LAPACK Working Note 9
A Test Matrix Generation Suite

James Demmel
Alan McKenney

Courant Institute
251 Mercer St.
New York, NY 10012, USA

March 1989

Abstract

We discuss the design and implementation of a suite of test matrix generators for testing linear algebra software. These routines generate random matrices with certain properties which are useful for testing linear equation solving, least squares, and eigendecomposition software. These properties include the spectrum, symmetry, bandwidth, norm, sparsity, conditioning (with respect to inversion or for the eigenproblem), type (real or complex), and storage scheme (dense, packed, or banded).

1 Introduction

Testing a large numerical linear algebra library such as LAPACK [3] requires generating many different kinds of test matrices. For example, LAPACK contains linear equation solving routines for real and complex matrices which may be symmetric, Hermitian, or non-symmetric, banded or dense, packed or unpacked, positive definite or indefinite, and so on. Thorough testing requires test matrices with a range of condition numbers and scalings (i.e., with norms near the overflow and underflow thresholds). There are similar requirements for testing the eigendecomposition (and singular value decomposition) routines.

To meet this need we have developed a suite of test matrix generation software, written in standard FORTRAN 77, which generates random matrices with various controlled properties. For example, one option permits the generation of random real rectangular matrices with singular values forming a geometric sequence between 1 and a user specified condition number, and with user specified upper and lower bandwidths. Of course, not all options can be specified independently; for example, there is no known way to generate nontrivial random nonsymmetric matrices with a given spectrum and arbitrary given upper and lower bandwidths. The software we describe checks for consistency among the specified matrix properties. The software can also generate consistent random examples in the sense that one can generate random matrices differing only in their storage scheme (dense vs. packed vs. banded), or in the order of their rows and columns.

The three main routines we have developed are called xLATMR, xLATMS and xLATME, where the first letter of each name (the 'x') is either 'S', 'D', 'C', or 'Z'; for example, "SLATMS" or "ZLATME". If the first letter is 'S', then the matrix generated will have single precision real (`REAL`) entries, if it is 'D', then double precision real (`DOUBLE PRECISION`) entries, if 'C', then single precision complex (COMPLEX) entries, and if 'Z', then double precision complex (DOUBLE COMPLEX) entries. If the first letter is 'S' or 'C', all floating-point values and variables will be single-precision; if the first letter is 'D' or 'Z', all floating-point values and variables will be double-precision. xLATMR generates matrices with random off-diagonal entries. xLATMS generates random real symmetric and complex Hermitian matrices with given eigenvalues and bandwidth, and random nonsymmetric and complex symmetric matrices with given singular values and bandwidth. xLATME generates random nonsymmetric matrices with given eigenvalues and either a given Jordan form (with certain restrictions) or given condition number for its eigenproblem. The routines can generate output in any legal LAPACK storage scheme (dense, packed, or banded).

The rest of this paper is organized as follows: sections 2, 3, and 4 present high level descriptions of xLATMR, xLATMS, and xLATME, respectively. Section 5 discusses naming and semantic conventions for common arguments of the three routines. Sections 6, 7, and 8 present the detailed calling sequence of the three routines. Finally, section 9 has implementation notes.

All variable names and FORTRAN fragments will appear in `typewriter font`.

2 xLATMR — High Level Description

xLATMR generates a matrix with random off-diagonal entries and given diagonal entries. It is the simplest (and fastest) of the three routines in the suite, and permits no direct control over the eigenvalues or singular values of the generated matrix. It is also fast and space efficient because both the time and the space required to generate a band matrix are proportional to the number of entries inside the desired bandwidth rather than the square of the matrix dimension; this efficiency is not generally possible if we wish to specify the spectrum of the resulting matrix. A high level description of its operation is as follows:

(1) Generate a matrix A with random entries. If the entries are real, they are chosen from either (a) a uniform distribution on $(0,1)$, (b) a uniform distribution on $(-1,1)$, or (c) a normal distribution with mean zero and variance one (normal$(0,1)$), as specified by the user. If the entries are complex, either their real and imaginary parts are independently chosen from the same distribution, which must be one of the three distributions available for real entries, or else the entries may be chosen from a uniform distribution on the unit disk. A may be nonsymmetric, real symmetric, complex symmetric, or complex Hermitian.

