.

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

LAPACK Working Note #10

Installing and Testing the Initial Release of LAPACK

Unix and Non-Unix Versions

Edward Anderson and Jack Dongarra

Mathematics and Computer Science Division

Technical Memorandum No. MCS-TM-130

May 1989

Work supported in part by the National Science Foundation, under Contract NSF ASC-8715728.

.

Installing and Testing the Initial Release of LAPACK

Unix and Non-Unix Versions

Edward Anderson and Jack Dongarra

Abstract — This working note describes how to install and test the initial release of LAPACK.

1. Introduction

This working note describes how to install and test the initial release of LAPACK. LAPACK is intended to provide a uniform set of subroutines to solve the most common linear algebra problems and to run efficiently on a wide range of architectures. The routines presented at this time are intended not for general distribution, but only for initial testing. 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. Section 4 provides instructions for Unix users installing a *tar* tape, and Section 5 contains instructions for non-Unix users. We assume the installer has experience in compiling and running Fortran programs and in creating object libraries. The installation process involves reading a tape, creating a library from the Fortran source, running the tests, and sending the results to Argonne.

This release contains only a fraction of the routines that will be part of LAPACK. We have included routines to deal with solving systems of equations for the following types of matrices and storage:

Data type	Matrix type	Storage type
S, D, C, Z	General	general, banded
S, D	Symmetric positive definite	general, packed, banded
S, D, C, Z	Symmetric indefinite	general, packed, banded
C, Z	Hermitian positive definite	general, packed, banded
C, Z	Hermitian indefinite	general, packed, banded
S, D, C, Z	Triangular	general, packed

In addition, we have provided routines to perform the QR factorization of a general matrix.

We expect the next test release of LAPACK to include routines for computing eigenvalues and eigenvectors. We hope to have the next release ready in the late summer or early fall of

1989.

2. Format

LAPACK is distributed in the form of a tape which contains the Fortran source for LAPACK, as well as the Basic Linear Algebra Subprograms (the Level 1, 2, and 3 BLAS) needed by LAPACK, the testing programs, and the timing programs.

2.1. Unix Version

For Unix users, the software is distributed in the form of a *tar* tape containing a number of directories structured as follows:

"LAPACK" at 3.25,5.0 line from 3.25,4.9 to 0.95,4.15 line from 3.25,4.9 to 2.65,4.15 line from 3.25,4.9 to 4.05,4.15 line from 3.25,4.9 to 5.6,4.15 "BLAS" at 0.9,4.0 line from 0.9,3.9 to 0.25,3.15 line from 0.9,3.9 to 1.35,3.15 "SRC" at 0.25,3.0 line from 0.25,2.9 to 0.25,2.15 "makefiles" at 1.35, 3.0 "input files" at 1.35, 2.85 "BLAS2 & 3 test" at 1.35,2.7 "routines" at 1.35,2.55 "BLAS2 & 3 timing" at 1.35,2.4 "routines" at 1.35,2.25 "makefile" at 0.25,2.0 "Level 1 BLAS" at 0.25,1.85 "Level 2 BLAS" at 0.25,1.7 "Level 3 BLAS" at 0.25,1.55 "SRC" at 2.65,4.0 line from 2.65,3.9 to 2.65,3.15 "makefile" at 2.65,3.0 "LAPACK routines" at 2.65,2.85 "LAPACK auxiliary" at 2.65,2.7 "routines" at 2.65,2.55 "TESTING" at 4.05,4.0 line from 4.05,3.9 to 3.65,3.15 line from 4.05,3.9 to 4.55,3.15 "SRC" at 3.65,3.0 line from 3.65,2.9 to 3.65,2.15 "input files" at 4.55,3.0 "makefile" at 3.65,2.0 "LAPACK test" at 3.65,1.85 "routines" at 3.65,1.7 "TIMING" at 5.65,4.0 line from 5.65,3.9 to 5.25,3.15 line from 5.65,3.9 to 6.15,3.15 "SRC" at 5.25,3.0 line from 5.25,2.9 to 5.25,2.15 "input files" at 6.15,3.0 "makefile" at 5.25,2.0 "LAPACK timing" at 5.25,1.85 "routines" at 5.25,1.7

2.2. Non-Unix Version

For non-Unix users, the software is distributed on an unlabeled ASCII tape containing 73 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 (see Section 5.1). Files whose names end in 'F' contain Fortran source code; those whose names end in 'D' contain data for input to the test and timing programs. Most of the files occur in groups of four, corresponding to the different versions of the routines (see Section 3). In the installation instructions we shall frequently refer to these files generically, using 'x' in place of the first letter (for example, xLASRCF).

File number	File name	Description of contents
1	README	This file
2	ALLAUXF	LAPACK auxiliary routines common to more than one version
3	SCLAUXF	
4	DZLAUXF	
5	SLASRCF	LAPACK routines and auxiliary routines
6	CLASRCF	
7	DLASRCF	
8	ZLASRCF	
9	SBLAS1F	Level 1 BLAS routines
10	CBLAS1F	
11	DBLAS1F	
12	ZBLAS1F	
13	ALLBLASF	Auxiliary routines for the Level 2 and 3 BLAS
14	SBLAS2F	Level 2 BLAS routines
15	CBLAS2F	
16	DBLAS2F	
17	ZBLAS2F	
18	SBLAS3F	Level 3 BLAS routines
19	CBLAS3F	
20	DBLAS3F	
21	ZBLAS3F	
22	SBLAT2F	Level 2 BLAS test programs
23	CBLAT2F	
24	DBLAT2F	
25	ZBLAT2F	
26	SBLAT2D	Data files for Level 2 BLAS test programs
27	CBLAT2D	
28	DBLAT2D	
29	ZBLAT2D	
30	SBLAT3F	Level 3 BLAS test programs
31	CBLAT3F	
32	DBLAT3F	
33	ZBLAT3F	

