[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.

A.9 Send the Results to Tennessee

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

sost@cs.utk.edu

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

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

If you would like to do more, please contact us so that we may coordinate your efforts with the development of the final test release of LAPACK. One option is to look at ways to improve the performance of LAPACK on your machine. If you do not have optimized BLAS, tuning the BLAS would likely have a dramatic effect on performance. Other suggestions on fine-tuning specific algorithms are also welcome. For example, one of our test sites noticed that the row interchanges in the LU factorization routine SGETRF were degrading performance on the IBM 3090 because of the non-unit stride in SSWAP [2]. In response we added the auxiliary routine SLASWP to interchange a block of rows, so that users of the IBM 3090 could easily replace this routine with one in which the row interchanges are applied to one column at a time.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *Preliminary LAPACK User's Guide*, University of Tennessee, July 1991.

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

[3] C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, *LAPACK Working Note #5: Provisional Contents*, Argonne National Laboratory, 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

A.8.3 Timing the Eigensystem Routines

Four input files are provided in each data type for timing the eigensystem routines, one for the nonsymmetric eigenvalue problem, one for the symmetric eigenvalue problem, one for the singular value decomposition, and one for the generalized nonsymmetric eigenvalue problem. For the REAL version, the small data sets are SGEPTIM.IN, SNEPTIM.IN, SSEPTIM.IN, and SSVDTIM.IN and the large data sets are SGEPTIM2.IN, SNEPTIM2.IN, SSEPTIM2.IN, and SSVDTIM2.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) Compile the files xEIGSRC.F and create an object library. If you have compiled either the S or C version, you must also compile and include the file SCIGSRC.F, and if you have compiled either the D or Z version, you must also compile and include the file DZIGSRC.F. If you did not compile the file ALLBLAS.F 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 xEIGTIM.F with AEIGTIM.F 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 SCIGTIM.F, and if you have compiled either the D or Z version, you must also compile and include the file DZIGTIM.F.

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 6 for further details.

   Associate the appropriate input file with Fortran unit number 5.

d) The output file is written to Fortran unit number 6. Associate a suitably named file with this unit number (e.g., 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) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

c) 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) Send the output files to the authors as directed in section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A.8.2 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 SBLTIMA, SBLTIMB, and SBLTIMC and the large data sets are SBLTIM2A, SBLTIM2B, and SBLTIM2C. 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 SBLTIMA, M and N are large but K is small, while in SBLTIMB the small parameter is M, and in SBLTIMC 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.

d) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

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 ‘TIM2D (except for the BLAS timing files, see Section A8.2).

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

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

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

A8.1 Timing the Linear Equations Routines

Three input files are provided in each data type for timing the linear equation routines, one for square matrices, one for band matrices, and one for rectangular matrices. The small data sets are in xLINTIMD, xBNDTIMD, and xRECTIMD, and the large data sets are in xLINTIM2D, xBNDTIM2D, and xRECTIM2D.

a) Compile the files xLINTIMF, and link them to your LAPACK library and your BLAS library or libraries in that order (on some systems you may get unsatisfied external references if you specify the libraries in the wrong order). If you have compiled either the S or C version, you must also compile and include the file SCINSTIF, and if you have compiled either the D or Z version, you must also compile and include the file DZINSTIF.

b) Make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value, or to restrict the size of the tests if you are using a computer with performance in between that of a workstation and that of a supercomputer. The computational requirements can be cut in half by using only one value of LDA. If it is necessary to also reduce the matrix sizes or the values of the blocksize, corresponding changes should be made to the BLAS input files (see Section A8.2).

   Associate the appropriate input file with Fortran unit number 5.

b) The data files for the linear equation test program are called xLINTST.D. For each of the test programs, associate the appropriate data file with Fortran unit number 5.

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

d) Run the test programs.

e) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A7.2 Testing the Eigensystem Routines

a) Compile the files 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 SCIGSTF, and if you have compiled either the D or Z version, you must also compile and include the file DZIGSTF.

b) There are ten sets of data files for the eigensystem test program, xBAKTST.D, xBALTST.D, xECTST.D, xEDTST.D, xGGTST.D, xSBTST.D, xSGTST.D, NEPTST.D, SEPTST.D, and SVDTST.D. Note that three of the input files (NEPTST.D, SEPTST.D, and SVDTST.D) 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.

e) Send the output files to the authors as directed in Section A.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

A.8 Run the LAPACK Timing Programs

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

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

c) The name of the output file is indicated on the first line of each input file and is currently defined to be SBLAT2.SUMM for the REAL Level 2 BLAS, with similar names for the other files. If necessary, edit the name of the output file to ensure that it is valid on your system.

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

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

A.5 Create the LAPACK Library

Compile the files xLASRC.F with ALLAUX.F and create an object library. If you have compiled either the S or C version, you must also compile and include the files SCLAUX.F, SLAMCH.F, and SECOND.F, and if you have compiled either the D or Z version, you must also compile and include the files DZLAUX.F, DLAMCH.F, and DSECND.F. If you did not compile the file ALLBLAS.F and include it in your BLAS library as described in Section A.3, you must compile it now and include it in your LAPACK library.

A.6 Create the Test Matrix Generator Library

Compile the files xMATGEN.F and create an object library. If you have compiled either the S or C version, you must also compile and include the file SCATGEN.F, and if you have compiled either the D or Z version, you must also compile and include the file DZATGEN.F.

A.7 Run the LAPACK Test Programs

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

A.7.1 Testing the Linear Equation Routines

a) Compile the files xLINTST.F 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).

A.2.3 Installing SECOND and DSECND

Both the timing routines and the test routines call SECOND (DSECND), a real function with no arguments that returns the time in seconds from some fixed starting time. Our version of this routine returns only "user time", and not "user time + system time". The version of second in SECOND.F calls ETIME, a Fortran library 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 TSECOND.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. Compile SECOND.F and TSECOND.F 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 DSECND.F and TDSECND.F and save SECOND and DSECND for inclusion in the LAPACK library in Section A.5.

A.3 Create the BLAS Library

Ideally, a highly optimized version of the BLAS library already exists on your machine. In this case you can go directly to Section A.4 to make the BLAS test programs. Otherwise, you must create a library using the files xBLAS1.F, xBLAS2.F, xBLAS3.F, and ALLBLAS.F. 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 ALLBLAS.F must be included if any part of xBLAS2.F or xBLAS3.F is used. Compile these files and create an object library.

A.4 Run the BLAS Test Programs

Test programs for the Level 2 and 3 BLAS are in the files xBLAT2.F and xBLAT3.F. A test program for the Level 1 BLAS is not included, in part because only a subset of the original set of Level 1 BLAS is actually used in LAPACK, and the old test program was designed to test the full set of Level 1 BLAS. The original Level 1 BLAS test program is available from netlib as TOMS algorithm 539.

a) Compile the files xBLAT2.F and xBLAT3.F 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 ALLBLAS.F and recompile the BLAS library.

b) Each BLAS test program has a corresponding data file xBLAT2.D or xBLAT3.D. Associate this file with Fortran unit number 5.

```
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 SLAMCH.F and TSLAMCH.F and run the test program. If the results from the test program are correct, save SLAMCH for inclusion in the LAPACK library. Repeat these steps with DLAMCH.F and TDLAMCH.F. 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 $1/(\text{overflow}) \geq (\text{underflow})$ it is recomputed as $(1/(\text{overflow})) * (1 + \varepsilon)$, where ε is the machine precision.

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

If you are working only in single precision, you do not need to install DLAMCH and DSECND; and if you are working only in double precision, you do not need to install SLAMCH and SECOND. These five subroutines and their test programs are provided in the files LSAME.F and TLSAME.F, SLAMCH.F and TSLAMCH.F, etc.

A.2.1 Installing LSAME

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

