LAPSVD(R) (LAPACK singular values and N right singular vectors of a dense matrix, using SGEBRD, SORGEN and SBDSQR(R))
11. LAPSVD(B) (LAPACK singular values, min(M,N) left singular vectors, and N right singular vectors of a dense matrix, using SGEBRD, SORGEN and SBDSQR(B))
12. LINSVD (LINPACK singular values only of a dense matrix using SSVDC, to be compared to LAPSVD)
13. LINSVD(l) (LINPACK singular values and min(M,N) left singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(l))

14. LINSVD(L) (LINPACK singular values and M left singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(L))

15. LINSVD(R) (LINPACK singular values and N right singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(R))

16. LINSVD(B) (LINPACK singular values, min(M,N) left singular vectors and N right singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(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 $1, \varepsilon, \ldots, \varepsilon$ 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
100 100 200 200 200 400 400         Values of M (row dimension)
100 100 200 200 200 400 400         Values of N (column dimension)
1                                   Number of values of parameters
1                                   Values of NB (blocksize)
401                                 Values of LDA (leading dimension)
0.0                                 Minimum time in seconds
5                                   Number of matrix types
SBD    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: NTYPES, the number of matrix types to be used

If $0 <$ NTYPES < 5, then line 9 specifies NTYPES 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 (see Section 7.3).

Acknowledgments

Jim Demmel 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 described in [2] and indicate by means of a table which subroutines are included in this release.

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 TXXYYY. 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. In this release, we include subroutines covering only a subset of the total collection of matrix types to be provided in LAPACK. 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:

GE	general (i.e., unsymmetric, in some cases rectangular)
GB	general band
PO	symmetric or Hermitian positive definite
PP	symmetric or Hermitian positive definite, packed storage
PB	symmetric or Hermitian positive definite band
SY	symmetric (i.e., indefinite)
SP	symmetric, packed storage
HE	(complex) Hermitian (i.e., indefinite)
HP	(complex) Hermitian, packed storage
OR	(real) orthogonal
UN	(complex) unitary
TR	triangular
TP	triangular, packed storage
HS	Hessenberg
ST	symmetric tridiagonal
BD	bidiagonal

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

TRF	perform a triangular factorization (LU, Cholesky, etc.)
TF2	unblocked triangular factorization, if TRF is blocked
TRS	solve systems of linear equations (based on triangular factorization)
TRI	compute inverse (based on triangular factorization)
TI2	unblocked computation of inverse, if TRI is blocked
CON	estimate condition number
RFS	refine initial solution returned by TRS routines
QRF	perform the QR factorization without pivoting
QR2	unblocked version of QRF

QRS	solve linear least squares problems (based on QR factorization)
LQF	perform the LQ factorization without pivoting
LQ2	unblocked version of LQF
LQS	solve underdetermined linear systems (based on LQ factorization)
GEN	generate a real orthogonal or complex unitary matrix as a product of Householder matrices
GN2	unblocked version of GEN
MUL	multiply a matrix by a real orthogonal or complex unitary matrix by applying a product of Householder matrices
ML2	unblocked version of MUL
HRD	reduce a square matrix to upper Hessenberg form
HD2	unblocked version of HRD
TRD	reduce a symmetric matrix to real symmetric tridiagonal form
TD2	unblocked version of TRD
BD2	reduce a rectangular matrix to bidiagonal form
EQR	compute eigenvalues and, optionally, Schur factorization or eigenvectors using the QR algorithm
EIN	compute selected eigenvectors by inverse iteration
EVC	compute eigenvectors from Schur factorization
ERF	compute eigenvectors using the Pal-Walker-Kahan variant of the QL or QR algorithm
SQR	compute singular values and, optionally, singular vectors using the QR algorithm