34	SBLAT3D	Data files for Level 3 BLAS test programs
35	CBLAT3D	
36	DBLAT3D	
37	ZBLAT3D	
38	SB2TIMF	Level 2 BLAS timing programs
39	CB2TIMF	
40	DB2TIMF	
41	ZB2TIMF	
42	SB2TIMD	Data files for Level 2 BLAS timing programs
43	CB2TIMD	
44	DB2TIMD	
45	ZB2TIMD	
46	SB3TIMF	Level 3 BLAS timing programs
47	CB3TIMF	
48	DB3TIMF	
49	ZB3TIMF	
50	SB3TIMD	Data files for Level 3 BLAS timing programs
51	CB3TIMD	
52	DB3TIMD	
53	ZB3TIMD	
54	SLATSTF	LAPACK test programs
55	CLATSTF	
56	DLATSTF	
57	ZLATSTF	
58	SLATSTD	Small data files for LAPACK test programs
59	CLATSTD	
60	DLATSTD	
61	ZLATSTD	
62	SLATS2D	Large data files for LAPACK test programs
63	CLATS2D	
64	DLATS2D	
65	ZLATS2D	
66	SLATIMF	LAPACK timing programs
67	CLATIMF	
68	DLATIMF	
69	ZLATIMF	

70	SLATIMD	Data files for LAPACK timing programs
71	CLATIMD	
72	DLATIMD	
73	ZLATIMD	

3. Overview of Tape Contents

Most routines in LAPACK occur in four versions: REAL, DOUBLE PRECISION, COMPLEX, and COMPLEX*16 (or DOUBLE COMPLEX). The first three versions (REAL, DOUBLE PRECISION, and COMPLEX) are written in standard Fortran 77; the COMPLEX*16 version is provided for those compilers that allow this data type. For convenience, we 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 double complex. 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

The LAPACK routines and the auxiliary routines (except for the BLAS) called by the LAPACK routines are contained in the directory LAPACK/SRC for the Unix version and in the files xLASRCF and xxLAUXF for the non-Unix version.

An LAPACK routine is one of the computational routines described in [1] to perform a distinct algorithmic task, such as performing an LU factorization or solving a system of equations given the LU factorization of the coefficient matrix. An LAPACK auxiliary routine is a routine to perform a specific task which is called from one of the LAPACK routines. The tasks performed by the auxiliary routines are more general and may be applicable in more than one context. 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. More details on the scope of the LAPACK project are available in [1].

3.2. Level 1, 2, and 3 BLAS

LAPACK employs the Level 1, 2, and 3 BLAS to carry out basic operations. The key to getting good performance from LAPACK lies in having an efficient version of the BLAS optimized for your particular machine. If you already have a library of the BLAS on your machine, we recommend that, before using it, you first run the BLAS tests provided on the tape to verify that the routines in your library are correct.

For the Unix version, the directory LAPACK/BLAS/SRC contains the Level 1, 2, and 3 BLAS written in portable Fortran 77. Test routines and their input files can be found in LAPACK/BLAS. Timing routines and their input files are also found in LAPACK/BLAS. A

quick reference to the BLAS is provided in Appendix C; more details are in [2, 4, 6].

For the non-Unix version, the files xBLAS1F, xBLAS2F, and xBLAS3F contain the Level 1, 2, and 3 BLAS written in portable Fortran 77. Test programs for the Level 2 and 3 BLAS can be found in the files xBLAT2F and xBLAT3F, with input files xBLAT2D and xBLAT3D. Timing programs can be found in the files xB2TIMF and xB3TIMF, with input files xB2TIMD and xB3TIMD.

3.3. LAPACK Test Routines

The source code for all of the LAPACK test routines is in LAPACK/TESTING/SRC for the Unix version and in the files xLATSTF for the non-Unix version. For the Unix version, the main program in the single-precision case is in the file *slats1.f* ; for the non-Unix version, each file begins with the main program, which in the real single-precision case is called SLATS1.

The main program reads in the input file and directs the testing of each LAPACK path (by a path we mean the set of routines associated with a particular type of matrix or type of operation). The main procedure calls separate routines with names like SCHK01 to check each path. For example, SCHK01 tests the SGE routines involving the *LU* decomposition of a general matrix. For each matrix order N and each matrix type indicated in the input file, SCHK01 generates a test matrix, calls the LAPACK routines to factor, solve, invert, and compute the condition number of this matrix, and computes certain test ratios after each operation to verify that it has been successfully completed. The functions of generating a matrix and computing the test ratios are carried out in other test subroutines.

3.4. LAPACK Timing Routines

The source code for all of the LAPACK timing routines is in LAPACK/TIMING/SRC for the Unix version and in the files xLATIMF for the non-Unix version. For the Unix version, the main program in the single-precision case is in the file *stime.f* ; for the non-Unix version, each file begins with the main program, which in the real single-precision case is called STIME.

The main program reads in the input file and directs the timing of the routines in each LAPACK path. The main procedure calls separate routines with names like STIM01 to time each path. For example, STIM01 times the SGE routines involving the *LU* decomposition of a general matrix. For each matrix order N and each SGE routine indicated in the input file, STIM01 generates a test matrix and calls the LAPACK routine, repeating the operation if necessary until the time elapsed exceeds some minimum time. Then STIM01 computes the number of operations performed and determines the megaflop rate for this run, dividing the number of operations (in millions) by the time in seconds. The functions of generating a matrix and computing the number of operations performed are carried out in other timing subroutines.