```
LSAME( UPLO, 'U' )
```

is equivalent to

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

The supplied version works correctly on all systems that use the ASCII code for internal representations of characters. For systems that use the EBCDIC code, one constant must be changed. For CDC systems with 6-12 bit representation, alternative code is provided in the comments. The test program in TLSAME.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.

Compile LSAME.F and TLSAME.F 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 ALLBLAS.F. 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.

A.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 TSLAMCH.F simply prints out the different values computed by SLAMCH, so you need to know something about what the values should be. For example, the output of the test program for SLAMCH on a Sun SPARCstation is

194. DNEPTIM2D
195. DSEPTIM2D
196. DSVDTIM2D

197. ZGEPTIM2D
198. ZNEPTIM2D
199. ZSEPTIM2D
200. ZSVDTIM2D

A Installing LAPACK on a non-Unix System

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

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

A.1 Read the Tape

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

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

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

159. SEIGTIMF Timing program for the eigensystem routines
160. CEIGTIMF
161. DEIGTIMF
162. ZEIGTIMF

163. SEIGSRCF Instrumented LAPACK routines
164. CEIGSRCF
165. DEIGSRCF
166. ZEIGSRCF

167. SCIGSRCF Instrumented auxiliary routines used in S and C versions
168. DZIGSRCF Instrumented auxiliary routines used in D and Z versions

169. SGEPTIMD Data file 1 for timing Generalized Nonsymmetric Eigenvalue Problem
170. SNEPTIMD Data file 1 for timing Nonsymmetric Eigenvalue Problem
171. SSEPTIMD Data file 1 for timing Symmetric Eigenvalue Problem
172. SSVDTIMD Data file 1 for timing Singular Value Decomposition

173. CGEPTIMD
174. CNEPTIMD
175. CSEPTIMD
176. CSVDTIMD

177. DGEPTIMD
178. DNEPTIMD
179. DSEPTIMD
180. DSVDTIMD

181. ZGEPTIMD
182. ZNEPTIMD
183. ZSEPTIMD
184. ZSVDTIMD

185. SGEPTIM2D Data file 2 for timing Generalized Nonsymmetric Eigenvalue Problem
186. SNEPTIM2D Data file 2 for timing Nonsymmetric Eigenvalue Problem
187. SSEPTIM2D Data file 2 for timing Symmetric Eigenvalue Problem
188. SSVDTIM2D Data file 2 for timing Singular Value Decomposition

189. CGEPTIM2D
190. CNEPTIM2D
191. CSEPTIM2D
192. CSVDTIM2D

193. DGEPTIM2D

123. ZBLTIMAD

124. SBLTIMBD Data file 1-b for timing the BLAS
125. DBLTIMBD
126. CBLTIMBD
127. ZBLTIMBD

128. SBLTIMCD Data file 1-c for timing the BLAS
129. DBLTIMCD
130. CBLTIMCD
131. ZBLTIMCD

132. SLINTIM2D Data file 2 for timing dense square linear equations
133. DLINTIM2D
134. CLINTIM2D
135. ZLINTIM2D

136. SRECTIM2D Data file 2 for timing dense rectangular linear equations
137. DRECTIM2D
138. CRECTIM2D
139. ZRECTIM2D

140. SBNDTIM2D Data file 2 for timing banded linear equations
141. DBNDTIM2D
142. CBNDTIM2D
143. ZBNDTIM2D

144. SBLTIM2AD Data file 2-a for timing the BLAS
145. DBLTIM2AD
146. CBLTIM2AD
147. ZBLTIM2AD

148. SBLTIM2BD Data file 2-b for timing the BLAS
149. DBLTIM2BD
150. CBLTIM2BD
151. ZBLTIM2BD

152. SBLTIM2CD Data file 2-c for timing the BLAS
153. DBLTIM2CD
154. CBLTIM2CD
155. ZBLTIM2CD

156. AEIGTIMF Auxiliary routines for the eigensystem timing program
157. SCIGTIMF
158. DZIGTIMF

87. SSGTSTD Data file for testing symmetric generalized eigenvalue routines
88. DSGTSTD
89. CSGTSTD
90. ZSGTSTD

91. AEIGTSTF Auxiliary routines for the eigensystem test program
92. SCIGTSTF
93. DZIGTSTF

94. SEIGTSTF Test program for eigensystem routines
95. CEIGTSTF
96. DEIGTSTF
97. ZEIGTSTF

98. NEPTSTD Data file for testing Nonsymmetric Eigenvalue Problem
99. SEPTSTD Data file for testing Symmetric Eigenvalue Problem
100. SVDTSTD Data file for testing Singular Value Decomposition

101. ALINTIMF Auxiliary routines for the linear system timing program
102. SCINTIMF
103. DZINTIMF

104. SLINTIMF Timing program for linear equations
105. CLINTIMF
106. DLINTIMF
107. ZLINTIMF

108. SLINTIMD Data file 1 for timing dense square linear equations
109. DLINTIMD
110. CLINTIMD
111. ZLINTIMD

112. SRECTIMD Data file 1 for timing dense rectangular linear equations
113. DRECTIMD
114. CRECTIMD
115. ZRECTIMD

116. SBNDTIMD Data file 1 for timing banded linear equations
117. DBNDTIMD
118. CBNDTIMD
119. ZBNDTIMD

120. SBLTIMAD Data file 1-a for timing the BLAS
121. DBLTIMAD
122. CBLTIMAD

52. DMATGENF
53. ZMATGENF

54. ALINTSTF Auxiliary routines for the linear equation test program

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

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

63. SBAKTSTD Data file for testing SGEBAK
64. DBAKTSTD Data file for testing DGEBAK
65. CBAKTSTD Data file for testing CGEBAK
66. ZBAKTSTD Data file for testing ZGEBAK

67. SBALTSTD Data file for testing SGEBAL
68. DBALTSTD Data file for testing DGEBAL
69. CBALTSTD Data file for testing CGEBAL
70. ZBALTSTD Data file for testing ZGEBAL

71. SECTSTD Data file for testing eigencondition routines
72. DECTSTD
73. CECTSTD
74. ZECTSTD

75. SEDTSTD Data file for testing nonsymmetric eigenvalue driver routines
76. DEDTSTD
77. CEDTSTD
78. ZEDTSTD

79. SGGTSTD Data file for testing nonsymmetric generalized eigenvalue routines
80. DGGTSTD
81. CGGTSTD
82. ZGGTSTD

83. SSBTSTD Data file for testing SSBTRD
84. DSBTSTD Data file for testing DSBTRD
85. CSBTSTD Data file for testing CHBTRD
86. ZSBTSTD Data file for testing ZHBTRD

17. DSECNDF DSECND function to return time in seconds
18. TDSECNDF Test program for DSECND

19. ALLBLASF Auxiliary routines for the BLAS (and LAPACK)

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

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

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

32. SBLAT2F Test program for Level 2 BLAS
33. CBLAT2F
34. DBLAT2F
35. ZBLAT2F

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

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

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

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

50. SMATGENF Test matrix generators
51. CMATGENF

Appendix F: Implementation Guide for Non-Unix Systems

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

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

The leading one or two characters of the file name generally indicates which of the different versions of the library or test 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. ISAMEF 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

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

Table 15: Cray YMP – 1 processor, execution times (in seconds)

Appendix E: Estimated Time

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

On the Sun, the Fortran files were compiled with `f77 -O`. Tests were done using two versions of the Sun-4 compiler, with an older version, `ctrti2.f`, `chetrs.f`, `cchol.f`, `zchol.f`, and `dstein.f` (from the LAPACK libraries) had to be compiled without optimization. On the Cray, the Fortran files were compiled with `cf77 -Zp` using cf77 5.0 and UNICOS 7.0.

Test/timing run	Data set	S	C	D	Z
Linear eqn testing	_test.in	439	2046	516	1921
Eigensystem testing	nep.in	43	282	69	314
	sep.in	147	647	267	726
	svd.in	210	1224	347	1341
	_ec.in	340	40	434	44
	_ed.in	154	509	238	621
	_gg.in	91	555	152	610
	_sg.in	55	355	87	340
	_sb.in	5	36	6	34
	_bal.in	<1	<1	<1	<1
	_bak.in	<1	<1	<1	<1
Linear eqn timing	_time.in	116	941	151	786
	_time2.in	136	1180	188	983
	_band.in	38	322	56	266
BLAS timing	_blas.in1	120	925	139	779
	_blas.in2	35	182	36	158
	_blas.in3	35	178	36	158
Eigensystem timing	_neptim.in	64	544	113	596
	_septim.in	41	887	80	907
	_svdtim.in	42	259	60	261
	_geptim.in	163	4239	322	1784

Table 14: Sun SPARCstation execution times (in seconds)

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

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

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

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

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

[2] ANSI X3.9-1978, sec. 15.9.3.6

Appendix D: Caveats

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

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

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

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

ZBEG	from the Level 2 BLAS test program
ZBEG	from the Level 3 BLAS test program
ZLADIV	from the LAPACK library
ZLARND	from the test matrix generator library
ZLATM2	from the test matrix generator library
ZLATM3	from the test matrix generator library

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

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

We have not included test programs for the Level 1 BLAS. Users should therefore beware of a common problem in machine-specific implementations of xNRM2, the function to compute the 2-norm of a vector. The Fortran version of xNRM2 avoids underflow or overflow by scaling intermediate results, but some library versions of xNRM2 are not so careful about scaling. If xNRM2 is implemented without scaling intermediate results, some of the LAPACK test ratios may be unusually high, or a floating point exception may occur in the problems scaled near underflow or overflow. The solution to these problems is to link the Fortran version of xNRM2 with the test program.

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

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

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

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:	$4mnk - 2(m+n)k^2 + 4/3k^3 + 3nk - mk - k^2 - 4/3k$

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

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 12: 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 13: Operation counts for the Level 3 BLAS

Appendix C: Operation Counts for the BLAS and LAPACK

In this appendix we reproduce in tabular form the formulas we have used to compute operation counts for the BLAS and LAPACK routines. In single precision, the functions SOPBL2, SOPBL3, SOPAUX, and SOPLA return the operation counts for the Level 2 BLAS, Level 3 BLAS, LAPACK auxiliary routines, and LAPACK routines, respectively. All four functions are found in the directory `LAPACK/TIMING/LIN` in the Unix version and in `SCINTSTF` in the non-Unix version.

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

Operation Counts for the Level 2 BLAS

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

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

SGBMV	multiplications:	$mn - (m - k_l - 1)(m - k_l)/2 - (n - k_u - 1)(n - k_u)/2$
	additions:	$mn - (m - k_l - 1)(m - k_l)/2 - (n - k_u - 1)(n - k_u)/2$
	total flops:	$2mn - (m - k_l - 1)(m - k_l) - (n - k_u - 1)(n - k_u)$

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

Operation Counts for the Level 3 BLAS

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

Operation Counts for the LAPACK Routines