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

	GE	GB	PO	PP	PB	HE SY	HP SP	UN OR	TR	TP
TRF	×	×	×	×	×	×	×			
TF2	×	×	×		×	×				
TRS	×	×	×	×	×	×	×			
TRI	×		×	×		×	×		×	×
TI2									×	
CON	×	×	×	×	×	×	×			
RFS	×	×	×	×	×	×	×			
QRF	×									
QR2	×									
QRS	×									
LQF	×									
LQ2	×									
LQS	×									
GEN								×		
GN2								×		
MUL								×		
ML2								×		

The following table indicates the routines provided in this release for finding eigenvalues and eigenvectors or singular values and singular vectors:

	GE	HS	TR	SY	SP	ST	BD
HRD	×						
HD2	×						
TRD				×	×		
TD2				×			
BD2	×						
EQR		×				×	
EIN		×					
EVC			×				
ERF						×	
SQR							×

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 xxLAUXF and xLASRCF 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:

ENVIR	Return parameters for use in the algorithms, such as block size and number of shifts
XENVIR	Set parameters in ENVIR's COMMON block
XERBLA	Error handler for LAPACK routines

Special functions:

LSAME	LOGICAL	Return .TRUE. if two characters are the same regardless of case
LSAMEN	LOGICAL	Return .TRUE. if two character strings are the same regardless of case
SLAMCH	REAL	Return single precision machine parameters
DLAMCH	DOUBLE PRECISION	Return double precision machine parameters
R1MACH	REAL	Return single precision machine parameters
D1MACH	DOUBLE PRECISION	Return double precision machine parameters

Functions for computing norms:

_LANGE	General matrix
_LANGB	General band matrix
_LANSY	Symmetric matrix
_LANSP	Symmetric packed matrix
_LANSB	Symmetric band matrix
_LANHE	(complex) Hermitian matrix
_LANHP	(complex) Hermitian packed matrix
_LANHB	(complex) Hermitian band matrix
_LANTR	Trapezoidal matrix
_LANTP	Triangular packed matrix
_LANTB	Triangular band matrix
_LANHS	Upper Hessenberg matrix

Extensions to the Level 1 and 2 BLAS:

_SROT	(complex) Apply a real plane rotation to a complex vector
SCSUM1	Sum absolute values of a complex vector
ICMAX1	Find the index of element whose real part has max. abs. value
DZSUM1	Sum absolute values of a complex*16 vector

IZMAX1	Find the index of element whose real part has max. abs. value
_SYMV	(complex) Symmetric matrix times vector
_SPMV	(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

Other LAPACK auxiliary routines:

_LACGV	(complex) Conjugate a complex vector
_LACON	Estimate the norm of a matrix for use in condition estimation
_LACPY	Copy a matrix to another matrix
_LAE2	Compute eigenvalues of a 2 x 2 real symmetric or complex Hermitian matrix
_LAEIN	Use inverse iteration to find a specified right and/or left eigenvector of an upper Hessenberg matrix
_LAESY	(complex) Compute eigenvalues and eigenvectors of a complex symmetric 2 x 2 matrix
_LAEV2	Compute eigenvalues and eigenvectors of a 2 x 2 real symmetric or complex Hermitian matrix
_LAHBR	(complex) Factor a panel of a complex Hermitian indefinite matrix
_LAHQR	Find the Schur factorization of a Hessenberg matrix (modified version of HQR from EISPACK)
_LAHR2	Reduce a panel of a matrix to Hessenberg form
_LAHRD	Chase a K x K bulge in the reduction to Hessenberg form
_LALN2	(real) Solve a 1 x 1 or 2 x 2 linear system
_LAN2	(real) Compute the eigenvalues of a 2 x 2 nonsymmetric matrix:
_LAPY2	Compute square root of X**2 + Y**2
_LAPY3	(real) Compute square root of X**2 + Y**2 + Z**2
_LARAN	(real) Generate a real random number in the interval (0,1)
_LARF	Multiply by a Householder matrix
_LARFB	Multiply by a block Householder matrix
_LARFG	Generate a Householder matrix
_LARFT	Accumulate the triangular factor of a block Householder matrix
_LARND	Generate a random number from one of several distributions
_LARTG	Generate Givens rotations
_LAS2	(real) Compute singular values of a 2 x 2 triangular matrix
_LASBR	Factor a panel of a symmetric indefinite matrix
_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
_LASWP	Perform a series of row interchanges
_LASYK	Apply a rank-K update to a symmetric matrix of the form $C := C - A * B$
_LATRD	Reduce a panel of a symmetric matrix to tridiagonal form
_LATRS	Solve a triangular system (includes scaling)
_LAULM	Compute the product U*L (blocked version)

