LAPACK Working Note 41
Installation Guide for LAPACK[1]

Susan Blackford[2], and Jack Dongarra
Department of Computer Science
University of Tennessee
Knoxville, Tennessee 37996-1301

REVISED: VERSION 3.0, June 30, 1999

Abstract

This working note describes how to install, test, and time version 3.0 of LAPACK, a linear algebra package for high-performance computers. Separate instructions are provided for the Unix and non-Unix versions of the test package. Further details are also given on the design of the test and timing programs.

[1]This work was supported by NSF Grant No. ASC-8715728.
[2]formerly Susan Ostrouchov

Contents

1 Introduction

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

This working note describes how to install, test, and time this release of LAPACK.

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 untarring the file, 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 installations of LAPACK (the Windows 98/NT PC installation is described in Appendix E, and the VAX installation is described in Appendix F). At this time, the VAX installation of LAPACK can only be obtained after first untarring the Unix tar file and then following the instructions in Appendix F.

Section 3 describes how the files are organized in the file, and Section 4 gives a general overview of the parts of the test package. Step-by-step instructions appear in Section 5 for the Unix version and in the appendices for Windows 98/NT and VAX-type installations.

For users desiring additional information, Sections 7 and 8 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.

It is strongly advised that the user read Appendix D before proceeding with the installation process.

Appendices E and F contain the instructions to install LAPACK on a non-Unix system.

2 Revisions Since the First Public Release

Since its first public release in February, 1992, LAPACK has had several updates, which have encompassed the introduction of new routines as well as extending the functionality of existing routines. The first update, June 30, 1992, was version 1.0a; the second update, October 31, 1992, was version 1.0b; the third update, March 31, 1993, was version 1.1; version 2.0 on September 30, 1994, coincided with the release of the Second Edition of the LAPACK Users' Guide; and finally, version 3.0 was released on June 30, 1999 in conjunction wih the publication of the Third Edition of the LAPACK Users' Guide. All LAPACK routines reflect the current version number with the date on the routine indicating when it was last modified. For more information on revisions in the latest release, please refer to the file

```
http://www.netlib.org/lapack/revisions.info
```

in the lapack directory on netlib. The `tar` file `lapack.tgz` that is available on netlib is always the most up-to-date.

On-line manpages (troff files) for LAPACK driver and computational routines, as well as most of the BLAS routines, are available via the `lapack` index on netlib.

3 File Format

The software for LAPACK is distributed in the form of a gzipped tar file (via anonymous ftp or the World Wide Web), 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. Users who wish to have a Windows 98/NT installation should go to Appendix E, and users who would like a VAX-type installation should refer to Appendix F. The overview in section 4 applies to both the Unix and non-Unix versions.

The package may be accessed via the World Wide Web through the URL address:

```
http://www.netlib.org/lapack/lapack.tgz
```

Or, you can retrieve the file via anonymous ftp at netlib:

```
ftp ftp.netlib.org
login:  anonymous
password:  <your email address>
cd lapack
binary
get lapack.tgz
quit
```

The software in the `tar` file is organized in a number of essential directories as shown in Figure 1. Please note that this figure does not reflect everything that is contained in the `LAPACK` directory. Input and instructional files are also located at various levels. Libraries are created in the LAPACK directory and executable files are created in one of the directories BLAS, TESTING, or TIMING. Input files for the test and timing programs are also found in these three directories so that testing may be carried out in the directories LAPACK/BLAS, LAPACK/TESTING, and LAPACK/TIMING. A top-level makefile in the LAPACK directory is provided to perform the entire installation procedure.

4 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

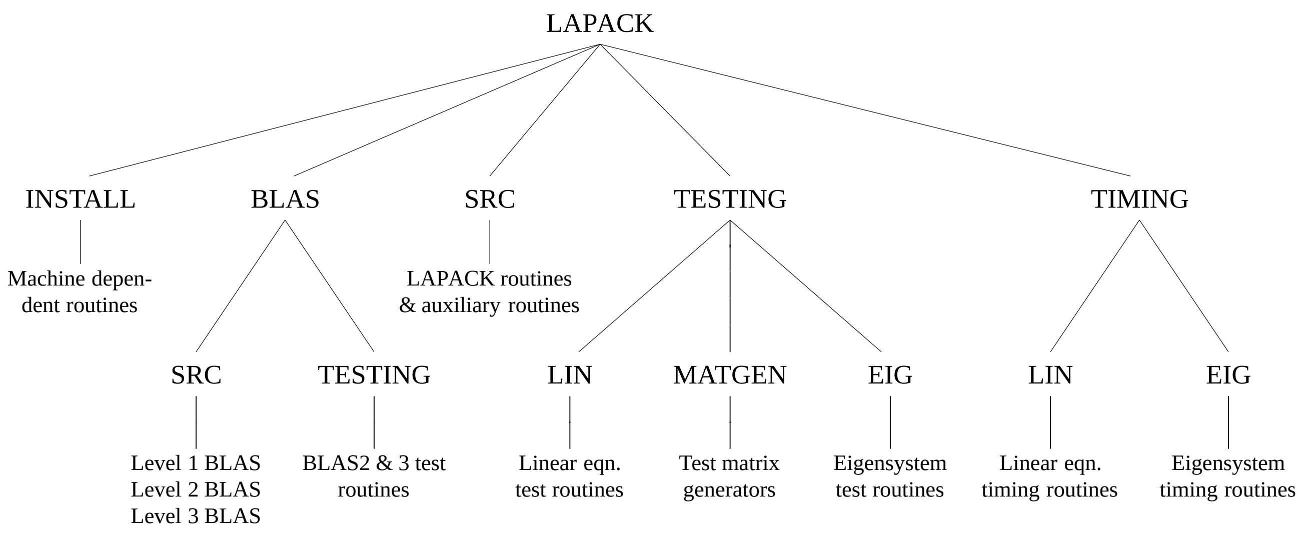
LAPACK
INSTALL
BLAS
SRC
TESTING
TIMING
Machine dependent routines
LAPACK routines & auxiliary routines
SRC
TESTING
LIN
MATGEN
EIG
LIN
EIG
Level 1 BLAS
Level 2 BLAS
Level 3 BLAS
BLAS2 & 3 test routines
Linear eqn. test routines
Test matrix generators
Eigensystem test routines
Linear eqn. timing routines
Eigensystem timing routines

Figure 1: Unix organization of LAPACK

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).

4.1 LAPACK Routines

There are three classes of LAPACK routines:

- **driver** routines solve a complete problem, such as solving a system of linear equations or computing the eigenvalues of a real symmetric matrix. Users are encouraged to use a driver routine if there is one that meets their requirements. The driver routines are listed in Appendix A and the LAPACK Users' Guide [1].
- **computational** routines, also called simply LAPACK routines, perform a distinct computational task, such as computing the LU decomposition of an m-by-n matrix or finding the eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm. The LAPACK routines are listed in Appendix A and the LAPACK Users' Guide [1].
- **auxiliary** routines are all the other subroutines called by the driver routines and computational routines. The auxiliary routines are listed in Appendix B and the LAPACK Users' Guide [1].

4.2 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. Optimized BLAS libraries are available on a variety of architectures, refer to the BLAS FAQ on netlib for further information.

```
http://www.netlib.org/blas/faq.html
```

There are also freely available BLAS generators that automatically tune a subset of the BLAS for a given architecture. E.g.,

```
http://www.netlib.org/atlas/
```

And, if all else fails, there is the Fortran 77 reference implementation of the Level 1, 2, and 3 BLAS available on netlib (also included in the LAPACK distribution tar file).

```
http://www.netlib.org/blas/blas.tgz
```

No matter which BLAS library is used, the BLAS test programs should always be run.

Users should not expect too much from the Fortran 77 reference implementation 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 [9], [7], and [5]. The BLAS Quick Reference card is available on netlib.

4.3 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, and the other tests routines for the matrix eigenvalue problem. The routines for generating test matrices are used by both test programs and are compiled into a library for use by both test programs.

4.4 LAPACK Timing Routines

This release also contains two distinct timing programs for the LAPACK routines in each data type. The linear equation timing program gathers performance data in megaflops on the factor, solve, and inverse routines for solving linear systems, the routines to generate or apply an orthogonal matrix given as a sequence of elementary transformations, and the reductions to bidiagonal, tridiagonal, or Hessenberg form for eigenvalue computations. The operation counts used in computing the megaflop rates are computed from a formula; see Appendix C. The eigenvalue 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.

5 Installing LAPACK on a Unix System

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

1. Gunzip and tar the file.
2. Edit the file `LAPACK/make.inc`.
3. Edit the file `LAPACK/Makefile` and type `make`.

5.1 Untar the File

If you received a tar file of LAPACK via the World Wide Web or anonymous ftp, enter the following two commands to untar the file:

```
gunzip -c lapack.tgz | tar xvf -
```

This will create a top-level directory called `LAPACK`, which requires approximately 34 Mbytes of disk space. The total space requirements including the object files and executables is approximately 100 Mbytes for all four data types.

5.2 Edit the file `LAPACK/make.inc`

Before the libraries can be built, or the testing and timing programs run, you must define all machine-specific parameters for the architecture to which you are installing LAPACK. All machine-specific parameters are contained in the file `LAPACK/make.inc`.

The first line of this `make.inc` file is:

SHELL = /bin/sh

and it will need to be modified to `SHELL = /sbin/sh` if you are installing LAPACK on an SGI architecture. Second, you will need to modify the `PLAT` definition, which is appended to all library names, to specify the architecture to which you are installing LAPACK. This features avoids confusion in library names when you are installing LAPACK on more than one architecture. Next, you will need to modify `FORTRAN`, `OPTS`, `DRVOPTS`, `NOOPT`, `LOADER`, `LOADOPTS`, `ARCH`, `ARCHFLAGS`, and `RANLIB` to specify the compiler, compiler options, compiler options for the testing and timing main programs, loader, loader options, archiver, archiver options, and ranlib for your machine. If your architecture does not require `ranlib` to be run after each archive command (as is the case with CRAY computers running UNICOS, Hewlett Packard computers running HP-UX, or SUN SPARCstations running Solaris), set `ranlib=echo`. And finally, you must modify the `BLASLIB` definition to specify the BLAS library to which you will be linking. If an optimized version of the BLAS is available on your machine, you are highly recommended to link to that library. Otherwise, by default, `BLASLIB` is set to the Fortran 77 version.

NOTE: Example `make.inc` include files are contained in the `LAPACK/INSTALL` directory. Please refer to Appendix D for machine-specific installation hints, and/or the `release_notes` file on `netlib`.

```
http://www.netlib.org/lapack/release_notes
```

5.3 Edit the file `LAPACK/Makefile`

This `Makefile` can be modified to perform as much of the installation process as the user desires. Ideally, this is the ONLY makefile the user must modify. However, modification of lower-level makefiles may be necessary if a specific routine needs to be compiled with a different level of optimization.

First, edit the definitions of `blaslib`, `lapacklib`, `tmglib`, `testing`, and `timing` in the file `LAPACK/Makefile` to specify the data types desired. For example, if you only wish to compile the single precision real version of the LAPACK library, you would modify the `lapacklib` definition to be:

```
lapacklib:
        ( cd SRC; $(MAKE) single )
```

Likewise, you could specify `double, complex, or complex16` to build the double precision real, single precision complex, or double precision complex libraries, respectively. By default, the presence of no arguments following the `make` command will result in the building of all four data types. The make command can be run more than once to add another data type to the library if necessary.

Next, if you will be using a locally available BLAS library, you will need to remove `blaslib` from the `lib` definition. And finally, if you do not wish to build all of the libraries individually and likewise run all of the testing and timing separately, you can modify the `all` definition to specify the amount of the installation process that you want performed. By default, the `all` definition is set to

```
all: install lib blas_testing testing timing blas_timing
```

which will perform all phases of the installation process – testing of machine-dependent routines, building the libraries, BLAS testing, LAPACK testing, LAPACK timing, and BLAS timing.

The entire installation process will then be performed by typing `make`.

Questions and/or comments can be directed to the authors as described in Section 6.8. If test failures occur, please refer to the appropriate subsection in Section 6.

If disk space is limited, I would suggest building each data type separately and/or deleting all object files after building the libraries. Likewise, all testing and timing executables can be deleted after the testing and timing process is completed. The removal of all object files and executables can be accomplished by the following:

```
cd LAPACK

make clean
```

6 Further Details of the Installation Process

Alternatively, you can choose to run each of the phases of the installation process separately. The following sections give details on how this may be achieved.

6.1 Test and Install the Machine-Dependent Routines.

There are six 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
ILAENV	INTEGER	Checks that NaN and infinity arithmetic are IEEE-754 compliant

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 in `LAPACK/INSTALL`, along with five test programs. To compile the five test programs and run the tests, go to `LAPACK` and type `make install`. The test programs are called `testlsame, testslamch, testdlamch, testsecond, testdsecnd` and `testieee`. If you do not wish to run all tests, you will need to modify the `install` definition in the `LAPACK/Makefile` to only include the tests you wish to run. Otherwise, all tests will be performed. The expected results of each test program are described below.

6.1.1 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 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 after the execution of `testlsame` is

```
 ASCII character set
 Tests completed
```

The file `lsame.f` is automatically copied to `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.

6.1.2 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 executable `testslamch` for SLAMCH on a Sun SPARCstation is

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

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

Likewise, the test executable `testdlamch` is run for DLAMCH.

The files `slamch.f` and `dlamch.f` are automatically copied to to `LAPACK/SRC/`. If both tests were successful, go to Section 6.1.3.

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

'B': Base of the machine

'E': Epsilon (relative machine precision)

'O': Overflow threshold

'P': Precision = Epsilon*Base

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

'U': Underflow threshold

Some people may be familiar with 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.

BE AWARE that the initial call to SLAMCH or DLAMCH is expensive. We suggest that installers run it once, save the results, and hard-code the constants in the version they put in their library.

6.1.3 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 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.

On some IBM architectures such as IBM RS/6000s, the timing function `ETIME` *is instead called* `ETIME_`*, and therefore the routines* `LAPACK/INSTALL/second.f` *and* `LAPACK/INSTALL/dsecnd.f` *should be modified. Usually on HPPA architectures, the compiler and loader flag* `+U77` *should be included to access the function* `ETIME`.

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. The test program executable is called `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 files `second.f` and `dsecnd.f` are automatically copied to `LAPACK/SRC/` for inclusion in the LAPACK library.

6.1.4 Testing IEEE arithmetic and ILAENV

As some new routines in LAPACK rely on IEEE-754 compliance, two settings (`ISPEC=10` and `ISPEC=11`) have been added to ILAENV (`LAPACK/SRC/ilaenv.f`) to denote IEEE-754 compliance for NaN and infinity arithmetic, respectively. By default, ILAENV assumes an IEEE machine, and does a test for IEEE-754 compliance. **NOTE: If you are installing LAPACK on a non-IEEE machine, you MUST modify ILAENV, as this test inside ILAENV will crash!**

If `ILAENV( 10, ... )` or `ILAENV( 11, ... )` is issued, then `ILAENV=1` is returned to signal IEEE-754 compliance, and `ILAENV=0` if the architecture is non-IEEE-754 compliant.

Thus, for non-IEEE machines, the user must hard-code the setting of (`ILAENV=0`) for (`ISPEC=10` and `ISPEC=11`) in the version of `LAPACK/SRC/ilaenv.f` to be put in his library.

There are also specialized testing and timing versions of ILAENV that will also need to be modified.

- Testing/timing version of `LAPACK/TESTING/LIN/ilaenv.f`
- Testing/timing version of `LAPACK/TESTING/EIG/ilaenv.f`
- Testing/timing version of `LAPACK/TIMING/LIN/ilaenv.f`
- Testing/timing version of `LAPACK/TIMING/EIG/ilaenv.f`

The test program in `LAPACK/INSTALL/tstiee.f` checks an installation architecture to see if infinity arithmetic and NaN arithmetic are IEEE-754 compliant. A warning message to the user is printed if non-compliance is detected. This same test is performed inside the function ILAENV. If `ILAENV( 10, ... )` or `ILAENV( 11, ... )` is issued, then `ILAENV=1` is returned to signal IEEE-754 compliance, and `ILAENV=0` if the architecture is non-IEEE-754 compliant.

To avoid this IEEE test being run every time you call `ILAENV( 10, ...)` or `ILAENV( 11, ... )`, we suggest that the user hard-code the setting of `ILAENV=1` or `ILAENV=0` in the version of `LAPACK/SRC/ilaenv.f` to be put in his library. As aforementioned, there are also specialized testing and timing versions of ILAENV that will also need to be modified.

6.2 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 6.3 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` and edit the definition of `blaslib` in the file `Makefile` to specify the data types desired, as in the example in Section 5.3.

   If you already have some of the BLAS, you will need to edit the file `LAPACK/BLAS/SRC/-Makefile` to comment out the lines defining the BLAS you have.

b) Type `make blaslib`. 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_PLAT.a`, where `PLAT` is the user-defined architecture suffix specified in the file `LAPACK/make.inc`.

6.3 Run the BLAS Test Programs

Test programs for the Level 1, 2, and 3 BLAS are in the directory `LAPACK/BLAS/TESTING`.

To compile and run the Level 1, 2, and 3 BLAS test programs, go to `LAPACK` and type `make blas_testing`. The executable files are called `xblat_s`, `xblat_d`, `xblat_c`, and `xblat_z`, where the _ (underscore) is replaced by 1, 2, or 3, depending upon the level of BLAS that it is testing. All executable and output files are created in `LAPACK/BLAS/`. For the Level 1 BLAS tests, the output file names are `sblat1.out`, `dblat1.out`, `cblat1.out`, and `zblat1.out`. For the Level 2 and 3 BLAS, 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 Level 2 REAL version, and `SBLAT3.SUMM` for the Level 3 REAL version, 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 [8] and [6].

6.4 Create the LAPACK Library

a) Go to the directory `LAPACK` and edit the definition of `lapacklib` in the file `Makefile` to specify the data types desired, as in the example in Section 5.3.

b) Type `make lapacklib`. 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_PLAT.a`, where `PLAT` is the user-defined architecture suffix specified in the file `LAPACK/make.inc`.

6.5 Create the Test Matrix Generator Library

a) Go to the directory `LAPACK` and edit the definition of `tmglib` in the file `Makefile` to specify the data types desired, as in the example in Section 5.3.

b) Type `make tmglib`. 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_PLAT.a`, where `PLAT` is the user-defined architecture suffix specified in the file `LAPACK/make.inc`.

6.6 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. In each data type, there is one input file for testing the linear equation routines and seventeen input files for testing the eigenvalue routines. The input files reside in `LAPACK/TESTING`. For more information on the test programs and how to modify the input files, see Section 7.

If you do not wish to run each of the tests individually, you can go to `LAPACK`, edit the definition `testing` in the file `Makefile` to specify the data types desired, and type `make testing`. This will compile and run the tests as described in sections 6.6.1 and 6.6.2.

6.6.1 Testing the Linear Equations Routines

a) Go to `LAPACK/TESTING/LIN` and type `make` followed by the data types desired. The executable files are called `xlintsts, xlintstc, xlintstd`, or `xlintstz` and are created in `LAPACK/TESTING`.

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

```
xlintsts < stest.in > stest.out
```

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

If you encountered failures in this phase of the testing process, please refer to Section 6.8.

6.6.2 Testing the Eigensystem Routines

a) Go to `LAPACK/TESTING/EIG` and type `make` followed by the data types desired. The executable files are called `xeigtsts, xeigtstc, xeigtstd`, and `xeigtstz` and are created in `LAPACK/TESTING`.

b) Go to `LAPACK/TESTING` and run the tests for each data type. The tests for the eigensystem routines use eighteen separate input files for testing the nonsymmetric eigenvalue problem, the symmetric eigenvalue problem, the banded symmetric eigenvalue problem, the generalized symmetric eigenvalue problem, the generalized nonsymmetric eigenvalue problem, the singular value decomposition, the banded singular value decomposition, the generalized singular value decomposition, the generalized QR and RQ factorizations, the generalized linear regression model, and the constrained linear least squares problem. The tests for the REAL version are as follows:

```
xeigtsts < nep.in > snep.out
xeigtsts < sep.in > ssep.out
xeigtsts < svd.in > ssvd.out
xeigtsts < sec.in > sec.out
xeigtsts < sed.in > sed.out
xeigtsts < sgg.in > sgg.out
xeigtsts < sgd.in > sgd.out
xeigtsts < ssg.in > ssg.out
xeigtsts < ssb.in > ssb.out
xeigtsts < sbb.in > sbb.out
xeigtsts < sbal.in > sbal.out
xeigtsts < sbak.in > sbak.out
xeigtsts < sgbal.in > sgbal.out
xeigtsts < sgbak.in > sgbak.out
xeigtsts < glm.in > sglm.out
xeigtsts < gqr.in > sgqr.out
xeigtsts < gsv.in > sgsv.out
xeigtsts < lse.in > slse.out
```

The tests using `xeigtstc`, `xeigtstd`, and `xeigtstz` also use the input files `nep.in`, `sep.in`, `svd.in`, `glm.in`, `gqr.in`, `gsv.in`, and `lse.in`, but the leading 's' in the other input file names must be changed to 'c', 'd', or 'z'.

If you encountered failures in this phase of the testing process, please refer to Section 6.8.

6.7 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 and a large set. The small data sets are appropriate for a standard workstation or other non-vector machine. The large data sets are appropriate for supercomputers, vector computers, and high-performance workstations. We are mainly interested in results from the large data sets, and it is not necessary to run both the large and small sets. The values of N in the large data sets are about five times larger than those in the small data set, and the large data sets use additional values for parameters such as the block size NB and the leading array dimension LDA. Small data sets are indicated by lower case names, such as `stime.in`, and large data sets are indicated by upper case names, such as `STIME.in`. Except as noted, the leading 's' (or 'S') in the input file name must be replaced by 'd', 'c', or 'z' ('D', 'C', or 'Z') for the other data types.

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

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

For more information on the timing programs and how to modify the input files, see Section 8.

If you do not wish to run each of the timings individually, you can go to `LAPACK`, edit the definition `timing` in the file `Makefile` to specify the data types desired, and type `make timing`. This will compile and run the timings for the linear equation routines and the eigensystem routines (see Sections 6.7.1 and 6.7.3).

If you encounter failures in any phase of the timing process, please feel free to contact the authors as directed in Section 6.8.

Please note that the BLAS timing runs will still need to be run as instructed in 6.7.2.

6.7.1 Timing the Linear Equations Routines

The linear equation timing program is found in `LAPACK/TIMING/LIN` and the input files are in `LAPACK/TIMING`. Three input files are provided in each data type for timing the linear equation routines, one for square matrices, one for band matrices, and one for rectangular matrices. The small data sets for the REAL version are `stime.in`, `sband.in`, and `stime2.in`, respectively, and the large data sets are `STIME.in`, `SBAND.in`, and `STIME2.in`.

The timing program for the least squares routines uses special instrumented versions of the LAPACK routines to time individual sections of the code. 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/LIN/LINSRC` and type `make` followed by the data types desired, as in the examples of Section 5.3.

   The library of instrumented code is created in `LAPACK/TIMING/LIN/linsrc_PLAT.a`, where `PLAT` is the user-defined architecture suffix specified in the file `LAPACK/make.inc`.

b) To make the linear equation timing programs, go to `LAPACK/TIMING/LIN` and type `make` followed by the data types desired, as in the examples in Section 5.3. The executable files are called `xlintims`, `xlintimc`, `xlintimd`, and `xlintimz` 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 6.7.2).

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

```
xlintims < stime.in > stime.out
xlintims < sband.in > sband.out
xlintims < stime2.in > stime2.out
```

or the commands for the large data sets are

```
xlintims < STIME.in > STIME.out
xlintims < SBAND.in > SBAND.out
xlintims < STIME2.in > STIME2.out
```

Similar commands should be used for the other data types.

6.7.2 Timing the BLAS

The linear equation timing program is also used to time the BLAS. Three input files are provided in each data type for timing the Level 2 and 3 BLAS. These input files time the BLAS using the matrix shapes encountered in the LAPACK routines, and we will use the results to analyze the performance of the LAPACK routines. For the REAL version, the small data files are `sblasa.in`, `sblasb.in`, and `sblasc.in` and the large data files are `SBLASA.in`, `SBLASB.in`, and `SBLASC.in`. 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 `sblasa.in`, M and N are large but K is small, while in `sblasb.in` the small parameter is M, and in `sblasc.in` 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 6.7.1, 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 for the small data sets are

```
xlintims < sblasa.in > sblasa.out
xlintims < sblasb.in > sblasb.out
xlintims < sblasc.in > sblasc.out
```

or the commands for the large data sets are

```
xlintims < SBLASA.in > SBLASA.out
xlintims < SBLASB.in > SBLASB.out
xlintims < SBLASC.in > SBLASC.out
```

Similar commands should be used for the other data types.

6.7.3 Timing the Eigensystem Routines

The eigensystem timing program is found in `LAPACK/TIMING/EIG` and the input files are in `LAPACK/TIMING`. Four input files are provided in each data type for timing the eigensystem routines, one for the generalized nonsymmetric eigenvalue problem, one for the nonsymmetric eigenvalue problem, one for the symmetric and generalized symmetric eigenvalue problem, and one for the singular value decomposition. For the REAL version, the small data sets are called `sgeptim.in`, `sneptim.in`, `sseptim.in`, and `ssvdtim.in`, respectively. and the large data sets are called `SGEPTIM.in`, `SNEPTIM.in`, `SSEPTIM.in`, and `SSVDTIM.in`. Each of the four input files reads a different set of parameters, and the format of the input is indicated by a 3-character code on the first line.