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

_LAQSY equilibrate a symmetric matrix
_LAQTR solve a real or complex quasi-triangular system
_LAR2V apply real plane rotations from both sides to a sequence
of 2 by 2 real symmetric matrices
_LARF apply (multiply by) an elementary reflector
_LARFB apply (multiply by) a block reflector
_LARFG generate an elementary reflector
_LARFT form the triangular factor of a block reflector
_LARFX unrolled version of xLARF
_LARGV generate a vector of plane rotations
_LARTG generate a plane rotation
_LARTV apply a vector of plane rotations to a pair of vectors
_LAS2 (real) Compute singular values of a 2 x 2 triangular matrix
_LASCL scale a matrix by CTO/CFROM
_LASET initializes a matrix to BETA on the diagonal and ALPHA on
the offdiagonals
_LASR Apply a sequence of plane rotations to a rectangular matrix
_LASSQ Compute a scaled sum of squares of the elements of a vector
_LASV2 (real) Compute singular values and singular vectors of a 2 x 2 triangular
matrix
_LASWP Perform a series of row interchanges
_LASY2 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
_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
_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)
_LAXPY Add a multiple of a matrix to another matrix
_LAZRO Initialize a rectangular matrix (usually to zero)

_UNML2 (complex) multiply by the unitary matrix from xGELQF
_UNMR2 (complex) multiply by the unitary matrix from xGERQF

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
_LACON estimate the norm of a matrix for use in condition estimation
_LACPY copy a matrix to another matrix
_LADIV perform complex division in real arithmetic
_LAE2 compute eigenvalues of a 2 x 2 real symmetric or complex Hermitian matrix
_LAEBC chases a bulge down an upper Hessenberg block
_LAEBZ compute and use the count of eigenvalues of a symmetric tridiagonal matrix
_LAEIN Use inverse iteration to find a specified right and/or left eigenvector of an upper Hessenberg matrix
_LAEQU perform an orthogonal similarity transformation to standardize a 2 by 2 diagonal block of a quasi-triangular matrix
_LAEQZ unblocked single-/double-shift version of QZ method
_LAESY (complex) Compute eigenvalues and eigenvectors of a complex symmetric 2 x 2 matrix
_LAEV2 Compute eigenvalues and eigenvectors of a 2 x 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
_LAGBC bulge-chasing for the multishift QZ method
_LAGTF 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
_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 x 1 or 2 x 2 linear system
_LAORD sort the elements of a vector in increasing or decreasing order
_LAPTM multiply a matrix by a symmetric tridiagonal matrix
_LAPY2 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
_LAQSB equilibrate a symmetric band matrix
_LAQSP equilibrate a symmetric packed matrix

_SBMV (complex) Symmetric band 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
_GETF2 compute the LU factorization of a general matrix
_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
_PBTF2 compute the Cholesky factorization of a positive definite band matrix
_POTF2 compute the Cholesky factorization of a positive definite matrix
_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
_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

Appendix B: LAPACK Auxiliary Routines

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

Special subroutines:

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
_LANSB Symmetric band matrix
_LANSP Symmetric packed matrix
_LANST 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

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

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

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
LS   solve over- or underdetermined linear system using orthogonal factorizations
LSX   compute a minimum-norm solution using a complete orthogonal factorization (using QR with column pivoting)
LSS   solve least squares problem using the SVD
EV   compute all eigenvalues and/or eigenvectors
EVX   compute selected eigenvalues and eigenvectors
ES   compute all eigenvalues, Schur form, and/or Schur vectors
ESX   compute all eigenvalues, Schur form, and/or Schur vectors and the conditioning of selected eigenvalues or eigenvectors
GV   compute generalized eigenvalues and/or generalized eigenvectors
GS   compute generalized eigenvalues, Schur form, and/or Schur vectors
SVD   compute the SVD and/or singular vectors

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

	GE	GB	GT	PO	PP	PB	PT	HE SY	HP SP	HB SB	ST
SV	×	×	×	×	×	×	×	×	×		
SVX	×	×	×	×	×	×	×	×	×		
LS	×										
LSX	×										
LSS	×										
EV	×							×	×	×	×
EVX	×							×	×	×	×
ES	×										
ESX	×										
GV	×							×	×		
GS	×										
SVD	×										

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

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

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

† – also RQ, QL, and LQ

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

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

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
EIN compute selected eigenvectors by inverse iteration
EQR compute eigenvalues and/or the Schur form using the QR algorithm
EQU equilibrate a matrix to reduce its condition number
EQZ compute generalized eigenvalues and/or generalized Schur form by QZ method
ERF compute eigenvectors using the Pal-Walker-Kahan variant of the QL or QR
algorithm
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
LQS compute a minimum-norm solution using the LQ factorization
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
MTR multiply by the orthogonal/unitary matrix from xxxTRD
QLF compute a QL factorization without pivoting
QLS solve a least squares problem using the QL factorization
QPF compute a QR factorization with column pivoting
QRF compute a QR factorization without pivoting
QRS solve a least squares problem using the QR factorization
RFS refine initial solution returned by TRS routines
RQF compute an RQ factorization without pivoting
RQS compute a minimum-norm solution using the RQ factorization
SEN compute a basis and/or reciprocal condition number (sensitivity) of an
invariant subspace
SNA estimate reciprocal condition numbers of eigenvalue/-vector pairs
SQR compute singular values and/or singular vectors using the QR algorithm

Appendix A: LAPACK Routines

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

Each subroutine name in LAPACK is a coded specification of the computation done by the subroutine. All names consist of six characters in the form 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
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
OR (real) orthogonal
OP (real) orthogonal, packed storage
PB symmetric or Hermitian positive definite band
PO symmetric or Hermitian positive definite
PP symmetric or Hermitian positive definite, packed storage
PT symmetric or Hermitian positive definite 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

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

If NTYPES $>= 4$, all the types are used. If $0 <$ NTYPES < 4, then line 13 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 SHGEQZ routine, the table has one line for each different combination of NB, NS, MAXB, MINNB, and MINBLK.

Acknowledgments

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

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 $(\varepsilon, 1)$.

6.6.1 Input File for Timing the Generalized Nonsymmetric Eigenpr

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
10  10  10  10                   Values of NB (blocksize)
2   2   4   4                    Values of NS (no. of shifts)
200 2   4   4                    Values of MAXB (multishift crossover pt)
200 200 200 10                   Values of MINNB (minimum blocksize)
200 200 200 10                   Values of MINBLK (minimum blocksize)
201 201 201 201                  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:

2. SGGHRD(Q) (LAPACK reduction to generalized upper Hessenberg form, computing U but not V.)

3. SGGHRD(Z) (LAPACK reduction to generalized upper Hessenberg form, computing V but not U.)

4. SGGHRD(Q,Z) (LAPACK reduction to generalized upper Hessenberg form, computing U and V.)

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(A,L) (LAPACK computation of the the left generalized eigenvectors of a matrix pair in generalized Schur form)

11. STGEVC(B,L) (LAPACK computation of the the left generalized eigenvectors of a matrix pair in generalized Schur form, back transformed by Q)

12. STGEVC(A,R) (LAPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form)

13. STGEVC(B,R) (LAPACK computation of the the right generalized eigenvectors of a matrix pair in generalized Schur form, back transformed by Z)

14. QZHES(F) (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)

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 $0 < \text{NTYPES} < 5$, 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.

6.6 Timing the Generalized Nonsymmetric Eigenproblem

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

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

The parameters NB, MINNB, MINBLK, NS, and MAXB apply only to the QZ iteration routine xHGEQZ. A goal of this timing code is to determine the values of NB, NS, and MAXB (as well as MINNB and MINBLK) 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, MAXB, MINNB, or MINBLK	400
LDAMAX	Maximum value for LDA	420
MAXIN	Maximum number of values of N	12
MAXPRM	Maximum number of parameter sets (NB, NS, MAXB, MINNB, MINBLK, 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.)

13. LINSVD(1) (LINPACK singular values and min(M,N) left singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(1))

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

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

16. LINSVD(B) (LINPACK singular values, min(M,N) left singular vectors and N right singular vectors of a dense matrix using SSVDC, to be compared to LAPSVD(B)).

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

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

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

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

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

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

- 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
NMAX	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. LINSVD (LINPACK singular values only of a dense matrix using SSVDC, to be compared to LAPSVD)

- 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
50 100 200 300 400               Values of N (dimension)
5                                Number of values of parameters
1   16  32  48  64               Values of NB (blocksize)
401 401 401 401 401              Values of LDA (leading dimension)
0.0                              Minimum time in seconds
4                                Number of matrix types
SST    T T T T T T T T
```