_LAUL2	Unblocked version of _LAULM
_LAUUM	Compute the product U*U' or L'*L (blocked version)
_LAUU2	Unblocked version of _LAUUM
_LAVHE	(complex) Multiply a vector by a matrix that has been factored by _HETRF
_LAVHP	(complex) Multiply a vector by a matrix that has been factored by _HPTRF
_LAVSY	Multiply a vector by a matrix that has been factored by _SYTRF
_LAVSP	Multiply a vector by a matrix that has been factored by _SPTRF
_LAXPY	Add a multiple of a matrix to another matrix
_LAZRO	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 SOPBL2, SOPBL3, SOPAUX, and SOPLA 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 SGBMV from the Level 2 BLAS is as follows:

SGBMV	multiplications:	$mn - (m - k_l - 1)(m - k_l)/2 - (n - k_u - 1)(n - k_u)/2$
	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 6.

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 7 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), NRHS, the number of right hand sides in the solution phase, and q, an integer offset for storing the Householder vectors in SORGEN and SORMUL.

Level 2 BLAS	multiplications	additions	total flops
SGEMV [1,2]	mn	mn	$2mn$
SSYMV [3,4]	n^2	n^2	$2n^2$
SSBMV [3,4]	$n(2k+1)-k(k+1)$	$n(2k+1)-k(k+1)$	$n(4k+2)-2k(k+1)$
STRMV [3,4,5]	$n(n+1)/2$	$(n-1)n/2$	n^2
STBMV [3,4,5]	$n(k+1)-k(k+1)/2$	$nk-k(k+1)/2$	$n(2k+1)-k(k+1)$
STRSV [5]	$n(n+1)/2$	$(n-1)n/2$	n^2
STBSV [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 – Plus m multiplies if $\alpha \neq \pm 1$
2 – Plus m multiplies if $\beta \neq \pm 1$ or 0
3 – Plus n multiplies if $\alpha \neq \pm 1$
4 – Plus n multiplies if $\beta \neq \pm 1$ or 0
5 – Less n multiplies if matrix is unit triangular

Table 6: Operation counts for the Level 2 BLAS

Level 3 BLAS	multiplications	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	$kn(n+1)/2$	$kn(n+1)/2$	$kn(n+1)$
SSYR2K	kn^2	kn^2+n	$2kn^2+n$
STRMM (SIDE = 'L')	$nm(m+1)/2$	$nm(m-1)/2$	nm^2
STRMM (SIDE = 'R')	$mn(n+1)/2$	$mn(n-1)/2$	mn^2
STRSM (SIDE = 'L')	$nm(m+1)/2$	$nm(m-1)/2$	nm^2
STRSM (SIDE = 'R')	$mn(n+1)/2$	$mn(n-1)/2$	mn^2

Table 7: Operation counts for the Level 3 BLAS

LAPACK routines:

SGETRF	multiplications:	$1/2mn^2 - 1/6n^3 + 1/2mn - 1/2n^2 + 2/3n$
	additions:	$1/2mn^2 - 1/6n^3 - 1/2mn + 1/6n$
	total flops:	$mn^2 - 1/3n^3 - 1/2n^2 + 5/6n$
SGETRI	multiplications:	$2/3n^3 + 1/2n^2 + 5/6n$
	additions:	$2/3n^3 - 3/2n^2 + 5/6n$
	total flops:	$4/3n^3 - n^2 + 5/3n$
SGETRS	multiplications:	NRHS $[n^2]$
	additions:	NRHS $[n^2 - n]$
	total flops:	NRHS $[2n^2 - n]$
SGECON	multiplications:	$8n^2 + 6n + 10$
	additions:	$8n^2 - 6$
	total flops:	$16n^2 + 6n + 4$
SPOTRF	multiplications:	$1/6n^3 + 1/2n^2 + 1/3n$
	additions:	$1/6n^3 - 1/6n$
	total flops:	$1/3n^3 + 1/2n^2 + 1/6n$
SPOTRI	multiplications:	$1/3n^3 + n^2 + 2/3n$
	additions:	$1/3n^3 - 1/2n^2 + 1/6n$
	total flops:	$2/3n^3 + 1/2n^2 + 5/6n$
SPOTRS	multiplications:	NRHS $[n^2 + n]$
	additions:	NRHS $[n^2 - n]$
	total flops:	NRHS $[2n^2]$
SPOCON	multiplications:	$8n^2 + 14n + 10$
	additions:	$8n^2 - 6$
	total flops:	$16n^2 + 14n + 4$
SPBTRF	multiplications:	$n(1/2k^2 + 3/2k + 1) - 1/3k^3 - k^2 - 2/3k$
	additions:	$n(1/2k^2 + 1/2k) - 1/3k^3 - 1/2k^2 - 1/6k$
	total flops:	$n(k^2 + 2k + 1) - 2/3k^3 - 3/2k^2 - 5/6k$
SPBTRS	multiplications:	NRHS $[2nk + 2n - k^2 - k]$
	additions:	NRHS $[2nk - k^2 - k]$
	total flops:	NRHS $[4nk + 2n - 2k^2 - 2k]$
SPBCON	multiplications:	$8nk + 11n - 4k^2 - 4k + 5$
	additions:	$8nk + 4n - 4k^2 - 4k - 3$
	total flops:	$16nk + 15n - 8k^2 - 8k + 2$

SSYTRF	multiplications:	$1/6n^3 + 1/2n^2 + 10/3n$
	additions:	$1/6n^3 - 1/6n$
	total flops:	$1/3n^3 + 1/2n^2 + 19/6n$
SSYTRI	multiplications:	$1/3n^3 + 2/3n$
	additions:	$1/3n^3 - 1/3n$
	total flops:	$2/3n^3 + 1/3n$
SSYTRS	multiplications:	NRHS $[n^2 + n]$
	additions:	NRHS $[n^2 - n]$
	total flops:	NRHS $[2n^2]$
SSYCON	multiplications:	$8n^2 + 6n + 10$
	additions:	$8n^2 - 6$
	total flops:	$16n^2 + 6n + 4$
SGEQRF	multiplications:	$mn^2 - 1/3n^3 + mn + 1/2n^2 + 23/6n$
	additions:	$mn^2 - 1/3n^3 + 1/2n^2 + 5/6n$
	total flops:	$2mn^2 - 2/3n^3 + mn + n^2 + 14/3n$
SGEQRS	multiplications:	NRHS $[2mn - 1/2n^2 + 5/2n]$
	additions:	NRHS $[2mn - 1/2n^2 + 1/2n]$
	total flops:	NRHS $[4mn - n^2 + 3n]$
SORMUL (SIDE = 'L')		
	multiplications:	$2nmk - 2nqk + 2nk - nk^2$
	additions:	$2nmk - 2nqk + nk - nk^2$
	total flops:	$4nmk - 4nqk + 3nk - 2nk^2$
SORMUL (SIDE = 'R')		
	multiplications:	$2nmk - 2mqk + mk + 1/2k + nk - qk - 1/2k^2 - mk^2$
	additions:	$2nmk - 2mqk + mk - mk^2$
	total flops:	$4nmk - 4mqk + 2mk + 1/2k + nk - qk - 1/2k^2 - 2mk^2$
STRTRI	multiplications:	$1/6n^3 + 1/2n^2 + 1/3n$
	additions:	$1/6n^3 - 1/2n^2 + 1/3n$
	total flops:	$1/3n^3 + 2/3n$
SGEHRD	multiplications:	$5/3n^3 + 1/2n^2 - 7/6n - 13$
	additions:	$5/3n^3 - n^2 - 2/3n - 8$
	total flops:	$10/3n^3 - 1/2n^2 - 11/6n - 21$
SSYTRD	multiplications:	$2/3n^3 + 5/2n^2 - 1/6n - 15$
	additions:	$2/3n^3 + n^2 - 8/3n - 4$
	total flops:	$4/3n^3 + 3n^2 - 17/6n - 19$