3.5. Libraries and Test Programs

For the Unix version, the libraries and test programs are created using the makefiles in each directory. Target names are supplied for each of the four precisions and are called

```
single  double  complex  complex16
```

To create a library from one of the makefiles called *makefile*, you simply type *make* followed by the precisions desired. Examples:

```
make  single
make  double  complex16
make  single  double  complex  complex16
```

Alternatively,

```
make
```

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

The makefiles for the test routines create separate test programs for each precision. These programs can be created one at a time:

```
make  single
make  double
. . .
```

or all at once:

```
make  single  double  complex  complex16
```

As before, the last command is equivalent to typing *make* by itself. In cases 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
```

For the non-Unix implementation, each of the four versions of LAPACK (S, C, D, and Z), together with the corresponding versions of the BLAS, can be installed and tested independently. The only files that are common to more than one version are ALLBLASF, ALLAUXF, SCLAUXF, and DZLAUXF; their use is described in Sections 5.2 and 5.4.

4. Instructions to Unix Installers of LAPACK

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

1) Read the tape.
2) If you do not have all the Level 1, 2, and 3 BLAS on your system, make the BLAS library.
3) Make the Level 2 and 3 BLAS test programs.
4) Run the Level 2 and 3 BLAS test programs.
5) Make the LAPACK library.
6) Make the LAPACK test programs.
7) Run the LAPACK test programs.
8) Make the LAPACK timing programs.
9) Run the LAPACK timing programs.
10) Make the BLAS timing programs.
11) Run the BLAS timing programs.

Results from steps 7, 9, and 11 should be sent to Argonne.

4.1. Read the Tape

To unload the tape, type one of the commands

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 7.1 megabytes to unload the complete tape, plus room for the libraries and executable files. On a Sun 3/260, the libraries used 1.4 MB and the executables used 7.9 MB. In addition, the object files used 3.8 MB, but the object files can be deleted after creating the libraries and executable files. The total space requirement including the object files is approximately 20 MB.

4.2. Make the BLAS Library

In an ideal world, a highly optimized version of this library already exists on your machine. In this case you can go straight to Section 4.3 to make the BLAS test programs. If you have some of the BLAS, but not all (Level 1 and 2, but not 3, for example), you can edit out the BLAS you do not need in the makefile.

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

b) If you already have some of the BLAS, comment out the lines defining the BLAS you have.

c) Type *make* followed by the precisions desired, as in the examples in Section 3.5. The make command can be run more than once to add another precision to the library if necessary.

d) Move the library *blas.a* to the directory LAPACK/BLAS with the command

```
mv blas.a ..
```

4.3. Make the BLAS Test Programs

It is not uncommon to find bugs in the system-supported BLAS library or in the Fortran compiler if you compile the BLAS from the Fortran source code. Before proceeding to the testing of LAPACK, you should get some assurance that the BLAS are functioning correctly.

a) To make the BLAS 2 test programs, go to the directory LAPACK/BLAS 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. Also define BLAS to point to your system's BLAS library or to the library *blas.a* created in Section 4.2.

b) Type *make -f makeblat2* followed by the precisions desired, as in the examples in Section 3.5. The test programs are called *xblat2s, xblat2c, xblat2d,* and *xblat2z*.

c) Repeat steps a) and b) for the makefile *makeblat3* to make the BLAS 3 test programs. These programs are called *xblat3s, xblat3c, xblat3d*, and *xblat3z*.

4.4. Run the Level 2 and 3 BLAS Test Programs

a) Go to the directory LAPACK/BLAS and run the Level 2 BLAS tests:

```
xblat2s  <sblat2.in
xblat2c  <cblat2.in
xblat2d  <dblat2.in
xblat2z  <zblat2.in
```

The name of the output file is indicated on the first line of each input file and is currently defined to be SBLAT2.SUMM in the single-precision case, with similar names for the other precisions. If the runs have been successful and the tests have passed, each run should produce a couple pages of output; see [5] for more details.

b) From the directory LAPACK/BLAS, run the Level 3 BLAS tests:

```
xblat3s  <sblat3.in
xblat3c  <cblat3.in
xblat3d  <dblat3.in
xblat3z  <zblat3.in
```

The name of the output file is indicated on the first line of each input file and is currently defined to be SBLAT3.SUMM in the single-precision case, with similar names for the other

precisions. If the runs have been successful and the tests have passed, each run should produce a couple pages of output; see [3] for more details.

4.5. Make 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 precisions desired, as in the examples in Section 3.5. The make command can be run more than once to add another precision to the library if necessary.

c) Move the library *lapack.a* to the LAPACK directory with the command

```
mv lapack.a ..
```

4.6. Make the LAPACK Test Programs

a) Go to the directory LAPACK/TESTING/SRC 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. Also define BLAS to point to your system's BLAS library or to the library *blas.a* created in Section 4.2, and define LAPACK to point to the library *lapack.a* created in Section 4.5.

b) Type *make* followed by the precisions desired, as in the examples in Section 3.5.

c) Move the executable files up a level with the command

```
mv x* ..
```

The test programs are called *xchks, xchkc, xchkd,* and *xchkz* and should reside in the directory LAPACK/TESTING.

4.7. Run the LAPACK Test Programs

We have provided two sets of input files for the LAPACK test programs, a small set containing matrices up to order 10 and another larger set that includes matrices up to order 129. The small test set is included to provide a quick check of the software; it should run for at most a few minutes on a typical workstation. The larger test set includes all of the test cases in the smaller set plus some larger matrices, at least one of which should be larger than the vector register length. We would like you to send the results from only the larger test set to Argonne.

We also encourage you to conduct other tests of your own and inform us of any unusual results or areas for improvement. For more information on the test programs, see Section 6.

To run the small set of tests (optional):

a) Go to the directory LAPACK/TESTING and execute the commands