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

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

6.5 Timing the Singular Value Decomposition

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

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

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

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

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

14. TRIDIB (EISPACK computation of the eigenvalues of)(compare with SSTEBZ – RANGE='I')

15. BISECT (EISPACK computation of the eigenvalues of)(compare with SSTEBZ – RANGE='V)

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

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

9. 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).

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

- 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. SSTEQR(N) (LAPACK computation of eigenvalues only of a symmetric tridiagonal matrix)
3. SSTEQR(V) (LAPACK computation of the eigenvalues and eigenvectors of a symmetric tridiagonal matrix)
4. SSTERF (LAPACK computation of the eigenvalues only of a symmetric tridiagonal matrix using a square-root free algorithm)
5. SPTEQR(COMPZ='N') (LAPACK computation of the eigenvalues of a symmetric positive definite tridiagonal matrix)
6. SPTEQR(COMPZ='V') (LAPACK computation of the eigenvalues and eigenvectors of a symmetric positive definite tridiagonal matrix)
7. SSTEBZ(RANGE='I') (LAPACK computation of the eigenvalues in a specified interval for a symmetric tridiagonal matrix)
8. SSTEBZ(RANGE='V') (LAPACK computation of the eigenvalues in a half-open interval for a symmetric tridiagonal matrix)
9. SSTEIN (LAPACK computation of the eigenvectors of a symmetric tridiagonal matrix corresponding to specified eigenvalues using inverse iteration)
10. TRED1 (EISPACK reduction to symmetric tridiagonal form, to be compared to SSYTRD)

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

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

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

6.4 Timing the Symmetric Eigenproblem

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

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
50 100 200 300                    Values of N (dimension)
```

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

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

6.3 Timing the Nonsymmetric Eigenproblem

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

- N, the matrix size
- four-tuples of parameter values (NB, NS, 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)

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

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

6.2 Timing the Level 2 and 3 BLAS

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

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

```
BLAS timing, REAL data, K small
5                           Number of values of M
100 200 300 400 500         Values of M
5                           Number of values of N
100 200 300 400 500         Values of N
5                           Number of values of K
2 16 32 48 64               Values of K
1                           Number of values of INCX
```

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

The first 13 lines of the input file are read using list-directed input and are used to specify the values of M, N, K, NB, NX, LDA, and 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
1000                              Values of M (row dimension)
0                                 Number of values of N
200                               Values of N (column dimension)
5                                 Number of values of K
25 50 100 150 200                 Values of K (bandwidth)
5                                 Number of values of NB
1 16  32  48  64                  Values of NB (blocksize)
0 48 128 128 128                  Values of NX (crossover point)
2                                 Number of values of LDA
601 602                           Values of LDA (leading dimension)
0.0                               Minimum time in seconds
BAND                              Time sample banded BLAS
SGB
SPB
STB
```

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

We also provide a separate input file for timing the orthogonal factorization and reduction routines that operate on rectangular matrices. For these routines, the values of M and block variants of the LU factorization (left-looking, Crout, and right-looking) for $N \times N$ matrices, as well as the corresponding unblocked variants on matrices of size $N \times NB$ and the Linpack routine xGEFA. The xCH timing path requests timing of three block variants of the Cholesky factorization and the corresponding Linpack routine xPOFA. The LU and Cholesky variants are not strictly part of LAPACK and will not be included in the public release.