(2) Set the diagonal of A to D (an array), where the entries of D are computed as follows (N is the matrix dimension):

1. Input by the user.
2. $\mathtt{D}(1) = 1$ and the other $\mathtt{D}(i) = 1/\mathtt{COND}$, where $\mathtt{COND} \geq 1$ is a user input.
3. $\mathtt{D}(\mathtt{N}) = 1/\mathtt{COND}$ and the other $\mathtt{D}(i) = 1$.
4. The $\mathtt{D}(i)$ form a geometric sequence from 1 to $1/\mathtt{COND}$.
5. The $\mathtt{D}(i)$ form an arithmetic sequence from 1 to $1/\mathtt{COND}$.

6. The $\mathtt{D}(i)$ are random in the range $[1/\mathtt{COND}, 1]$ with uniformly distributed logarithms.
7. The $\mathtt{D}(i)$ are random with the same distribution as the other matrix entries.

In addition, each $\mathtt{D}(i)$ may optionally be multiplied by a random number with absolute value 1.

(3) Grade `A`, if desired, by pre- and postmultiplying it by diagonal matrices `DL` and `DR`, respectively. The entries of `DL` and `DR` are chosen just as the entries of `D` above.

(4) Permute the rows and columns of `A`, if desired.

(5) Set random entries of `A` to zero, if desired, to get a matrix with a given fraction of zero entries.

(6) Make `A` a band matrix, if desired, by zeroing out its entries outside given upper and lower bandwidths.

(7) Scale `A`, if desired, to have a given maximum absolute entry.

(8) Pack `A`, if desired. The allowable options are no packing, zeroing out the upper or lower half (if symmetric or Hermitian), storing the upper or lower half columnwise (if symmetric, Hermitian or triangular), using triangular band storage (upper half or lower half, and only if the matrix is symmetric, Hermitian or triangular), and full band storage.

If two calls to xLATMR differ only in the desired packing, they will generate mathematically equivalent matrices; this is convenient for testing routines which accept matrices in different storage schemes. If two calls to xLATMR both specify matrices with full bandwidth, and differ only in the order in which they permute the rows and columns (and possibly in the packing), then the matrices generated will differ only in the order of the rows and columns, and otherwise contain the same data. This consistency (which clearly cannot be attained for banded matrices, since permutations generally destroy any band structure) is useful for testing linear system solvers which perform pivoting.

3 xLATMS — High Level Description

xLATMS generates either a random real symmetric or complex Hermitian matrix with given eigenvalues and bandwidth, or a random nonsymmetric or complex symmetric matrix with given singular values and upper and lower bandwidth. The same packing options are available as for xLATMR. A high level description of its operation is as follows:

(1) Set the diagonal of the matrix `A` to `D`, where the entries of `D` can be chosen using the same options as for xLATMR. The entries of `D` will be the eigenvalues (and/or singular values) of the final matrix.

(2) Pre- and postmultiply `A` by random orthogonal matrices (if `A` is real) or unitary matrices (if `A` is complex.) If `A` is to be (real or complex) symmetric, then the premultiplying

matrix is the transpose of the postmultiplying one. If A is to be Hermitian, then the premultiplying matrix is the conjugate transpose of the postmultiplying one. Otherwise, the pre- and postmultipying matrices are chosen independently of one another.

(3) Reduce A to have the desired bandwidth using Householder transformations.

(4) Pack A, if desired. The same options are available as for xLATMR.

For non-banded and large-bandwidth matrices, the preceding description accurately describes the procedure actually followed. For small-bandwidth matrices, steps 2–4 are replaced by a method which uses Givens rotations to increase the bandwidth to the desired value, rather than generating a dense matrix and then reducing its bandwidth; this method is described in section 9.2. To be more precise, the Givens rotation method is used, if

(a) the matrix is symmetric or Hermitian and the bandwidth (KL or KU) is less than N/2,

(b) the matrix is non-symmetric and $\mathtt{KL} + \mathtt{KU} < 0.3(\mathtt{M} + \mathtt{N})$, or

(c) the first dimension of the array A is less than M (only possible if the matrix is to be stored in band storage format, i.e., PACK = 'B', 'Q', or 'Z' — see the description of LDA in section 5.1.)

4 xLATME — High Level Description

xLATME generates a random nonsymmetric matrix with given eigenvalues, either a given condition number for the eigenvalues or a given Jordan form (with certain restrictions), and a given one-sided bandwidth. Thus, for example, we can generate random Hessenberg matrices with given eigenvalues and sensitivities; this is useful for testing QR iteration algorithms for the nonsymmetric eigenproblem. Only dense storage is provided, since the nonsymmetric eigenroutines only work on matrices in dense storage format; thus, xLATME uses $n^2 + O(n)$ space and $O(n^3)$ time.