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

a) To make a library of the instrumented LAPACK routines, first go to `LAPACK/TIMING/EIG/EIGSRC` and type `make` followed by the data types desired, as in the examples of Section 5.3. The library of instrumented code is created in `LAPACK/TIMING/EIG/eigsrc_PLAT.a`, where `PLAT` is the user-defined architecture suffix specified in the file `LAPACK/make.inc`.

b) To make the eigensystem timing programs, go to `LAPACK/TIMING/EIG` and type `make` followed by the data types desired, as in the examples of Section 5.3. The executable files are called `xeigtims`, `xeigtimc`, `xeigtimd`, and `xeigtimz` 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 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 8 for further details.

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

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

or the commands for the large data sets are

```
xeigtims < SGEPTIM.in > SGEPTIM.out
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.

6.8 Send the Results to Tennessee

Congratulations! You have now finished installing, testing, and timing LAPACK. If you encountered failures in any phase of the testing or timing process, please consult our release_notes file on netlib.

```
http://www.netlib.org/lapack/release_notes
```

This file contains machine-dependent installation clues which hopefully will alleviate your difficulties or at least let you know that other users have had similar difficulties on that machine. If there is not an entry for your machine or the suggestions do not fix your problem, please feel free to contact the authors at

lapack@cs.utk.edu.

Tell us the type of machine on which the tests were run, the version of the operating system, the compiler and compiler options that were used, and details of the BLAS library or libraries that you used. You should also include a copy of the output file in which the failure occurs.

We would like to keep our release_notes file as up-to-date as possible. Therefore, if you do not see an entry for your machine, please contact us with your testing results.

Comments and suggestions are always welcome.

We encourage you to make the LAPACK library available to your users and provide us with feedback from their experiences.

7 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 17 different sets of input, although four of these may be concatenated into one data set, for a total of 14 input files. The following sections describe the different input formats and testing styles.

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. 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.

7.1 The Linear Equation Test Program

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
- NRHS, the number of right hand sides
- NB, the blocksize for the blocked routines
- NX, the crossover point, the point in a block algorithm at which we switch to an unblocked algorithm

For symmetric or Hermitian matrices, 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 test programs. For the linear equation test program, these are:

Parameter	Description	Value
NMAX	Maximum value of M or N for rectangular matrices	132
MAXIN	Maximum number of values of M, N, NB, or NX	12
MAXRHS	Maximum value of NRHS	10

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 band matrices
{S, C, D, Z} GT	General tridiagonal
{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 band
{S, C, D, Z}	PT	Positive definite tridiagonal
{C, Z}	HE	Hermitian indefinite matrices
{C, Z}	HP	Hermitian indefinite packed
{S, C, D, Z}	SY	Symmetric indefinite matrices
{S, C, D, Z}	SP	Symmetric indefinite packed
{S, C, D, Z}	TR	Triangular matrices
{S, C, D, Z}	TP	Triangular packed
{S, C, D, Z}	TB	Triangular band
{S, C, D, Z}	QR	QR decomposition
{S, C, D, Z}	RQ	RQ decomposition
{S, C, D, Z}	LQ	LQ decomposition
{S, C, D, Z}	QL	QL decomposition
{S, C, D, Z}	QP	QR decomposition with column pivoting
{S, C, D, Z}	TZ	Trapezoidal matrix (RQ factorization)
{S, C, D, Z}	LS	Least Squares driver routines
{S, C, D, Z}	EQ	Equilibration routines

The xQR, xRQ, xLQ, and xQL test paths also test the routines for generating or multiplying by an orthogonal or unitary matrix expressed as a sequence of elementary Householder transformations.

7.1.1 Tests for General and Symmetric Matrices

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 the test paths for general and symmetric matrices are shown in Table 1. Both the computational routines and the driver routines are tested with the same set of matrix types. In this context, ε is the machine epsilon and κ is the condition number of the matrix A. Matrix types with one or more columns set to zero (or rows and columns, if the matrix is symmetric) are used to test the error return codes. For band matrices, all combinations of the values 0, 1, $n-1$, $(3n-1)/4$, and $(n-1)/4$ are used for KL and KU in the GB path, and for KD in the PB path. For the tridiagonal test paths xGT and xPT, types 1-6 use matrices of predetermined condition number, while types 7-12 use random tridiagonal matrices.

For the LAPACK test paths shown in Table 1, each test matrix is subjected to the following tests:

- Factor the matrix using xxxTRF, and compute the ratio

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

  This form is for the paths xGE, xGB, and xGT. For the paths xPO, xPP, or xPB, replace LU by LL^T or U^TU; for xPT, replace LU by LDL^T or U^TDU, where D is diagonal; and for the paths xSY, xSP, xHE, or xHP, replace LU by LDL^T or UDU^T, where D is diagonal with 1-by-1 and 2-by-2 diagonal blocks.

Test matrix type	GE	GB	GT	PO, PP	PB	PT	SY, SP, HE, HP
Diagonal	1		1	1		1	1
Upper triangular	2						
Lower triangular	3						
Random, $\kappa = 2$	4	1	2	2	1	2	2
Random, $\kappa = \sqrt{0.1/\varepsilon}$	8	5	3	6	5	3	7
Random, $\kappa = 0.1/\varepsilon$	9	6	4	7	6	4	8
First column zero	5	2	8	3	2	8	3
Last column zero	6	3	9	4	3	9	4
Middle column zero				5	4	10	5
Last $n/2$ columns zero	7	4	10				6
Scaled near underflow	10	7	5, 11	8	7	5, 11	9
Scaled near overflow	11	8	6, 12	9	8	6, 12	10
Random, unspecified κ			7			7	
Block diagonal							$11^{\dagger}$

†– complex symmetric test paths only

Table 1: Test matrices for general and symmetric linear systems

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

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

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

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

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

  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

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

- Compute the reciprocal condition number RCOND using xxxCON, and compare to the value RCONDC which was computed as 1/(ANORM * AINVNM) where AINVNM is the explicitly computed norm of A^{-1}. The larger of the ratios

  RCOND/RCONDC and RCONDC/RCOND

  is returned. Since the same value of ANORM is used in both cases, this test measures the accuracy of the estimate computed for A^{-1}.

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 general and symmetric test paths are listed in Table 2. Here we use $||LU - A||$ to describe the difference in the recomputed matrix, even though it is actually $||LL^T - A||$ or some other form for other paths.

	GE, PO, PP, SY, SP		GB, GT, PB, PT	
Test ratio	routines	drivers	routines	drivers
$\lVert LU - A\rVert/(n\lVert A\rVert\varepsilon)$	1	1	1	1
$\lVert I - AA^{-1}\rVert/(n\lVert A\rVert\,\lVert A^{-1}\rVert\varepsilon)$	2			
$\lVert b - Ax\rVert/(\lVert A\rVert\,\lVert x\rVert\varepsilon)$	3	2	2	2
$\lVert x - x^*\rVert/(\lVert x^*\rVert\kappa\varepsilon)$	4		3	
$\lVert x - x^*\rVert/(\lVert x^*\rVert\kappa\varepsilon)$, refined	5	3	4	3
(backward error)/ε	6	4	5	4
$\lVert x - x^*\rVert/(\lVert x^*\rVert$(error bound))	7	5	6	5
RCOND $*\ \kappa$	8	6	7	6

Table 2: Tests performed for general and symmetric linear systems

7.1.2 Tests for Triangular Matrices

The triangular test paths, xTR, xTP, and xTB, include a number of pathological test matrices for testing the auxiliary routines xLATRS, xLATPS, and xLATBS, which are robust triangular solves used in condition estimation. The triangular test matrices are summarized in Table 3. To generate unit triangular matrices of predetermined condition number, we choose a special unit triangular matrix and use plane rotations to fill in the zeros without destroying the ones on the diagonal. For the xTB path, all combinations of the values 0, 1, $n-1$, $(3n-1)/4$, and $(n-1)/4$ are used for the number of offdiagonals KD, so the diagonal type is not necessary.

Types 11-18 for the xTR and xTP paths, and types 10-17 for xTB, are used only to test the scaling options in xLATRS, xLATPS, and xLATBS. These subroutines solve a scaled triangular system $Ax = sb$ or $A^Tx = sb$, where s is allowed to underflow to 0 in order to prevent overflow in x. A growth factor is computed using the norms of the columns of A, and if the solution can not overflow, the Level 2 BLAS routine is called. Types 11 and 18 test the scaling of b when b is initially large, types 12-13 and 15-16 test scaling when the diagonal of A is small or zero, and type 17 tests the scaling if overflow occurs when adding multiples of the columns to the right hand side. In type 14, no scaling is done, but the growth factor is too large to call the equivalent BLAS routine.

The tests performed for the triangular routines are similar to those for the general and symmetric routines, including tests of the inverse, solve, iterative refinement, and condition estimation routines. One additional test ratio is computed for the robust triangular solves:

$$||sb - Ax||/(||A||\,||x||\,\varepsilon)$$

Table 4 shows the test ratios computed for the triangular test paths.

Test matrix type	TR, TP	TB
Diagonal	1	
Random, $\kappa = 2$	2	1
Random, $\kappa = \sqrt{0.1/\varepsilon}$	3	2
Random, $\kappa = 0.1/\varepsilon$	4	3
Scaled near underflow	5	4
Scaled near overflow	6	5
Identity	7	6
Unit triangular, $\kappa = 2$	8	7
Unit triangular, $\kappa = \sqrt{0.1/\varepsilon}$	9	8
Unit triangular, $\kappa = 0.1/\varepsilon$	10	9
Matrix elements are O(1), large right hand side	11	10
First diagonal causes overflow, offdiagonal column norms < 1	12	11
First diagonal causes overflow, offdiagonal column norms > 1	13	12
Growth factor underflows, solution does not overflow	14	13
Small diagonal causes gradual overflow	15	14
One zero diagonal element	16	15
Large offdiagonals cause overflow when adding a column	17	16
Unit triangular with large right hand side	18	17

Table 3: Test matrices for triangular linear systems

Test ratio	TR, TP	TB
$\|\|I - AA^{-1}\|\|/(n\|\|A\|\|\,\|\|A^{-1}\|\|\varepsilon)$	1	
$\|\|b - Ax\|\|/(\|\|A\|\|\,\|\|x\|\|\varepsilon)$	2	1
$\|\|x - x^*\|\|/(\|\|x^*\|\|\kappa\varepsilon)$	3	2
$\|\|x - x^*\|\|/(\|\|x^*\|\|\kappa\varepsilon)$, refined	4	3
(backward error)$/\varepsilon$	5	4
$\|\|x - x^*\|\|/(\|\|x^*\|\|$(error bound))	6	5
RCOND $* \kappa$	7	6
$\|\|sb - Ax\|\|/\|\|A\|\|\,\|\|x\|\|\,\varepsilon)$	8	7

Table 4: Tests performed for triangular linear systems

7.1.3 Tests for the Orthogonal Factorization Routines

The orthogonal factorization routines are contained in the test paths xQR, xRQ, xLQ, xQL, xQP, and xTZ. The first four of these test the QR, RQ, LQ, and QL factorizations without pivoting. The subroutines to generate or multiply by the orthogonal matrix from the factorization are also tested in these paths. There is not a separate test path for the orthogonal transformation routines, since the important thing when generating an orthogonal matrix is not whether or not it is, in fact, orthogonal, but whether or not it is the orthogonal matrix we wanted. The xQP test path is used for QR with pivoting, and xTZ tests the reduction of a trapezoidal matrix by an RQ factorization.

The test paths xQR, xRQ, xLQ, and xQL all use the same set of test matrices and compute similar test ratios, so we will only describe the xQR path. Also, we will refer to the subroutines by their single precision real names, SGEQRF, SGEQRS, SORGQR, and SORMQR. In the complex case, the orthogonal matrices are unitary, so the names beginning with SOR- are changed to CUN-. Each of the orthogonal factorizations can operate on m-by-n matrices, where $m > n$, $m = n$, or $m < n$.

Eight test matrices are used for SQR and the other orthogonal factorization test paths. All are generated with a predetermined condition number (by default, $\kappa = 2.$).

1. Diagonal	5. Random, $\kappa = \sqrt{0.1/\varepsilon}$
2. Upper triangular	6. Random, $\kappa = 0.1/\varepsilon$
3. Lower triangular	7. Scaled near underflow
4. Random, $\kappa = 2.$	8. Scaled near overflow

The tests for the SQR path are as follows:

- Compute the QR factorization using SGEQRF, generate the orthogonal matrix Q from the Householder vectors using SORGQR, 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 SORMQR with UPLO = 'L', and compare the result to the product of C and Q (or Q^H) using the explicit matrix Q generated by SORGQR. The different options for SORMQR are tested by computing the 4 ratios

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

  where the first product is computed using SORMQR 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. $||b - Ax||/(||A||\,||x||\varepsilon)$

In the SQP test path, we test the QR factorization with column pivoting (SGEQPF or SGEQP3), which decomposes a matrix A into a product of a permutation matrix P, an orthogonal matrix Q, and an upper triangular matrix R such that $AP = QR$. We generate three types of matrices A with singular values s as follows:

- all singular values are zero,
- all singular values are 1, except for $\sigma_{\min(m,n)} = 1/\epsilon$, and
- the singular values are $1, r, r^2, \ldots, r^{\min(m,n)-1} = 1/\epsilon$.

The following tests are performed:

- Compute the QR factorization with column pivoting using SGEQPF (or SGEQP3), compute the singular values $\tilde{s}$ of R using SGEBD2 and SBDSQR, and compute the ratio
$$||\tilde{s} - s||/(m||s||\epsilon)$$
- Generate the orthogonal matrix Q from the Householder vectors using SORMQR, and compute the ratio
$$||AP - QR||/(m||A||\epsilon)$$
- Test the orthogonality of the computed matrix Q by computing the ratio
$$||I - Q^H Q||/(m\epsilon)$$

In the STZ path, we test the trapezoidal reduction (STZRQF or STZRZF), which decomposes an m-by-n (m < n) upper trapezoidal matrix R (i.e. $r_{ij} = 0$ if $i > j$) into a product of a strictly upper triangular matrix T (i.e. $t_{ij} = 0$ if $i > j$ or $j > m$) and an orthogonal matrix Z such that $R = TZ$. We generate matrices with the following three singular value distributions s:

- all singular values are zero,
- all singular values are 1, except for $\sigma_{\min(m,n)} = 1/\epsilon$, and
- the singular values are $1, r, r^2, \ldots, r^{\min(m,n)-1} = 1/\epsilon$.

To obtain an upper trapezoidal matrix with the specified singular value distribution, we generate a dense matrix using SLATMS and reduce it to upper triangular form using SGEQR2. The following tests are performed:

- Compute the trapezoidal reduction STZRQF (or STZRZF), compute the singular values $\tilde{s}$ of T using SGEBD2 and SBDSQR, and compute the ratio
$$||\tilde{s} - s||/(m||s||\epsilon)$$

- Apply the orthogonal matrix Z to T from the right using SLATZM, and compute the ratio
$$||R - TZ||/(m||R||\epsilon)$$
- Form $Z^T Z$ using SLATZM, and compute the ratio
$$||I - Z^T Z||/(m\epsilon)$$

7.1.4 Tests for the Least Squares Driver Routines

In the SLS path, driver routines are tested for computing solutions to over- and underdetermined, possibly rank-deficient systems of linear equations $AX = B$ (A is m-by-n). For each test matrix type, we generate three matrices: One which is scaled near underflow, a matrix with moderate norm, and one which is scaled near overflow.

The SGELS driver computes the least-squares solutions (when $m \geq n$) and the minimum-norm solution (when $m < n$) for an m-by-n matrix A of full rank. To test SGELS, we generate a diagonally dominant matrix A, and for $C = A$ and $C = A^H$, we

- generate a consistent right-hand side B such that X is in the range space of C, compute a matrix X using SGELS, and compute the ratio
$$||AX - B||/(\max(m, n)||A||||X||\epsilon)$$
- If C has more rows than columns (i.e. we are solving a least-squares problem), form $R = AX - B$, and check whether R is orthogonal to the column space of A by computing
$$||R^H C||/(\max(m, n, nrhs)||A||||B||\epsilon)$$
- If C has more columns than rows (i.e. we are solving an overdetermined system), check whether the solution X is in the row space of C by scaling both X and C to have norm one, and forming the QR factorization of $D = [A, X]$ if $C = A^H$, and the LQ factorization of $D = [A^H, X]^H$ if $C = A$. Letting $E = D(n : n+nrhs, n+1, n+nrhs)$ in the first case, and $E = D(m+1 : m+nrhs, m+1 : m+nrhs)$ in the latter, we compute
$$\max|d_{ij}|/(\max(m, n, nrhs)\epsilon)$$

The SGELSX, SGELSY, SGELSS and SGELSD drivers solve a possibly rank-deficient system $AX = B$ using a complete orthogonal factorization (SGELSX or SGELSY) or singular value decomposition (SGELSS or SGELSD), respectively. We generate matrices A that have rank $r = \min(m, n)$ or rank $r = 3\min(m, n)/4$ and are scaled to be near underflow, of moderate norm, or near overflow. We also generate the null matrix (which has rank $r = 0$). Given such a matrix, we then generate a right-hand side B which is in the range space of A.

In the process of determining X, SGELSX (or SGELSY) computes a complete orthogonal factorization $AP = QTZ$, whereas SGELSS (or SGELSD) computes the singular value decomposition $A = U\text{diag}(\sigma)V^T$.

- If s are the true singular values of A, and $\tilde{s}$ are the singular values of T, we compute

$$||s - \tilde{s}||/(||s||\epsilon)$$

for SGELSX (or SGELSY), and

$$||s - \sigma||/(||s||\epsilon)$$

for SGELSS (or SGELSD).

- Compute the ratio

$$||AX - B||/(\max(m, n)||A||||X||\epsilon)$$

- If $m > r$, form $R = AX - B$, and check whether R is orthogonal to the column space of A by computing

$$||R^H A||/(\max(m, n, nrhs)||A||||B||\epsilon)$$

- If $n > r$, check if X is in the row space of A by forming the LQ factorization of $D = [A^H, X]^H$. Letting $E = D(m + 1 : m + nrhs, m + 1 : m + nrhs)$, we return

$$\max |d_{ij}|/(\max(m, n, nrhs)\epsilon)$$

7.1.5 Tests for the Equilibration Routines

The equilibration routines are xGEEQU, xGBEQU, xPOEQU, xPPEQU and xPBEQU. These routines perform diagonal scaling on various kinds of matrices to reduce their condition number prior to linear equation solving. All of them attempt to somehow equalize the norms of the rows and/or columns of the input matrix by diagonal scaling. This is tested by generating a few matrices for which the answer is known exactly, and comparing the output with the correct answer. There are no testing parameters for the user to set.

Equilibration is also an option to the driver routines for the test paths xGE, xGB, xPO, xPP, and xPB, so it is tested in context there.

7.1.6 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 and crossover point 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 eqn. routines
7                       Number of values of M
0 1 2 3 5 10 16         Values of M (row dimension)
7                       Number of values of N
0 1 2 3 5 10 16         Values of N (column dimension)
3                       Number of values of NRHS
1 2 15                  Values of NRHS (number of right hand sides)
```

```
5                       Number of values of NB
1 3 3 3 20              Values of NB (the blocksize)
1 0 5 9 1               Values of NX (crossover point)
30.0                    Threshold value of test ratio
T                       Put T to test the LAPACK routines
T                       Put T to test the driver routines
T                       Put T to test the error exits
SGE   11                List types on next line if 0 < NTYPES < 11
SGB    8                List types on next line if 0 < NTYPES <  8
SGT   12                List types on next line if 0 < NTYPES < 12
SPO    9                List types on next line if 0 < NTYPES <  9
SPP    9                List types on next line if 0 < NTYPES <  9
SPB    8                List types on next line if 0 < NTYPES <  8
SPT   12                List types on next line if 0 < NTYPES < 12
SSY   10                List types on next line if 0 < NTYPES < 10
SSP   10                List types on next line if 0 < NTYPES < 10
STR   18                List types on next line if 0 < NTYPES < 18
STP   18                List types on next line if 0 < NTYPES < 18
STB   17                List types on next line if 0 < NTYPES < 17
SQR    8                List types on next line if 0 < NTYPES <  8
SRQ    8                List types on next line if 0 < NTYPES <  8
SLQ    8                List types on next line if 0 < NTYPES <  8
SQL    8                List types on next line if 0 < NTYPES <  8
SQP    6                List types on next line if 0 < NTYPES <  6
STZ    3                List types on next line if 0 < NTYPES <  3
SLS    6                List types on next line if 0 < NTYPES <  6
SEQ
```

The first 11 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). Lines 12-14 specify if the LAPACK routines, the driver routines, or the error exits are to be tested. 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, followed by the number of test matrix types. If the number of matrix types is omitted, as in the above example for SEQ, or if a character is encountered before an integer, all the possible matrix types are tested. 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    11
```

requests all of the matrix types for path SGE, while

```
SGE    3
 4 5 6
```

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

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 ) =   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)
```

7.2 Testing the Eigenproblem Balancing and Backward Transformation Routines

The balancing routine, xGEBAL, is tested. xGEBAL balances a matrix and isolates some of its eigenvalues. The backward transformation routine to be tested is xGEBAK. xGEBAK back transforms the computed right or left eigenvectors if the original matrix was preprocessed by balance subroutine xGEBAL.

No parameters can be varied for either of the routines tested; the data files contain precomputed test problems along with their precomputed solutions. The reason for this approach is threefold. First, there is no simple residual test ratio which can test correctness of a condition estimator. Second, no comparable code in another library exists to compare solutions. Third, the condition numbers we compute can themselves be quite ill-conditioned, so that we need the precomputed solution to verify that the computed result is within acceptable bounds.

The test program `xeigtsts` reads in the data from the data files `sbal.in` and `sbak.in` respectively (for the REAL code). If there are no errors, a single message saying that all the routines pass the tests will be printed. If any routine fails its tests, an error message is printed with the name of the failed routine along with the number of failures, the number of the example with the worst failure, and the test ratio of the worst failure.

7.3 Testing the Nonsymmetric Eigenvalue Routines

The test routine for the LAPACK nonsymmetric 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
- 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.

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	40
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). Similar restrictions exist for the other input files for the eigenvalue test program.

7.3.1 The Nonsymmetric Eigenvalue Drivers

The driver routines for the nonsymmetric eigenvalue problem are

xGEEV eigenvalue/eigenvector driver,

xGEEVX expert version of xGEEV (includes condition estimation),

xGEES Schur form driver, and

xGEESX expert version of xGEES (includes condition estimation).

For these subroutines, some tests are done by generating random matrices of a dimension and type chosen by the user, and computing error bounds similar to those used for the nonsymmetric eigenvalue computational routines. Other tests use a file of precomputed matrices and condition numbers, identical to that used for testing the nonsymmetric eigenvalue/vector condition estimation routines.

The parameters that can be varied in the random matrix tests are:

- the order N of the matrix A
- the type of test matrix A
- five numerical parameters: NB (the block size), NBMIN (minimum block size), NX (minimum dimension for blocking), NS (number of shifts in xHSEQR), and NBCOL (minimum column dimension for blocking).

7.3.2 Test Matrices for the Nonsymmetric Eigenvalue Routines

Twenty-one different types of test matrices may be generated for the nonsymmetric eigenvalue routines. Table 5 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries.

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

Type	Eigenvalue Distribution				
	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 5: Test matrices for the nonsymmetric eigenvalue problem

(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

7.3.3 Test Matrices for the Nonsymmetric Eigenvalue Drivers

All four drivers are tested with up to 21 types of random matrices. These are nearly the same as the types of matrices used to test the nonsymmetric eigenvalue computational routines, and are given in Table 3. The only differences are that matrix types 7 and 17 are scaled by a number close to the underflow threshold (rather than its square root), types 8 and 18 are scaled by a number close to the overflow threshold, and types 20 and 21 have certain rows and columns zeroed out. The reason for these changes is to activate the automatic scaling features in the driver, and to test the balancing routine.

In addition, the condition estimation features of the expert drivers xGEEVX and xGEESX are tested by the same precomputed sets of test problems used to test their constituent pieces xTRSNA and xTRSEN.

7.3.4 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 5, "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.

7.3.5 Tests Performed on the Nonsymmetric Eigenvalue Drivers

The four drivers have slightly different tests applied to them.

xGEEV takes the input matrix A and computes a matrix of its right eigenvectors VR, a matrix of its left eigenvectors VL, and a (block) diagonal matrix W of eigenvalues. If W is real it may have 2 by 2 diagonal blocks corresponding to complex conjugate eigenvalues. The test ratios computed are:

$$
\begin{array}{ll}
r_1 = \frac{\|A \cdot VR - VR \cdot W\|}{n\epsilon \|A\|} & r_2 = \frac{\|A' \cdot VL - VL \cdot W\|}{n\epsilon \|A\|} \\
r_3 = \frac{|\|VR_i\| - 1|}{\epsilon} & r_4 = \frac{|\|VL_i\| - 1|}{\epsilon} \\
r_5 = (W(full) = W(partial)) & r_6 = (VR(full) = VR(partial)) \\
r_7 = (VL(full) = VL(partial)) &
\end{array}
$$

r_5, r_6 and r_7 check whether W or VR or VL is computed identically independent of whether other quantities are computed or not. r_3 and r_4 also check that the component of VR or VL of largest absolute value is real.

These test ratios are compared to the input parameter THRESH. If a ratio exceeds THRESH, a message is printed specifying the test matrix, the ratio that failed and its value, just like the tests performed on the nonsymmetric eigenvalue problem computational routines.