The timing programs have their own matrix generator that supplies random Toeplitz matrices (constant along a diagonal) for many of the timing paths. Toeplitz matrices are used because they can be generated more quickly than dense matrices, and the call to the matrix generator is inside the timing loop. The 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
100 200 300 400 500               Values of M (row dimension)
5                                 Number of values of N
100 200 300 400 500               Values of N (column dimension)
2                                 Number of values of K
100 400                           Values of K
5                                 Number of values of NB
1 16  32  48  64                  Values of NB (blocksize)
0 48 128 128 128                  Values of NX (crossover point)
2                                 Number of values of LDA
512 513                           Values of LDA (leading dimension)
0.0                               Minimum time in seconds
SGE    T T T
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 F
SLQ    T T F
SQL    T T F
```

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 $(\text{LDA}+K)M \le 3(\text{LDAMAX})(\text{NMAX})$, where $\text{LDA} \ge 3K+1$, 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 $(2K+1)M \le 3(\text{NMAX})(\text{LDAMAX})$.

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} LU	Variants of the LU factorization
{S, C, D, Z} CH	Variants of the Cholesky factorization

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

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

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

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

The main timing procedure for the REAL linear equation routines is found in `LAPACK/TIMING/LIN/stimaa.f` in the Unix version and is the first program unit in STIMF 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.

6.1 The Linear Equation Timing Program

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

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

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

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

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

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

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

6 More About Timing

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

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

When the test 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=18, seed=2548,1429,1713,1411, result  8 is   11.33
```

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

The normalization of the eigenvectors will also be checked. If the absolute value of the largest entry in an eigenvector is not within $\varepsilon \times$ 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.
```

5.6.5 Tests Performed on the Generalized Nonsymmetric Eigenvalu

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

$$r_1 = \frac{\left\|A - QSZ^T\right\|}{\|A\|\, n\varepsilon} \qquad r_2 = \frac{\left\|B - QTZ^T\right\|}{\|B\|\, n\varepsilon}$$

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

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

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

For SGEGV the following tests are computed:

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

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

5.6.6 Input File for Testing the Generalized Nonsymmetric Eigen tines and Drivers

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

5.6.3 Test Matrices for the Generalized Nonsymmetric Eigenvalue

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

5.6.4 Tests Performed on the Generalized Nonsymmetric Eigenvalue

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

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

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

$$r_1 = \frac{\|A - UHV^*\|}{n\varepsilon\,\|A\|} \qquad r_2 = \frac{\|B - UTV^*\|}{n\varepsilon\,\|B\|}$$

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

$$r_5 = \frac{\|H - QSZ^*\|}{n\varepsilon\,\|H\|} \qquad r_6 = \frac{\|T - QPZ^*\|}{n\varepsilon\,\|T\|}$$

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

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

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

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

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

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

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

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

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

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

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

Arithmetic: Difference between adjacent eigenvalues is a constant.

Geometric: Ratio of adjacent eigenvalues is a constant.

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

Random: Eigenvalues are logarithmically distributed.

Random entries: Matrix entries are uniformly distributed random numbers.

5.6.1 The Generalized Nonsymmetric Eigenvalue Drivers

The driver routines for the generalized nonsymmetric eigenvalue problem are

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

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

5.6.2 Test Matrices for the Generalized Nonsymmetric Eigenvalue

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

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

The following symbols and abbreviations are used:

0: The zero matrix.

I**:** The identity matrix.

ω**:** Generally, the underflow 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.

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

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).

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

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

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

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

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

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

5.5.6 Input File for Testing the Singular Value Decomposition Ro

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

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

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

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

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

$$r_3 = \frac{||I - P^*P||}{n\varepsilon} \qquad r_4 = \frac{||B - U\Sigma V^*||}{\tilde{n}\varepsilon\,||B||}$$

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

$$r_6 = \frac{||I - U^*U||}{\tilde{n}\varepsilon} \qquad r_7 = \frac{||I - VV^*||}{\tilde{n}\varepsilon}$$

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

$$r_9 = \begin{cases} 0 & \text{if eigenvalues of } B \text{ are within } THRESH \text{ of those in } S1. \\ 2 * THRESH & \text{otherwise} \end{cases}$$

$$r_{10} = \frac{||S1 - S2||}{\varepsilon\,||S1||} \qquad r_{11} = \frac{||A - (QU)\Sigma(PV)^*||}{\tilde{n}\varepsilon\,||A||}$$

$$r_{12} = \frac{||X - (QU)Z||}{\max(m, k)\varepsilon\,||X||} \qquad r_{13} = \frac{||I - (QU)^*(QU)||}{m\varepsilon}$$

$$r_{14} = \frac{||I - (VP)(VP)^*||}{n\varepsilon}$$

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

5.5.5 Tests Performed on the Singular Value Decomposition Driver

For the driver routine, the following tests are computed:

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

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

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

5.5.3 Test Matrices for the Singular Value Decomposition Driver

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

	Eigenvalue Distribution				
Type	Arithmetic	Geometric	Clustered	Random	Other
Zero					1
Identity					2
UDV	3, $4^{\dagger}$, $5^{\ddagger}$				

$\dagger$ – matrix entries are multiplied by the underflow-threshold$/\varepsilon$
$\ddagger$ – matrix entries are multiplied by the overflow-threshold $* \varepsilon$

Table 9: Test matrices for the singular value decomposition driver

5.5.4 Tests Performed on the Singular Value Decomposition Routines

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

1. A is decomposed as QBP^*, where Q and P are unitary and B is real bidiagonal.
2. B is decomposed as $U \Sigma V$, where U and V are real orthogonal and Σ is a positive real diagonal matrix of singular values. 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	$m \times n$
Q	$m \times \tilde{n}$ (but $m \times m$ if NRHS > 0)
P	$\tilde{n} \times n$
B	$\tilde{n} \times \tilde{n}$
U, V	$\tilde{n} \times \tilde{n}$
$S1$, $S2$	diagonal, order $\tilde{n}$
X	$m \times$ NRHS

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

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

5.5.1 The Singular Value Decomposition Driver

The driver routine for the singular value decomposition is

xGESVD singular value decomposition of A

5.5.2 Test Matrices for the Singular Value Decomposition Routines

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

	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

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

Table 8: Test matrices for the singular value decomposition

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

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

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

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

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

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

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

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

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

5.5 Testing the Singular Value Decomposition Routines

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

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

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

$$\begin{aligned}
r_{13} &= \max_i \frac{\|D4(i) - WR(i)\|}{\|D4(i)\|\,\omega}, \\
&\quad \text{where} \quad \omega = 2(2n-1)\varepsilon \frac{1+8*\gamma^2}{(1-\gamma)^4} \\
r_{14} &= \frac{\|WA1 - D3\|}{\varepsilon\,\|D3\|} \\
r_{15} &= \frac{\max_i(\min_j(\|WA2(i) - WA3(j)\|)) + \max_i(\min_j(\|WA3(i) - WA2(j)\|))}{\varepsilon\,\|D3\|} \\
r_{16} &= \frac{\|S - YWA1Y^*\|}{n\varepsilon\,\|S\|} \\
r_{17} &= \frac{\|I - YY^*\|}{n\varepsilon}
\end{aligned}$$

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.

5.4.5 Tests Performed on the Symmetric Eigenvalue Drivers

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

$$\begin{aligned}
r_1 &= \frac{\|A - ZDZ^*\|}{n\varepsilon\,\|A\|} \\
r_2 &= \frac{\|I - ZZ^*\|}{n\varepsilon} \\
r_3 &= \frac{\|D1 - D2\|}{\varepsilon\,\|D1\|}
\end{aligned}$$

where Z is the matrix of eigenvectors returned when the eigenvector option is given and $D1$ and $D2$ are the eigenvalues returned with and without the eigenvector option.

5.4.6 Input File for Testing the Symmetric Eigenvalue Routines a

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
7                                   Number of values of N
```

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.

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

$$r_1 = \frac{||A - VSV^*||}{n\varepsilon\,||A||}$$
computed by $SSYTRD(UPLO = 'U')$

$$r_2 = \frac{||I - UV^*||}{n\varepsilon}$$
test of $SORGTR(UPLO = 'U')$

$$r_3 = \frac{||A - VSV^*||}{n\varepsilon\,||A||}$$
computed by $SSYTRD(UPLO = 'L')$

$$r_4 = \frac{||I - UV^*||}{n\varepsilon}$$
test of $SORGTR(UPLO = 'L')$

$$r_5 = \frac{||S - ZD1Z^*||}{n\varepsilon\,||S||}$$

$$r_6 = \frac{||I - ZZ^*||}{n\varepsilon}$$

$$r_7 = \frac{||D1 - D2||}{\varepsilon\,||D1||}$$

$$r_8 = \frac{||D1 - D3||}{\varepsilon\,||D1||}$$

$$r_9 = \begin{cases} 0 & \text{if eigenvalues of } S \text{ are within } THRESH \text{ of those in } D1. \\ 2 * THRESH & \text{otherwise} \end{cases}$$

For S positive definite,

$$r_{10} = \frac{||S - Z4D4Z4^*||}{n\varepsilon\,||S||}$$

$$r_{11} = \frac{||I - Z4Z4^*||}{n\varepsilon}$$

$$r_{12} = \frac{||D4 - D5||}{100\varepsilon\,||D4||}$$

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^{*}, $19^{\star}$, $20^{\bullet}$	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 6: Test matrices for the symmetric eigenvalue problem

Type	Eigenvalue Distribution			
	Arithmetic	Geometric	Clustered	Other
Zero				1
Identity				2
Diagonal	3	4, $6^{\dagger}$, $7^{\ddagger}$	5	
UDU^{-1}	8, $11^{\dagger}$, $12^{\ddagger}$	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 7: Test matrices for the symmetric eigenvalue drivers

5.4.4 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 $ZD1Z^*$, 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.