```
xchks  <slats.in
xchkc  <clats.in
xchkd  <dlats.in
xchkz  <zlats.in
```

On a Sun 3/260, these tests took from 30 seconds to 4 minutes to run. The results appear in the files

```
slats.out  clats.out  dlats.out  zlats.out
```

To run the large set of tests:

b) If you are running on a non-vector machine, edit the input files, such as *slats.in2*, and use only the first 10 values of N and the first 2 values of NB. See Section 6.2 for more information on the test program input file.

c) Execute the commands

```
xchks  <slats.in2
xchkc  <clats.in2
xchkd  <dlats.in2
xchkz  <zlats.in2
```

On a Sun 3/260, using the first 10 values of N and the first 2 values of NB, the running time of these tests varied from 10 minutes for *xchks* to nearly 3 hours for *xchkz*.

d) Send the files

```
slats.chk  clats.chk  dlats.chk  zlats.chk
```

by e-mail to

andersn@mcs.anl.gov

Please tell us the type of machine on which the tests were run and the compiler options that were used.

4.8. Make the LAPACK Timing Programs

All of the timing programs call a REAL function SECOND with no arguments, which is assumed to return the central processor time in seconds from some fixed starting time. We have not supplied this routine and you must provide the correct interface on your machine. (This may be simply a call to some other timing function on your machine.) See Appendix E for a Unix version of SECOND.

a) Go to the directory LAPACK/TIMING/SRC 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. Also define BLAS to point to your system's BLAS library or to the library *blas.a* created in Section 4.2, and define LAPACK to point to the library *lapack.a* created in Section 4.5.

b) Define a REAL function called SECOND that returns the time from a fixed starting time. If SECOND is an intrinsic function on your machine, remove the reference to the file *second.o* in the makefile.

c) Type *make* followed by the precisions desired, as in the previous examples.

d) Move the executable files up a level with the command

```
mv x* ..
```

The timing programs are called *xtims, xtimc, xtimd,* and *xtimz* and should reside in the directory LAPACK/TIMING.

4.9. Run the LAPACK Timing Routines

We have set up the timing programs to produce benchmark results suitable for a supercomputer. This means that the timing programs may use too much memory to run on your machine. If this is the case, you will need to decrease the declared size of the matrices (set in the parameter NMAX) and eliminate any values of N or LDA in the input file that exceed the new maximums. For more information on the timing programs, see Section 7.

a) Go to the directory LAPACK/TIMING and, if necessary, edit the input files. In particular, you may find it necessary to change the minimum time a subroutine will be timed, currently set at 0.1 seconds, or the values of N, currently consisting of powers of 2 from 32 to 512. For a description of the input file and how to modify it, see Section 7.2.

b) Run the timing programs by executing the commands

```
xtims <stime.in >stime.out
xtimc <ctime.in >ctime.out
xtimd <dtime.in >dtime.out
xtimz <ztime.in >ztime.out
```

By default, the results appear on standard output; these commands will redirect them to a file.

c) Send the results to

andersn@mcs.anl.gov

along with any comments you may have. Please tell us the type of machine on which the tests were run and the compiler options that were used.

4.10. Make the BLAS Timing Programs

The BLAS timing programs are also included to provide a standard of comparison for the LAPACK timing results. Most of the directions for making the BLAS timing programs are the same as for making the LAPACK timing programs (Section 4.8).

a) Go to the directory LAPACK/BLAS and edit the makefile called *makeblas2time*. 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. Also define BLAS to point to your system's BLAS library or to the library *blas.a* created in Section 4.2.

b) Define a REAL function called SECOND that returns the time from a fixed starting time.

c) Type *make -f makeblas2time* followed by the precisions desired, as in the previous examples.

d) Repeat steps a-c for the makefile *makeblas3time*.

4.11. Run the BLAS Timing Programs

a) Go to the directory LAPACK/BLAS and, if necessary, edit the input files to include only the matrix sizes your machine is capable of running.

b) Run the timing programs by executing the commands

```
xb2times <sb2tim.in >sb2tim.out
xb2timec <cb2tim.in >cb2tim.out
xb2timed <db2tim.in >db2tim.out
xb2timez <zb2tim.in >zb2tim.out