In addition to the above tests, xGEEVX is tested by computing the test ratios r_8 through r_{11}. r_8 tests whether the output quantities SCALE, ILO, IHI, and ABNRM are identical independent of which other output quantities are computed. r_9 tests whether the output quantity RCONDV is independent of the other outputs. r_{10} and r_{11} are only applied to the matrices in the precomputed examples:

$$
r_{10} = \max \frac{|RCONDV - RCDVIN|}{cond(RCONDV)} \quad r_{11} = \max \frac{|RCONDE - RCDEIN|}{cond(RCONDE)}
$$

RCONDV (RCONDE) is the array of output reciprocal condition numbers of eigenvectors (eigenvalues), RCDVIN (RCDEIN) is the array of precomputed reciprocal condition numbers, and *cond*(RCONDV) (*cond*(RCONDE)) is the condition number of RCONDV (RCONDE).

xGEES takes the input matrix A and computes its Schur decomposition $A = VS \cdot T \cdot VS'$ where VS is orthogonal and T is (quasi) upper triangular, optionally sorts the eigenvalues on the diagonal of T, and computes a vector of eigenvalues W. The following test ratios are computed without sorting eigenvalues in T, and compared to THRESH:

$$
\begin{array}{ll}
r_1 = (T \text{ in Schur form?}) & r_2 = \frac{\|A - VS \cdot T \cdot VS'\|}{n\epsilon \|A\|} \\
r_3 = \frac{\|I - VS \cdot VS'\|}{n\epsilon} & r_4 = (W \text{ agrees with diagonal of } T) \\
r_5 = (T(partial) = T(full)) & r_6 = (W(partial) = W(full))
\end{array}
$$

r_7 through r_{12} are the same test ratios but with sorting the eigenvalues . r_{13} indicates whether the sorting was done successfully.

In addition to the above tests, xGEESX is tested via ratios r_{14} through r_{17}. r_{14} (r_{15}) tests if RCONDE (RCONDV) is the same no matter what other quantities are computed. r_{16} and r_{17} are only applied to the matrices in the precomputed examples:

$$r_{16} = \max \frac{|RCONDE - RCDEIN|}{cond(RCONDE)} \quad r_{17} = \max \frac{|RCONDV - RCDVIN|}{cond(RCONDV)}$$

RCONDV (RCONDE) is the output reciprocal condition number of the selected invariant subspace (eigenvalue cluster), RCDVIN (RCDEIN) is the precomputed reciprocal condition number, and *cond*(RCONDV) (*cond*(RCONDE)) is the condition number of RCONDV (RCONDE).

7.3.6 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 Nonsymmetric Eigenvalue Problem routines
7                                  Number of values of N
0 1 2 3 5 10 16                    Values of N (dimension)
5                                  Number of values of NB, NS, and MAXB
1  3  3  3  20                     Values of NB (blocksize)
2  2  2  2  2                      Values of NBMIN (minimum blocksize)
1  0  5  9  1                      Values of NX (crossover point)
2  4  2  4  6                      Values of NS (no. of shifts)
20 20 6  10 10                     Values of MAXB (min. blocksize)
20.0                               Threshold value
T                                  Put T to test the error exits
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 2–11 are read using list-directed input and specify the following values:

line 2:	The number of values of N
line 3:	The values of N, the matrix dimension
line 4:	The number of values of the parameters NB, NBMIN, NX, NS, and MAXB
line 5:	The values of NB, the blocksize
line 6:	The values of NBMIN, the minimum blocksize
line 7:	The values of NX, the crossover point
line 8:	The values of NS, the number of shifts
line 9:	The values of MAXB, the minimum blocksize
line 10:	The threshold value for the test ratios
line 11:	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 12:	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.

7.3.7 Input File for Testing the Nonsymmetric Eigenvalue Drivers

There is a single input file to test all four drivers. The input data for each path (testing xGEEV, xGEES, xGEEVX and xGEESX) is preceded by a single line identifying the path (SEV, SES, SVX and SSX, respectively, when x=S, and CEV, CES, CVX and CSX, respectively, when x=C). We discuss each set of input data in turn.

An annotated example of input data for testing SGEEV is shown below (testing CGEEV is identical except CEV replaces SEV):

```
SEV                Data file for the Real Nonsymmetric Eigenvalue Driver
6                  Number of matrix dimensions
0 1 2 3 5 10       Matrix dimensions
3 3 1 4 1          Parameters NB, NBMIN, NX, NS, NBCOL
20.0               Threshold for test ratios
T                  Put T to test the error exits
2                  Read another line with random number generator seed
2518 3899 995 397  Seed for random number generator
SEV 21             Use all matrix types
```

The first line must contain the characters SEV in columns 1-3. The remaining lines are read using list-directed input and specify the following values:

line 2:	The number of values of matrix dimension N
line 3:	The values of N, the matrix dimension
line 4:	The values of the parameters NB, NBMIN, NX, NS and NBCOL
line 5:	The threshold value THRESH for the test ratios
line 6:	T to test the error exits
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
line 9:	Contains 'SEV' in columns 1-3, followed by the number of matrix types (an integer from 0 to 21)
line 9:	(and following) if the number of matrix types is at least one and less than 21, a list of integers between 1 and 21 indicating which matrix types are to be tested.

The input data for testing xGEES has the same format as for xGEEV, except SES replaces SEV when testing SGEES, and CES replaces CEV when testing CGEES.

The input data for testing xGEEVX consists of two parts. The first part is identical to that for xGEEV (using SVX instead of SEV and CVX instead of CEV). The second consists of precomputed data for testing the eigenvalue/vector condition estimation routines. Each matrix is stored on 1+2*N lines, where N is its dimension (1+N+N**2 lines for complex data). The first line contains the dimension, a single integer (for complex data, a second integer ISRT indicating how the data is sorted is also provided). The next N lines contain the matrix, one row per line (N**2 lines for complex data, one item per row). The last N lines correspond to each eigenvalue. Each of these last N lines contains 4 real values: the real part of the eigenvalues, the imaginary part of the eigenvalue, the reciprocal condition number of the eigenvalues, and the reciprocal condition number of the vector eigenvector. The end of data is indicated by dimension N=0. Even if no data is to be tested, there must be at least one line containing N=0.

The input data for testing xGEESX also consists of two parts. The first part is identical to that for xGEES (using SSX instead of SES and CSX instead of CES). The second consists of precomputed data for testing the eigenvalue/vector condition estimation routines. Each matrix is stored on 3+N lines, where N is its dimension (3+N**2 lines for complex data). The first line contains the dimension N and the dimension M of an invariant subspace (for complex data, a third integer ISRT indicating how the data is sorted is also provided). The second line contains M integers, identifying the eigenvalues in the invariant subspace (by their position in a list of eigenvalues ordered by increasing real part (or imaginary part, depending on ISRT for complex data)). The next N lines contains the matrix (N**2 lines for complex data). The last line contains the reciprocal condition number for the average of the selected eigenvalues, and the reciprocal condition number for the corresponding right invariant subspace. The end of data is indicated by a line containing N=0 and M=0. Even if no data is to be tested, there must be at least one line containing N=0 and M=0.

7.4 Testing the Generalized Nonsymmetric Eigenvalue Routines

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

- the order N of the test matrix pair (A, B)
- the type of the test matrix pair (A, B)

The test program thus consists of a doubly-nested loop, the outer one over N and the inner one over matrix types. On each iteration of the innermost loop, a pair of matrices (A, B) is generated and used to test the eigenvalue routines.

7.4.1 The Generalized Nonsymmetric Eigenvalue Drivers

The driver routines for the generalized nonsymmetric eigenvalue problem are

xGGES generalized Schur form driver

xGGESX expert version of xGGES (includes condition estimation)

xGGEV generalized eigenvalue/eigenvector driver

xGGEVX expert version of xGGEV (includes condition estimation)

For these driver routines, some tests are done by generating random matrix pairs of a dimension and type chosen by users, and computing error bounds similar to those used for the nonsymmetric generalized eigenvalue computational routines. For some tests, one can use a file of precomputed matrix pairs and condition numbers.

The parameters that can be varied in the random matrix pair tests are:

- the order N of the test matrix pair (A, B)
- the type of the test matrix pair (A, B)

7.4.2 Test Matrices for the Generalized Nonsymmetric Eigenvalue Routines

Twenty-six different types of test matrix pairs may be generated for the generalized eigenvalue routines. Tables 6 and 7 show the available types, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries.

The following symbols and abbreviations are used:

0: The zero matrix.

I: The identity matrix.

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

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

Table 6: Sparse test matrices for the generalized nonsymmetric eigenvalue problem

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

Table 7: Dense test matrices for the generalized nonsymmetric eigenvalue problem

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

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

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

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

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

Except for matrices with random entries, all the matrix pairs include at least one infinite, one zero, and one singular eigenvalue (0/0, which is not well defined). For arithmetic, geometric, and clustered eigenvalue distributions, the eigenvalues lie between ϵ (the machine precision) and 1 in absolute value. The eigenvalue distributions have the following meanings:

Arithmetic: Difference between adjacent eigenvalues is a constant.

Geometric: Ratio of adjacent eigenvalues is a constant.

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

Random: Eigenvalues are logarithmically distributed.

Random entries: Matrix entries are uniformly distributed random numbers.

7.4.3 Test Matrices for the Generalized Nonsymmetric Eigenvalue Drivers

- Simple Drivers
  Twenty-six different types of test matrix pairs may be generated for the simple generalized eigenvalue drivers xGGES and xGGEV. Tables 6 and 7 show the available types, along with the numbers used to refer to the matrix types.

- Expert generalized Schur form driver

  For the expert generalized Schur form driver xGGESX, two kinds of tests are executed: one is to use the built-in test matrix pairs and another one is to use a precomputed set of test problems. The test matrix generator xLATM5 generates the five types of built-in test matrix pairs (A, B) of the form:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ & B_{22} \end{pmatrix},$$

where $m \times m$ matrices A_{11} and B_{11} and $(n-m) \times (n-m)$ matrices A_{22} and B_{11} are chosen as the following, and A_{12} and B_{12} are chosen so that the generalized Sylvester equation:

$$\begin{aligned} A_{11}R - LA_{22} &= -A_{12} \\ B_{11}R - LB_{22} &= -B_{12} \end{aligned}$$

has the prescribed solutions R and L.

Type 1: $A_{11} = J_m(1,-1), B_{11} = I_m$ and $A_{22} = J_{n-m}(1-\alpha, 1), B_{22} = I_{n-m}$, where $J_k(d,s)$ denotes a Jordan block of dimension k with d and s as diagonal and superdiagonal elements, respectively. In the tests, $\alpha = \sqrt{\epsilon}$, and $1/\sqrt{\epsilon}$.

Type 2: (A_{11}, B_{11}) and (A_{22}, B_{22}) are upper triangular with

$$a_{ij} = 2(0.5 - \sin(i)), \quad b_{ij} = 2(0.5 - \sin(i \cdot j)), \quad i = 1, \ldots, m, \quad j = i, \ldots, m.$$

$$a_{ij} = 2(0.5 - \sin(i+j)), \quad b_{ij} = 2(0.5 - \sin(i \cdot j)), \quad i = m+1, \ldots, n-m, \quad j = i, \ldots, n-m.$$

Type 3: (A_{11}, B_{11}) and (A_{22}, B_{22}) are upper quasi-triangular, where the entries are first set as for Type 2. Then each second diagonal block in A_{11} and each third block in A_{22} are made 2×2 by setting $a_{k+1,k+1} = a_{kk}$ and $a_{k+1,k} = -\sin(a_{k,k+1})$ for appropriate values of k.

Type 4: (A_{11}, B_{11}) and (A_{22}, B_{22}) are set as dense blocks:

$$a_{ij} = 20(0.5 - \sin(i \cdot j)), \quad b_{ij} = 2(0.5 - \sin(i+j)), \quad i = 1, \ldots, m, \quad j = 1, \ldots, m.$$

$$a_{ij} = 20(0.5 - \sin(i+j)), \quad b_{ij} = 2(0.5 - \sin(i \cdot j)), \quad i = m+1, \ldots, n-m, \quad j = m+1, \ldots, n-m.$$

Type 5: (A, B) has potentially close or common eigenvalues, and large or very large departure from block diagonality. First, A_{11} is chosen as the $m \times m$ leading submatrix of A_1, where

$$A_1 = \begin{pmatrix} 1 & \beta & & & & & & & \\ -\beta & 1 & & & & & & & \\ & & 1+\delta & \beta & & & & & \\ & & -\beta & 1+\delta & & & & & \\ & & & & \delta & 1 & & & \\ & & & & -1 & \delta & & & \\ & & & & & & -\delta & 1 & \\ & & & & & & -1 & -\delta & \\ & & & & & & & & 1 \end{pmatrix},$$

then A_{22} is chosen as the $(n-m) \times (n-m)$ leading submatrix of A_2, where

$$A_2 = \begin{pmatrix} -1 & \beta & & & & & & & \\ -\beta & -1 & & & & & & & \\ & & 1-\delta & \beta & & & & & \\ & & -\beta & 1-\delta & & & & & \\ & & & & \delta & 1+\beta & & & \\ & & & & -1-\beta & \delta & & & \\ & & & & & & -\delta & 1+\beta & \\ & & & & & & -1-\beta & -\delta & \\ & & & & & & & & 1-\delta \end{pmatrix}.$$

B_{11} and B_{22} are chosen as the identity matrices I_m and I_{n-m}, respectively. The values of $\beta = 20/\alpha$ and $\delta = -1.5/\alpha$ are used for $\alpha = \sqrt{\epsilon}$ and $1/\sqrt{\epsilon}$.

- Expert generalized eigenvalue/eigenvector driver
For the expert generalized eigenvalue/eigenvector driver xGGEVX, two kinds of tests can be executed: one is to use the built-in test matrix pairs and another one is to use a precomputed set of test problems. The test matrix generator xLATM6 generates the two types of built-in test matrix pairs of the form

$$(A, B) = Y^{-H}(D_a, D_b)X^{-1}$$

where

Type 1:

$$D_a = \begin{pmatrix} 1+\alpha & & & & \\ & 2+\alpha & & & \\ & & 3+\alpha & & \\ & & & 4+\alpha & \\ & & & & 5+\alpha \end{pmatrix}, \qquad D_b = I_5.$$

Type 2:

$$D_a = \begin{pmatrix} 1 & -1 & & & \\ 1 & 1 & & & \\ & & 1 & & \\ & & & 1+\alpha & 1+\beta \\ & & & -1-\beta & 1+\alpha \end{pmatrix}, \qquad D_b = I_5.$$

The exact left and right eigenvectors of (A, B) are the rows and columns of

$$Y^H = \begin{pmatrix} 1 & & -y & y & -y \\ & 1 & -y & y & -y \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} \quad \text{and} \quad X = \begin{pmatrix} 1 & & -x & -x & x \\ & 1 & x & -x & -x \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix},$$

respectively, where α, β, x, and y are chosen from the values $\{\epsilon^{\frac{1}{4}}, 0.1, 1, 10, \epsilon^{-\frac{1}{4}}\}$, independently. For these test matrix pairs, the left and right eigenvectors are known exactly. Thus, the eigenvalue condition numbers can be computed exactly. In addition, xLATM6 will compute reciprocal condition numbers for the first and fifth eigenvectors. A total of 1250 different pencils (A, B) are generated in the tests.

7.4.4 Tests performed on the Generalized Nonsymmetric Eigenvalue Routines

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

1. (A, B) is decomposed as (UHV^H, URV^H), where U and V are unitary (orthogonal in the real case), H is upper Hessenberg, R is upper triangular, and U^H is the conjugate transpose of U.

2. (H, R) is decomposed as (QSZ^H, QTZ^H), where Q and Z are unitary (orthogonal in the real case), (S, T) is in generalized Schur form, where S is upper (quasi)-triangular and T is upper triangular with non-negative real diagonal entries and S is in Schur form; this also gives the generalized eigenvalues λ_i, which are expressed as pairs (α_i, β_i), where $\lambda_i = \alpha_i/\beta_i$.

3. The left and right generalized eigenvectors l_i and r_i for the pair (S, T) are computed, and from them the back-transformed eigenvectors $\hat{l}_i$ and $\hat{r}_i$ for the matrix pair (H, R). The eigenvectors are normalized so that their largest element has absolute value 1[3]. (Note that eigenvectors corresponding to singular eigenvalues, i.e., eigenvalues for which $\alpha = \beta = 0$, are not well defined, these are not tested in the eigenvector tests described below.)

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

$$r_1 = \frac{\|A-UHV^H\|}{n\epsilon\|A\|} \qquad r_2 = \frac{\|B-URV^H\|}{n\epsilon\|B\|}$$

$$r_3 = \frac{\|I-UU^H\|}{n\epsilon} \qquad r_4 = \frac{\|I-VV^H\|}{n\epsilon}$$

$$r_5 = \frac{\|H-QSZ^H\|}{n\epsilon\|S\|} \qquad r_6 = \frac{\|R-QTZ^H\|}{n\epsilon\|R\|}$$

$$r_7 = \frac{\|I-QQ^H\|}{n\epsilon} \qquad r_8 = \frac{\|I-ZZ^H\|}{n\epsilon}$$

$$r_9 = \max_i \frac{\|l_i^H(\beta_i S - \alpha_i T)\|}{\epsilon \max(\|\beta_i S\|, \|\alpha_i T\|)} \qquad r_{10} = \max_i \frac{\|\hat{l}_i^H(\beta_i H - \alpha_i R)\|}{\epsilon \max(\|\beta_i H\|, \|\alpha_i R\|)}$$

$$r_{11} = \max_i \frac{\|(\beta_i S - \alpha_i T)r_i\|}{\epsilon \max(\|\beta_i S\|, \|\alpha_i T\|)} \qquad r_{12} = \max_i \frac{\|(\beta_i H - \alpha_i R)\hat{r}_i\|}{\epsilon \max(\|\beta_i H\|, \|\alpha_i R\|)}$$

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

[3]For the purpose of normalization, the "absolute value"of a complex number $z = x + iy$ is computed as $|x| + |y|$.

When the test program is run, these test ratios will be compared with a user-specified threshhold 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=18, seed=2548,1429,1713,1411, result  8 is   11.33
```

In this example, the test matrix was of order $n = 25$ and of type 18 from Table 6, "seed" is the initial 4-integer seed of the random number generator used to generate A and B, and "result" specifies that test ratio r_8 failed to pass the threshhold, and its value was 11.33.

The normalization of the eigenvectors will also be checked. If the absolute value of the largest entry in an eigenvector is not within $\epsilon \times$ THRESH of 1, then a message is printed specifying the error. A sample message is

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

7.4.5 Tests performed on the Generalized Nonsymmetric Eigenvalue Drivers

The four driver routines have slightly different tests applied to them.

- For SGGES the following tests are computed:

  SGGES with no sorting of eigenvalues

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

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

  $r_5 = 1/\epsilon$ if (A, B) is not in generalized Schur form

  $$r_6 = \max_j D(j) = \begin{cases} \frac{|\alpha(j)-S(j,j)|}{\max(|\alpha(j)|,|S(j,j)|)} + \frac{|\beta(j)-T(j,j)|}{\max(|\beta(j)|,|T(j,j)|)} & \text{if } \alpha(j) \text{ is real} \\ \frac{|\det(sS_j-wT_j)|}{\epsilon\max(s\|S_j\|,|w|\|T_j\|)\|sS_j-wT_j\|} & \text{if } \alpha(j) \text{ is complex,} \end{cases}$$

  SGGES with sorting of eigenvalues

  $r_7 = \frac{\|(A,B)-Q(S,T)Z^T\|}{n\epsilon\|(A,B)\|}$ $\qquad$ $r_8 = \frac{\|I-QQ^T\|}{n\epsilon}$

  $r_9 = \frac{\|I-ZZ^T\|}{n\epsilon}$

  $r_{10} = 1/\epsilon$ if (A, B) is not in generalized Schur form

  $$r_{11} = \max_j D(j) = \begin{cases} \frac{|\alpha(j)-S(j,j)|}{\max(|\alpha(j)|,|S(j,j)|)} + \frac{|\beta(j)-T(j,j)|}{\max(|\beta(j)|,|T(j,j)|)} & \text{if } \alpha(j) \text{ is real} \\ \frac{|\det(sS_j-wT_j)|}{\epsilon\max(s\|S_j\|,|w|\|T_j\|)\|sS_j-wT_j\|} & \text{if } \alpha(j) \text{ is complex,} \end{cases}$$

  $r_{12} = 1/\epsilon$ if the eigenvalues were sorted correctly

  where S_j and T_j are the 2×2 diagonal blocks of S and T corresponding to the jth and $(j+1)$th eigenvalues.

- For SGGESX the following tests are computed:

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

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

$$r_5 = 1/\epsilon \quad \text{if } (A, B) \text{ is not in generalized Schur form}$$

$$r_6 = \max_j D(j) = \begin{cases} \frac{|\alpha(j) - S(j,j)|}{\max(|\alpha(j)|, |S(j,j)|)} + \frac{|\beta(j) - T(j,j)|}{\max(|\beta(j)|, |T(j,j)|)} & \text{if } \alpha(j) \text{ is real} \\ \frac{|\det(sS_j - wT_j)|}{\epsilon \max(s\|S_j\|, |w|\|T_j\|)\|sS_j - wT_j\|} & \text{if } \alpha(j) \text{ is complex,} \end{cases}$$

where S_j and T_j are the 2×2 diagonal blocks of S and T corresponding to the jth and $(j+1)$th eigenvalues.

$$r_7 = 1/\epsilon \qquad \text{if eigenvalues were not sorted correctly}$$

$$r_8 = \begin{cases} \max\left(\frac{\texttt{DIFest}}{\texttt{DIFtru}}, \frac{\texttt{DIFtru}}{\texttt{DIFest}}\right) & \text{if } \ \texttt{DIFest} \neq 0 \ \text{ and } \ \texttt{DIFtru} \neq 0 \\ 1/\epsilon & \text{if either} \begin{cases} \texttt{DIFest} = 0 \ \text{ and } \ \texttt{DIFtru} > \epsilon\|(A, B)\| \\ \texttt{DIFtru} = 0 \ \text{ and } \ \texttt{DIFest} > \epsilon\|(A, B)\| \end{cases} \end{cases}$$

$$r_9 = 1/\epsilon \quad \text{if} \quad \texttt{DIFest} \neq 0 \quad \text{or} \quad \texttt{DIFtru} > \epsilon * \|(A, B)\| \quad \text{when reordering fails}$$

$$r_{10} = \begin{cases} \max\left(\frac{\texttt{PLest}}{\texttt{PLtru}}, \frac{\texttt{PLtru}}{\texttt{PLest}}\right) & \text{if } \ \texttt{PLest} \neq 0 \ \text{ and } \ \texttt{PLtru} \neq 0 \\ 1/\epsilon & \text{if either} \begin{cases} \texttt{PLest} = 0 \ \text{ and } \ \texttt{PLtru} > \epsilon\|(A, B)\| \\ \texttt{PLtru} = 0 \ \text{ and } \ \texttt{PLest} > \epsilon\|(A, B)\| \end{cases} \end{cases}$$

The test r_{10} is for precomputed test problems. `DIFtru` is the true value of DIF obtained by computing the minimum singular value of the matrix generated in solving the generalized Sylvester equation and `DIFest` is the estimated value of DIF obtained from `xTGSEN`, and where `PLtru` is the true value of PL obtained by computing the norm of the matrix L resulting from the solution of the generalized Sylvester equation and `PLest` is the estimated value of PL obtained from `xTGSEN`.
Test 8 is compared to 10*`THRESH` instead of `THRESH`.
Test 9 is a check for very ill-conditioned problems. It is possible with ill-conditioned problems for reordering to be (nearly) impossible which corresponds to very large condition numbers. So if reordering fails, it is checked that the true reciprocal condition number `DIFtru` is small (i.e. the condition is large) and it is checked that the estimated reciprocal condition number `DIFest` is zero (when reordering fails, the subroutine is supposed to return `DIFest` $= 0$).

These test ratios are compared to the input parameter `THRESH`. If a ratio exceeds `THRESH`, a message is printed specifying the test matrix, the ratio that failed and its value, just like the tests performed on the nonsymmetric eigenvalue problem computational routines.

- For SGGEV the following tests are computed:

$$
\begin{aligned}
r_1 &= \max_{\text{left eigenvalue/-vector pairs } (\beta/\alpha, l)} \frac{\|l^H(\beta A - \alpha B)\|}{\epsilon \max(\|\beta A\|, \|\alpha B\|)} \\
r_2 &= \text{normalization of the left eigenvector } l \\
r_3 &= \max_{\text{right eigenvalue/-vector pairs } (\beta/\alpha, r)} \frac{\|(\beta A - \alpha B)r\|}{\epsilon \max(\|\beta A\|, \|\alpha B\|)} \\
r_4 &= \text{normalization of the right eigenvector } r \\
r_5 &= (W(\text{full}) = W(\text{partial})) \\
r_6 &= (l(\text{full}) = l(\text{partial})) \\
r_7 &= (r(\text{full}) = r(\text{partial}))
\end{aligned}
$$

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

- For SGGEVX the following tests are computed:

$$
\begin{aligned}
r_1 &= \max_{\text{left eigenvalue/-vector pairs } (\beta/\alpha, l)} \frac{\|l^H(\beta A - \alpha B)\|}{\epsilon \max(\|\beta A\|, \|\alpha B\|)} \\
r_2 &= \max_{\text{right eigenvalue/-vector pairs } (\beta/\alpha, r)} \frac{\|(\beta A - \alpha B)r\|}{\epsilon \max(\|\beta A\|, \|\alpha B\|)} \\
r_3 &= \max\left(\frac{\texttt{Sest}(i)}{\texttt{Stru}(i)}, \frac{\texttt{Stru}(i)}{\texttt{Sest}(i)}\right) \quad \text{for} \quad i = 1, ..., 5 \\
r_4 &= \max\left(\frac{\texttt{Difest}(i)}{\texttt{DIFtru}(i)}, \frac{\texttt{DIFtru}(i)}{\texttt{DIFest}(i)}\right) \quad \text{for} \quad i = 1 \text{ or } 5
\end{aligned}
$$

  where `Stru`(i) is the exact reciprocal condition number for eigenvalue i and `Sest`(i) is the estimated reciprocal condition number computed by `xTGSNA`, and `DIFtru`(i) is the exact reciprocal condition number for eigenvector i and `DIFest`(i) is the estimated reciprocal condition number computed by `xTGSNA`.
  Test 4 is compared to 10*`THRESH` instead of `THRESH`.
  All norms are $\|\cdot\|_1$. The scalings in the test ratios assure that the ratios will be $O(1)$, independent of $\|A\|$ and ϵ, and nearly independent of n.

7.4.6 Input file for Testing the Generalized Nonsymmetric Eigenvalue Routines

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