- 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.

5.4.1 The Symmetric Eigenvalue Drivers

The driver routines for the symmetric eigenvalue problem are

xSTEV eigenvalue/eigenvector driver for symmetric tridiagonal matrix,

xSTEVX selected eigenvalue/eigenvectors for symmetric tridiagonal matrix,

xSYEV eigenvalue/eigenvector driver for symmetrix matrix,

xSYEVX selected eigenvalue/eigenvectors for symmetric matrix,

xSPEV eigenvalue/eigenvector 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,

xSBEVX selected eigenvalue/eigenvectors for symmetric band matrix.

5.4.2 Test Matrices for the Symmetric Eigenvalue Routines

Twenty-one different types of test matrices may be generated for the symmetric eigenvalue routines. Table 6 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).

5.4.3 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 6. Table 7 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).

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.

5.3 Testing the Nonsymmetric Eigenvalue Condition Estima tines

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].

5.4 Testing the Symmetric Eigenvalue Routines

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

5.2.7 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
15.0                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

```
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.

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).

5.2.6 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)
```

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.

5.2.5 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:

$$r_1 = \frac{||A \cdot VR - VR \cdot W||}{n\epsilon||A||} \qquad r_2 = \frac{||A' \cdot VL - VL \cdot W||}{n\epsilon||A||}$$

$$r_3 = \frac{|||VR_i|| - 1|}{\epsilon} \qquad r_4 = \frac{|||VL_i|| - 1|}{\epsilon}$$

$$r_5 = (W(full) = W(partial)) \qquad r_6 = (VR(full) = VR(partial))$$

$$r_7 = (VL(full) = VL(partial))$$

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.

(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

5.2.3 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

5.2.4 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.

5.2.1 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).

5.2.2 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.

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

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

Table 5: Test matrices for the nonsymmetric eigenvalue problem

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

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, ratio =   1.14270
```

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

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

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

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

5.2 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	132
MAXIN	Maximum number of values of the parameters	20

For the nonsymmetric eigenvalue input file, MAXIN is both the maximum number of values of N and the maximum number of 3-tuples (NB, NS, MAXB). Similar restrictions exist for the other input files for the eigenvalue test program.

```
1 3 3 3 20            Values of NB (the blocksize)
1 0 5 9 1             Values of NX (crossover point)
20.0                  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
SGB    8
SGT   12
SPO    9
SPP    9
SPB    8
SPT   12
SSY   10
SSP   10
STR   18
STP   18
STB   17
SQR    8
SRQ    8
SLQ    8
SQL    8
SQP    6
STZ    3
SLS    6
SEQ
SLU   11
SCH    9
```

The first 14 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    8
```

requests all of the matrix types for path SGE, while

```
SGE    3
 4 5 6
```

- 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, and

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

for SGELSS.

- 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

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

5.1.5 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.

5.1.6 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 equation routines
7                         Number of values of M
0 1 2 3 5 10 16 100       Values of M (row dimension)
7                         Number of values of N
0 1 2 3 5 10 16 100       Values of N (column dimension)
1                         Number of values of NRHS
2                         Values of NRHS (number of right hand sides)
5                         Number of values of NB
```

- 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

$$||I - Z^T Z||/(m\epsilon)$$

5.1.4 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 $m \times n$). 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 $m \times n$ 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 and SGELSS drivers solve a possibly rank-deficient system $AX = B$ using a complete orthogonal factorization (SGELSX) or singular value decomposition (SGELSS), 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 computes a complete orthogonal factorization $AP = QTZ$, whereas SGELSS computes the singular value decomposition $A = U\mathrm{diag}(\sigma)V^T$.

- 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), 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, 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), which decomposes an $m \times n (m < n)$ 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, compute the singular values $\tilde{s}$ of T using SGEBD2 and SBDSQR, and compute the ratio

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

5.1.3 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 $m \times n$ 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.

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

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

Table 2: Tests performed for general and symmetric linear systems

5.1.2 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 $A^T x = sb$, 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	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

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.

- 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 after forming the inverse. The larger of the ratios

  RCOND/RCONDC and RCONDC/RCOND

{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
{S, C, D, Z}	LU	LU variants
{S, C, D, Z}	CH	Cholesky variants

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 elmentary Householder transformations.

5.1.1 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

5 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 13 different sets of input, although four of these may be concatenated into one data set, for a total of 10 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.

5.1 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)

4.9 Send the Results to Tennessee

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

sost@cs.utk.edu

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

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

If you would like to do more, please contact us so that we may coordinate your efforts with the development of the final test release of LAPACK. One option is to look at ways to improve the performance of LAPACK on your machine. If you do not have optimized BLAS, tuning the BLAS would likely have a dramatic effect on performance. Other suggestions on fine-tuning specific algorithms are also welcome. For example, one of our test sites noticed that the row interchanges in the LU factorization routine SGETRF were degrading performance on the IBM 3090 because of the non-unit stride in SSWAP [2]. In response we added the auxiliary routine SLASWP to interchange a block of rows, so that users of the IBM 3090 could easily replace this routine with one in which the row interchanges are applied to one column at a time.

and `LOADOPTS` to refer to the loader and desired load options for your machine. Then type `make` followed by the data types desired, as in the examples of Section 3.5. The library of instrumented code is created in `LAPACK/TIMING/EIG/eigsrc.a`.

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

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

d) Go to `LAPACK/TIMING` and make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value, or to restrict the number of tests if you are using a computer with performance in between that of a workstation and that of a supercomputer. Instead of decreasing the matrix dimensions to reduce the time, it would be better to reduce the number of matrix types to be timed, since the performance varies more with the matrix size than with the type. For example, for the nonsymmetric eigenvalue routines, you could use only one matrix of type 4 instead of four matrices of types 1, 3, 4, and 6. See Section 6 for further details.

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

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

or the commands for the large data sets are