A high level description of its operation is as follows:

(1) Set the diagonal of the matrix A to D, where D is specified as for xLATMR and xLATMS.

(2) If A is real and complex conjugate pairs of eigenvalues are desired, certain pairs of adjacent elements of D are interpreted as the real and imaginary parts of a complex conjugate pair of eigenvalues, and A is made block diagonal with 2 by 2 blocks in the corresponding locations. If D is input by the user, the user also specifies which entries are to be interpreted as real and imaginary parts of complex conjugate eigenpairs. If D is not input by the user, the user may specify that pairs of adjacent entries be randomly designated as either a pair of real eigenvalues or as real and imaginary parts of an eigenpair.

(3) If the user so specifies, the upper triangle of A is filled with random numbers, which are chosen with the same range of options as for xLATMR. This option may be used to partially control the Jordan form of A as follows: if A has any multiple eigenvalues (as

determined by the last two steps), and the upper triangle is filled in, then there will be exactly one Jordan block per distinct eigenvalue; such a matrix is called defective. If the upper triangle is not filled in, there will only be 1 by 1 blocks in the Jordan form, even if there are multiple eigenvalues; such a matrix is called derogatory.

(4) If the user so specifies, `A` is premultiplied by a random matrix S and postmultiplied by S^{-1}. Here, S is a random dense nonsymmetric matrix whose singular values `DS` may be chosen with the same options as `D` in xLATMR. This option may be used to control the condition of `A`'s eigenproblem as follows: if the upper triangle has *not* been filled in in the last step, then the most sensitive eigenvalue of `A` will have sensitivity approximately equal to $\kappa \equiv \max_i |\mathtt{DS}(i)| / \min_i |\mathtt{DS}(i)|$, where sensitivity means that a perturbation of norm ϵ in `A` will cause a change an eigenvalue by at most approximately $\kappa \cdot \epsilon$ [6, 2]. The approximation arises because the true sensitivity need only be within a factor `N` (the dimension of `A`) of the condition number of SX, where X is a diagonal matrix chosen so that the columns of SX have equal norm. In general, the columns of S will not differ too much in norm, so that the condition number of the eigenproblem may indeed be approximately controlled with this approach.

(5) If the user so specifies, either the upper or lower bandwidth (but not both) is reduced to any positive value desired.

(6) Scale `A`, if desired, to have a given maximum absolute entry.

5 Common Argument Conventions

As can be seen from the last three sections, the three test matrix generators share various arguments and argument conventions. To simplify the later description of the detailed calling sequences, we collect those common conventions in this section. They are

1. Output matrix — `M`, `N`, `A`, `LDA`
2. Probability Distribution — `DIST`
3. Random Number Generator Seed — `ISEED`(4)
4. Symmetry — `SYM`
5. Diagonal Matrix Specification — `D`, `MODE`, `COND`, `DMAX`, and `RSIGN`.
6. Bandwidths — `KL`, `KU`
7. Norm — `ANORM`
8. Packing Option — `PACK`
9. Error Flag — `INFO`

In addition, we follow the convention that character arguments are one character (`CHARACTER*1`) and case independent. In this document we will always use upper case, although the software recognizes lower case as well.

5.1 Output Matrix — M, N, A, LDA

The output matrix is A; in xLATMR and xLATMS, it has M rows and N columns. In xLATME, where A must be square, it has N rows and columns. LDA is the leading dimension in the declaration of A in the calling routine; therefore within the test matrix generator A is dimensioned as A(LDA,*). A will be REAL, COMPLEX, DOUBLE PRECISION, or DOUBLE COMPLEX, according to whether the first letter of the name of the routine is 'S', 'D', 'C', or 'Z'. LDA must be at least 1; if A is not to be stored in band format (i.e., PACK is 'N', 'U', 'L', 'C', or 'R'), then LDA must be at least M; if A is to be stored in lower band storage format (i.e., PACK is 'B'), then LDA must be at least KL + 1; if A is to be stored in upper band storage format (i.e., PACK is 'Q'), then LDA must be at least KU + 1; if A is to be stored in general band storage format (i.e., PACK is 'Z'), then LDA must be at least KU + KL + 1. M, N and LDA are integers, and are read-only.

5.2 Probability Distribution — DIST

DIST is a read-only CHARACTER*1 variable. If A is real (REAL or DOUBLE PRECISION), the following options are available for DIST:

DIST = **'U'** — Uniform distribution on $(0, 1)$.

DIST = **'S'** — Uniform distribution on $(-1, 1)$.