```
SGG:  Data file for testing Nonsymmetric Eigenvalue Problem routines
7                               Number of values of N
0 1 2 3 5 10 16                 Values of N (dimension)
4                               Number of parameter values
1   1   2   2                   Values of NB (blocksize)
```

```
100 100 2   2                 Values of NBMIN (minimum blocksize)
2   4   2   4                 Values of NSHIFT (no. of shifts)
100 100 2   2                 Values of MAXB (multishift crossover pt)
100 100 2   2                 Values of NBCOL (minimum col. dimension)
20.0                          Threshold value
T                             Put T to test the LAPACK routines
T                             Put T to test the driver routines
T                             Put T to test the error exits
1                             Code to interpret the seed
SGG  26
```

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

line 2:	The number of values of N
line 3:	The values of N, the matrix dimension
line 4:	The threshold value for the test ratios
line 5:	TSTCHK, flag to test LAPACK routines
line 6:	TSTDRV, flag to test driver routines
line 7:	TSTERR, flag to test error exits from LAPACK and driver routines
line 8:	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 9:	If line 8 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 standard nonsymmetric case. The valid 3-character codes are SGG (CGG in complex, DGG in double precision, and ZGG in complex*16).

7.4.7 Input file for Testing the Generalized Nonsymmetric Eigenvalue Drivers

There is a single input file to test all drivers. The input data for each path (testing xGGEV, xGGES, xGGEVX and xGGESX) is preceeded by a single line identifying the path (SGV, SGS, SVX and SGX, respectively, when x=S, and CGV, CGS, CXV and CGX, respectively, when x=C). We discuss each set of input data in turn.

An annotated example of input data for testing SGGEV is shown below (testing CGGEV is identical except CGV replaces SGV):

```
SGV                 Data for the Real Nonsymmetric Eigenvalue Problem Driver
6                   Number of matrix dimensions
2 6 8 10 15 20      Matrix dimensions
1 1 1 2 1           Parameters NB, NBMIN, NXOVER, NS, NBCOL
10                  Threshold value
.FALSE.             Put .TRUE. to test the error exits
0                   Code to interpret the seed
SGV 26              Test all 26 matrix types
```

The first line must contain the characters SGV in columns 1-3. The remaining lines 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 values of the parameters NB, NBMIN, NXOVER, NS and NBCOL
line 5:	The threshold value THRESH for the test ratios
line 6:	T to test the error exits
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
line 9:	Contains 'SGV' in columns 1-3, followed by the number of matrix types (an integer from 0 to 26)
line 10:	(and following) if the number of matrix types is at least one and less than 26, a list of integers between 1 and 26 indicating which matrix types are to be tested.

The input data for testing xGGES has the same format as for xGGEV, except SGS replaces SGV when testing SGGES, and CGS replaces CGV when testing CGGES.

The input data for testing xGGEVX consists of two files. An annotated example of the first type is listed below

```
SVX                 Data for the Real Nonsymmetric Eigenvalue Expert Driver
5                   Largest matrix dimension
1 1 1 2 1           Parameters NB, NBMIN, NXOVER, NS, NBCOL
10                  Threshold for test ratios
.FALSE.             Put .TRUE. to test the error exits
0                   Code to interpret the seed
```

The first line must contain the characters SXV in columns 1-3. The remaining lines are read using list-directed input and specify the following values:

line 2:	The number of values of N
line 3:	The values of the parameters NB, NBMIN, NXOVER, NS and NBCOL
line 4:	The threshold value THRESH for the test ratios
line 5:	T to test the error exits
line 6:	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

The second consists of precomputed data for testing the eigenvalue/eigenvector condition estimation routines. It has the same format as above, except that on line 2, a zero is entered as the matrix size and on line 7 and the following, there is precomputed data. Each example is stored on 2*N+3 lines, where N is its dimension (2*N*N+3 lines for complex data). The first line contains the dimension (a single integer). The next N lines contain the matrix A, one row per line. The next N lines contain the matrix B. The next line contains the reciprocals of the eigenvalue condition numbers. The last line contains the reciprocals of the eigenvector condition numbers. The end of data is indicated by dimension N = 0. Even if no data is to be tested, there must be at least one line containing N = 0.

The input data for testing xGGESX consists of two parts. The first part is identical to that for xGGEVX (using SGX instead of SVX and CGX instead of CVX). The second consists of precomputed data for testing the eigenvalue/eigenvector condition estimation routines. Again, it has the same format as the second part for xGGEVX, except that each example is stored on 2*N+3 lines, where N is its dimension (2*N*N+3 lines for complex data). The first line contains the dimension (a single integer). The next line contains an integer k such that only the last k eigenvalues will be selected and appear in the leading diagonal blocks of A and B. The next N lines contain the matrix A, one row per line. The next N lines contain the matrix B. The last line contains the reciprocal of the eigenvalue cluster condition number and the reciprocal of the deflating subspace (associated with selected eigencluster) condition number. The end of data is indicated by dimension N = 0. Even if no data is to be tested, there must be at least one line containing N = 0.

7.5 Testing the Nonsymmetric Eigenvalue Condition Estimation Routines

The main routines tested are xTREXC, xTRSYL, xTRSNA and xTRSEN. xTREXC reorders eigenvalues on the diagonal of a matrix in Schur form, xTRSYL solves the Sylvester equation $AX+XB=C$ for X given A, B and C, xTRSNA computes condition numbers for individual eigenvalues and right eigenvectors, and xTRSEN computes condition numbers for the average of a cluster of eigenvalues, as well as their corresponding right invariant subspace. Several auxiliary routines xLAEQU, xLAEXC, xLALN2, xLAQTR, and xLASY2 are also tested; these are only used with real (x=S or x=D) data.

No parameters can be varied; the data files contain precomputed test problems along with their precomputed solutions. The reason for this approach is threefold. First, there is no simple residual test ratio which can test correctness of a condition estimator. Second, no comparable code in another library exists to compare solutions. Third, the condition numbers we compute can themselves be quite ill-conditioned, so that we need the precomputed solution to verify that the computed result is within acceptable bounds.

The test program `xeigtsts` reads in the data from the data file `sec.in` (for the REAL code). If there are no errors, a single message saying that all the routines pass the tests will be printed. If any routine fails its tests, an error message is printed with the name of the failed routine along with the number of failures, the number of the example with the worst failure, and the test ratio of the worst failure.

For more details on eigencondition estimation, see LAPACK Working Note 13 [4].

7.5.1 Testing the Generalized Nonsymmetric Eigenvalue Condition Estimation Routines

The main routines tested are xTGEXC, xTGSYL, xTGSNA, and xTGSEN. xTGEXC reorders the generalized Schur decomposition of a matrix pair, xTGSYL solves the generalized Sylvester equation $AR-LB=C$, $DR-LE=F$ for R and L given (A,D), (B,E), and (C,F), xTGSNA computes condition numbers for individual eigenvalues and eigenvectors of a matrix pair, and xTGSEN computes condition numbers for the average of a cluster of eigenvalues, as well as their corresponding right invariant subspace. Several auxiliary routines xGETC2, xGESC2, xTGDIF, xTGEX2, and xTGSY2 exist.

NOTE: There are known testing failures in condition number estimation routines in the generalized nonsymmetric eigenproblem testing. Specifically in sgd.out, dgd.out, cgd.out and zgd.out. The cause for the failures of some test cases is that the mathematical algorithm used for estimating the condition numbers could over- or under-estimate the true values in a certain factor in some rare cases.

For more details on eigencondition estimation, see LAPACK Working Note 87.

7.6 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.

However, there is one exception. The test routine for the LAPACK banded 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 testing program thus consists of a doubly-nested loop, the outer one 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.

7.6.1 The Symmetric Eigenvalue Drivers

The driver routines for the symmetric eigenvalue problem are

xSTEV eigenvalue/eigenvector driver for symmetric tridiagonal matrix,

xSTEVD divide and conquer driver for symmetric tridiagonal matrix,

xSTEVX selected eigenvalue/eigenvectors for symmetric tridiagonal matrix,

xSTEVR selected eigenvalue/eigenvectors for symmetric tridiagonal matrix using Relatively Robust Representations,

xSYEV eigenvalue/eigenvector driver for symmetrix matrix,

xSYEVD divide and conquer driver for symmetric matrix,

xSYEVX selected eigenvalue/eigenvectors for symmetric matrix,

xSYEVR selected eigenvalue/eigenvectors for symmetric matrix using Relatively Robust Representations,

xSPEV eigenvalue/eigenvector driver for symmetric matrix in packed storage,

xSPEVD divide and conquer driver for symmetric matrix in packed storage,

xSPEVX selected eigenvalue/eigenvectors for symmetric matrix in packed storage,

xSBEV eigenvalue/eigenvector driver for symmetric band matrix,

xSBEVD divide and conquer driver for symmetric band matrix,

xSBEVX selected eigenvalue/eigenvectors for symmetric band matrix.

7.6.2 Test Matrices for the Symmetric Eigenvalue Routines

Except for the banded matrices, twenty-one different types of test matrices may be generated for the symmetric eigenvalue routines. Table 8 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 7.3.2).

For banded matrices, fifteen different types of test matrices may be generated. These fifteen test matrices are the same as the first fifteen test matrices in Table 8.

	Eigenvalue Distribution			
Type	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3	4, $6^\dagger$, $7^\ddagger$	5	
UDU^{-1}	8, $11^\dagger$, $12^\ddagger$, 16^*, $19^\star$, $20^\bullet$	9, 17^*	10, 18^*	
Symmetric w/Random entries				13, $14^\dagger$, $15^\ddagger$
Diag. Dominant		21		

$\dagger$– matrix entries are $O(\sqrt{\text{overflow}})$
$\ddagger$– matrix entries are $O(\sqrt{\text{underflow}})$
$*$ – diagonal entries are positive
$\star$ – matrix entries are $O(\sqrt{\text{overflow}})$ and diagonal entries are positive
$\bullet$ – matrix entries are $O(\sqrt{\text{underflow}})$ and diagonal entries are positive

Table 8: Test matrices for the symmetric eigenvalue problem

7.6.3 Test Matrices for the Symmetric Eigenvalue Drivers

Eighteen different types of test matrices may be generated for the symmetric eigenvalue drivers. The first 15 test matrices are the same as the types of matrices used to test the symmetric eigenvalue computational routines, and are given in Table 8. Table 9 shows the types available, along with the numbers used to refer to the matrix types. Except as noted, all matrices have $O(1)$ entries. 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 5.2.1).

	Eigenvalue Distribution			
Type	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3	4, $6^{\dagger}$, $7^{\ddagger}$	5	
UDU^{-1}	8, $11^{\dagger}$, $12^{\ddagger}$	9	10	
Symmetric w/Random entries				13, $14^{\dagger}$, $15^{\ddagger}$
Band				16, $17^{\dagger}$, $18^{\ddagger}$

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

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

Table 9: Test matrices for the symmetric eigenvalue drivers

7.6.4 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. U is represented as a product of Householder transformations, whose vectors are stored in the first n-1 columns of V, and whose scale factors are in TAU.

2. S is decomposed as $Z\text{D}1Z^*$, where Z is real orthogonal and $D1$ is a real diagonal matrix of eigenvalues. $D2$ is the matrix of eigenvalues computed when Z is not computed.

3. The "PWK" method is used to compute $D3$, the matrix of eigenvalues, using a square-root-free method which does not compute Z.

4. S is decomposed as $Z4\ D4\ Z4^*$, for a symmetric positive definite tridiagonal matrix. D_5 is the matrix of eigenvalues computed when Z is not computed.

5. Selected eigenvalues ($WA1$, $WA2$, and $WA3$) are computed and denote eigenvalues computed to high absolute accuracy, with different range options. WR will denote eigenvalues computed to high relative accuracy.

6. Given the eigenvalues, the eigenvectors of S are computed in Y.

7. S is factored as $Z\ D1\ Z^*$.

To check these calculations, the following test ratios are computed (where banded matrices only compute test ratios 1-4):

$$\begin{aligned}
r_1 &= \frac{\|A - VSV^*\|}{n\varepsilon\,\|A\|} \\
&\quad \text{computed by } SSYTRD(UPLO =' U') \text{ or } SSBTRD(UPLO =' U') \\
r_2 &= \frac{\|I - UV^*\|}{n\varepsilon} \\
&\quad \text{test of } SORGTR(UPLO =' U') \\
r_3 &= \frac{\|A - VSV^*\|}{n\varepsilon\,\|A\|} \\
&\quad \text{computed by } SSYTRD(UPLO =' L') \text{ or } SSBTRD(UPLO =' L') \\
r_4 &= \frac{\|I - UV^*\|}{n\varepsilon} \\
&\quad \text{test of } SORGTR(UPLO =' L')
\end{aligned}$$

Tests 5-8 are the same as tests 1-4 but for SSPTRD and SOPGTR.

$$\begin{aligned}
r_9 &= \frac{\|S - ZD1Z^*\|}{n\,ulp\,\|S\|} \\
&\quad \text{test of } SSTEQR('V') \\
r_{10} &= \frac{\|I - ZZ^*\|}{n\,ulp} \\
&\quad \text{test of } SSTEQR('V') \\
r_{11} &= \frac{\|D1 - D2\|}{ulp\,\|D1\|} \\
&\quad \text{test of } SSTEQR('V') \\
r_{12} &= \frac{\|D1 - D3\|}{ulp\,\|D1\|} \\
&\quad \text{test of } SSTERF \\
r_{13} &= \begin{cases} 0 & \text{if eigenvalues of } S \text{ are within } THRESH \text{ of those in } D1. \\ 2 * THRESH & \text{otherwise} \end{cases}
\end{aligned}$$

For S positive definite,

$$\begin{aligned}
r_{14} &= \frac{\|S - Z4\,D4\,Z4^*\|}{n\,ulp\,\|S\|} \\
&\quad \text{test of } SPTEQR('V')
\end{aligned}$$

$$
\begin{aligned}
r_{15} &= \frac{\|I - Z4\, Z4^*\|}{n\, ulp} \\
&\quad \text{test of } SPTEQR('V') \\
r_{16} &= \frac{\|D4 - D5\|}{100\, ulp\, \|D4\|} \\
&\quad \text{test of } SPTEQR('N')
\end{aligned}
\tag{1}
$$

When S is also diagonally dominant by a factor $\gamma < 1$,

$$
\begin{aligned}
r_{17} &= \max_i \frac{\|D4(i) - WR(i)\|}{\|D4(i)\|\, \omega}, \\
&\quad \text{where } \omega = 2(2n-1)\, ulp \frac{1 + 8 * \gamma^2}{(1-\gamma)^4} \\
&\quad \text{test of } SSTEBZ('A', 'E')
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
r_{18} &= \frac{\|WA1 - D3\|}{ulp\, \|D3\|} \\
&\quad \text{test of } SSTEBZ('A', 'E') \\
r_{19} &= \frac{\max_i(\min_j(\|WA2(i) - WA3(j)\|)) + \max_i(\min_j(\|WA3(i) - WA2(j)\|))}{ulp\, \|D3\|} \\
&\quad \text{test of } SSTEBZ('I', 'E') \\
r_{20} &= \frac{\|S - Y\, WA1\, Y^*\|}{n\, ulp\, \|S\|} \\
&\quad \text{test of } SSTEBZ \text{ and } SSTEIN \\
r_{21} &= \frac{\|I - YY^*\|}{n\, ulp} \\
&\quad \text{test of } SSTEBZ \text{ and } SSTEIN \\
r_{22} &= \frac{\|S - Z\, D\, Z^*\|}{\|S\|\; n\, ulp} \\
&\quad \text{for SSTEDC('I')} \\
r_{23} &= \frac{\|I - Z\, Z^*\|}{n\, ulp} \\
&\quad \text{for SSTEDC('I')} \\
r_{24} &= \frac{\|S - Z\, D\, Z^*\|}{\|S\|\; n\, ulp} \\
&\quad \text{for SSTEDC('V')} \\
r_{25} &= \frac{\|I - Z\, Z^*\|}{n\, ulp} \\
&\quad \text{for SSTEDC('V')}
\end{aligned}
$$

$$r_{26} = \frac{\|D1 - D2\|}{\|D1\| \; ulp}$$

for SSTEDC('V') and SSTEDC('N')

$$r_{27} = \max_i \frac{\|D6(i) - WR(i)\|}{\|D6(i)\| \omega},$$

where $\omega = 2(2n-1) \; ulp \frac{1 + 8 * \gamma^2}{(1-\gamma)^4}$

test of $SSTEGR('V','A')$

$$r_{28} = \max_i \frac{\|D6(i) - WR(i)\|}{\|D6(i)\| \omega},$$

where $\omega = 2(2n-1) \; ulp \frac{1 + 8 * \gamma^2}{(1-\gamma)^4}$

test of $SSTEGR('V','I')$

$$r_{29} = \frac{\|S - Z \, D \, Z^*\|}{\|S\| \; n \, ulp}$$

for SSTEGR('V','I')

$$r_{30} = \frac{\|I - Z \, Z^*\|}{n \, ulp}$$

for SSTEGR('V','I')

$$r_{31} = \frac{\max_i(\min_j(\|WA2(i) - WA3(j)\|)) + \max_i(\min_j(\|WA3(i) - WA2(j)\|))}{ulp \; \|D3\|}$$

test of $SSTEGR('N','I')$ versus $SSTEGR('V','I')$

$$r_{32} = \frac{\|S - Z \, D \, Z^*\|}{\|S\| \; n \, ulp}$$

for $SSTEGR('V','V')$

$$r_{33} = \frac{\|I - Z \, Z^*\|}{n \, ulp}$$

for $SSTEGR('V','V')$

$$r_{34} = \frac{\max_i(\min_j(\|WA2(i) - WA3(j)\|)) + \max_i(\min_j(\|WA3(i) - WA2(j)\|))}{ulp \; \|D3\|}$$

test of $SSTEGR('N','V')$ versus $SSTEGR('V','V')$

$$r_{35} = \frac{\|S - Z \, D \, Z^*\|}{\|S\| \; n \, ulp}$$

for $SSTEGR('V','A')$

$$r_{36} = \frac{\|I - Z \, Z^*\|}{n \, ulp}$$

for $SSTEGR('V','A')$

$$
\begin{aligned}
r_{37} &= \frac{\max_i(\min_j(\|WA2(i) - WA3(j)\|)) + \max_i(\min_j(\|WA3(i) - WA2(j)\|))}{ulp\ \|D3\|} \\
&\text{test of } SSTEGR('N','A') \text{ versus } SSTEGR('V','A')
\end{aligned} \tag{3}
$$

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.

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.

NOTE: Test 27 is disabled at the moment because SSTEGR does not guarantee high relative accuracy. Tests 29 through 34 are disabled at present because SSTEGR does not handle partial spectrum requests.

7.6.5 Tests Performed on the Symmetric Eigenvalue Drivers

For each driver routine, the following tests will be performed:

$$
\begin{aligned}
r_1 &= \frac{\|A - USU^*\|}{n\ ulp\ \|A\|} \\
&\text{test of } SSTEV('V') \\
r_2 &= \frac{\|I - UU^*\|}{n\ ulp} \\
&\text{test of } SSTEV('V') \\
r_3 &= \frac{\|D1 - D2\|}{ulp\ \|D1\|} \\
&\text{test of } SSTEV('N') \\
r_4 &= \frac{\|A - USU^*\|}{n\ ulp\ \|A\|} \\
&\text{test of } SSTEVX('V','A') \\
r_5 &= \frac{\|I - UU^*\|}{n\ ulp} \\
&\text{test of } SSTEVX('V','A') \\
r_6 &= \frac{\|D1 - EVEIGS\|}{ulp\ \|D1\|} \\
&\text{test of } SSTEVX('N','A') \\
r_7 &= \frac{\|A - USU^*\|}{n\ ulp\ \|A\|} \\
&\text{test of } SSTEVR('V','A')
\end{aligned}
$$

$$r_8 = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSTEVR('V','A')$

$$r_9 = \frac{\|D1 - EVEIGS\|}{ulp\,\|D1\|}$$
test of $SSTEVR('N','A')$

$$r_{10} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSTEVX('V','I')$

$$r_{11} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSTEVX('V','I')$

$$r_{12} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSTEVX('N','I')$

$$r_{13} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSTEVX('V','V')$

$$r_{14} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSTEVX('V','V')$

$$r_{15} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSTEVX('N','V')$

$$r_{16} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSTEVD('V')$

$$r_{17} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSTEVD('V')$

$$r_{18} = \frac{\|D1 - EVEIGS\|}{ulp\,\|D1\|}$$
test of $SSTEVD('N')$

$$r_{19} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSTEVR('V','I')$

$$\begin{aligned}
r_{20} &= \frac{\|I - UU^*\|}{n\,ulp} \\
&\quad \text{test of } SSTEVR('V','I') \\
r_{21} &= \frac{\|D1 - D2\|}{ulp\,\|D1\|} \\
&\quad \text{test of } SSTEVR('N','I') \\
r_{22} &= \frac{\|A - USU^*\|}{n\,ulp\,\|A\|} \\
&\quad \text{test of } SSTEVR('V','V') \\
r_{23} &= \frac{\|I - UU^*\|}{n\,ulp} \\
&\quad \text{test of } SSTEVR('V','V') \\
r_{24} &= \frac{\|D1 - D2\|}{ulp\,\|D1\|} \\
&\quad \text{test of } SSTEVR('N','V') \\
r_{25} &= \frac{\|A - USU^*\|}{n\,ulp\,\|A\|} \\
&\quad \text{test of } SSYEV('L','V') \\
r_{26} &= \frac{\|I - UU^*\|}{n\,ulp} \\
&\quad \text{test of } SSYEV('L','V') \\
r_{27} &= \frac{\|D1 - D2\|}{ulp\,\|D1\|} \\
&\quad \text{test of } SSYEV('L','N') \\
r_{28} &= \frac{\|A - USU^*\|}{n\,ulp\,\|A\|} \\
&\quad \text{test of } SSYEVX('L','V','A') \\
r_{29} &= \frac{\|I - UU^*\|}{n\,ulp} \\
&\quad \text{test of } SSYEVX('L','V','A') \\
r_{30} &= \frac{\|D1 - D2\|}{ulp\,\|D1\|} \\
&\quad \text{test of } SSYEVX('L','N','A') \\
r_{31} &= \frac{\|A - USU^*\|}{n\,ulp\,\|A\|} \\
&\quad \text{test of } SSYEVX('L','V','I')
\end{aligned}$$

$$r_{32} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSYEVX('L','V','I')$

$$r_{33} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSYEVX('L','N','I')$

$$r_{34} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSYEVX('L','V','V')$

$$r_{35} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSYEVX('L','V','V')$

$$r_{36} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSYEVX('L','N','V')$

$$r_{37} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSPEV('L','V')$

$$r_{38} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSPEV('L','V')$

$$r_{39} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSPEV('L','N')$

$$r_{40} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSPEVX('L','V','A')$

$$r_{41} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSPEVX('L','V','A')$

$$r_{42} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSPEVX('L','N','A')$

$$r_{43} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSPEVX('L','V','I')$

$$r_{44} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSPEVX('L','V','I')$

$$r_{45} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSPEVX('L','N','I')$

$$r_{46} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSPEVX('L','V','V')$

$$r_{47} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSPEVX('L','V','V')$

$$r_{48} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSPEVX('L','N','V')$

$$r_{49} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSBEV('L','V')$

$$r_{50} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSBEV('L','V')$

$$r_{51} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSBEV('L','N')$

$$r_{52} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSBEVX('L','V','A')$

$$r_{53} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSBEVX('L','V','A')$

$$r_{54} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSBEVX('L','N','A')$

$$r_{55} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSBEVX('L','V','I')$

$$r_{56} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSBEVX('L','V','I')$

$$r_{57} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSBEVX('L','N','I')$

$$r_{58} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSBEVX('L','V','V')$

$$r_{59} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSBEVX('L','V','V')$

$$r_{60} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSBEVX('L','N','V')$

$$r_{61} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSYEVD('L','V')$

$$r_{62} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSYEVD('L','V')$

$$r_{63} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSYEVD('L','N')$

$$r_{64} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSPEVD('L','V')$

$$r_{65} = \frac{\|I - UU^*\|}{n\,ulp}$$
test of $SSPEVD('L','V')$

$$r_{66} = \frac{\|D1 - D2\|}{ulp\,\|D1\|}$$
test of $SSPEVD('L','N')$

$$r_{67} = \frac{\|A - USU^*\|}{n\,ulp\,\|A\|}$$
test of $SSBEVD('L','V')$

$$
\begin{aligned}
r_{68} &= \frac{\|I - UU^*\|}{n\, ulp} \\
&\text{test of } SSBEVD('L','V') \\
r_{69} &= \frac{\|D1 - D2\|}{ulp\ \|D1\|} \\
&\text{test of } SSBEVD('L','N') \\
r_{70} &= \frac{\|A - USU^*\|}{n\, ulp\ \|A\|} \\
&\text{test of } SSYEVR('L','V','A') \\
r_{71} &= \frac{\|I - UU^*\|}{n\, ulp} \\
&\text{test of } SSYEVR('L','V','A') \\
r_{72} &= \frac{\|D1 - D2\|}{ulp\ \|D1\|} \\
&\text{test of } SSYEVR('L','N','A') \\
r_{73} &= \frac{\|A - USU^*\|}{n\, ulp\ \|A\|} \\
&\text{test of } SSYEVR('L','V','I') \\
r_{74} &= \frac{\|I - UU^*\|}{n\, ulp} \\
&\text{test of } SSYEVR('L','V','I') \\
r_{75} &= \frac{\|D1 - D2\|}{ulp\ \|D1\|} \\
&\text{test of } SSYEVR('L','N','I') \\
r_{76} &= \frac{\|A - USU^*\|}{n\, ulp\ \|A\|} \\
&\text{test of } SSYEVR('L','V','V') \\
r_{77} &= \frac{\|I - UU^*\|}{n\, ulp} \\
&\text{test of } SSYEVR('L','V','V') \\
r_{78} &= \frac{\|D1 - D2\|}{ulp\ \|D1\|} \\
&\text{test of } SSYEVR('L','N','V')
\end{aligned}
\tag{4}
$$

Tests 25 through 78 are repeated (as tests 79 through 132) with UPLO='U'.

U is the matrix of eigenvectors returned when the eigenvector option is given, $D1$ and $D2$ are the eigenvalues returned with and without the eigenvector option, and ulp represents xLAMCH('P').

7.6.6 Input File for Testing the Symmetric Eigenvalue Routines and Drivers

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