xb3times <sb3tim.in >sb3tim.out
xb3timec <cb3tim.in >cb3tim.out
xb3timed <db3tim.in >db3tim.out
xb3timez <zb3tim.in >zb3tim.out
```

The results are printed on standard output; these command redirect them to a file.

c) Send the output files to

andersn@mcs.anl.gov

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. Instructions to Non-Unix Installers of LAPACK

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

1) Read the tape.

2) If you do not have all the Level 1, 2, and 3 BLAS on your system, create a BLAS library.

3) Run the Level 2 and 3 BLAS test programs.

4) Create the LAPACK library.

5) Run the LAPACK test programs.

6) Run the LAPACK timing programs.

7) Run the BLAS timing programs.

Results from steps 5, 6, and 7 should be sent to Argonne.

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 machine-readable copy of these directions, and you may be able to use the list of files and file names from Section 2 to name the files as the tape is read. You will need about 7.3 megabytes to read the complete tape, plus room for the libraries and executable files. If you do not wish to install all four versions of LAPACK, you may elect to read only those files which correspond to the version(s) you have chosen.

On a Sun 3/260, the libraries used 1.4 MB and the executables used 7.9 MB. In addition, the object files used 3.8 MB, but the object files can be deleted after creating the libraries and executable files. The total space requirement for all four versions, including the object files, is therefore approximately 20 MB.

5.2. Create a BLAS Library

In an ideal world, a highly optimized version of this library already exists on your machine. In this case you can go straight to Section 5.3 to run 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.

5.3. Run the BLAS Test Programs

It is not uncommon to find bugs in the system-supported BLAS library or in the Fortran compiler if you compile the BLAS from the Fortran source code. Before proceeding to the testing of LAPACK, you should get some assurance that the BLAS are functioning correctly.

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 single-precision 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 runs have been successful and the tests have passed, each run should produce a couple of pages of output; see [3] and [5] for more details.

5.4. 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 file SCLAUXF, and if you have compiled either the D or Z version, you must also compile and include the file DZLAUXF. If you did not compile the file ALLBLASF and include it in your BLAS library as described in Section 5.2, you must compile it now and include it in your LAPACK library.

5.5. Run the LAPACK Test Programs

Compile the files xLATSTF 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).

We have provided two sets of input files for the LAPACK test programs, a small set containing matrices up to order 10 and another larger set containing matrices up to order 129. The small test set is included to provide a quick check of the software; it should run for at most a few minutes on a typical workstation. The larger test set includes all of the test cases in the smaller set plus some larger matrices, at least one of which should be larger than the vector register length. We would like you to send the results from only the larger test set to Argonne.

We also encourage you to conduct other tests of your own and inform us of any unusual results or areas for improvement. For more information on the test programs, see Section 6.

To run the small set of tests (optional):

a) The input files for the small set of tests are named xLATSTD. Associate the appropriate file with Fortran unit number 5.

b) The name of the output file is indicated on the first line of each input file and is currently defined to be SLATS.OUT for the real single-precision version, 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.

c) Run the programs. On a Sun 3/260, these programs took from 30 seconds to 4 minutes to run.

To run the large set of tests:

d) The input files for the large set of tests are named xLATS2D. Associate the appropriate file with Fortran unit number 5.

e) If you are running on a non-vector machine, edit the input files and use only the first 10 values of N and the first 2 values of NB. See Section 6.2 for more information on the test program input file.

f) The name of the output file is indicated on the first line of each input file and is currently defined to be SLATS.CHK for the real single-precision version, 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.

g) Run the programs. On a Sun 3/260, these programs took from 10 minutes to nearly 3 hours to run.

h) Send the output files from the large set of tests by e-mail to

andersn@mcs.anl.gov

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.

5.6. Run the LAPACK Timing Routines

All of the timing programs call a REAL function SECOND with no arguments, which is assumed to return the central processor time in seconds from some fixed starting time. We have not supplied this routine, and you must provide the correct interface on your machine. (This may be simply a call to some other timing function on your machine.)

We have set up the timing programs to produce benchmark results suitable for a supercomputer. This means that the timing programs may use too much memory to run on your machine. If so, you will need to decrease the declared size of the matrices (set in the parameter NMAX) and

eliminate any values of N or LDA in the input file that exceed the new maximum. For more information on the timing programs, see Section 7.

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) The input files for the timing programs are named xLATIMD. Associate the appropriate file with Fortran unit number 5.

c) You may find it necessary to edit the input files in order to change the minimum time a subroutine will be timed, currently set at 0.1 seconds, or the values of N, currently consisting of powers of 2 from 32 to 512. For a description of the input file and how to modify it, see Section 7.2.

d) The output file is written to Fortran unit number 6. Associate a suitably named file (e.g., STIME.OUT for the real single-precision version) with this unit number.

e) Run the programs.

f) Send the output files by e-mail to

   andersn@mcs.anl.gov

   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. Run the BLAS Timing Programs

The BLAS timing programs are also included to provide a standard of comparison for the LAPACK timing results.

a) Compile the files xB2TIMF and xB3TIMF, and link them to your BLAS library or libraries.

b) The input files for the timing programs are named xB2TIMD and xB3TIMD. Associate the appropriate file with Fortran unit number 5.

c) The output file is written to Fortran unit number 6. Associate a suitably named file (e.g., SB2TIM.OUT) with this unit number.

d) Run the programs.

e) Send the output files by e-mail to

   andersn@mcs.anl.gov

   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.

6. More about Testing

A separate test program exists for each of the four data types (REAL, COMPLEX, DOUBLE PRECISION, and COMPLEX*16). The program is driven by a data file that specifies

— a set S_n of values of n
— a set S_{nb} of values of nb (the blocksize)
— a set S_{path} of LAPACK path names

Specified along with each LAPACK path name is either a list of test matrix types to be used in testing that path or an indication that all of the types should be used. The outline of the test procedure is as follows:

```
for nb ∈ S_nb
    for each path name ∈ S_path
        for n ∈ S_n
            for each matrix type
                {Test each routine in this path}
```

The main test program (in the file *slats1.f* in single precision in the Unix version) defines several program maximums which can be modified if necessary.

Parameter	Description	Value
NMAX	Maximum value for N*	400
NIDMAX	Number of different values of N	12
NNBMAX	Number of different values of NB	10

* *NMAX* must be at least $3N-2$ to test the xGB routines and at least $2N$ to test the xSB or xHB routines.

6.1. Test Matrices and Test Ratios

The test routine for each LAPACK path generates a number of different test matrices and, for each matrix, calls the LAPACK routines in that path and computes certain test ratios to verify that each operation has been successfully completed. Up to 12 different types of test matrices may be used:

Diagonal matrix
Upper triangular matrix
Lower triangular matrix
Banded matrix: $kl < m/2, ku < n/2$
Banded matrix: $kl < m/2, ku > n/2$
Banded matrix: $kl > m/2, ku < n/2$
Banded matrix: $kl > m/2, ku > n/2$
Random matrix with condition number = 2
Random matrix with condition number = [illegible]
Random matrix with condition number = $0.1/\varepsilon$
Matrix scaled near underflow limit
Matrix scaled near overflow limit

Here, ε is the machine epsilon, i.e., the smallest positive floating-point number such that $1.0 + \varepsilon \neq 1.0$. Upper and lower triangular matrices are valid only for routines that operate on non-symmetric or non-Hermitian matrices, and banded matrices are valid only for the band routines.

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

a) Factor the matrix using xxxTRF, and compute the ratio

$$\frac{||LU - A||}{(\sqrt{n}\ ||A||\varepsilon)}$$

b) Invert the matrix A using xxxTRI, and compute the ratio

$$\frac{||I - AA^{-1}||}{(\sqrt{n}\ ||A||\ ||A^{-1}||\varepsilon)}$$

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

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

$$\frac{||b - Ax||}{(||A||\ ||x||\varepsilon)} \quad \text{and} \quad \frac{||x - x_*||}{(||x_*||\kappa\varepsilon)}$$

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

d) If the options are available, repeat step c) with A replaced by A^T or A^H.

e) Compute the condition number using xxxCON, and compute the product *RCOND* * κ.

6.2. The Test Program Input File

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. As an illustration, the input file for the single-precision real test program is as follows:

```
´slats.chk´          NAME OF SUMMARY OUTPUT FILE
6                    UNIT NUMBER OF SUMMARY FILE
´SLATS1.SNAP´        NAME OF SNAPSHOT OUTPUT FILE
-1                   UNIT NUMBER OF SNAPSHOT FILE (NOT USED IF .LT. 0)
F                    PUT T TO REWIND SNAPSHOT FILE AFTER EACH RECORD.
F                    PUT T TO STOP ON FAILURES.
T                    PUT T TO TEST ERROR EXITS.
20.0                           THRESHOLD VALUE OF TEST RATIO
11                             NUMBER OF VALUES OF N
0 1 2 3 4 5 10 20 30 40 129    VALUES OF N
3                              NUMBER OF VALUES OF NB (BLOCK SIZE)
1 8 130                        VALUES OF NB
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
SSB     8            List types on next line if 0 < NTYPES < 8
SQR     8            List types on next line if 0 < NTYPES < 8
```

The first line directs the output to appear in a file called *slats.chk*, and the program will write the output to Fortran logical unit number 6. If any of the test ratios are above a threshold value of 20., a message will be printed. Eleven values of *N* and three values of the blocksize *NB* are specified in this example.

The remaining lines of the input file specify the matrix types to be used in testing each LAPACK path name. We have set the options in the input file to run through all the available tests and matrix types for each LAPACK path. For example,

```
SGE     8            List types on next line if 0 < NTYPES < 8
```

indicates that all 8 matrix types are to be tested for path SGE. We could skip testing the SGE

routines by placing a 0 instead of an 8 after SGE. If more than 0 but fewer than 8 matrix types are requested, a second line is required giving the numbers of the desired types. For example, to test the SGE path only on square matrices with small condition numbers, replace the line beginning with SGE with the following two lines:

```
SGE     1            List types on next line if 0 < NTYPES < 8.
  4
```

A complete list of matrix types and the tests that are performed for each LAPACK path name appears in Appendix D.

Any test that produces a ratio greater than or equal to the threshold will cause 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 this path. A sample line describing a test that did not pass the threshold when the threshold was set to 1.0 is as follows:

```
Matrix of order   10, type  2, test  7, ratio =   1.30901
```

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, as follows:

```
SGE:     2 out of  392 tests failed to pass the threshold
```

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

```
All tests for SGE passed the threshold ( 392 tests run)
```

Note that if any of the specifications in the input file are invalid, a diagnostic message will be printed to the output file and none of the tests will be run.

7. More about Timing

A separate timing program also exists for each of the four data types (REAL, COMPLEX, DOUBLE PRECISION, and COMPLEX*16). The program is driven by a data file that specifies

— a set S_n of values of n
— a set S_{nb} of values of nb (the blocksize)
— a set S_k of values of k (the half-bandwidth)
— a set S_{lda} of values of LDA (the leading dimension)
— a set S_{path} of LAPACK path names

Specified along with each LAPACK path name is a list of requests that tells whether or not to time each routine in that path. The general outline of the timing procedure is as follows (not all options are applicable for each path name):

for each relevant value of *UPLO*
  for $LDA \in S_{lda}$
    for $k \in S_k$
      for each path name $\in S_{path}$
        for $n \in S_n$
          for $nb \in S_{nb}$
            {Time the blocked routines in this path}
          {Time the unblocked routines in this path}

Results over all relevant values of n and nb for each path name are collected in a table and printed to standard output.

The main timing program (in the file *stime.f* in single-precision) defines several program maximums that can be modified if necessary.

Parameter	Description	Value
NMAX	Maximum value for N, NB, or K*	512
LDAMAX	Maximum value for LDA	544
MAXVAL	Maximum number of values of N, NB, or K	12
MXNLDA	Maximum number of values of LDA	12

* *NMAX* must be at least $3K+1$ to test the xGB routines and at least $2K+2$ to test the xSB or xHB routines.

7.1. Timing Matrices and Results

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.

The performance of the LAPACK routines is reported in megaflops. Since this measure of speed is relative to the architecture and the specific implementation of the BLAS, calls are always made to the matrix-vector multiply routine xGEMV and the matrix-matrix multiply routine xGEMM to provide a standard for comparison.

7.2. The Timing Program Input File

From the timing program's input file, one can control the order of the test matrices, the block size for the blocked routines, the bandwidth for the banded routines, the leading dimension of the work arrays, and the individual routines to be timed. We have set the options in the input

files to run through all the timing paths. As an illustration, the input file for the single-precision real timing program is as follows:

```
Data file for timing program for the REAL LAPACK routines
5                       Number of values of N (order, maximum 12)
32 64 128 256 512       The values of N  (maximum is NMAX)
5                       Number of values of NB (blocksize, maximum 12)
1 2 8 16 32             The values of NB (maximum is NMAX)
5                       Number of values of K (bandwidth, maximum 12)
31 63 127 255 511       The values of K  (maximum is NMAX)
2                       Number of values of LDA (maximum 12)
512 513                 The values of LDA (maximum is LDAMAX)
0.1                     Minimum time (repeat the routine if necessary)
SGE     T T T T         Put T to time:  TRF  TRS  TRI  CON
SGB     T T T                           TRF  TRS       CON
SPO     T T T T                         TRF  TRS  TRI  CON
SPP     T T T T                         TRF  TRS  TRI  CON
SPB     T T T                           TRF  TRS       CON
SSY     T T T T                         TRF  TRS  TRI  CON
SSP     T T T T                         TRF  TRS  TRI  CON
SSB     T T T                           TRF  TRS       CON
SGEQR   T T                             QRF  QRS
STR     T                                         TRI
STP     T                                         TRI
```

The first 10 lines of this file are read using list-directed input, so the spacing of the numbers on a line is not significant. The lines beginning with SGE specify the routines to be timed. The first 6 characters are reserved for the path name, which should begin in column 1, and the first few nonblank characters to the right of the path name determine whether a routine will be timed, where 'T' or 't' means ''Time this routine''. The list of suffixes on the right-hand part of the line is not read.

The output is in the form of a table which shows the megaflop rates for each routine over all values of N. For blocked routines, the table has one line for each different block size NB. For the solve routines, the table has one line for each of four different numbers of right-hand sides: 1, 2, $N/2$, and N. Separate tables are generated for different values of LDA and K, and also for upper triangular and lower triangular storage in the symmetric and Hermitian cases.

Appendix A. LAPACK Routines

In this appendix, we review the subroutine naming scheme for LAPACK as described in [1] and indicate by means of a table which subroutines are included in this initial 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 or DOUBLE COMPLEX (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
SB symmetric band
HE (complex) Hermitian (i.e., indefinite)
HP (complex) Hermitian, packed storage
HB (complex) Hermitian band
OR (real) orthogonal
UN (complex) unitary
TR triangular
TP triangular, packed storage

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

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
QRF	perform the QR factorization without pivoting
QR2	unblocked version of QRF
QRS	solve linear least squares problems (based on QR factorization)
GNC	generate a real orthogonal or complex unitary matrix as a product of Householder matrices, where each Householder vector is stored in a column of the matrix
GC2	unblocked version of GNC
MLC	multiply a matrix by a real orthogonal or complex unitary matrix by applying a product of Householder matrices, where each Householder vector is stored in a column of the matrix
MC2	unblocked version of MLC

Given these definitions, the following table indicates the LAPACK subroutines provided in this release:

	GE	GB	PO	PP	PB	HE SY	HP SP	HB SB	UN OR	TR	TP
TRF	×	×	×	×	×	×	×	×			
TF2	×		×		×						
TRS	×	×	×	×	×	×	×	×			
TRI	×		×	×		×	×			×	×
TI2										×	
CON	×	×	×	×	×	×	×	×			
QRF	×										
QR2	×										
QRS	×										
GNC									×		
GC2									×		
MLC									×		
MC2									×		

Appendix B. LAPACK Auxiliary Routines

This appendix lists all of the auxiliary routines (except for the BLAS) that are called from the LAPACK routines. On the tape, these routines reside in the directory LAPACK/SRC for Unix users or in the files xxLAUXF and xLASRCF for non-Unix users. Except as indicated, routines specified with an underscore as the first character are available in all four precisions (S, D, C, and Z).

Special subroutines:

ENVIR	Determine the block size
SMACHR	Single-precision routine for computing machine parameters
DMACHR	Double-precision routine for computing machine parameters
XERBLA	Error handler for LAPACK routines

Special functions:

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

Pseudo-BLAS2 routines used with complex symmetric matrices

_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
_LAEV2	Compute eigenvalues and eigenvectors of a real symmetric or complex Hermitian 2 x 2 matrix
_LAPY2	(real) Compute square root of X**2 + Y**2
_LAPY3	(real) Compute square root of X**2 + Y**2 + Z**2
_LARF	Multiply by a Householder matrix
_LARFB	Multiply by a block Householder matrix
_LARFG	Generate a Householder matrix
_LARFT	Accumulate a block Householder matrix
_LASSQ	Compute a scaled sum of squares of the elements of a vector
_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
_LAVHB	(complex) Multiply a vector by a matrix that has been factored by _HBTRF
_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
_LAVSB	Multiply a vector by a matrix that has been factored by _SBTRF
_LAZRO	Initialize a rectangular matrix (usually to zero)
_SYM22	Compute eigenvalues and eigenvectors of a complex symmetric 2 x 2 matrix

SCSUM1	Sum absolute values of a complex vector (used by CLACON)
ICMAX1	Find the index of element whose real part has max. abs. value (used by CLACON)