DIST = **'N'** — Normal $(0, 1)$.

If A is complex (COMPLEX or DOUBLE COMPLEX), the following options are available for DIST:

DIST = **'U'** — Both real and imaginary parts are independent and uniformly distributed on $(0, 1)$.

DIST = **'S'** — Both real and imaginary parts are independent and uniformly distributed on $(-1, 1)$.

DIST = **'N'** — Both real and imaginary parts are independent and normally distributed with zero mean and unit variance.

DIST = **'D'** — The complex number is uniformly distributed inside the unit disk in the complex plane.

5.3 Random Number Generator Seed — ISEED

ISEED is an array of 4 integers, which are the seeds of the random number generator. The random number generator will operate identically on any machine with at least 24 bit integers, and generate a random number sequence with period 2^{48}. The sequence of random numbers generated during an execution of xLATMR, xLATMS, or xLATME (and thus the matrix generated) is determined by the value of ISEED on entry to the routine; thus, to recreate a matrix previously generated by one of these routines, it is only necessary to know the argument values used to generate it the first time, including the value of ISEED. Moreover, if the matrix is recreated on a different machine, it will differ from the first only due to the effects of round-off and range errors. ISEED is modified by the routines.

5.4 Symmetry — SYM

SYM is a read-only CHARACTER*1 variable. It has the following interpretations (in all cases A may be real or complex):

SYM = 'N' — Nonsymmetric.

SYM = 'S' — Symmetric.

SYM = 'H' — Hermitian. If A is real, then this is equivalent to SYM = 'S'.

SYM = 'P' — Positive semidefinite.

5.5 Diagonal Matrix Specification — D, MODE, COND, DMAX, RSIGN

There are various array arguments like D which are meant to be the diagonal entries of matrices. All of them may be computed using the options MODE and COND; only some use DMAX and RSIGN as well. In all cases, D is an array of n REAL, DOUBLE PRECISION, COMPLEX or DOUBLE COMPLEX numbers which may be modified by the routine (n may equal either N or M, depending on the situation). MODE is a read-only integer. COND is a read-only REAL or DOUBLE PRECISION variable, which must be at least 1 if it is referenced. DMAX is a read-only variable of the same type as D. RSIGN is a read-only CHARACTER*1 variable. MODE and COND are interpreted as follows:

MODE = 0 — D is supplied on entry by the user, in which case it is not modified by the program.

MODE = 1 — D(1) is set to 1 and the other D(i) are set to 1/COND.

MODE = 2 — D(N) is set to 1/COND and the other D(i) are set to 1.

MODE = 3 — D(i) is set to $\mathtt{COND}^{-(i-1)/(n-1)}$, i.e., a geometric sequence ranging from 1 to 1/COND.

MODE = 4 — D(i) is set to $1 - (i-1)/(n-1) \cdot (1 - 1/\mathtt{COND})$, i.e., an arithmetic sequence ranging from 1 to 1/COND.

MODE = 5 — Each D(i) is an independent random number in the range from 1/COND to 1 with a uniformly distributed logarithm. This guarantees that the ratio $\max_i \mathtt{D}(i)/\min_i \mathtt{D}(i)$ is close to COND.

MODE = 6 — Each D(i) is an independent random number with distribution DIST.

If MODE < 0, the meaning is the same as for |MODE|, except that the order of the entries of D(i) is reversed. Thus, if $1 \leq \mathtt{MODE} \leq 4$, the D($i$) range from 1 down to 1/COND, and if $-4 \leq \mathtt{MODE} \leq -1$, the D($i$) range from 1/COND up to 1.

If DMAX is specified as well, each D is scaled by $\mathtt{DMAX}/\max_i |\mathtt{D}(i)|$, so that the maximum absolute entry is |DMAX|. Note that DMAX may be negative (or complex).

If RSIGN is specified, it has the following meaning:

RSIGN = **'F'** — D is unchanged.

RSIGN = **'T'** — If D is real, each D(i) is multiplied by $+1$ or -1 with a .5 probability. If D is complex, each D(i) is multiplied by an independent random number r uniformly distributed on the unit circle (thus r has unit absolute value).

5.6 Bandwidths — KL, KU

KL and KU are read-only integers, specifying the lower and upper bandwidths, respectively. Thus, if KL$=0$ (or KU$=0$), the matrix is upper (or lower) triangular. If KL$=1$ (or KU$=1$), the matrix is upper (or lower) Hessenberg. If KL $\geq$ M -1 (or KU $\geq$ N -1), the matrix is full below (or above) the diagonal. If A is specified to be symmetric or Hermitian, then KL must equal KU.