```
SEP:  Data file for testing Symmetric Eigenvalue Problem routines
6                                 Number of values of N
0 1 2 3 5 50                      Values of N (dimension)
5                                 Number of values of NB
1 3  3  3 20                      Values of NB (blocksize)
2 2  2  2  2                      Values of NBMIN (minimum blocksize)
1 0  5  9  1                      Values of NX (crossover point)
25.0                              Threshold value
T                                 Put T to test the LAPACK routines
T                                 Put T to test the driver routines
T                                 Put T to test the error exits
1                                 Code to interpret the seed
SEP 21
```

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

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

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

7.6.7 Input File for Testing the Banded Symmetric Eigenvalue Routines and Drivers

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

```
SSB:  Data file for testing Symmetric Eigenvalue Problem routines
2                                 Number of values of N
5 20                              Values of N (dimension)
5                                 Number of values of K
0 1 2 5 16                        Values of K (band width)
20.0                              Threshold value
T                                 Put T to test the error exits
1                                 Code to interpret the seed
SSB 15
```

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

line 2:	The number of values of N
line 3:	The values of N, the matrix dimension
line 4:	The number of values of K
line 5:	The values of K
line 6:	The threshold value for the test ratios
line 7:	TSTERR, flag to test error exits from LAPACK and driver routines
line 8:	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 9:	If line 12 was 2, four integer values for the random number seed

The remaining lines are used to specify the matrix types for one or more sets of tests. The valid 3-character code is SSB (CSB in complex, DSB in double precision, and ZSB in complex*16).

7.7 Testing the Generalized Symmetric Eigenvalue Drivers

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

- the order N of the test matrix pair (A, B)
- 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 pair (A, B) is generated and used to test the drivers.

7.7.1 The Generalized Symmetric Eigenvalue Drivers

The driver routines for the generalized symmetric eigenvalue problem are

xSYGV/xHEGV eigenvalue/vector driver for symmetric matrices A and B,

xSYGVD/xHEGVD divide and conquer eigenvalue/vector driver for symmetric matrices A and B,

xSYGVX/xHEGVX selected eigenvalue/vector driver for symmetric matrices A and B,

xSPGV/xHPGV eigenvalue/vector driver for symmetric packed matrices A and B,

xSPGVD/xHPGVD divide and conquer eigenvalue/vector driver for symmetric packed matrices A and B,

xSPGVX/xHPGVX selected eigenvalue/vector driver for symmetric packed matrices A and B,

xSBGV/xHBGV eigenvalue/vector driver for symmetric and banded matrices A and B,

xSBGVD/xHBGVD divide and conquer eigenvalue/vector driver for symmetric and banded matrices A and B,

xSBGVX/xHBGVX selected eigenvalue/vector driver for symmetric and banded matrices A and B.

In all these drivers, B is assumed to be positive definite.

7.7.2 Test Matrices for the Generalized Symmetric Eigenvalue Drivers

Twenty-one different types of test matrices may be generated for generalized symmetric eigenvalue drivers and they are given in Table 10. These test matrices are very similar to the test matrices in Table 9 for testing the symmetric eigenvalue drivers.

7.7.3 Tests Performed on the Generalized Symmetric Eigenvalue Drivers

Finding the eigenvalues and eigenvectors of symmetric matrices A and B, where B is also positive definite, follows the same stages as the symmetric eigenvalue problem except that the problem is first reduced from generalized to standard form using xSYGST, xSPGST or xSBGST.

The test ratio

$$\frac{\|A\,Z - B\,Z\,D\|}{\|A\|\;\|Z\|\;n\,ulp}$$

is calculated for

1. calling SSYGV with ITYPE=1 and UPLO='U'
2. calling SSPGV with ITYPE=1 and UPLO='U'
3. calling SSBGV with ITYPE=1 and UPLO='U'
4. calling SSYGV with ITYPE=1 and UPLO='L'
5. calling SSPGV with ITYPE=1 and UPLO='L'

Type	Eigenvalue Distribution			
	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3	4, $6^\dagger$, $7^\ddagger$	5	
UDU^{-1}	8, $11^\dagger$, $12^\ddagger$ $16^\bullet$, $17^\star$, $18^\diamond$ 19^*, 20°, $21^\odot$	9	10	
Symmetric w/Random entries				13, $14^\dagger$, $15^\ddagger$

$\dagger$– matrix entries are $O(\sqrt{\text{overflow}})$
$\ddagger$– matrix entries are $O(\sqrt{\text{underflow}})$
$\bullet$ – Banded with $KA = 1$ and $KB = 1$
$\star$ – Banded with $KA = 2$ and $KB = 1$
$\diamond$ – Banded with $KA = 2$ and $KB = 2$
$*$ – Banded with $KA = 3$ and $KB = 1$
$\circ$ – Banded with $KA = 3$ and $KB = 2$
$\odot$ – Banded with $KA = 3$ and $KB = 3$

Table 10: Test matrices for the symmetric eigenvalue drivers

6. calling SSBGV with ITYPE=1 and UPLO='L'
7. calling SSYGV with ITYPE=2 and UPLO='U'
8. calling SSPGV with ITYPE=2 and UPLO='U'
9. calling SSYGV with ITYPE=2 and UPLO='L'
10. calling SSPGV with ITYPE=2 and UPLO='L'
11. calling SSYGV with ITYPE=3 and UPLO='U'
12. calling SSPGV with ITYPE=3 and UPLO='U'
13. calling SSYGV with ITYPE=3 and UPLO='L'
14. calling SSPGV with ITYPE=3 and UPLO='L'

For the divide and conquer drivers: SSYGVD, SSPGVD and SSBGVD, all above 14 tests are also applied, where SSYGV, SSPGV and SSBGV are replaced by SSYGVD, SSPGVD and SSBGVD, respectively.

For the selected eigenvalue/vector drivers: SSYGVX, SSPGVX and SSBGVX, these 14 tests are applied in association with the parameter RANGE = 'A', 'N' and 'I', respectively, where SSYGV, SSPGV and SSBGV are replaced by SSYGVX, SSPGVX and SSBGVX, respectively.

7.7.4 Input File for Testing the Generalized Symmetric Eigenvalue Drivers

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

```
SEP:  Data file for testing Symmetric Eigenvalue Problem routines
7                                  Number of values of N
0 1 2 3 5 10 16                    Values of N (dimension)
3                                  Number of values of NB, NBMIN, NX
1 3 20                             Values of NB (blocksize)
2 2  2                             Values of NBMIN (minimum blocksize)
1 1  1                             Values of NX (crossover point)
20.0                               Threshold value
T                                  Put T to test the LAPACK routines
T                                  Put T to test the driver routines
T                                  Put T to test the error exits
1                                  Code to interpret the seed
SSG 21
```

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

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

The remaining lines are used to specify the matrix types for one or more sets of tests, as in the symmetric case. The valid 3-character code is SSG (CSG in complex, DSG in double precision, and ZSG in complex*16).

7.8 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.

7.8.1 The Singular Value Decomposition Drivers

The driver routines for the singular value decomposition are

xGESVD singular value decomposition of A

xGESDD singular value decomposition of A using divide-and-conquer

7.8.2 Test Matrices for the Singular Value Decomposition Routines

Sixteen different types of test matrices may be generated for the singular value decomposition routines. Table 11 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†, 7‡	4	5	
UDV	8, 11†, 12‡	9	10	
Random entries				13, 14†, 15‡
Random bidiagonal				16

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

Table 11: 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)-by-min(M, N) identity matrix with zero rows or columns added to the bottom or right to make it M-by-N

UDV: Real M-by-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 (or xBDSDC) 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).

7.8.3 Test Matrices for the Banded Singular Value Decomposition Routines

Fifteen different types of test matrices may be generated for the banded singular value decomposition routines. These test matrices are the same as the first fifteen test matrices in Table 11.

7.8.4 Test Matrices for the Singular Value Decomposition Drivers

Five different types of test matrices may be generated for the singular value decomposition drivers. Table 12 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
UDV	3, $4^{\dagger}$, $5^{\ddagger}$				

†– matrix entries are multiplied by the underflow-threshold/ε

†– matrix entries are multiplied by the overflow-threshold * ε

Table 12: Test matrices for the singular value decomposition drivers

7.8.5 Tests Performed on the Singular Value Decomposition Routines

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

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

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

A	m-by-n
Q	m-by-$\tilde{n}$ (but m-by-m if nrhs > 0)
P	$\tilde{n}$-by-n
B	$\tilde{n}$-by-$\tilde{n}$
U, V	$\tilde{n}$-by-$\tilde{n}$
$S1$, $S2$	diagonal, order $\tilde{n}$
X	m-by-$nrhs$

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

To check these calculations, the following test ratios are computed, where $PT = P^*$ and $VT = V^*$. Tests 1–3 test SGEBRD and SORGBR. Tests 4–10 test SBDSQR on a bidiagonal matrix B. Tests 11–14 test SBDSQR on a matrix A. Tests 15–19 test SBDSDC on a bidiagonal matrix B.

$$r_1 = \frac{\|A - QBPT\|}{\max(m, \tilde{n})\varepsilon \, \|A\|} \qquad r_2 = \frac{\|I - Q^*Q\|}{m\varepsilon}$$

$$r_3 = \frac{\|I - PT\,PT^*\|}{n\varepsilon} \qquad r_4 = \frac{\|B - U\Sigma VT\|}{\min(m, \tilde{n})\varepsilon \, \|B\|}$$

$$r_5 = \frac{\|Y - UZ\|}{\max(\min(m, \tilde{n}), k)\varepsilon \, \|Y\|}, \quad \text{where } Y = Q^*X \text{ and } Z = U^*Y.$$

$$r_6 = \frac{\|I - U^*U\|}{\min(m, \tilde{n})\varepsilon} \qquad r_7 = \frac{\|I - VV^*\|}{\min(m, \tilde{n})\varepsilon}$$

$$r_8 = \begin{cases} 0 & \text{if } S1 \text{ contains } \tilde{n} \text{ nonnegative values in decreasing order.} \\ \frac{1}{\varepsilon} & \text{otherwise} \end{cases}$$

$$r_9 = \frac{\|S1 - S2\|}{\varepsilon \, \|S1\|}$$

$$r_{10} = \begin{cases} 0 & \text{if eigenvalues of } B \text{ are within } THRESH \text{ of those in } S1. \\ 2 * THRESH & \text{otherwise} \end{cases}$$

$$r_{11} = \frac{\|A - (QU)\Sigma(VT\,PT)\|}{\max(m, \tilde{n})\varepsilon \, \|A\|} \qquad r_{12} = \frac{\|X - (QU)Z\|}{\max(m, k)\varepsilon \, \|X\|}$$

$$r_{13} = \frac{\|I - (QU)^*(QU)\|}{m\varepsilon} \qquad r_{14} = \frac{\|I - (VT\,PT)(PT^*\,VT^*)\|}{n\varepsilon}$$

Tests 15–19 are the same as tests 4, 6, 7, 8, and 9, respectively, except that SBDSDC is tested. 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.

7.8.6 Tests Performed on the Banded Singular Value Decomposition Routines

Testing the reduction of a general m-by-n band matrix A to bidiagonal form is done in the following stages:

1. A is factored as QBP^*, where Q and P are orthogonal and B is upper bidiagonal.
2. A given matrix C is overwritten with $Q{*}C$.

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

A	m-by-n
Q	m-by-$\tilde{n}$ (but m-by-m if nrhs > 0)
P	$\tilde{n}$-by-n
B	$\tilde{n}$-by-$\tilde{n}$
C	m-by-$nrhs$

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

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

$$r_1 = \frac{\|A - QBP^*\|}{\|A\|\; max(m,n)\; ulp} \qquad r_2 = \frac{\|I - Q^*Q\|}{m\; ulp}$$

$$r_3 = \frac{\|I - P^*P\|}{n\; ulp} \qquad r_4 = \frac{\|Y - Q^*C\|}{\|Y\|\; max(m, nrhs)\; ulp}, \text{ where } Y = Q^*C.$$

7.8.7 Tests Performed on the Singular Value Decomposition Drivers

For the driver routines, the following tests are computed:

$$r_1 = \frac{\|A - U\text{diag}(S)VT\|}{\|A\|\max(M,N)\varepsilon}$$

$$r_2 = \frac{\left\|I - U^TU\right\|}{M\varepsilon}$$

$$r_3 = \frac{\left\|I - VT(VT)^T\right\|}{N\varepsilon}$$

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

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

$$r_6 = \frac{\|VT - VT_p\|}{N\varepsilon}, \text{ where } VT_p \text{ is a partially computed } VT.$$
$$r_7 = \frac{\|S - S_p\|}{MNMIN\varepsilon\,\|S\|}, \text{ where } S_p \text{ is the vector of singular values from the partial SVD}$$

Tests 1–7 test xGESVD, and tests 8–14 are the same as tests 1–7 except that they test xGESDD.

7.8.8 Input File for Testing the Singular Value Decomposition Routines

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

```
SVD:  Data file for testing Singular Value Decomposition routines
19                                                Number of values of M
0 0 0 1 1 1 2 2 3 3 3 10 10 16 16 30 30 50 50 Values of M
0 1 3 0 1 2 0 1 0 1 3 10 16 10 16 30 50 30 50 Values of N
5                                                 Number of parameter values
1 3  3  3 20                                      Values of NB (blocksize)
2 2  2  2  2                                      Values of NBMIN (minimum blocksize)
1 0  5  9  1                                      Values of NX (crossover point)
2 0  2  2  2                                      Values of NRHS
35.0                                              Threshold value
T                                                 Put T to test the LAPACK routines
T                                                 Put T to test the driver routines
T                                                 Put T to test the error exits
1                                                 Code to interpret the seed
SVD 16
```

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

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

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

7.8.9 Input File for Testing the Banded Singular Value Decomposition Routines

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

```
SBB:  Data file for testing banded Singular Value Decomposition routines
20                                 Number of values of M
0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 10  10  16  16    Values of M
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 10  16  10  16    Values of N
5                                  Number of values of K
0 1 2 3 16                         Values of K (band width)
2                                  Number of values of NRHS
1 2                                Values of NRHS
20.0                               Threshold value
F                                  Put T to test the error exits
1                                  Code to interpret the seed
SBB 15
```

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

line 2:	The number of values of 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 K
line 6:	The values of K, the bandwidth
line 7:	The number of values of NRHS
line 8:	The values of NRHS, the number of right hand sides
line 9:	The threshold value for the test ratios
line 10:	TSTERR, the flag to test error exits
line 11:	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 12:	If line 14 was 2, four integer values for the random number seed

The remaining lines are used to specify the matrix types for the set of tests. The valid 3-character code is `SBB` (`CBB` in complex, `DBB` in double precision, and `ZBB` in complex*16).

7.9 Testing the Generalized Singular Value Decomposition Driver

The driver routine for the generalized singular value decomposition is

xGGSVD computes the generalized singular value decomposition of matrices A and B

The test routine for this driver has the following parameters which may be varied:

- the number of rows M of the test matrix A
- the number of rows P of the test matrix B
- the number of columns N of the test matrices A and B
- the number of matrix types to be tested

The test program thus consists of a doubly-nested loop, the outer one over ordered triples $(\mathrm{M}, \mathrm{P}, \mathrm{N})$, and the inner one over matrix types. On each iteration of the innermost loop, matrices A and B are generated and used to test the GSVD routines.

Please note that the block size NB is not an input parameter since at the present time no blocked version of GSVD exists.

7.9.1 Test Matrices for the Generalized Singular Value Decomposition Driver

Eight different test matrix combinations are used for the GSV test paths. All are generated with a predetermined condition number. The following test matrices are used:

NTYPES	Matrix A	Matrix B	$\|\|A\|\|$	$\|\|B\|\|$	$\kappa(A)$	$\kappa(B)$
1	Diagonal	Upper triangular	10	1000	100	10
2	Upper triangular	Upper triangular	10	1000	100	10
3	Lower triangular	Upper triangular	10	1000	100	10
4	Random dense	Random dense	10	1000	100	10
5	Random dense	Random dense	10	1000	$\sqrt{0.1/\varepsilon}$	$\sqrt{0.1/\varepsilon}$
6	Random dense	Random dense	10	1000	$0.1/\varepsilon$	$0.1/\varepsilon$
7	Random dense	Random dense	10	1000	$\sqrt{0.1/\varepsilon}$	$0.1/\varepsilon$
8	Random dense	Random dense	10	1000	$0.1/\varepsilon$	$\sqrt{0.1/\varepsilon}$

7.9.2 Tests Performed on the Generalized Singular Value Decomposition Driver

For each set of matrix dimensions (M, N, P) and each selected matrix type, an m-by-n matrix A and a p-by-n matrix B are generated. The problem dimensions are as follows:

A	m-by-n
B	p-by-n
Q	n-by-n
U	m-by-m
V	p-by-p

The tests for the GSV path are as follows:

- Compute the Generalized Singular Value Decomposition using xGGSVD, and compute the test ratios

$$\begin{aligned}
r_1 &= \frac{\left\|U^H AQ - D1\,R\right\|}{\|A\|\max(M,N)ulp} \\
r_2 &= \frac{\left\|V^H BQ - D2\,R\right\|}{\|B\|\max(P,N)ulp} \\
r_3 &= \frac{\left\|I - U^H U\right\|}{M\,ulp} \\
r_4 &= \frac{\left\|I - V^H V\right\|}{P\,ulp} \\
r_5 &= \frac{\left\|I - Q^H Q\right\|}{N\,ulp} \\
r_6 &= \begin{cases} 0 & \text{if } ALPHA \text{ is in decreasing order.} \\ \frac{1}{ulp} & \text{otherwise} \end{cases}
\end{aligned} \tag{5}$$

where $D1$ and $D2$ are "diagonal" matrices, and form the generalized singular pairs of the matrices A and B, and ulp represents xLAMCH('P').

7.9.3 Input File for Testing the Generalized Singular Value Decomposition Driver

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

```
GSV:  Data file for testing Generalized SVD routines
8                       Number of values of M, P, N
0  5  9  10 20 12 12 40 Values of M (row dimension)
4  0  12 14 10 10 20 15 Values of P (row dimension)
3  10 15 12  8 20 8  20 Values of N (column dimension)
40.0                    Threshold value of test ratio.
T                       Put T to test the error exits
1                       Code to interpret the seed
GSV 8                   List matrix types on next line if 0 < NTYPES < 8
```

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

line 2:	The number of values M, P, and N
line 3:	Values of M (row dimension)
line 4:	Values of P (row dimension)
line 5:	Values of N (column dimension)
line 6:	THRESH, the threshold value for the test ratios
line 7:	TSTERR, flag to test the error exits
line 8:	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 9 :	If line 8 was 2, four integer values for the random number seed
	Otherwise, the path GSV followed by the number of matrix types NTYPES
line 10:	If NTYPES < 8, then specifies matrix types to be tested.

7.10 Testing the Generalized QR and RQ Factorization Routines

The test routine for the GQR and GRQ factorization routines has the following parameters which may be varied:

- the values of M of the test matrix A
- the values of P of the test matrix B
- the number of columns or rows N of the test matrices A and B
- the number of matrix types to be tested

The test program thus consists of a doubly-nested loop, the outer one over ordered triples $(\mathrm{M}, \mathrm{P}, \mathrm{N})$, and the inner one over matrix types. On each iteration of the innermost loop, matrices A and B are generated and used to test the GQR and GRQ routines.

Please note that the block size NB is not an input test parameter since the GQR and GRQ factorizations are implemented by calling the QR and RQ factorizations which have been tested for the parameter block size NB.

7.10.1 Test Matrices for the Generalized QR and RQ Factorization Routines

Eight different test matrix combinations are used for the GQR and GRQ test paths. All are generated with a predetermined condition number. For the GQR path, the following test matrices are used:

NTYPES	Matrix A	Matrix B	$\|\|A\|\|$	$\|\|B\|\|$	$\kappa(A)$	$\kappa(B)$
1	Diagonal	Lower triangular	10	1000	100	10
2	Lower triangular	Diagonal	10	1000	100	10
3	Lower triangular	Upper triangular	10	1000	100	10
4	Random dense	Random dense	10	1000	100	10
5	Random dense	Random dense	10	1000	$\sqrt{0.1/\varepsilon}$	$\sqrt{0.1/\varepsilon}$
6	Random dense	Random dense	10	1000	$0.1/\varepsilon$	$0.1/\varepsilon$
7	Random dense	Random dense	†	‡	$\sqrt{0.1/\varepsilon}$	$0.1/\varepsilon$
8	Random dense	Random dense	‡	†	$0.1/\varepsilon$	$\sqrt{0.1/\varepsilon}$

†– near underflow threshold
‡– near overflow threshold

For the GRQ path, the following test matrices are used:

NTYPES	Matrix A	Matrix B	$\|\|A\|\|$	$\|\|B\|\|$	$\kappa(A)$	$\kappa(B)$
1	Diagonal	Upper triangular	10	1000	100	10
2	Upper triangular	Upper triangular	10	1000	100	10
3	Lower triangular	Upper triangular	10	1000	100	10
4	Random dense	Random dense	10	1000	100	10
5	Random dense	Random dense	10	1000	$\sqrt{0.1/\varepsilon}$	$\sqrt{0.1/\varepsilon}$
6	Random dense	Random dense	10	1000	$0.1/\varepsilon$	$0.1/\varepsilon$
7	Random dense	Random dense	†	‡	$\sqrt{0.1/\varepsilon}$	$0.1/\varepsilon$
8	Random dense	Random dense	‡	†	$0.1/\varepsilon$	$\sqrt{0.1/\varepsilon}$

†– near underflow threshold
‡– near overflow threshold

7.10.2 Tests Performed on the Generalized QR and RQ Factorization Routines

For the GQR test path, and each set of matrix dimensions (M, N, P) and each selected matrix type, an n-by-m matrix A and an n-by-p matrix B are generated. The problem dimensions are as follows:

A	n-by-m
B	n-by-p
Q	n-by-n
Z	p-by-p

The tests for the GQR path are as follows:

- Compute the Generalized QR factorization using xGGQRF, generate the orthogonal matrix Q from the Householder vectors using xORGQR, generate the matrix Z using xORGRQ, and compute the test ratios

  1. $||R - Q^H A||/(max(m,n)\,||A||\,ulp)$
  2. $||TZ - Q^H B||/(max(p,n)\,||B||\,ulp)$
  3. $||I - Q^H Q||/(m\,ulp)$
  4. $||I - Z^H Z||/(p\,ulp)$

where ulp represents xLAMCH('P').

For the GRQ test path, and each set of matrix dimensions (M, N, P) and each selected matrix type, an m-by-n matrix A and a p-by-n matrix B are generated. The problem dimensions are as follows:

A	m-by-n
B	p-by-n
Q	n-by-n
Z	p-by-p

The tests for the GRQ path are as follows:

- Compute the Generalized RQ factorization using xGGRQF, generate the orthogonal matrix Q from the Householder vectors using xORGRQ, generate the matrix Z from the Householder vectors using xORGQR, and compute the test ratios

  1. $||R - AQ^H||/(max(m,n)\,||A||\,ulp)$
  2. $||TQ - Z^H B||/(max(p,n)\,||A||\,ulp)$
  3. $||I - Q^H Q||/(n\,ulp)$
  4. $||I - Z^H Z||/(p\,ulp)$

where ulp represents xLAMCH('P').

7.10.3 Input File for Testing the Generalized QR and RQ Factorization Routines

An annotated example of an input file for testing the generalized QR and RQ factorization routines is shown below.