```
xeigtims < SNEPTIM.in > SNEPTIM.out
xeigtims < SSEPTIM.in > SSEPTIM.out
xeigtims < SSVDTIM.in > SSVDTIM.out
xeigtims < SGEPTIM.in > SGEPTIM.out
```

Similar commands should be used for the other data types.

f) Send the output files to the authors as directed in Section 4.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

a) Go to `LAPACK/TIMING` and make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value. If you modified the values of N or NB in Section 4.8.1, set M, N, and K accordingly. The large parameters among M, N, and K should be the same as the matrix sizes used in timing the linear equation routines, and the small parameter should be the same as the blocksizes used in timing the linear equation routines. If necessary, the large data set can be simplified by using only one value of LDA.

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

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

or the commands for the large data sets are

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

Similar commands should be used for the other data types.

c) Send the output files to the authors as directed in Section 4.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

4.8.3 Timing the Eigensystem Routines

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 nonsymmetric eigenvalue problem, one for the symmetric eigenvalue problem, one for the singular value decomposition, and one for the generalized nonsymmetric eigenvalue problem. For the REAL version, the small data sets are called `sneptim.in`, `sseptim.in`, `ssvdtim.in`, and `sgeptim.in`, respectively, and the large data sets are called `SNEPTIM.in`, `SSEPTIM.in`, `SSVDTIM.in`, and `SGEPTIM.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 edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` desired load options for your machine. If you are not using the Fortran BLAS, define BLAS to point to your system's BLAS library, instead of `../../blas.a`.

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

c) Go to `LAPACK/TIMING` and make any necessary modifications to the input files. You may need to set the minimum time a subroutine will be timed to a positive value, or to restrict the size of the tests if you are using a computer with performance in between that of a workstation and that of a supercomputer. The computational requirements can be cut in half by using only one value of LDA. If it is necessary to also reduce the matrix sizes or the values of the blocksize, corresponding changes should be made to the BLAS input files (see Section 4.8.2).

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

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

or the commands for the large data sets are

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

Similar commands should be used for the other data types.

e) Send the output files to the authors as directed in Section 4.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

4.8.2 Timing the BLAS

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 `sblas.in1`, `sblas.in2`, and `sblas.in3` and the large data files are `SBLAS.in1`, `SBLAS.in2`, and `SBLAS.in3`. There are three sets of inputs because there are three parameters in the Level 3 BLAS, M, N, and K, and in most applications one of these parameters is small (on the order of the blocksize) while the other two are large (on the order of the matrix size). In `sblas.in1`, M and N are large but K is small, while in `sblas.in2` the small parameter is M, and in `sblas.in3` the small parameter is N. The Level 2 BLAS are timed only in the first data set, where K is also used as the bandwidth for the banded routines.

4.8 Run the LAPACK Timing Programs

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

Two sets of input files are provided, a small set 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 6.

4.8.1 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`.

a) To make the linear equation timing programs, go to `LAPACK/TIMING/LIN` and edit the makefile. Define `FORTRAN` and `OPTS` to refer to the compiler and desired compiler options for your machine, and define `LOADER` and `LOADOPTS` to refer to the loader and

d) Send the output files to the authors as directed in Section 4.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

4.7.2 Testing the Eigensystem Routines

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

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

c) Go to LAPACK/TESTING and run the tests for each data type. The tests for the eigensystem routines use ten separate input files, for testing the generalized nonsymmetric eigenvalue problem routines, the nonsymmetric eigenvalue problem drivers, the nonsymmetric eigenvalue problem routines, the symmetric eigenvalue problem routines, and the singular value decomposition routines. 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 < ssg.in > ssg.out
xeigtsts < ssb.in > ssb.out
xeigtsts < sbal.in > sbal.out
xeigtsts < sbak.in > sbak.out
```

The tests using xeigtstc, xeigtstd, and xeigtstz also use the input files nep.in, sep.in, and svd.in, but the leading 's' in the other input file names must be changed to 'c', 'd', or 'z'. We have shown the output of these ten tests going to ten different output files, but we would prefer to receive one file containing the results of all the tests.

d) Send the output files to the authors as directed in Section 4.9. Please tell us the type of machine on which the tests were run, the compiler options that were used, and details of the BLAS library or libraries that you used.

4.5 Create the LAPACK Library

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

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

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

4.6 Create the Test Matrix Generator Library

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

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

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

4.7 Run the LAPACK Test Programs

There are two distinct test programs for LAPACK routines in each data type, one for the linear equation routines and one for the eigensystem routines. In each data type, there is one input file for testing the linear equation routines and ten 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 5.

4.7.1 Testing the Linear Equations Routines

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

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

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

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

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

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

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

4.4 Run the BLAS Test Programs

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

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

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

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

```
xblat2s < sblat2.in
```

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

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

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

'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 $1/(\text{overflow}) \geq (\text{underflow})$ it is recomputed as $(1/(\text{overflow})) * (1 + \varepsilon)$, where ε is the machine precision.

4.2.3 Installing SECOND and DSECND

Both the timing routines and the test routines call SECOND (DSECND), a real function with no arguments that returns the time in seconds from some fixed starting time. Our version of this routine returns only "user time", and not "user time + system time". The version of SECOND in `second.f` calls ETIME, a Fortran library 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 `secondtst.f` performs a million operations using 5000 iterations of the SAXPY operation $y := y + \alpha x$ on a vector of length 100. The total time and megaflops for this test is reported, then the operation is repeated including a call to SECOND on each of the 5000 iterations to determine the overhead due to calling SECOND. Run the test program by typing `testsecond` (or `testdsecnd`). There is no single right answer, but the times in seconds should be positive and the megaflop ratios should be appropriate for your machine. If you modify SECOND or DSECND, copy `second.f` and/or `dsecnd.f` to `LAPACK/SRC/` for inclusion in the LAPACK library.

4.3 Create the BLAS Library

Ideally, a highly optimized version of the BLAS library already exists on your machine. In this case you can go directly to Section 4.4 to make the BLAS test programs. You may already have a library containing some of the BLAS, but not all (Level 1 and 2, but not Level 3, for example). If so, you should use your local version of the BLAS wherever possible.

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

```
ASCII character set
Tests completed
```

If any modifications were required to LSAME, copy `lsame.f` to both `LAPACK/BLAS/SRC/` and `LAPACK/SRC/`. The function LSAME is needed by both the BLAS and LAPACK, so it is safer to have it in both libraries as long as this does not cause trouble in the link phase when both libraries are used.

4.2.2 Installing SLAMCH and DLAMCH

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

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

Run the test program by typing `testslamch`. If any modifications were made to SLAMCH, copy `slamch.f` to `LAPACK/SRC/`. Do the same for DLAMCH and the test program `testdlamch`. If both tests were successful, go to Section 4.2.3.

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

'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) requirements will be less if you do not use all four data types. The total space requirements including the object files is approximately 70 MB for all four data types.

If you received a tar file of LAPACK via the file transfer program `ftp`, enter the following command to untar the file:

`tar xvf` *file* `(where` *file* `is the name of the tar file)`

Since single precision (REAL+COMPLEX) and double precision (DOUBLE PRECISION +COMPLEX*16) are separated into two separate tar files, the space requirements for each tar file will be half what they are for the tape.

4.2 Test and Install the Machine-Dependent Routines.

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

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

If you are working only in single precision, you do not need to install DLAMCH and DSECND, and if you are working only in double precision, you do not need to install SLAMCH and SECOND.

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

4.2.1 Installing LSAME

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

```
LSAME( UPLO, 'U' )
```

is equivalent to

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

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

Run the test program by typing `testlsame`. If LSAME works correctly, the only message you should see is

```
xeigtsts < ssb.in > ssb.out
```

repeat for c, d, and z (except for nep.in, sep.in, and svd.in, replace the leading s in the input file by c, d, or z, respectively)

8. Run the LAPACK Timing Programs

```
cd LAPACK/TIMING/LIN
make
cd LAPACK/TIMING
xtims < stime.in > stime.out
xtims < sband.in > sband.out
xtims < stime2.in > stime2.out
```

repeat for c, d, and z

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

repeat for c, d, and z

```
cd LAPACK/TIMING/EIG/EIGSRC
make
cd LAPACK/TIMING/EIG
make
cd LAPACK/TIMING
xeigtims < sgeptim.in > sgeptim.out
xeigtims < sneptim.in > sneptim.out
xeigtims < sseptim.in > sseptim.out
xeigtims < ssvdtim.in > ssvdtim.out
```

repeat for c, d, and z

4.1 Read the Tape or Untar the File

If you received a tar tape of LAPACK, type one of the following commands to unload the tape (the device name may be different at your site):

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

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

This will create a top-level directory called `LAPACK`. You will need about 28 megabytes to read in the complete tape. On a Sun SPARCstation, the libraries used 14 MB and the LAPACK executable files used 20 MB. In addition, the object files used 18 MB, but the object files can be deleted after creating the libraries and executable files. Your actual space

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

5. Create the LAPACK Library

```
cp LAPACK/INSTALL/lsame.f LAPACK/SRC/
cp LAPACK/INSTALL/slamch.f LAPACK/SRC/
cp LAPACK/INSTALL/dlamch.f LAPACK/SRC/
cp LAPACK/INSTALL/second.f LAPACK/SRC/
cp LAPACK/INSTALL/dsecnd.f LAPACK/SRC/
cd LAPACK/SRC
make
```

6. Create the Library of Test Matrix Generators

```
cd LAPACK/TESTING/MATGEN
make
```

7. Run the LAPACK Test Programs

```
cd LAPACK/TESTING/LIN
make
cd LAPACK/TESTING
xchks < stest.in > stest.out
xchkd < dtest.in > dtest.out
xchkc < ctest.in > ctest.out
xchkz < ztest.in > ztest.out
cd LAPACK/TESTING/EIG
make
cd LAPACK/TESTING
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 < ssg.in > ssg.out
xeigtsts < sgg.in > sgg.out
xeigtsts < sbal.in > sbal.out
xeigtsts < sbak.in > sbak.out
```

Quick Reference Guide for Installation of LAPACK

If you insist on not reading the instructions, here is an abbreviated set of directions for installing and testing LAPACK.