5.7 Norm — ANORM

This read-only REAL or DOUBLE PRECISION variable specifies the maximum absolute entry of the output A matrix. If ANORM is zero or positive, A will be scaled so that the largest absolute entry is ANORM. If it is negative, A will not be scaled.

5.8 Packing Option — PACK

This read-only CHARACTER*1 variable specifies the storage scheme of the output A matrix. All storage schemes in LAPACK (which includes all storage schemes in LINPACK [1] and the BLAS [5, 4]) are accounted for. In this section we list the options and describe these storage schemes. PACK is interpreted as follows:

PACK = **'N'** — No packing (i.e., dense storage of all entries).

PACK = **'U'** — Zero out all the subdiagonal entries, but store A in dense format. This is allowed only if A is symmetric or Hermitian.

PACK = **'L'** — Zero out all the superdiagonal entries, but store A in dense format. This is allowed only if A is symmetric or Hermitian.

PACK = **'C'** — Store the upper triangle columnwise. This is allowed only if A is symmetric, Hermitian or upper triangular. This is a packed format, requiring about half the space of dense storage.

PACK = **'R'** — Store the lower triangle columnwise. This is allowed only if A is symmetric, Hermitian or lower triangular. This is a packed format, requiring about half the space of dense storage.

PACK = **'B'** — Store the lower triangle in band storage format. This is allowed only if A is symmetric, Hermitian, or lower triangular. To illustrate this storage scheme, suppose the original matrix is

$$\begin{bmatrix} 11 & 12 & 13 & 0 & 0 & 0 \\ 21 & 22 & 23 & 24 & 0 & 0 \\ 31 & 32 & 33 & 34 & 35 & 0 \\ 0 & 42 & 43 & 44 & 45 & 46 \\ 0 & 0 & 53 & 54 & 55 & 56 \\ 0 & 0 & 0 & 64 & 65 & 66 \end{bmatrix} \tag{5.1}$$

, where we have labeled each nonzero entry by its concatenated indices. In lower band format this matrix would be stored as

$$\begin{bmatrix} 11 & 22 & 33 & 44 & 55 & 66 \\ 21 & 32 & 43 & 54 & 65 & 0 \\ 31 & 42 & 53 & 64 & 0 & 0 \end{bmatrix} \tag{5.2}$$

. Thus, the columns of the band storage format correspond to columns of the original matrix, and rows of the band storage format correspond to diagonals of the original matrix. This is true for the `PACK` = 'Q' and `PACK` = 'Z' options below as well.

`PACK` = **'Q'** — Store the upper triangle in band storage format This is allowed only if `A` is symmetric, Hermitian, or upper triangular. To illustrate, the matrix in (5.1) would be stored as

$$\begin{bmatrix} 0 & 0 & 13 & 24 & 35 & 46 \\ 0 & 12 & 23 & 34 & 45 & 56 \\ 11 & 22 & 33 & 44 & 55 & 66 \end{bmatrix} \tag{5.3}$$

.

`PACK` = **'Z'** — Store the matrix in band storage format. To illustrate, the matrix in (5.1) would be stored as

$$\begin{bmatrix} 0 & 0 & 13 & 24 & 35 & 46 \\ 0 & 12 & 23 & 34 & 45 & 56 \\ 11 & 22 & 33 & 44 & 55 & 66 \\ 21 & 32 & 43 & 54 & 65 & 0 \\ 31 & 42 & 53 & 64 & 0 & 0 \end{bmatrix} \tag{5.4}$$

.

5.9 Error Flag — `INFO`

Following the convention used in LAPACK, the variable `INFO` will be set to zero to indicate successful completion. A negative value of `INFO` on return means that argument number −`INFO` in the calling sequence in incorrect; in this case the comments in the front of the program explain the error in more detail. If `INFO` is positive on return, then a numerical or other error was encountered during execution; again the comments at the front of the routines describe the situation in more detail.