```
GQR:  Data file for testing Generalized QR and RQ routines
3                   Number of values of M, P and N
0 3 10              Values of M
0 5 20              Values of P
0 3 30              Values of N
20.0                Threshold value of test ratio.
T                   Put T to test the error exits
1                   Code to interpret the seed
GQR 8               List matrix types on next line if 0 < NTYPES < 8
```

The first line of the input file must contain the characters GQR or GRQ 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, P and N
line 3:	The values of M
line 4:	The values of P
line 5:	The values of N
line 6:	The threshold value for the test ratios
line 7:	TSTERR, flag to test the error exits
line 8:	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 9 :	If line 8 was 2, four integer values for the random number seed
	Otherwise, the path GQR or GRQ followed by the number of matrix types NTYPES
line 10:	If NTYPES < 8, then specifies matrix types to be tested.

7.11 Testing the Generalized Linear Regression Model Driver

The driver routine for the generalized linear regression model is

xGGGLM solves generalized linear regression model problem using the generalized QR factorization

The test routine for this driver has the following parameters which may be varied:

- the number of rows M of the test matrix A
- the number of rows P of the test matrix B
- the number of columns N of the test matrices A and B
- the number of matrix types to be tested

The test program thus consists of a doubly-nested loop, the outer one over ordered triples $(\mathrm{M}, \mathrm{P}, \mathrm{N})$, and the inner one over matrix types. On each iteration of the innermost loop, matrices A and B are generated and used to test the GLM driver routine.

Please note that the block size NB is not an input test parameter since the GLM problem is solved by calling GQR factorization. The GQR is implemented by calling the QR and RQ factorizations which have been tested for the parameter block size NB.

7.11.1 Test Matrices for the Generalized Linear Regression Model Driver

Eight different test matrix combinations are used for the GLM test path. All are generated with a predetermined condition number. The following test matrices are used:

Please note that the current version of the GLM driver only addresses well-conditioned problems (like xGELS does). Therefore, we do not test the code with ill-conditioned matrices.

NTYPES	Matrix A	Matrix B	$\|\|A\|\|$	$\|\|B\|\|$	$\kappa(A)$	$\kappa(B)$
1	Diagonal	Lower triangular	10	1000	100	10
2	Lower triangular	Diagonal	10	1000	100	10
3	Lower triangular	Upper triangular	10	1000	100	10
4	Random dense	Random dense	10	1000	100	10
5	Random dense	Random dense	10	1000	100	10
6	Random dense	Random dense	10	1000	100	10
7	Random dense	Random dense	10	1000	100	10
8	Random dense	Random dense	10	1000	100	10

7.11.2 Tests Performed on the Generalized Linear Regression Model Driver

For each set of matrix dimensions (M, N, P) and each selected matrix type, an n-by-m matrix A and an n-by-p matrix B are generated.

The test for the GLM path is as follows:

- Solve the Generalized Linear Regression Model problem using xGGGLM, and compute the test ratio

  1. $||d - Ax - Bu|| / ((||A|| + ||B||)\,(||x|| + ||u||)\,\varepsilon)$

where d is the left hand side vector of length n, u is the solution vector of length p, and x is the solution vector of length m.

7.11.3 Input File for Testing the Generalized Linear Regression Model Driver

An annotated example of an input file for testing the generalized linear regression model driver is shown below.

```
GLM:  Data file for testing Generalized Linear Regression Model routines
6                          Number of values of NN
0  5  8  15 20 40          Values of M (row dimension),
9  0  15 12 15 30          Values of P (row dimension),
5  5  10 25 30 50          Values of N (column dimension)  M <= N <= M+P
20.10                      Threshold value of test ratio.
T                          Put T to test the error exits
1                          Code to interpret the seed
GLM 8                      List matrix types on next line if 0 < NTYPES < 8
```

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

line 2:	The number of values M, P, and N
line 3:	Values of M (row dimension)
line 4:	Values of P (row dimension)
line 5:	Values of N (column dimension), note M $\leq$ N $\leq$ M+P
line 6:	THRESH, the threshold value for the test ratios
line 7:	TSTERR, flag to test the error exits
line 8:	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 9 :	If line 8 was 2, four integer values for the random number seed
	Otherwise, the path GLM followed by the number of matrix types NTYPES
line 10:	If NTYPES < 8, then specifies matrix types to be tested.

7.12 Testing the Constrained Linear Least Squares Driver

The driver routine for the constrained linear least squares problem is

xGGLSE solves the constrained linear least squares problem using the generalized RQ factorization

The test routine for this driver has the following parameters which may be varied:

- the number of rows M of the test matrix A
- the number of rows P of the test matrix B
- the number of columns N of the test matrices A and B
- the number of matrix types to be tested

The test program thus consists of a doubly-nested loop, the outer one over ordered triples $(\mathrm{M}, \mathrm{P}, \mathrm{N})$, and the inner one over matrix types. On each iteration of the innermost loop, matrices A and B are generated and used to test the LSE driver routine.

Please note that the block size NB is not an input test parameter since the LSE problem is solved by calling GRQ factorization. The GQR is implemented by calling the QR and RQ factorizations which have been tested for the parameter block size NB.

7.12.1 Test Matrices for the Constrained Linear Least Squares Driver

Eight different test matrix combinations are used for the LSE test path. All are generated with a predetermined condition number. The following test matrices are used:

Please note that the current version of the LSE driver only addresses well-conditioned problems (like xGELS does). Therefore, we do not test the code with ill-conditioned matrices.

NTYPES	Matrix A	Matrix B	$\|\|A\|\|$	$\|\|B\|\|$	$\kappa(A)$	$\kappa(B)$
1	Diagonal	Upper triangular	10	1000	100	10
2	Upper triangular	Upper triangular	10	1000	100	10
3	Lower triangular	Upper triangular	10	1000	100	10
4	Random dense	Random dense	10	1000	100	10
5	Random dense	Random dense	10	1000	100	10
6	Random dense	Random dense	10	1000	100	10
7	Random dense	Random dense	10	1000	100	10
8	Random dense	Random dense	10	1000	100	10

7.12.2 Tests Performed on the Constrained Linear Least Squares Driver

For each set of matrix dimensions (M, N, P) and each selected matrix type, an m-by-n matrix A and an p-by-n matrix B are generated.

The tests for the LSE path are as follows:

- Solve the Constrained Linear Least Squares problem using xGGLSE, and compute the test ratio

  1. $||Ax - c||/(||A||\,||x||\,\varepsilon)$
  2. $||Bx - d||/(||B||\,||x||\,\varepsilon)$

where x is the solution vector of length n, c is the right hand side vector of the least squares part of length m, and d is the right hand side vector for the constrained equation of length p.

7.12.3 Input File for Testing the Constrained Linear Least Squares Driver

An annotated example of an input file for testing the constrained linear least squares driver is shown below.

```
LSE:  Data file for testing Constrained Linear Least Squares routines
6                    Number of values of NN
6  0  5  8  10 30    Values of M
0  5  5  5  8  20    Values of P
5  5  6  8  12 45    Values of N,  note P <= N <= P+M
20.1                 Threshold value of test ratio.
T                    Put T to test the error exits
1                    Code to interpret the seed
LSE 8                List matrix types on next line if 0 < NTYPES < 8
```

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

line 2:	The number of values M, P, and N
line 3:	Values of M
line 4:	Values of P
line 5:	Values of N, note $P \leq N \leq P+M$
line 6:	THRESH, the threshold value for the test ratios
line 7:	TSTERR, flag to test the error exits
line 8:	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 9 :	If line 8 was 2, four integer values for the random number seed
	Otherwise, the path LSE followed by the number of matrix types NTYPES
line 10:	If NTYPES < 8, then specifies matrix types to be tested.

8 More About Timing

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

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

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

8.1 The Linear Equation Timing Program

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

- M, the matrix row dimension
- N, the matrix column dimension, or the half-bandwidth for the band routines
- K, the number of right-hand sides for the linear solvers, or the third dimension for the Level 3 BLAS
- NB, the block size for the blocked routines, or INCX for the Level 2 BLAS
- NX, the crossover point, the point in a block algorithm at which we switch to an unblocked algorithm

- LDA, the leading dimension of the dense and banded matrices.

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

The number and size of the input values are limited by certain program 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 $(\text{LDA} + K)M \leq 3(\text{LDAMAX})(\text{NMAX})$, where $\text{LDA} \geq 3K + 1$, which restricts the value of K. These limits allow K to be as big as 200 for M = 1000. For the xPB and xTB paths, the condition is $(2K + 1)M \leq 3(\text{NMAX})(\text{LDAMAX})$.

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

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

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

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

The paths xGT, xPT, xHR and xTD include timing of the equivalent LINPACK solvers or EISPACK reductions for comparison.

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

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

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

```
LAPACK timing, REAL square matrices
5                                 Number of values of M
10 20 40 60 80                    Values of M (row dimension)
5                                 Number of values of N
10 20 40 60 80                    Values of N (column dimension)
2                                 Number of values of K
20 80                             Values of K
2                                 Number of values of NB
1  8                              Values of NB (blocksize)
0  8                              Values of NX (crossover point)
1                                 Number of values of LDA
81                                Values of LDA (leading dimension)
0.05                              Minimum time in seconds
SGE    T T T
SPO    T T T
SPP    T T T
SSY    T T T
SSP    T T T
STR    T T
STP    T T
SQR    T T T
SLQ    T T T
SQL    T T T
SRQ    T T T
SQP    T
```

```
SHR    T T T T
STD    T T T T
SBR    T T T
SLS    T T T T T T
```

The first 13 lines of the input file are read using list-directed input and are used to specify the values of M, N, K, NB, NX, LDA, and 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 14:

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

```
LAPACK timing, REAL band matrices
1                                Number of values of M
200                              Values of M (row dimension)
5                                Number of values of K
10 20 30 40 50                   Values of K (bandwidth)
4                                Number of values of NRHS
1 2 16 100                       Values of NRHS (the number of right-hand sides)
2                                Number of values of NB
1  8                             Values of NB (blocksize)
0  8                             Values of NX (crossover point)
1                                Number of values of LDA
152                              Values of LDA (leading dimension)
0.05                             Minimum time in seconds
BAND                             Time sample banded BLAS
SGB
SPB
STB
```

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

We also provide a separate input file for timing the orthogonal factorization and reduction routines that operate on rectangular matrices. For these routines, the values of M and N are specified in ordered pairs (M, N). An annotated example of an input file for timing the REAL linear equation routines that operate on dense rectangular matrices is shown below. The input file is read in the same way as the one for dense square matrices.

```
LAPACK timing, REAL rectangular matrices
7                                 Number of values of M
20 40 20 40 80 40 80              Values of M (row dimension)
7                                 Number of values of N
20 20 40 40 40 80 80              Values of N (column dimension)
4                                 Number of values of K
1 2 16 100                        Values of K
2                                 Number of values of NB
1  8                              Values of NB (blocksize)
0  8                              Values of NX (crossover point)
1                                 Number of values of LDA
81                                Values of LDA (leading dimension)
0.05                              Minimum time in seconds
none
SQR    T T T
SLQ    T T T
SQL    T T T
SRQ    T T T
SQP    T
SBR    T T F
```

8.2 Timing the Level 2 and 3 BLAS

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

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

```
BLAS timing, REAL data, K small
5                           Number of values of M
10 20 40 60 80              Values of M
5                           Number of values of N
10 20 40 60 80              Values of N
2                           Number of values of K
2 16                        Values of K
1                           Number of values of INCX
1                           Values of INCX
1                           Number of values of LDA
81                          Values of LDA
0.05                        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.

8.3 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 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(V) (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(V) plus STREVC(R))
12. INVIT (EISPACK computation of the right eigenvectors of an upper Hessenberg matrix using inverse iteration, to be compared to SHSEIN(R)).

Eight different matrix types are provided for timing the nonsymmetric eigenvalue routines. A variety of matrix types is allowed because the number of iterations to compute the eigenvalues, and hence the timing, can depend on the type of matrix whose eigendecomposition is desired. The matrices used for timing are of the form XTX^{-1} where X is either orthogonal (for types 1–4) or random with condition number $1/\sqrt{\varepsilon}$ (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
10 20 30 40                      Values of N (dimension)
4                                Number of values of parameters
1   1   1   1                    Values of NB (blocksize)
2   4   6   2                    Values of NS (number of shifts)
12  12  12  50                   Values of MAXB (multishift crossover pt)
81  81  81  81                   Values of LDA (leading dimension)
0.05                             Minimum time in seconds
```

```
4                               Number of matrix types
1 3 4 6
SHS    T T T T T T T T T T T T
```

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

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

If $0 < \text{NTYPES} < 8$, 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 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.

8.4 Timing the Generalized Nonsymmetric Eigenproblem

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

- N, the matrix size,
- LDA, the leading dimension,
- the test matrix types,

- the routines or sequences of routines from LAPACK or EISPACK to be timed.

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	400
LDAMAX	Maximum value for LDA	420
MAXIN	Maximum number of values of N	12
MAXPRM	Maximum number of values of LDA LDA	10

The computations that may be timed for the REAL version are

1. SGGHRD(N) (LAPACK reduction to generalized upper Hessenberg form, without computing U or V, including a call to SGEQRF and SORMQR to reduce B to upper triangular form.)
2. SGGHRD(Q) (LAPACK reduction to generalized upper Hessenberg form, computing U but not V, including a call to SGEQRF, SORGQR, and SORMQR to reduce B to upper triangular form.)
3. SGGHRD(Z) (LAPACK reduction to generalized upper Hessenberg form, computing V but not U, including a call to SGEQRF and SORMQR to reduce B to upper triangular form.)
4. SGGHRD(Q,Z) (LAPACK reduction to generalized upper Hessenberg form, computing U and V, including a call to SGEQRF, SORGQR, and SORMQR to reduce B to upper triangular form.)
5. SHGEQZ(E) (LAPACK computation of generalized eigenvalues only of a pair of matrices in generalized Hessenberg form)
6. SHGEQZ(S) (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form)
7. SHGEQZ(Q) (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Q)
8. SHGEQZ(Z) (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Z)
9. SHGEQZ(Q,Z) (LAPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Q and Z)
10. STGEVC(L,A) (LAPACK computation of the the left generalized eigenvectors of a matrix pair in generalized Schur form)
11. STGEVC(L,B) (LAPACK computation of the the left generalized eigenvectors of a matrix pair in generalized Schur form, back transformed by Q)

12. STGEVC(R,A) (LAPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form)
13. STGEVC(R,B) (LAPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form, back transformed by Z)
14. QZHES(F) (EISPACK reduction to generalized upper Hessenberg form, with MATZ =.FALSE., so V is not computed.)
15. QZHES(T) (EISPACK reduction to generalized upper Hessenberg form, with MATZ =.TRUE., so V is computed.)
16. QZIT(F) (QZIT followed by QZVAL with MATZ=.FALSE.: EISPACK computation of generalized eigenvalues only of a pair of matrices in generalized Hessenberg form)
17. QZIT(T) (QZIT followed by QZVAL with MATZ=.TRUE.: EISPACK computation of generalized Schur form of a pair of matrices in generalized Hessenberg form and Z)
18. QZVEC (EISPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form, back transformed by Z)

Note that SGGHRD is timed along with the QR routines that reduce B to upper-triangular form; this is to allow a fair comparison with the EISPACK routine QZHES.

Four different matrix types are provided for timing the generalized 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 have at least one zero, one infinite, and one singular ($\alpha = \beta = 0$) generalized eigenvalue. The remaining eigenvalues are sometimes real and sometimes complex, distributed in magnitude as follows:

- "clustered" entries $1, \varepsilon, \ldots, \varepsilon$ with random signs;
- evenly spaced entries from 1 down to ε with random signs;
- geometrically spaced entries from 1 down to ε with random signs;
- eigenvalues randomly chosen from the interval $(\varepsilon, 1)$.

8.4.1 Input File for Timing the Generalized Nonsymmetric Eigenproblem

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

```
GEP:  Data file for timing Generalized Nonsymmetric Eigenvalue Problem
4                                Number of values of N
50 100 150 200                   Values of N (dimension)
4                                Number of parameter values
1   10   1  10                   Values of NB (blocksize -- used by SGEQRF, etc.)
```

```
201 201 200 200                 Values of LDA (leading dimension)
0.0                             Minimum time in seconds
5                               Number of matrix types
SHG   T T T T T T T T T T T T T T T T T T
```

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

line 2:	The number of values of N
line 3:	The values of N, the matrix dimension
line 4:	Number of values of the parameters
line 5:	The values for NB, the blocksize
line 6:	The values for the leading dimension LDA
line 7:	The minimum time (in seconds) that a subroutine will be timed. If TIMMIN is zero, each routine should be timed only once.
line 8:	NTYPES, the number of matrix types to be used

If NTYPES $>= 4$, all the types are 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 routines to be timed. For the generalized nonsymmetric eigenvalue problem, the path names for the four data types are `SHG`, `CHG`, `DHG`, and `ZHG`. A line to request all the routines in the REAL path has the form

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

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

```
SHG
```

is equivalent to the line above.

The output is in the form of a table which shows the absolute times in seconds, floating point operation counts, and megaflop rates for each routine over all relevant input parameters. For the timings of SGGHRD plus appropriate QR routines, the table has one line for each different combination of LDA and NB. For other routines, the table has one line for each distinct value of LDA.

8.5 Timing the Symmetric and Generalized 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 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. SORGTR (LAPACK generation of orthogonal martrix as returned by SSYTRD)
3. SORMTR (LAPACK multiplication of orthogonal matrix as returned by SSYTRD)
4. SSTEQR(N) (LAPACK computation of eigenvalues only of a symmetric tridiagonal matrix)
5. SSTEQR(V) (LAPACK computation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix)
6. SSTERF (LAPACK computation of the eigenvalues only of a symmetric tridiagonal matrix using a square-root free algorithm)
7. SPTEQR(COMPZ='N') (LAPACK computation of the eigenvalues of a symmetric positive definite tridiagonal matrix)
8. SPTEQR(COMPZ='V') (LAPACK computation of the eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix)
9. SSTEBZ(RANGE='I') (LAPACK computation of the eigenvalues in a specified interval for a symmetric tridiagonal matrix)
10. SSTEBZ(RANGE='V') (LAPACK computation of the eigenvalues in a half-open interval for a symmetric tridiagonal matrix)
11. SSTEIN (LAPACK computation of the eigenvectors of a symmetric tridiagonal matrix corresponding to specified eigenvalues using inverse iteration)

12. SSTEDC(COMPQ='N')
13. SSTEDC(COMPQ='I')
14. SSTEDC(COMPQ='V')
15. SSTEGR(COMPQ='N')
16. SSTEGR(COMPQ='V')
17. TRED1 (EISPACK reduction to symmetric tridiagonal form, to be compared to SSYTRD)
18. IMTQL1 (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to SSTEQR(N))
19. IMTQL2 (EISPACK computation of eigenvalues and eigenvectors of a symmetric tridiagonal matrix, to be compared to SSTEQR(V))
20. TQLRAT (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to SSTERF).
21. TRIDIB (EISPACK computation of the eigenvalues of)(compare with SSTEBZ – RANGE='I')
22. BISECT (EISPACK computation of the eigenvalues of)(compare with SSTEBZ – RANGE='V')
23. TINVIT (EISPACK computation of the eigenvectors of a triangular matrix using inverse iteration) (compare with SSTEIN)

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 symmetric tridiagonal matrix)
3. CUNGTR+CSTEQR(V) (LAPACK computation of the eigenvalues and eigenvectors of a symmetric diagonal matrix)
4. CPTEQR(VECT='N') (LAPACK computation of the eigenvalues only of a symmetric positive definite tridiagonal matrix)
5. CUNGTR+CPTEQR(VECT='V') (LAPACK computation of the eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix)
6. SSTEBZ+CSTEIN+CUNMTR (LAPACK computation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix)
7. CSTEDC(COMPQ='I')+CUNMTR

8. CSTEGR(COMPQ='V')
9. HTRIDI (EISPACK reduction to symmetric tridiagonal form, to be compared to CHETRD)
10. IMTQL1 (EISPACK computation of eigenvalues only of a symmetric tridiagonal matrix, to be compared to CSTEQR(V))
11. 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 CUNGTR+CSTEQR(V)).

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
5                               Number of values of N
10 20 40 60 80                  Values of N (dimension)
2                               Number of values of parameters
1  16                           Values of NB (blocksize)
81 81                           Values of LDA (leading dimension)
0.05                            Minimum time in seconds
4                               Number of matrix types
SST    T T T T T T T T T T T T T T T T T T T T T T T
```

The first line of the input file must contain the characters 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 $0 < \text{NTYPES} < 4$, 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.

8.6 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 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, SORGBR and SBDSQR(L))
9. LAPSVD(L) (LAPACK singular values and M left singular vectors of a dense matrix, using SGEBRD, SORGBR and SBDSQR(L))
10. LAPSVD(R) (LAPACK singular values and N right singular vectors of a dense matrix, using SGEBRD, SORGBR 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, SORGBR and SBDSQR(B))
12. SBDSDC(B) (LAPACK singular values and left and right singular vectors; assume original matrix min(M,N) by min(M,N))
13. SGESDD(B) (LAPACK singular values and min(M,N) left singular vectors and N right singular vectors if $M \geq N$, singular values and M left singular vectors and min(M,N) right singular vectors otherwise.)
14. LINSVD (LINPACK singular values only of a dense matrix using SSVDC, to be compared to LAPSVD)
15. LINSVD(l) (LINPACK singular values and min(M,N) left singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(l))
16. LINSVD(L) (LINPACK singular values and M left singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(L))
17. LINSVD(R) (LINPACK singular values and N right singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(R))
18. 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
10 10 20 20 20 40 40            Values of M (row dimension)
10 20 10 20 40 20 40            Values of N (column dimension)
1                               Number of values of parameters
1                               Values of NB (blocksize)
81                              Values of LDA (leading dimension)
0.05                            Minimum time in seconds
4                               Number of matrix types
1 2 3 4
SBD    T T T T T T T T T T T T T T T T 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 $0 < \mathrm{NTYPES} < 5$, 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 SVD, the path names for the four data types are SBD, DBD, CBD, and ZBD. The (optional) characters after the path name indicate the computations to be timed, as in the input file for the nonsymmetric eigenvalue problem.

8.7 Timing the Generalized Singular Value Decomposition

At the present time, no timing program for GSVD is provided. The main reason for this omission is because the GSVD subroutine is essentially BLAS 1 sequential code in the current implementation.

8.8 Timing the Generalized QR and RQ Factorizations

At the present time, no timing program for the GQR and GRQ factorizations is provided. The main reason for this omission is because these codes rely heavily on the QR and RQ factorizations which already have existing timing code.

8.9 Timing the Generalized Linear Regression Model Problem

At the present time, no timing program for GLM is provided. The main reason for this omission is because the major floating point operations of this code is in the GQR factorization. The GQR factorization relies heavily on the QR and RQ factorizations which already have existing timing code.

8.10 Timing the Constrained Linear Least Squares Problem

At the present time, no timing program for LSE is provided. The main reason for this omission is because the major floating point operations of this code is in the GRQ factorization. The GRQ factorization relies heavily on the QR and RQ factorizations which already have existing timing code.

Acknowledgments

Ed Anderson contributed to previous versions of this report. Other contributors include: Zhaojun Bai of the University of Kentucky, Jim Demmel of the University of California-Berkeley, Sven Hammarling of NAG Ltd., and Alan McKenney of the Courant Institute of Mathematical Sciences, New York University.

Appendix A

LAPACK Routines

In this appendix, we review the subroutine naming scheme for LAPACK as presented in [1] and indicate by means of a table which subroutines are included in this release. We also list the driver routines.