DZSUM1	Sum absolute values of a complex*16 vector (used by ZLACON)
IZMAX1	Find the index of element whose real part has max. abs. value (used by ZLACON)

Appendix C. BLAS Quick Reference

.

.

Appendix D. Matrix Types and Tests Performed for Each LAPACK Path

_GE routines:

	Matrix types		Real test ratios	Complex test ratios
1	Diagonal	1	$\frac{\| \|LU-A\| \|}{}$	$\frac{\| \|LU-A\| \|}{}$
2	Upper triangular	2	$\frac{(\sqrt{n} \cdot \| \|A\| \| \varepsilon) \cdot \| \|I-AA^{-1}\| \|}{}$	$\frac{(\sqrt{n} \cdot \| \|A\| \| \varepsilon) \cdot \| \|I-AA^{-1}\| \|}{}$
3	Lower triangular	3	$\frac{(\sqrt{n} \cdot \| \|A\| \| \cdot \| \|A^{-1}\| \| \varepsilon)}{\| \|b-Ax\| \|}$	$\frac{(\sqrt{n} \cdot \| \|A\| \| \cdot \| \|A^{-1}\| \| \varepsilon)}{\| \|b-Ax\| \|}$
4	$\kappa = 2$	4	$\frac{(\| \|A\| \| \cdot \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$	$\frac{(\| \|A\| \| \cdot \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$
5	$\kappa = \sqrt{0.1/\varepsilon}$	5	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{\| \|b-A^T x\| \|}$	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{\| \|b-A^T x\| \|}$
6	$\kappa = 0.1/\varepsilon$	6	$\frac{(\| \|A^T\| \| \cdot \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$	$\frac{(\| \|A^T\| \| \cdot \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$
7	Scaled near underflow	7	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{RCOND * \kappa}$	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{\| \|b-A^H x\| \|}$
8	Scaled near overflow	8		$\frac{(\| \|A^H\| \| \cdot \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$
		9		$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{RCOND * \kappa}$

where the system to be solved is $Ax = b$ and

x_* is the exact solution, $A^{-1}b$
x is the computed solution
n is the order of the matrix A
ε is the machine epsilon
κ is the condition number of A
$RCOND$ is the computed estimate of $1/\kappa$

_GB routines:

	Matrix types		Real test ratios	Complex test ratios
1	Diagonal	1	$\frac{\| \|LU-A\| \|}{}$	$\frac{\| \|LU-A\| \|}{}$
2	Upper triangular	2	$\frac{(\sqrt{n} \| \|A\| \| \varepsilon)}{\| \|b-Ax\| \|}$	$\frac{(\sqrt{n} \| \|A\| \| \varepsilon)}{\| \|b-Ax\| \|}$
3	Lower triangular	3	$\frac{(\| \|A\| \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$	$\frac{(\| \|A\| \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$
4	Band: $KL < M/2$, $KU < n/2$	4	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{\| \|b-A^T x\| \|}$	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{\| \|b-A^T x\| \|}$
5	Band: $KL < M/2$, $KU > n/2$	5	$\frac{(\| \|A^T\| \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$	$\frac{(\| \|A^T\| \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$
6	Band: $KL > M/2$, $KU < n/2$	6	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{RCOND * \kappa}$	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{\| \|b-A^H x\| \|}$
7	Band: $KL > M/2$, $KU > n/2$	7		$\frac{(\| \|A^H\| \| \|x\| \| \varepsilon)}{\| \|x-x_*\| \|}$
8	$\kappa = 2$	8		$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{RCOND * \kappa}$
9	$\kappa = \sqrt{0.1/\varepsilon}$			
10	$\kappa = 0.1/\varepsilon$			
11	Scaled near underflow			
12	Scaled near overflow			

_PO, _PP, _SY, _SP, _HE, and _HP routines: (for Hermitian matrices, replace U^T by U^H)

	Matrix types		Test ratios
1	Diagonal	1	$\frac{\| \|U^T U - A\| \|}{}$
2	$\kappa = 2$	2	$\frac{(\sqrt{n} \| \|A\| \| \varepsilon)^{-1} \| \|I - AA^{-1}\| \|}{}$
3	$\kappa = \sqrt{0.1/\varepsilon}$	3	$\frac{(\sqrt{n} \| \|A\| \| \|A^{-1}\| \| \varepsilon)}{\| \|b - Ax\| \|}$
4	$\kappa = 0.1/\varepsilon$	4	$\frac{(\| \|A\| \| \|x\| \| \varepsilon)}{\| \|x - x_*\| \|}$
5	Scaled near underflow	5	$\frac{(\| \|x_*\| \| \kappa\varepsilon)}{}$ *RCOND* * κ
6	Scaled near overflow	6	6-10: like 1-5 but the lower triangle of A is stored

_PB, _SB, and _HB routines: (for Hermitian matrices, replace U^T by U^H)

	Matrix types		Test ratios
1	Diagonal	1	$\frac{\| \|U^T U - A\| \|}{}$
2	$\kappa = 2$	2	$\frac{(\sqrt{n} \| \|A\| \| \varepsilon)}{\| \|b - Ax\| \|}$
3	$\kappa = \sqrt{0.1/\varepsilon}$	3	$\frac{(\| \|A\| \| \|x\| \| \varepsilon)}{\| \|x - x_*\| \|}$
4	$\kappa = 0.1/\varepsilon$	4	$(\| \|x_*\| \| \kappa\varepsilon)$ $RCOND * \kappa$
5	Scaled near underflow	5	5- 8: like 1-4 but the lower triangle of A is stored
6	Scaled near overflow		

Appendix E. Example of SECOND Timing Function for Unix Systems

This appendix presents an example of the routine SECOND for Unix-based machines. It calls the C routine *mclock* which in turn calls the system routine *times*.

```
      real function second(t)
c
c     this routine will gather the user time for a process.
c     it has a resolution of 1/60 of a second
c     and uses the unix c program times.
c     see the unix manual for details.
c     reports time in seconds.
c
      itime = mclock(i)
      second = float(itime)/60.
c
c     this statement is here to bump the time by a bit
c     in case the interval was too small.
c
      second = second + second*1.0e-6
      return
      end
```

```
long mclock_()
 {
 long  buf[4];
 times(buf);
 return(buf[0]);
 }
```

References

[1] 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.

[2] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, Argonne National Laboratory, ANL-MCS-P88-1, August 1988.

[3] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs*, Argonne National Laboratory, ANL-MCS-TM-119, June 1988.

[4] 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.

[5] 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.

[6] 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.

References