6 xLATMR — Detailed Calling Sequence

Here is the specification of the calling sequence of xLATMR:

```
      SUBROUTINE xLATMR( M, N, DIST, ISEED, SYM, D, MODE, COND, DMAX,
     $                   RSIGN, GRADE, DL, MODEL, CONDL, DR, MODER,
     $                   CONDR, PIVTNG, IPIVOT, KL, KU, SPARSE, ANORM,
     $                   PACK, A, LDA, IWORK, INFO )
*
*     .. Scalar arguments ..
*
      CHARACTER*1        DIST, SYM, RSIGN, GRADE, PIVTNG, PACK
      INTEGER            MODE, MODEL, MODER, KL, KU, M, N, LDA, INFO
      type1              COND, CONDL, CONDR, SPARSE, ANORM, DMAX
      type2              DMAX
*
*     .. Array arguments ..
*
      INTEGER            ISEED( 4 ), IPIVOT( * ), IWORK( * )
      type2              D( * ), DL( * ), DR( * ), A( LDA, * )
```

Here, type1 and type2 are defined as follows:

Type of output matrix A	x in xLATMR	type1	type2
REAL	S	REAL	REAL
DOUBLE PRECISION	D	DOUBLE PRECISION	DOUBLE PRECISION
COMPLEX	C	REAL	COMPLEX
DOUBLE COMPLEX	Z	DOUBLE PRECISION	DOUBLE COMPLEX

Many of the arguments were described in the last section, so we only describe the new ones and variations here.

6.1 Grading — GRADE, DL, MODEL, CONDL, DR, MODER, CONDR

GRADE is a read-only CHARACTER*1 variable which specifies if the matrix A is to be pre- or postmultiplied by diagonal matrices with diagonal entries DL and DR, respectively.

GRADE = **'N'** — No grading.

GRADE = **'L'** — A is premultiplied by $diag$(DL). This is allowed only if A is nonsymmetric.

GRADE = **'R'** — A is postmultiplied by $diag$(DR). This is allowed only if A is nonsymmetric.

GRADE = **'B'** — A is premultiplied by $diag$(DL) and postmultiplied by $diag$(DR). This is allowed only if A is nonsymmetric.

GRADE = **'S'** — A is pre- and postmultiplied by $diag$(DL). This is allowed only if A is symmetric or nonsymmetric, but not complex Hermitian.

GRADE = **'H'** — A is premultiplied by $diag$(DL) and postmultiplied by the complex conjugate of $diag$(DL). This is allowed only if A is nonsymmetric or Hermitian, but not complex symmetric.

GRADE = **'E'** — A is premultiplied by *diag*(DL) and postmultiplied by the inverse of *diag*(DL). This is allowed only if A is nonsymmetric. This preserves the original eigenvalues.

DL is specified by MODEL and CONDL just as D is specified by MODE and COND. DR is specified by MODER and CONDR the same way.

6.2 Pivoting — PIVTNG and IPIVOT

PIVTNG is a read-only CHARACTER*1 variable which specifies how the rows and columns of A are to be permuted. IPIVOT is a read-only integer array which specified the order itself. PIVTNG is interpreted as follows:

PIVTNG = **'N'** — No pivoting.

PIVTNG = **'L'** — Left or row pivoting. A must be nonsymmetric.

PIVTNG = **'R'** — Right or column pivoting. A must be nonsymmetric.

PIVTNG = **'B' or 'F'** — Both or Full pivoting, i.e., on both sides. A must be square.

The IPIVOT array specifies the permutation used. After the basic matrix is generated, the rows, columns, or both are permuted. If, say, row pivoting is selected, xLATMR starts with the *last* row and interchanges the M-th and IPIVOT(M)-th rows, then moves to the next to last row, interchanging the (M − 1)-st and the IPIVOT(M − 1)-st rows, and so on. In terms of '2-cycles', the permutation is (1 IPIVOT(1)) (2 IPIVOT(2)) ...(M IPIVOT(M)) where the rightmost cycle is applied first. This is the *inverse* of the effect of pivoting in LINPACK or LAPACK. The idea is that factoring (with pivoting) an identity matrix which has been inverse-pivoted in this way should result in a pivot vector output from LAPACK or LINPACK identical to IPIVOT.

6.3 Sparsifying — SPARSE

This read-only real variable must be between 0 and 1, inclusive. For each matrix entry an independent random variable r uniformly distributed on $(0, 1)$ is generated and compared to SPARSE. If $r >$ SPARSE, the matrix entry is unchanged, but if $r \leq$ SPARSE, the matrix entry is set to zero (preserving symmetry if the matrix is symmetric or Hermitian). Thus, on average a fraction SPARSE of the entries will be set to zero. If SPARSE = 0, no entries will be set to zero.

6.4 Workspace — IWORK

This integer workspace array must have dimension as large as IPIVOT, if IPIVOT is referenced (PIVTNG ≠ 'N').