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. Most of these two-letter codes apply to both real and complex routines; a few apply specifically to one or the other, as indicated below:

BD	bidiagonal
DI	diagonal
GB	general band
GE	general (i.e. unsymmetric, in some cases rectangular)
GG	general matrices, generalized problem (i.e. a pair of general matrices)
GT	general tridiagonal
HB	(complex) Hermitian band
HE	(complex) Hermitian
HG	upper Hessenberg matrix, generalized problem (i.e., a Hessenberg and a triangular matrix)
HP	(complex) Hermitian, packed storage
HS	upper Hessenberg
OP	(real) orthogonal, packed storage
OR	(real) orthogonal
PB	symmetric or Hermitian positive definite band
PO	symmetric or Hermitian positive definite
PP	symmetric or Hermitian positive definite, packed storage

PT	symmetric or Hermitian positive definite tridiagonal
SB	(real) symmetric band
SP	symmetric, packed storage
ST	symmetric tridiagonal
SY	symmetric
TB	triangular band
TG	triangular matrices, generalized problem (i.e., a pair of triangular matrices)
TP	triangular, packed storage
TR	triangular (or in some cases quasi-triangular)
TZ	trapezoidal
UN	(complex) unitary
UP	(complex) unitary, packed storage

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

BAK	back transformation of eigenvectors after balancing
BAL	permute and/or balance to isolate eigenvalues
BRD	reduce to bidiagonal form by orthogonal transformations
CON	estimate condition number
EBZ	compute selected eigenvalues by bisection
EDC	compute eigenvectors using divide and conquer
EIN	compute selected eigenvectors by inverse iteration
EQR	compute eigenvalues and/or the Schur form using the QR algorithm
EGR	compute selected eigenvalues, and optionally, eigenvectors using Relatively Robust Representations
EQU	equilibrate a matrix to reduce its condition number
EQZ	compute generalized eigenvalues and/or generalized Schur form by QZ method
ERF	compute eigenvectors using the Pal-Walker-Kahan variant of the QL or QR algorithm
EVC	compute eigenvectors from Schur factorization
EXC	swap adjacent diagonal blocks in a quasi-upper triangular matrix
GBR	generate the orthogonal/unitary matrix from xGEBRD
GHR	generate the orthogonal/unitary matrix from xGEHRD
GLQ	generate the orthogonal/unitary matrix from xGELQF
GQL	generate the orthogonal/unitary matrix from xGEQLF
GQR	generate the orthogonal/unitary matrix from xGEQRF
GRQ	generate the orthogonal/unitary matrix from xGERQF
GST	reduce a symmetric-definite generalized eigenvalue problem to standard form
GTR	generate the orthogonal/unitary matrix from xxxTRD
HRD	reduce to upper Hessenberg form by orthogonal transformations
LQF	compute an LQ factorization without pivoting
MBR	multiply by the orthogonal/unitary matrix from xGEBRD
MHR	multiply by the orthogonal/unitary matrix from xGEHRD
MLQ	multiply by the orthogonal/unitary matrix from xGELQF

MQL	multiply by the orthogonal/unitary matrix from xGEQLF
MQR	multiply by the orthogonal/unitary matrix from xGEQRF
MRQ	multiply by the orthogonal/unitary matrix from xGERQF
MRZ	multiply by the orthogonal/unitary matrix from xTZRZF
MTR	multiply by the orthogonal/unitary matrix from xxxTRD
QLF	compute a QL factorization without pivoting
QPF	compute a QR factorization with column pivoting
QRF	compute a QR factorization without pivoting
RFS	refine initial solution returned by TRS routines
RQF	compute an RQ factorization without pivoting
RZF	compute an RZ factorization without pivoting
SDC	compute using divide and conquer SVD
SEN	compute a basis and/or reciprocal condition number (sensitivity) of an invariant subspace
SJA	obtain singular values, and optionally vectors, using Jacobi's method
SNA	estimate reciprocal condition numbers of eigenvalue/-vector pairs
SQR	compute singular values and/or singular vectors using the QR algorithm
SVP	preprocessing for GSVD
SYL	solve the Sylvester matrix equation
TRD	reduce a symmetric matrix to real symmetric tridiagonal form
TRF	compute a triangular factorization (LU, Cholesky, etc.)
TRI	compute inverse (based on triangular factorization)
TRS	solve systems of linear equations (based on triangular factorization)

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

									HE	HP				UN
	GE	GG	GB	GT	PO	PP	PB	PT	SY	SP	TR	TP	TB	OR
TRF	×		×	×	×	×	×	×	×	×				
TRS	×		×	×	×	×	×	×	×	×	×	×	×	
RFS	×		×	×	×	×	×	×	×	×	×	×	×	
TRI	×				×	×			×	×	×	×		
CON	×		×	×	×	×	×	×	×	×	×	×	×	
EQU	×		×		×	×	×							
QPF	×													
QRF†	×	×												
GQR†														×
MQR†														×

†– also RQ, QL, LQ, and RZ

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

								HE	HP	HB			
	GE	GB	GG	HS	HG	TR	TG	SY	SP	SB	ST	PT	BD
HRD	×		×										
TRD								×	×	×			
BRD	×	×											
EDC											×		
EQR				×							×	×	
EQZ					×								
EIN				×							×		
EVC						×	×						
EBZ											×		
ERF											×		
SQR													×
SDC													×
SEN						×	×						
SJA							×						
SNA						×	×						
SVP			×										
SYL						×	×						
EXC						×	×						
BAL	×		×										
BAK	×		×										
GST								×	×	×			

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

	UN	UP
	OR	OP
GHR	×	
GTR	×	×
GBR	×	
MHR	×	
MTR	×	×
MBR	×	
MRZ	×	

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

SV	factor the matrix and solve a system of equations
SVX	equilibrate, factor, solve, compute error bounds and do iterative refinement, and estimate the condition number
GLM	solves the generalized linear regression model
LS	solve over- or underdetermined linear system using orthogonal factorizations
LSE	solves the constrained linear least squares problem
LSX	compute a minimum-norm solution using a complete orthogonal factorization

	(using QR with column pivoting xGEQPF, xTZRQF, and xLATZM)
LSY	compute a minimum-norm solution using a complete orthogonal factorization (using QR with column pivoting xGEQP3, xTZRZF, and xORMRZ)
LSS	solve least squares problem using the SVD
LSD	solve least squares problem using the divide and conquer SVD
EV	compute all eigenvalues and/or eigenvectors
EVD	compute all eigenvalues and/or eigenvectors; if eigenvectors are desired, it uses a divide and conquer algorithm.
EVX	compute selected eigenvalues and eigenvectors
EVR	compute selected eigenvalues, and optionally, eigenvectors

using the Relatively Robust Representations

ES	compute all eigenvalues, Schur form, and/or Schur vectors
ESX	compute all eigenvalues, Schur form, and/or Schur vectors and the conditioning of selected eigenvalues or eigenvectors
GV	compute generalized eigenvalues and/or generalized eigenvectors
GVD	compute generalized eigenvalues, and optionally, generalized eigenvectors using a divide and conquer algorithm
GVX	compute selected generalized eigenvalues, and optionally, generalized eigenvectors
GS	compute generalized eigenvalues, Schur form, and/or Schur vectors
SDD	compute the divide-and-conquer SVD
SVD	compute the SVD and/or singular vectors

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

									HE	HP	HB	
	GE	GG	GB	GT	PO	PP	PB	PT	SY	SP	SB	ST
SV	×		×	×	×	×	×	×	×	×		
SVX	×		×	×	×	×	×	×	×	×		
GLM		×										
LS	×											
LSE		×										
LSX	×											
LSY	×											
LSS	×											
LSD	×											
EV	×	×							×	×	×	×
EVD									×	×	×	×
EVX	×	×							×	×	×	×
EVR									×			×
ES	×	×										
ESX	×	×										
GV									×	×	×	
GVD									×	×	×	
GVX									×	×	×	
SVD	×	×										
SDD	×											

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`. Routines specified with an underscore as the first character are available in all four data types (S, D, C, and Z), except those marked (real), for which the first character may be 'S' or 'D', and those marked (complex), for which the first character may be 'C' or 'Z'.

Special subroutines:

XERBLA	Error handler for the BLAS and LAPACK routines

Special functions:

ILAENV	INTEGER	Return block size and other parameters
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

Functions for computing norms:

_LANGB	General band matrix
_LANGE	General matrix
_LANGT	General tridiagonal matrix
_LANHB	(complex) Hermitian band matrix
_LANHE	(complex) Hermitian matrix
_LANHP	(complex) Hermitian packed matrix
_LANHS	Upper Hessenberg matrix
_LANHT	(complex) Hermitian tridiagonal matrix
_LANSB	Symmetric band matrix
_LANSP	Symmetric packed matrix
_LANST	(real) Symmetric tridiagonal matrix
_LANSY	Symmetric matrix

_LANTB	Triangular band matrix
_LANTP	Triangular packed matrix
_LANTR	Trapezoidal matrix

Extensions to the Level 1 and 2 BLAS:

CROT	Apply a plane rotation to a pair of complex vectors, where the cos is real and the sin is complex
CSROT	Apply a real plane rotation to a pair of complex vectors
ZDROT	Double precision version of CSROT
_SYMV	(complex) Symmetric matrix times vector
_SPMV	(complex) Symmetric packed matrix times vector
_SYR	(complex) Symmetric rank-1 update
_SPR	(complex) Symmetric rank-1 update of a packed matrix
ICMAX1	Find the index of element whose real part has max. abs. value
IZMAX1	Find the index of element whose real part has max. abs. value
SCSUM1	Sum absolute values of a complex vector
DZSUM1	Double precision version of SCSUM1
_RSCL	(real) Scale a vector by the reciprocal of a constant
CSRSCL	Scale a complex vector by the reciprocal of a real constant
ZDRSCL	Double precision version of CSRSCL

Level 2 BLAS versions of the block routines:

_GBTF2	compute the LU factorization of a general band matrix
_GEBD2	reduce a general matrix to bidiagonal form
_GEHD2	reduce a square matrix to upper Hessenberg form
_GELQ2	compute an LQ factorization without pivoting
_GEQL2	compute a QL factorization without pivoting
_GEQR2	compute a QR factorization without pivoting
_GERQ2	compute an RQ factorization without pivoting
_GESC2	solves the system of linear equations $A * X = scale * RHS$ using the factorization computed by xGETC2
_GETC2	computes the LU factorization of a general matrix with complete pivoting
_GETF2	compute the LU factorization of a general matrix
_GTTS2	
_HEGS2	(complex) reduce a Hermitian-definite generalized eigenvalue problem to standard form
_HETD2	(complex) reduce a Hermitian matrix to real tridiagonal form
_HETF2	(complex) compute diagonal pivoting factorization of a Hermitian matrix
_ORG2L	(real) generate the orthogonal matrix from xGEQLF
_ORG2R	(real) generate the orthogonal matrix from xGEQRF
_ORGL2	(real) generate the orthogonal matrix from xGEQLF
_ORGR2	(real) generate the orthogonal matrix from xGERQF
_ORM2L	(real) multiply by the orthogonal matrix from xGEQLF
_ORM2R	(real) multiply by the orthogonal matrix from xGEQRF
_ORML2	(real) multiply by the orthogonal matrix from xGELQF

_ORMR2	(real) multiply by the orthogonal matrix from xGERQF
_ORMR3	(real) multiply by the orthogonal matrix from xTZRZF
_PBTF2	compute the Cholesky factorization of a positive definite band matrix
_POTF2	compute the Cholesky factorization of a positive definite matrix
_PTTS2	
_SYGS2	(real) reduce a symmetric-definite generalized eigenvalue problem to standard form
_SYTD2	(real) reduce a symmetric matrix to tridiagonal form
_SYTF2	compute the diagonal pivoting factorization of a symmetric matrix
_TGEX2	swaps adjacent diagonal blocks in an upper (quasi) triangular matrix pair (A, B) by an orthogonal equivalence transformation
_TGSY2	Solves the generalized Sylvester equation
_TRTI2	compute the inverse of a triangular matrix
_UNG2L	(complex) generate the unitary matrix from xGEQLF
_UNG2R	(complex) generate the unitary matrix from xGEQRF
_UNGL2	(complex) generate the unitary matrix from xGEQLF
_UNGR2	(complex) generate the unitary matrix from xGERQF
_UNM2L	(complex) multiply by the unitary matrix from xGEQLF
_UNM2R	(complex) multiply by the unitary matrix from xGEQRF
_UNML2	(complex) multiply by the unitary matrix from xGELQF
_UNMR2	(complex) multiply by the unitary matrix from xGERQF
_UNMR3	(complex) multiply by the unitary matrix from xTZRZF

Other LAPACK auxiliary routines:

_LABAD	(real) returns square root of underflow and overflow if exponent range is large
_LABRD	reduce NB rows or columns of a matrix to upper or lower bidiagonal form
_LACGV	(complex) conjugates a complex vector of length n
_LACRM	(complex) matrix multiply $C = A * B$, where A is complex, B is real, and C is complex.
_LACRT	(complex) applies a plane rotation to two complex vectors
_LACON	estimate the norm of a matrix for use in condition estimation
_LACP2	copies all or part of a real matrix to a complex matrix
_LACPY	copy a matrix to another matrix
_LADIV	perform complex division in real arithmetic
_LAE2	(real) compute eigenvalues of a 2-by-2 real symmetric matrix
_LAEBZ	compute and use the count of eigenvalues of a symmetric tridiagonal matrix
_LAED0	Used by xSTEDC.
_LAED1	(real) Used by xSTEDC.
_LAED2	(real) Used by xSTEDC.
_LAED3	(real) Used by xSTEDC.
_LAED4	(real) Used by xSTEDC.
_LAED5	(real) Used by xSTEDC.
_LAED6	(real) Used by xSTEDC.
_LAED7	Used by xSTEDC.

_LAED8	Used by xSTEDC.
_LAED9	(real) Used by xSTEDC.
_LAEDA	Used by xSTEDC.
_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-by-2 matrix
_LAEV2	Compute eigenvalues and eigenvectors of a 2-by-2 real symmetric or complex Hermitian matrix
_LAEXC	swap adjacent diagonal blocks in a quasi-upper triangular matrix
_LAG2	compute the eigenvalues of a 2-by-2 generalized eigenvalue problem with scaling to avoid over-/underflow
_LAGS2	computes 2-by-2 orthogonal matrices
_LAGTF	(real) factorizes the matrix $(T - \lambda I)$
_LAGTM	matrix-vector product where the matrix is tridiagonal
_LAGTS	solves a system of equations $(T - \lambda I)x = y$ where T is a tridiagonal matrix
_LAGV2	computes the generalized Schur factorization of a 2-by-2 matrix pencil (A,B), where B is upper triangular.
_LAHEF	(complex) compute part of the diagonal pivoting factorization of a Hermitian matrix
_LAHQR	Find the Schur factorization of a Hessenberg matrix (modified version of HQR from EISPACK)
_LAHRD	reduce NB columns of a general matrix to Hessenberg form
_LAIC1	apply one step of incremental condition estimation
_LALN2	(real) Solve a 1-by-1 or 2-by-2 linear system
_LALS0	Used by xGELSD.
_LALSA	Used by xGELSD.
_LALSD	Used by xGELSD.
_LAMRG	(real) Creates a permutation list to merge entries of two independently sorted sets
_LANV2	(real) computes the Schur factorization of a real 2-by-2 nonsymmetric matrix
_LAPLL	measures linear dependence of two vectors
_LAPMT	applies forward or backward permutations to the columns of a matrix
_LAPY2	(real) Compute square root of X**2 + Y**2
_LAPY3	(real) Compute square root of X**2 + Y**2 + Z**2
_LAQGB	equilibrate a general band matrix
_LAQGE	equilibrate a general matrix
_LAQP2	Used by xGEQP3.
_LAQPS	Used by xGEQP3.
_LAQSB	equilibrate a symmetric band matrix
_LAQSP	equilibrate a symmetric packed matrix
_LAQSY	equilibrate a symmetric matrix
_LAQTR	(real) solve a real quasi-triangular system
_LAR1V	Computes the (scaled) column of the inverse of the

	submatrix in rows B1 through BN of the tridiagonal matrix $LDL^T\ \sigma I$
_LAR2V	apply real plane rotations from both sides to a sequence of 2-by-2 real symmetric matrices
_LARCM	(complex) matrix multiply $C = A * B$, where A is real, B is complex, and C is complex.
_LARF	apply (multiply by) an elementary reflector
_LARFB	apply (multiply by) a block reflector
_LARFG	generate an elementary reflector
_LARFT	form the triangular factor of a block reflector
_LARFX	unrolled version of xLARF
_LARGV	generate a vector of plane rotations
_LARNV	returns a vector of random numbers from a uniform or normal distribution
_LARRB	(real) performs limited bisection to locate eigenvalues
_LARRE	(real) sets "small" off-diagonal elements of tridiagonal matrix T to zero, and find the numbers σ_i, the base $T_i - \sigma_i I = L_i D_i L_i^T$ representations and the eigenvalues of each $L_i D_i L_i^T$.
_LARRF	(real) finds a new relatively robust representation such that at least one of the eigenvalues is relatively isolated.
_LARRV	(real) computes the eigenvectors of the tridiagonal matrix $T = LDL^T$ given L, D and the eigenvalues of LDL^T.
_LARTG	generate a plane rotation
_LARTV	apply a vector of plane rotations to a pair of vectors
_LARUV	(real) returns a vector of real random numbers from a uniform distribution
_LARZ	apply (multiply by) an elementary reflector as returned by xTZRZF
_LARZB	apply (multiply by) a block reflector as returned by xTZRZF
_LARZT	form the triangular factor of a block reflector as returned by xTZRZF
_LAS2	(real) Compute singular values of a 2-by-2 triangular matrix
_LASCL	scale a matrix by CTO/CFROM
_LASD0	(real) Used by SBDSDC.
_LASD1	(real) Used by SBDSDC.
_LASD2	(real) Used by SBDSDC.
_LASD3	(real) Used by SBDSDC.
_LASD4	(real) Used by SBDSDC.
_LASD5	(real) Used by SBDSDC.
_LASD6	(real) Used by SBDSDC.
_LASD7	(real) Used by SBDSDC.
_LASD8	(real) Used by SBDSDC.
_LASD9	(real) Used by SBDSDC.
_LASDA	(real) Used by SBDSDC.
_LASDQ	(real) Used by SBDSDC.
_LASDT	(real) Used by SBDSDC.
_LASET	initializes a matrix to BETA on the diagonal and ALPHA on the off-diagonals

_LASQ1	(real) Used by SBDSQR.
_LASQ2	(real) Used by SBDSQR.
_LASQ3	(real) Used by SBDSQR.
_LASQ4	(real) Used by SBDSQR.
_LASQ5	(real) Used by SBDSQR and SSTEGR.
_LASQ6	(real) Used by SBDSQR and SSTEGR.
_LASR	Apply a sequence of plane rotations to a rectangular matrix
_LASRT	Sorts numbers in increasing or decreasing order using Quick Sort, reverting to Insertion sort on arrays of size ≤ 20.
_LASSQ	Compute a scaled sum of squares of the elements of a vector
_LASV2	(real) Compute singular values and singular vectors of a 2-by-2 triangular matrix
_LASWP	Perform a series of row interchanges
_LASY2	(real) solve for a matrix X that satisfies the equation $TL * X + ISGN * X * TR = SCALE * B$
_LASYF	compute part of the diagonal pivoting factorization of a symmetric matrix
_LATBS	solve a triangular band system with scaling to prevent overflow
_LATDF	computes a contribution to the reciprocal Dif-estimate
_LATPS	solve a packed triangular system with scaling to prevent overflow
_LATRD	reduce NB rows and columns of a real symmetric or complex Hermitian matrix to tridiagonal form
_LATRS	solve a triangular system with scaling to prevent overflow
_LATRZ	factors a real upper trapezoidal matrix by orthogonal transformations
_LATZM	apply a Householder matrix generated by xTZRQF to a matrix
_LAUU2	Unblocked version of _LAUUM
_LAUUM	Compute the product U*U' or L'*L (blocked version)

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 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 1.

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 2 we assume these parameters are always ± 1 or 0, since that is how they are used in the LAPACK routines.

Operation Counts for the LAPACK Routines

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

Level 2 BLAS	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 1: 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 2: 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 $[2n^2 - n]$

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 $[2n^2]$

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 $[4nk + 2n - 2k^2 - 2k]$

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 $[2n^2]$

SGEQRF or SGEQLF $(m \geq n)$

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:	$2mn^2 - 2/3n^3 + mn + n^2 + 14/3n$

SGEQRF or SGEQLF $(m \leq n)$

multiplications:	$nm^2 - 1/3m^3 + 2nm - 1/2m^2 + 23/6m$
additions:	$nm^2 - 1/3m^3 + nm - 1/2m^2 + 5/6m$
total flops:	$2nm^2 - 2/3m^3 + 3nm - m^2 + 14/3n$

SGERQF or SGELQF ($m \geq n$)

multiplications:	$mn^2 - 1/3n^3 + mn + 1/2n^2 + 29/6n$
additions:	$mn^2 - 1/3n^3 + mn - 1/2n^2 + 5/6n$
total flops:	$2mn^2 - 2/3n^3 + 2mn + 17/3n$

SGERQF or SGELQF ($m \leq n$)

multiplications:	$nm^2 - 1/3m^3 + 2nm - 1/2m^2 + 29/6m$
additions:	$nm^2 - 1/3m^3 + 1/2m^2 + 5/6m$
total flops:	$2nm^2 - 2/3m^3 + 2nm + 17/3n$

SORGQR or SORGQL

multiplications:	$2mnk - (m+n)k^2 + 2/3k^3 + 2nk - k^2 - 5/3k$
additions:	$2mnk - (m+n)k^2 + 2/3k^3 + nk - mk + 1/3k$
total flops:	$4mnk - 2(m+n)k^2 + 4/3k^3 + 3nk - mk - k^2 - 4/3k$

SORGLQ or SORGRQ

multiplications:	$2mnk - (m+n)k^2 + 2/3k^3 + mk + nk - k^2 - 2/3k$
additions:	$2mnk - (m+n)k^2 + 2/3k^3 + mk - nk + 1/3k$
total flops:	$4mnk - 2(m+n)k^2 + 4/3k^3 + 2mk - k^2 - 1/3k$

SGEQRS	multiplications:	NRHS $[2mn - 1/2n^2 + 5/2n]$
	additions:	NRHS $[2mn - 1/2n^2 + 1/2n]$
	total flops:	NRHS $[4mn - n^2 + 3n]$

SORMQR, SORMLQ, SORMQL or SORMRQ (SIDE = 'L')

multiplications:	$2nmk - nk^2 + 2nk$
additions:	$2nmk - nk^2 + nk$
total flops:	$4nmk - 2nk^2 + 3nk$

SORMQR, SORMLQ, SORMQL or SORMRQ (SIDE = 'R')

multiplications:	$2nmk - mk^2 + mk + nk - 1/2k^2 + 1/2k$
additions:	$2nmk - mk^2 + mk$
total flops:	$4nmk - 2mk^2 + 2mk + nk - 1/2k^2 + 1/2k$

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:	$4/3n^3 + 3n^2 - 17/6n - 19$

SGEBRD $(m \geq n)$

multiplications:	$2mn^2 - 2/3n^3 + 2n^2 + 20/3n$
additions:	$2mn^2 - 2/3n^3 + n^2 - mn + 5/3n$
total flops:	$4mn^2 - 4/3n^3 + 3n^2 - mn + 25/3n$

SGEBRD $(m < n)$

exchange m and n in above

Appendix D

Caveats

In this appendix we list a few of the machine-specific difficulties we have encountered in our own experience with LAPACK. A more detailed list of machine-dependent problems, bugs, and compiler errors encountered in the LAPACK installation process is maintained on *netlib*.

```
http://www.netlib.org/lapack/release_notes
```

We assume the user has installed the machine-specific routines correctly and that the Level 1, 2 and 3 BLAS test programs have run successfully, so we do not list any warnings associated with those routines.

D.1 `LAPACK/make.inc`

All machine-specific parameters are specified in the file `LAPACK/make.inc`.

The first line of this `make.inc` file is:

SHELL = /bin/sh

and will need to be modified to `SHELL = /sbin/sh` if you are installing LAPACK on an SGI architecture.

D.2 ETIME

The LAPACK Testing and Timing Suites assume the use of the timing function `ETIME`.

On some IBM architectures such as IBM RS/6000s, the timing function `ETIME` is instead called `ETIME_`, and therefore the routines `LAPACK/INSTALL/second.f` and `LAPACK/INSTALL/dsecnd.f` should be modified.

On HPPA architectures, the compiler and loader flag `+U77` should be included to access the function `ETIME`.

D.3 ILAENV and IEEE-754 compliance

As some new routines in LAPACK rely on IEEE-754 compliance, two settings (`ISPEC=10` and `ISPEC=11`) have been added to ILAENV (`LAPACK/SRC/ilaenv.f`) to denote IEEE-754 compliance for NaN and infinity arithmetic, respectively. By default, ILAENV assumes an IEEE machine, and does a test for IEEE-754 compliance. **NOTE: If you are installing LAPACK on a non-IEEE machine, you MUST modify ILAENV, as this test inside ILAENV will crash!**

Thus, for non-IEEE machines, the user must hard-code the setting of (`ILAENV=0`) for (`ISPEC=10` and `ISPEC=11`) in the version of `LAPACK/SRC/ilaenv.f` to be put in his library. For further details, refer to section 6.1.4.

Be aware that some IEEE compilers by default do not enforce IEEE-754 compliance, and a compiler flag must be explicitly set by the user.

On SGIs for example, you must set the `-OPT:IEEE_NaN_inf=ON` compiler flag to enable IEEE-754 compliance.

And lastly, the test inside ILAENV to detect IEEE-754 compliance, will result in IEEE exceptions for "Divide by Zero" and "Invalid Operation". Thus, if the user is installing on a machine that issues IEEE exception warning messages (like a Sun SPARCstation), the user can disregard these messages. To avoid these messages, the user can hard-code the values inside ILAENV as explained in section 6.1.4.

D.4 Lack of `/tmp` space

If `/tmp` space is small (i.e., less than approximately 16 MB) on your architecture, you may run out of space when compiling. There are a few possible solutions to this problem.

1. You can ask your system administrator to increase the size of the `/tmp` partition.
2. You can change the environment variable `TMPDIR` to point to your home directory for temporary space. E.g.,

   ```
   setenv TMPDIR /home/userid/
   ```

   where `/home/userid/` is the user's home directory.
3. If your archive command has an `l` option, you can change the archive command to `ar crl` so that the archive command will only place temporary files in the current working directory rather than in the default temporary directory /tmp.

D.5 BLAS

If you suspect a BLAS-related problem and you are linking with an optimized version of the BLAS, we would strongly suggest as a first step that you link to the Fortran 77 version of the suspected BLAS routine and see if the error has disappeared.

We have 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. *On some CRAY architectures, the Fortran77 version of xNRM2 should be used.*

D.6 Optimization

If a large numbers of test failures occur for a specific matrix type or operation, it could be that there is an optimization problem with your compiler. Thus, the user could try reducing the level of optimization or eliminating optimization entirely for those routines to see if the failures disappear when you rerun the tests.

D.7 Compiling testing/timing drivers

The testing and timing main programs (xCHKAA, xCHKEE, xTIMAA, and xTIMEE) allocate large amounts of local variables. Therefore, it is vitally important that the user know if his compiler by default allocates local variables statically or on the stack. It is not uncommon for those compilers which place local variables on the stack to cause a stack overflow at runtime in the testing or timing process. The user then has two options: increase your stack size, or force all local variables to be allocated statically.