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.

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.

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 IMAG(C), where C is COMPLEX, to DBLE(Z) and DIMAG(Z), where Z is COMPLEX*16. IBM compilers use DREAL(Z) and DIMAG(Z) to take the real and imaginary parts of a double complex number. Some effort has been made to avoid this situation altogether, but in some subroutines IBM users still may have to change DBLE to DREAL manually 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
ZLARND	from the LAPACK library
ZLATM2	from the test matrix generator library
ZLATM3	from the test matrix generator library

The functions ZDOTC and ZDOTU from the Level 1 BLAS are already declared DOUBLE COMPLEX.

We have not included test programs for the Level 1 BLAS. Users should therefore beware of a common problem in machine-specific implementations of xNRM2, the function to compute the 2-norm of a vector. The Fortran version of xNRM2 avoids underflow or overflow by scaling intermediate results, but some library versions of xNRM2 are not so careful about scaling. If xNRM2 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 xNRM2 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 LINPACK 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 LINPACK 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 LINPACK 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." [1] If this causes any difficulties, the equivalence can be commented out as explained in the comments for the main eigensystem timing programs.

[1] ANSI 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 Cray YMP. The small data sets were used for the SPARCstation and the large data sets for the Cray YMP. In both cases, the Fortran BLAS were used. These times (particularly for the Cray) were obtained on a loaded machine and should be considered rough approximations.

Test/timing run	Data set	S	C	D	Z
Linear eqn testing		86	657	121	594
Eigensystem testing	NEP	58	421	96	423
	SEP	7	48	12	46
	SVD	6	45	10	43
Linear eqn timing	Dense	101	1228	159	1011
	Banded	32	318	50	265
BLAS timing	Set 1	120	2126	184	1755
	Set 2	16	298	23	254
	Set 3	14	279	21	246
Eigensystem timing	NEP	782	7718	1574	7930
	SEP	36	631	66	675
	SVD	189	2253	322	2003

Table 8: Sun SPARCstation execution times (in seconds)

Test/timing run	Data set	S	C
Linear eqn testing		112	197
Eigensystem testing	NEP	137	236
	SEP	7	12
	SVD	5	8
Linear eqn timing	Dense	569	1852
	Banded	65	160
BLAS timing	Set 1	477	3540
	Set 2	106	583
	Set 3	68	532
Eigensystem timing	NEP	4424	5775
	SEP	142	693
	SVD	555	890

Table 9: Cray YMP – 1 processor, execution times (in seconds)

References

[1] E. Anderson and J. Dongarra, *LAPACK Working Note 16: Results from the Initial Release of LAPACK*, University of Tennessee, CS-89-89, November 1989.

[2] 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, ANL-88-38, September 1988.

[3] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, to appear in *ACM Trans. Math. Soft.*, March 1990.

[4] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs*, to appear in *ACM Trans. Math. Soft.*, March 1990.

[5] J. Dongarra, J. Du 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.

[6] J. Dongarra, J. Du 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.

[7] 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.