7 xLATMS — Detailed Calling Sequence

Here is the specification of the calling sequence for xLATMS:

```
      SUBROUTINE xLATMS( M, N, DIST, ISEED, SYM, D, MODE, COND,
     $                   DMAX, KL, KU, PACK, A, LDA, WORK, INFO )
*
*     .. Scalar arguments ..
*
      CHARACTER*1        DIST, SYM, PACK
      INTEGER            MODE, KL, KU, M, N, LDA, INFO
      type1              COND, DMAX
*
*     .. Array arguments ..
*
      INTEGER            ISEED( 4 )
      type1              D( * ),
      type2              A( LDA, * ), WORK( * )
```

Here, type1 and type2 are defined just as for xLATMR. As before, we only describe the arguments which were not completely described in section 5.

7.1 Symmetry — SYM

The use of SYM depends on whether A is real or complex. If A is real then

SYM = **'N'** means that A is nonsymmetric and the vector D, as determined by MODE, COND and DMAX, determines its singular values.

SYM = **'P'** means that A is positive semidefinite and the vector D, as determined by MODE, COND and DMAX, determines its eigenvalues; in this case the eigenvalues will be nonnegative (unless the user inputs negative eigenvalues in D with MODE $= 0$).

SYM = **'S' or 'H'** means that A is symmetric and the vector D, as determined by MODE, COND, and DMAX. determines the eigenvalues. In addition, when MODE $\neq 0$, each entry of D is multiplied at random by either $+1$ or -1. Thus A will have both positive and negative eigenvalues (unless the user inputs eigenvalues of all one sign in D with MODE $= 0$).

If A is complex then

SYM = **'N'** means that A is nonsymmetric and the vector D, as determined by MODE, COND and DMAX, determines the singular values.

SYM = **'P'** means that A is positive semidefinite and the vector D, as determined by MODE, COND and DMAX, determines its eigenvalues; in this case the eigenvalues will be nonnegative (unless the user inputs negative eigenvalues in D with MODE $= 0$).

SYM = **'S'** means that A is complex symmetric and the vector D, as determined by MODE, COND and DMAX, determines the singular values.

SYM = **'H'** means that A is Hermitian the vector D, as determined by MODE, COND, and DMAX determines the eigenvalues. In addition, when MODE $\neq 0$, each entry of D is multiplied at random by either $+1$ or -1. Thus A will have both positive and negative eigenvalues (unless the user inputs eigenvalues of all one sign in D with MODE $= 0$).

7.2 Workspace — WORK

The array WORK, which has the same type as A, must be dimensioned at least $3 \cdot \max(\mathtt{M}, \mathtt{N})$. It is modified by the program.

8 xLATME — Detailed Calling Sequence

Here is the specification of the calling sequence for xLATME:

```
      SUBROUTINE xLATME( N, DIST, ISEED, D, MODE, COND, DMAX, EI,
     $                   RSIGN, UPPER, SIM, DS, MODES, CONDS,
     $                   KL, KU, ANORM, A, LDA, WORK, INFO )
*
*     .. Scalar arguments ..
*
      CHARACTER*1        DIST, RSIGN, UPPER, SIM, EI
      INTEGER            MODE, MODES, KL, KU, N, LDA, INFO
      type1              COND, CONDS, ANORM
      type2              DMAX
*
*     .. Array arguments ..
*
      INTEGER            ISEED( 4 )
      type1              DS( * )
      type2              D( * ), A( LDA, * ), WORK( * )
```

Here, type1 and type2 are determined just as for xLATMR and xLATMS. As before, we only discuss the arguments not completely described in section in section 5.

8.1 Eigenvalues — D, MODE, COND, DMAX, RSIGN, EI

These variables determine the eigenvalues of A. Their interpretations depend on whether A is real or complex.

If A is real, the eigenvalues are determined as follows. Recall that a real matrix either has real eigenvalues or complex eigenvalues appearing in complex conjugate pairs. D is first computed from MODE, COND, DMAX and RSIGN as described in section 5. In order to get complex conjugate eigenvalue, the following additional computations are done when MODE $= 0$ or MODE $= 5$.

When MODE $= 0$, both D and the array EI of CHARACTER*1 variables must be supplied by the user; EI specifies which entries of D are real eigenvalues, and which are the real and imaginary parts of a pair of complex conjugate eigenvalue. If EI(1) = ' ' (a blank), then all D(i) are taken to be real eigenvalues. Otherwise, all entries EI(i) must either be 'R' (for real part) or 'I' (for imaginary part). Each 'I' must follow an 'R'; thus if EI(i) = 'R' and EI($i+1$) = 'I', then D(i) is the real part and D($i+1$) is the imaginary part of a complex conjugate pair of eigenvalue. The assembly of A is begun by putting 1 by 1 blocks D(i) corresponding to real eigenvalues and 2 by 2 blocks

$$\begin{bmatrix} \mathtt{D(i)} & \mathtt{D(i+1)} \\ -\mathtt{D(i+1)} & \mathtt{D(i)} \end{bmatrix}, \tag{8.5}$$

which have complex conjugate eigenvalues $\mathtt{D}(i) \pm \imath \cdot \mathtt{D}(i+1)$, on the diagonal of A. When MODE $\neq 0$, EI is ignored.