On HPPA architectures, the compiler and loader flag `-K` should be used when compiling these testing and timing main programs to avoid such a stack overflow. I.e., set `DRVOPTS = -K` in the `LAPACK/make.inc` file.

For similar reasons, on SGI architectures, the compiler and loader flag `-static` should be used. I.e., set `DRVOPTS = -static` in the `LAPACK/make.inc` file.

D.8 IEEE arithmetic

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 $\log_{10}$(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.

Usually on HPPA architectures, a similar restriction in SLABAD should be enforced for all testing involving complex arithmetic. SLABAD is located in `LAPACK/SRC`.

For machines which have a narrow exponent range or lack gradual underflow (DEC VAXes for example), it is not uncommon to experience failures in sec.out and/or dec.out with SLAQTR/DLAQTR or DTRSYL. The failures in SLAQTR/DLAQTR and DTRSYL occur with test problems which are very badly scaled when the norm of the solution is very close to the underflow threshold (or even underflows to zero). We believe that these failures could probably be avoided by an even greater degree of care in scaling, but we did not want to delay the release of LAPACK any further. These tests pass successfully on most other machines. An example failure in dec.out on a MicroVAX II looks like the following:

```
Tests of the Nonsymmetric eigenproblem condition estimation routines
DLALN2, DLASY2, DLANV2, DLAEXC, DTRSYL, DTREXC, DTRSNA, DTRSEN, DLAQTR

Relative machine precision (EPS) =     0.277556D-16
Safe minimum (SFMIN)             =     0.587747D-38

Routines pass computational tests if test ratio is less than   20.00

DEC routines passed the tests of the error exits ( 35 tests done)
Error in DTRSYL: RMAX =   0.155D+07
LMAX =      5323 NINFO=    1600 KNT=   27648
Error in DLAQTR: RMAX =   0.344D+04
LMAX =     15792 NINFO=   26720 KNT=   45000
```

D.9 Timing programs

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

Installation Guide for PCs using Windows 98/NT

LAPACK requires unix-style make and /bin/sh commands in order to install on a Windows system. A fairly complete unix-style environment is available free of charge at the cygnus website,

`http://sourceware.cygnus.com/cygwin/`

From this website, you can download the package, get installation instructions, etc. You will want to download the "full" version of cygwin, which includes compilers, shells, make, etc. You will need to download the fortran compiler separately.

The installation is quite simple, involving downloading an executable and installing with Windows' usual install procedure (you can remove it from your machine with Windows' ADD/REMOVE if you later decide you don't want it).

IMPORTANT:

Windows 95/98 does a poor job of process load balance. If you change the focus from the cygnus window, performance will immediately drop by at least 1/3, and the timings will be inaccurate. When doing timings, it is recommended that you leave the focus on the window throughout the entire timing suite. This is not necessary for Windows NT.

Because people often miss them in the install instructions, we repeat two very important pieces of information about the cygnus install here:

1. `If, after installing cygnus, you get the message:`
   ```
       Out of environment space
   add the line
       shell=C:\command.com /e:4096 /p
   to your c:\config.sys
   ```

2. `For installation, LAPACK needs to find /bin/sh, so you should (assuming you don't already have this directory made):`

```
   mkdir -p /bin
Then, you should copy sh.exe from the cygwin bin directory to this one.
The location of the cygwin bin directory changes depending on where you
did the install, what type of machine you have, and the version of cygnus.
Here is an example:
    /cygnus/cygwin-b20/H-i586-cygwin32/bin
the cygwin-b20 is a version number, so you might see cygwin-b21, if you have
a newer release, for instance.  The i586 refers to your processor, you might
expect to see i386, i486, i586 or i686, for instance.
```

- NOTE:
  Gnu g77 and gcc provide better performance than MSVC++ (Digital Fortran) (or Watcom F77 or C), so we recommend that you use g77 and gcc.

- NOTE:
  Be careful. Many PC compilers often perform optimization by default at compile time! Thus, for routines such as LAPACK/SRC/slamch.f and LAPACK/SRC/dlamch.f, you will need to explicitly set a compile flag to turn OFF optimization.

- NOTE:
  Be aware that Microsoft *nmake* and Watcom *wmake* contain only a subset of the functionality of unix-style make. Therefore, if you choose to use this form of *make*, you will need to simply the makefiles. In the future, these makefiles may be made available.

- NOTE:
  Timing functions and Windows 98/NT... You will need to modify the LAPACK/INSTALL/second.f and dsecnd.f to call `clock()`, as there are no sophisticate timing functions available. Many users have written their own timing functions for Windows 98/NT applications.

- NOTE:
  An optimized BLAS library for Windows on an Intel Pentium is available. Refer to the BLAS FAQ on netlib for further details.

  `http://www.netlib.org/blas/faq.html`

Appendix F

Installation Guide for VAX-type Systems

The non-Unix version of LAPACK is created in two steps. First, the user must untar the Unix tar file according to the directions in section 4. Second, after the file has been tarred, the user must then go to the LAPACK directory and type `latape`. The execution of this file creates a directory called `ASCII` in the user's main directory. This ASCII directory contains the grouped files needed for a VAX-type installation. The layout of the `ASCII` directory is as described in this appendix.

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

The leading one or two characters of the file name generally indicates which of the different versions of the library or test 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.	TILAENVF	Test program for ILAENV
20.	ALLBLASF	Auxiliary routines for the BLAS (and LAPACK)
21.	SBLAS1F	Level 1 BLAS
22.	CBLAS1F	
23.	DBLAS1F	
24.	ZBLAS1F	
25.	CB1AUXF	Auxiliary routines for Complex Level 1 BLAS
26.	ZB1AUXF	Auxiliary routines for D.P. Complex Level 1 BLAS
27.	SBLAS2F	Level 2 BLAS
28.	CBLAS2F	
29.	DBLAS2F	
30.	ZBLAS2F	
31.	SBLAS3F	Level 3 BLAS
32.	CBLAS3F	
33.	DBLAS3F	
34.	ZBLAS3F	
35.	SBLAT1F	Test program for Level 1 BLAS
36.	CBLAT1F	
37.	DBLAT1F	
38.	ZBLAT1F	
39.	SBLAT2F	Test program for Level 2 BLAS

40. CBLAT2F	
41. DBLAT2F	
42. ZBLAT2F	
43. SBLAT2D	Data file for testing Level 2 BLAS
44. CBLAT2D	
45. DBLAT2D	
46. ZBLAT2D	
47. SBLAT3F	Test program for Level 3 BLAS
48. CBLAT3F	
49. DBLAT3F	
50. ZBLAT3F	
51. SBLAT3D	Data file for testing Level 3 BLAS
52. CBLAT3D	
53. DBLAT3D	
54. ZBLAT3D	
55. SCATGENF	Auxiliary routines for the test matrix generators
56. DZATGENF	
57. SMATGENF	Test matrix generators
58. CMATGENF	
59. DMATGENF	
60. ZMATGENF	
61. ALINTSTF	Auxiliary routines for the linear equation test program
62. SLINTSTF	Test program for linear equation routines
63. CLINTSTF	
64. DLINTSTF	
65. ZLINTSTF	
66. SCLNTSTF	Auxiliary routines for linear equation test programs
67. DZLNTSTF	
68. SLINTSTD	Data file 1 for linear equation test program
69. DLINTSTD	
70. CLINTSTD	
71. ZLINTSTD	
72. SBAKTSTD	Data file for testing SGEBAK
73. DBAKTSTD	Data file for testing DGEBAK
74. CBAKTSTD	Data file for testing CGEBAK

75.	ZBAKTSTD	Data file for testing ZGEBAK
76.	SBALTSTD	Data file for testing SGEBAL
77.	DBALTSTD	Data file for testing DGEBAL
78.	CBALTSTD	Data file for testing CGEBAL
79.	ZBALTSTD	Data file for testing ZGEBAL
80.	SECTSTD	Data file for testing eigencondition routines
81.	DECTSTD	
82.	CECTSTD	
83.	ZECTSTD	
84.	SEDTSTD	Data file for testing nonsymmetric eigenvalue driver routines
85.	DEDTSTD	
86.	CEDTSTD	
87.	ZEDTSTD	
88.	SSBTSTD	Data file for testing SSBTRD
89.	DSBTSTD	Data file for testing DSBTRD
90.	CSBTSTD	Data file for testing CHBTRD
91.	ZSBTSTD	Data file for testing ZHBTRD
92.	SGGTSTD	Data file for testing nonsymmetric generalized eigenvalue routines
93.	DGGTSTD	
94.	CGGTSTD	
95.	ZGGTSTD	
96.	SGDTSTD	Data file for testing nonsymmetric generalized eigenvalue driver routines
97.	DGDTSTD	
98.	CGDTSTD	
99.	ZGDTSTD	
100.	SSGTSTD	Data file for testing symmetric generalized eigenvalue routines
101.	DSGTSTD	
102.	CSGTSTD	
103.	ZSGTSTD	
104.	AEIGTSTF	Auxiliary routines for the eigensystem test program
105.	SCIGTSTF	
106.	DZIGTSTF	
107.	SEIGTSTF	Test program for eigensystem routines
108.	CEIGTSTF	
109.	DEIGTSTF	
110.	ZEIGTSTF	

111. NEPTSTD	Data file for testing Nonsymmetric Eigenvalue Problem
112. GEPTSTD	Data file for testing Generalized Nonsymmetric Eigenvalue Problem
113. SEPTSTD	Data file for testing Symmetric Eigenvalue Problem
114. SVDTSTD	Data file for testing Singular Value Decomposition
115. GLMTSTD	Data file for testing Generalized Linear Regression Model
116. GQRTSTD	Data file for testing Generalized QR and RQ
117. GSVTSTD	Data file for testing Generalized Singular Value Decomposition
118. LSETSTD	Data file for testing Constrained Linear Least Squares Problem
119. SGKTSTD	Data file for testing SGGBAK
120. DGKTSTD	Data file for testing DGGBAK
121. CGKTSTD	Data file for testing CGGBAK
122. ZGKTSTD	Data file for testing ZGGBAK
123. SGLTSTD	Data file for testing SGGBAL
124. DGLTSTD	Data file for testing DGGBAL
125. CGLTSTD	Data file for testing CGGBAL
126. ZGLTSTD	Data file for testing ZGGBAL
127. SBBTSTD	Data file for testing SGBBRD
128. DBBTSTD	Data file for testing DGBBRD
129. CBBTSTD	Data file for testing CGBBRD
130. ZBBTSTD	Data file for testing ZGBBRD
131. ALINTIMF	Auxiliary routines for the linear system timing program
132. SCINTIMF	
133. DZINTIMF	
134. SLINTIMF	Timing program for linear equations
135. CLINTIMF	
136. DLINTIMF	
137. ZLINTIMF	
138. SLINSRCF	Instrumented LAPACK routines
139. CLINSRCF	
140. DLINSRCF	
141. ZLINSRCF	
142. SCINSRCF	Instrumented auxiliary routines used in S and C versions
143. DZINSRCF	Instrumented auxiliary routines used in D and Z versions
144. SLINTIMD	Data file 1 for timing dense square linear equations
145. DLINTIMD	
146. CLINTIMD	

147.	ZLINTIMD	
148.	SRECTIMD	Data file 1 for timing dense rectangular linear equations
149.	DRECTIMD	
150.	CRECTIMD	
151.	ZRECTIMD	
152.	SBNDTIMD	Data file 1 for timing banded linear equations
153.	DBNDTIMD	
154.	CBNDTIMD	
155.	ZBNDTIMD	
156.	SBLTIMAD	Data file 1-a for timing the BLAS
157.	DBLTIMAD	
158.	CBLTIMAD	
159.	ZBLTIMAD	
160.	SBLTIMBD	Data file 1-b for timing the BLAS
161.	DBLTIMBD	
162.	CBLTIMBD	
163.	ZBLTIMBD	
164.	SBLTIMCD	Data file 1-c for timing the BLAS
165.	DBLTIMCD	
166.	CBLTIMCD	
167.	ZBLTIMCD	
168.	SLINTM2D	Data file 2 for timing dense square linear equations
169.	DLINTM2D	
170.	CLINTM2D	
171.	ZLINTM2D	
172.	SRECTM2D	Data file 2 for timing dense rectangular linear equations
173.	DRECTM2D	
174.	CRECTM2D	
175.	ZRECTM2D	
176.	SBNDTM2D	Data file 2 for timing banded linear equations
177.	DBNDTM2D	
178.	CBNDTM2D	
179.	ZBNDTM2D	
180.	SBLTM2AD	Data file 2-a for timing the BLAS
181.	DBLTM2AD	
182.	CBLTM2AD	

183. ZBLTM2AD	
184. SBLTM2BD	Data file 2-b for timing the BLAS
185. DBLTM2BD	
186. CBLTM2BD	
187. ZBLTM2BD	
188. SBLTM2CD	Data file 2-c for timing the BLAS
189. DBLTM2CD	
190. CBLTM2CD	
191. ZBLTM2CD	
192. AEIGTIMF	Auxiliary routines for the eigensystem timing program
193. SCIGTIMF	
194. DZIGTIMF	
195. SEIGTIMF	Timing program for the eigensystem routines
196. CEIGTIMF	
197. DEIGTIMF	
198. ZEIGTIMF	
199. SEIGSRCF	Instrumented LAPACK routines
200. CEIGSRCF	
201. DEIGSRCF	
202. ZEIGSRCF	
203. SCIGSRCF	Instrumented auxiliary routines used in S and C versions
204. DZIGSRCF	Instrumented auxiliary routines used in D and Z versions
205. SGEPTIMD	Data file 1 for timing Generalized Nonsymmetric Eigenvalue Problem
206. SNEPTIMD	Data file 1 for timing Nonsymmetric Eigenvalue Problem
207. SSEPTIMD	Data file 1 for timing Symmetric Eigenvalue Problem
208. SSVDTIMD	Data file 1 for timing Singular Value Decomposition
209. CGEPTIMD	
210. CNEPTIMD	
211. CSEPTIMD	
212. CSVDTIMD	
213. DGEPTIMD	
214. DNEPTIMD	
215. DSEPTIMD	
216. DSVDTIMD	
217. ZGEPTIMD	

218. ZNEPTIMD	
219. ZSEPTIMD	
220. ZSVDTIMD	

221. SGEPTM2D	Data file 2 for timing Generalized Nonsymmetric Eigenvalue Problem
222. SNEPTM2D	Data file 2 for timing Nonsymmetric Eigenvalue Problem
223. SSEPTM2D	Data file 2 for timing Symmetric Eigenvalue Problem
224. SSVDTM2D	Data file 2 for timing Singular Value Decomposition

225. CGEPTM2D	
226. CNEPTM2D	
227. CSEPTM2D	
228. CSVDTM2D	

229. DGEPTM2D	
230. DNEPTM2D	
231. DSEPTM2D	
232. DSVDTM2D	

233. ZGEPTM2D	
234. ZNEPTM2D	
235. ZSEPTM2D	
236. ZSVDTM2D	

F.1 Installing LAPACK on a non-Unix System

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

1. Gunzip and tar the file.
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.

F.1.1 Gunzip and Tar the File

Gunzip and tar the file as instructed in section 4. You will need about 33 megabytes to read in the complete tape. The total space requirements including the object files is approximately 100 MB for all four data types.

F.1.2 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
ILAENV	INTEGER	Checks that NaN and infinity arithmetic are IEEE-754 comp liant

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.

F.1.2.1 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 test program in TLSAMEF tests all combinations of the same character in upper and lower case for A and B, and two cases where A and B are different characters.

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

```
 ASCII character set
 Tests completed
```

The working version of LSAME should be appended to the file ALLBLASF. This file, which also contains the error handler XERBLA, will be compiled with either the BLAS library in Section A.3 or the LAPACK library in Section A.5.

F.1.2.2 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
Precision                    =     1.19209E-07
Number of digits in mantissa =     24.0000
Rounding mode                =     1.00000
Minimum exponent             =    -125.000
Underflow threshold          =     1.17549E-38
Largest exponent             =     128.000
Overflow threshold           =     3.40282E+38
Reciprocal of safe minimum   =     8.50706E+37
```

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

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 A.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:

'B': Base of the machine

'E': Epsilon (relative machine precision)

'O': Overflow threshold

'P': Precision = Epsilon*Base

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

'U': Underflow threshold

Some people may be familiar with 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/(overflow)) $*$ $(1 + \varepsilon)$, where ε is the machine precision.

F.1.2.3 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 A.5.

F.1.2.4 Testing IEEE arithmetic and ILAENV

As some new routines in LAPACK rely on IEEE-754 compliance, two settings (`ISPEC=10` and `ISPEC=11`) have been added to ILAENV (ILAENV routine inside ALLAUXF) to denote IEEE-754 compliance for NaN and infinity arithmetic, respectively. By default, ILAENV assumes an IEEE machine, and does a test for IEEE-754 compliance. **NOTE: If you are installing LAPACK on a non-IEEE machine, you MUST modify ILAENV, as this test inside ILAENV will crash!**

If `ILAENV( 10, ... )` or `ILAENV( 11, ... )` is issued, then `ILAENV=1` is returned to signal IEEE-754 compliance, and `ILAENV=0` if the architecture is non-IEEE-754 compliant.

Thus, for non-IEEE machines, the user must hard-code the setting of (`ILAENV=0`) for (`ISPEC=10` and `ISPEC=11`) in the version of (ILAENV in ALLAUXF) to be put in his library. There are also specialized testing and timing versions of ILAENV that will also need to be modified.

- Testing/timing version of ILAENV in ALINTSTF
- Testing/timing version of ILAENV in AEIGTSTF
- Testing/timing version of ILAENV in ALINTIMF
- Testing/timing version of ILAENV in AEIGTIMF

The test program in TILAENVF checks an installation architecture to see if infinity arithmetic and NaN arithmetic are IEEE-754 compliant. A warning message to the user is printed if non-compliance is detected. This same test is performed inside the function ILAENV. If `ILAENV( 10, ... )` or `ILAENV( 11, ... )` is issued, then `ILAENV=1` is returned to signal IEEE-754 compliance, and `ILAENV=0` if the architecture is non-IEEE-754 compliant.

To avoid this IEEE test being run every time you call `ILAENV( 10, ...)` or `ILAENV( 11, ... )`, we suggest that the user hard-code the setting of `ILAENV=1` or `ILAENV=0` in the version of (ILAENV in ALLAUXF) to be put in his library. As aforementioned, there are also specialized testing and timing versions of ILAENV that will also need to be modified.

F.1.3 Create the BLAS Library

Ideally, a highly optimized version of the BLAS library already exists on your machine. In this case you can go directly to Section A.4 to make the BLAS test programs. Otherwise, you must create a library using the files xBLAS1F, xBLAS2F, xBLAS3F, CB1AUXF, ZB1AUXF, 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.

F.1.4 Run the BLAS Test Programs

Test programs for the Level 1, 2, and 3 BLAS are in the files xBLAT1F, xBLAT2F, and xBLAT3F.

a) Compile the files xBLAT1F, 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 xBLAT1D, 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 1, 2, and 3 BLAS test programs.

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

F.1.5 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 A.3, you must compile it now and include it in your LAPACK library.

F.1.6 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.

F.1.7 Run the LAPACK Test Programs

There are two distinct test programs for LAPACK routines in each data type, one for the linear equations routines and one for the eigensystem routines. In each data type, there is one input file for testing the linear equation routines and fourteen input files for testing the eigenvalue routines. For more information on the test programs and how to modify the input files, see Section 6.

F.1.7.1 Testing the Linear Equation Routines

a) Compile the files xLINTSTF and either SCLNTSTF (for single precision real and complex) or DZLNTSTF (for double precision and double complex) and link them to your matrix generator library, your LAPACK library, and your BLAS library or libraries in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order).

b) The data files for the linear equation test program are called xLINTSTD. 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.

If you encountered failures in this phase of the testing process, please refer to Section 6.8.

F.1.7.2 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). If you have compiled either the S or C version, you must also compile and include the file SCIGTSTF, and if you have compiled either the D or Z version, you must also compile and include the file DZIGTSTF.

b) There are seventeen sets of data files for the eigensystem test program, xBAKTSTD, xBALTSTD, xECTSTD, xEDTSTD, xSBTSTD, xGGTSTD, xSGTSTD, NEPTSTD, SEPTSTD, SVDTSTD, GLMTSTD, GQRTSTD, GSVTSTD, LSETSTD, xGKT-STD, xGLTSTD, and xBBTSTD. Note that seven of the input files (NEPTSTD, SEPTSTD, SVDTSTD, GLMTSTD, GQRTSTD, GSVTSTD, and LSETSTD) 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, SBAKTST.OUT, etc.).

d) Run the test programs.

If you encountered failures in this phase of the testing process, please refer to Section 6.8.

F.1.8 Run the LAPACK Timing Programs

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

Two sets of input files are provided, a small set and a large set. The small data sets are appropriate for a standard workstation or other non-vector machine. The large data sets are appropriate for supercomputers, vector computers, and high-performance workstations. We are mainly interested in results from the large data sets, and it is not necessary to run both the large and small sets. The values of N in the large data sets are about five times larger than those in the small data set, and the large data sets use additional values for parameters such as the block size NB and the leading array dimension LDA. The small input files end with the four characters 'TIMD' and the large input files end with the characters 'TM2D' (except for the BLAS timing files, see Section A.8.2).

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

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

For more information on the timing programs and how to modify the input files, see Section 7.

If you encountered failures in this phase of the testing process, please refer to Section 6.8.

F.1.8.1 Timing the Linear Equations Routines

Three input files are provided in each data type for timing the linear equation routines, one for square matrices, one for band matrices, and one for rectangular matrices. The small data sets are in xLINTIMD, xBNDTIMD, and xRECTIMD, and the large data sets are in xLINTM2D, xBNDTM2D, and xRECTM2D.

a) Compile the files xLINSRCF and create an object library. If you have compiled either the S or C version, you must also compile and include the file SCINSRCF, and if you have compiled either the D or Z version, you must also compile and include the file DZINSRCF. If you did not compile the file ALLBLASF and include it in your BLAS library as described in Section A.3, you must compile it now and include it in the instrumented LAPACK library.

b) Compile the files xLINTIMF with ALINTIMF and link them to your test matrix generator library, the instrumented LAPACK library created in the previous step, your LAPACK library from Section A.5, and your BLAS library in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order). If you have compiled either the S or C version, you must also compile and include the file SCINTIMF, and if you have compiled either the D or Z version, you must also compile and include the file DZINTIMF.

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 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 A.8.2).

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

e) Run the timing programs in each data type you are using for each of the three input files.

F.1.8.2 Timing the BLAS

The linear equation timing program is also used to time the BLAS. Three input files are provided in each data type for timing the Level 2 and 3 BLAS. These input files time the BLAS using the matrix shapes encountered in the LAPACK routines, and we will use the results to analyze the performance of the LAPACK routines. For the REAL version, the small data sets are 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 A.8.1, set M, N, and K accordingly. The large parameters among M, N, and K should be the same as the matrix sizes used in timing the linear equation routines, and the small parameter should be the same as the 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.

F.1.8.3 Timing the Eigensystem Routines

Four input files are provided in each data type for timing the eigensystem routines, one for the generalized nonsymmetric eigenvalue problem, one for the nonsymmetric eigenvalue problem, one for the symmetric eigenvalue problem and generalized symmetric eigenvalue problem, and one for the singular value decomposition. For the REAL version, the small data sets are SGEPTIMD, SNEPTIMD, SSEPTIMD, and SSVDTIMD and the large data sets are SGEPTM2D, SNEPTM2D, SSEPTM2D, and SSVDTM2D. Each of the four input files reads a different set of parameters and the format of the input is indicated by a 3-character code on the first line.

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

a) Compile the files 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 A.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 A.5, and your BLAS library in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order). If you have compiled either the S or C version, you must also compile and include the file SCIGTIMF, and if you have compiled either the D or Z version, you must also compile and include the file DZIGTIMF.

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., SGEPTIM.OUT, SNEPTIM.OUT, SSEPTIM.OUT, and SSVDTIM.OUT for the four runs of the REAL version).

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

F.1.9 Send the Results to Tennessee

Congratulations! You have now finished installing, testing, and timing LAPACK. If you encountered failures in any phase of the testing or timing process, please consult our release_notes file on netlib.

`http://www.netlib.org/lapack/release_notes`

This file contains machine-dependent installation clues which hopefully will alleviate your difficulties or at least let you know that other users have had similar difficulties on that machine. If there is not an entry for your machine or the suggestions do not fix your problem, please feel free to contact the authors at

`lapack@cs.utk.edu`.

Tell us the type of machine on which the tests were run, the version of the operating system, the compiler and compiler options that were used, and details of the BLAS library or libraries that you used. You should also include a copy of the output file in which the failure occurs.

We would like to keep our `release_notes` file as up-to-date as possible. Therefore, if you do not see an entry for your machine, please contact us with your testing results.

Comments and suggestions are also welcome.

We encourage you to make the LAPACK library available to your users and provide us with feedback from their experiences.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, Second Edition, SIAM, Philadelphia, PA, 1995.

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

[3] C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, *LAPACK Working Note #5: Provisional Contents*, Argonne National Laboratory, ANL-88-38, September 1988.

[4] Z. Bai, J. Demmel, and A. McKenney, *LAPACK Working Note #13: On the Conditioning of the Nonsymmetric Eigenvalue Problem: Theory and Software*, University of Tennessee, CS-89-86, October 1989.

[5] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft.*, 16, 1:1-17, March 1990

[6] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs," *ACM Trans. Math. Soft.*, 16, 1:18-28, March 1990.

[7] 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.

[8] 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.

[9] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage," *ACM Trans. Math. Soft.*, 5, 3:308-323, September 1979.