To install, test, and time LAPACK:

1. Read the tape or untar the file.

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

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

   `tar xvf file` (from a file)

2. Test and Install the Machine-Dependent Routines

```
cd LAPACK/INSTALL
make
testlsame
testslamch
testdlamch
testsecond
testdsecnd
```

3. Create the BLAS Library, *if necessary*
   (NOTE: For best performance, it is recommended you use the manufacturers' BLAS)

```
cp LAPACK/INSTALL/lsame.f LAPACK/BLAS/SRC/
cd LAPACK/BLAS/SRC
make
```

4. Run the Level 2 and 3 BLAS Test Programs

```
cd LAPACK/BLAS/TESTING
make -f makeblat2
cd LAPACK/BLAS
xblat2s < sblat2.in
xblat2d < dblat2.in
xblat2c < cblat2.in
xblat2z < zblat2.in
cd LAPACK/BLAS/TESTING
make -f makeblat3
cd LAPACK/BLAS
xblat3s < sblat3.in
```

without any options creates a library of all four data types. The `make` command can be run more than once to add another data type to the library if necessary. Because of the quantity of software in LAPACK, compiling all four data types into one library may not be advisable; see Appendix D for alternate suggestions.

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

```
make single

make double

. . .
```

or all at once:

```
make single double complex complex16
```

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

```
make -f makeblat2 single
```

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

4 Installing LAPACK on a Unix System

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

1. Read the tape or untar 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.
9. Send the results from steps 7 and 8 to the authors at the University of Tennessee.

use it (but be sure to run the BLAS test programs). If an optimized library of the BLAS is not available, Fortran source code for the Level 1, 2, and 3 BLAS is provided on the tape. Users should not expect too much from the Fortran BLAS; these versions were written to define the basic operations and do not employ the standard tricks for optimizing Fortran code.

The formal definitions of the Level 1, 2, and 3 BLAS are in [9], [7], and [5]. Copies of the BLAS Quick Reference card are available from the authors.

3.3 LAPACK Test Routines

This release contains two distinct test programs for LAPACK routines in each data type. One test program tests the routines for solving linear equations and linear least squares problems, 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.

3.4 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.

3.5 makefiles

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

```
make single

make double complex16

make single double complex complex16
```

Alternatively,

```
make
```

3 Overview of Tape Contents

Most routines in LAPACK occur in four versions: REAL, DOUBLE PRECISION, COMPLEX, and COMPLEX*16. The first three versions (REAL, DOUBLE PRECISION, and COMPLEX) are written in standard Fortran 77 and are completely portable; the COMPLEX*16 version is provided for those compilers which allow this data type. For convenience, we often refer to routines by their single precision names; the leading 'S' can be replaced by a 'D' for double precision, a 'C' for complex, or a 'Z' for complex*16. For LAPACK use and testing you must decide which version(s) of the package you intend to install at your site (for example, REAL and COMPLEX on a Cray computer or DOUBLE PRECISION and COMPLEX*16 on an IBM computer).

3.1 LAPACK Routines

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.
- **computational** routines, also called simply LAPACK routines, perform a distinct computational task, such as computing the LU decomposition of an $m \times n$ matrix or finding the eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the QR algorithm. The LAPACK routines are listed in Appendix A; see also LAPACK Working Note #5 [3].
- **auxiliary** routines are all the other subroutines called by the driver routines and computational routines. Among them are subroutines to perform subtasks of block algorithms, in particular, the unblocked versions of the block algorithms; extensions to the BLAS, such as matrix-vector operations involving complex symmetric matrices; the special routines LSAME and XERBLA which first appeared with the BLAS; and a number of routines to perform common low-level computations, such as computing a matrix norm, generating an elementary Householder transformation, and applying a sequence of plane rotations. Many of the auxiliary routines may be of use to numerical analysts or software developers, so we have documented the Fortran source for these routines with the same level of detail used for the LAPACK routines and driver routines. The auxiliary routines are listed in Appendix B.

3.2 Level 1, 2, and 3 BLAS

The BLAS are a set of Basic Linear Algebra Subprograms that perform vector-vector, matrix-vector, and matrix-matrix operations. LAPACK is designed around the Level 1, 2, and 3 BLAS, and nearly all of the parallelism in the LAPACK routines is contained in the BLAS. Therefore, the key to getting good performance from LAPACK lies in having an efficient version of the BLAS optimized for your particular machine. If you have access to a library containing optimized versions of some or all of the BLAS, you should certainly

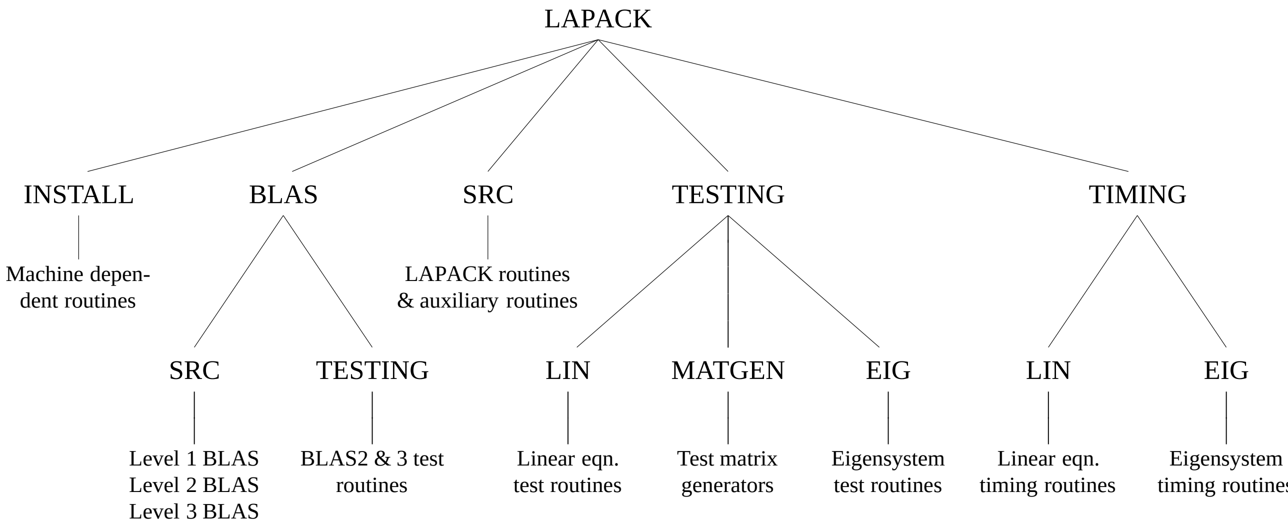

Figure 1: Unix organization of LAPACK

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

2 File Format

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

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

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

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

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

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

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

LAPACK Working Note 35
Implementation Guide for LAPACK *

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

August 9, 1991

Abstract

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

1 Introduction

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

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

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

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

†Current address: Cray Research Inc., 655F Lone Oak Drive, Eagan, MN 55121