When MODE $= 5$, complex conjugate pairs of eigenvalues are obtained as follows. For each adjacent pair $(\mathtt{D}(2i-1), \mathtt{D}(2i))$ of entries of D, an independent random number r is chosen which has takes the values ± 1 with probability .5. If $r = +1$, the pair is treated as two real eigenvalues, and if $r = -1$, the pair is treated as the real and imaginary parts of a complex conjugate pair of eigenvalues, and incorporated into A as in (8.5).

When A is complex, the eigenvalues are determined as follows. D is first computed from MODE, COND, DMAX as described in section 5. Then, if RSIGN= 'T', and MODE is neither 0 (input by the user) nor ± 6 (random with distribution DIST), each D(i) is multiplied by an independent random complex number r uniformly distributed on the unit circle. These D(i) are the eigenvalues; EI is ignored.

8.2 Jordan Form — UPPER

If UPPER = 'T', the upper triangle of A (above the eigenvalues in the 1 by 1 and 2 by 2 diagonal blocks) is filled in with random numbers from distribution DIST. This means there will be exactly one Jordan block per distinct eigenvalue. If UPPER = 'F', the upper triangular is left zero. Thus A's Jordan form will only have 1 by 1 blocks, even if the eigenvalues are multiple.

8.3 Similarity Transform — SIM, DS, MODES, CONDS

If SIM = 'T', then A is premultiplied by S and postmultiplied by S^{-1}, where $S = U \cdot diag(\mathtt{DS}) \cdot V$, U and V are random orthogonal (or unitary) matrices, and DS is constructed from MODES and CONDS just as D is constructed from MODE and COND (see section 5). If SIM = 'F', this pre- and postmultiplication is not performed. The condition number $\kappa \equiv \max_i |\mathtt{DS}(i)| / \min_i |\mathtt{DS}(i)|$ of S approximately determines the sensitivity of A's eigenproblem, as described in section 4.

8.4 Bandwidth — KL, KU

As before, KL is the lower bandwidth and KU is the upper bandwidth. However, at most one of them may be less than $\mathtt{N} - 1$, i.e., less than full bandwidth, and both must be positive. This is because no way is known to generate nontrivial random nonsymmetric matrices with fixed spectrum and arbitrary bandwidth. If a triangular matrix (`KL` = 0 or `KU` = 0) is desired, set `SIM` = 'F' so that no multiplication by S and S^{-1} is performed.

8.5 Workspace — `WORK`

The array `WORK`, which has the same type as `A`, must be dimensioned at least $3 \cdot$ `N`. It is modified by the program.

9 Implementation Notes

9.1 Generating Random Orthogonal and Unitary Matrices

We use a method developed by Stewart [7], which we just outline here. The following pseudocode shows how to postmultiply a given matrix A by a random orthogonal (unitary) matrix U:

for $i = 2$ to n

- generate a vector v with i entries each of which is an independent normal $(0, 1)$ random real (complex) number.
- let x be an n-vector whose first $n - i$ entries are zero and the remainder equal v.
- Let H be an n by n Householder transformation which reduces the vector x to one whose only nonzero entry is at location $n - i + 1$.
- apply the Householder transformation to $A : A \leftarrow AH$

endfor

for $i = 1$ to n

- multiply the i-th column of A by an independent real (complex) random number which equals ± 1 with probability .5 (is uniformly distributed on the unit circle)

endfor

Thus, U is represented as a product of random Householder transformations and a random diagonal orthogonal (unitary) matrix. The random orthogonal (unitary) matrix generated is distributed according to Haar measure [7], which is analogous to the uniform distribution. In other words, if $\mathbf{U}$ is a set of orthogonal (unitary) matrices with probability μ, then the sets $P \cdot \mathbf{U}$ and $\mathbf{U} \cdot P$, where P is any fixed orthogonal (unitary) matrix, also have probability μ.

The subroutine, xLAROR, also permits premultiplication $U \cdot A$, as well as $U \cdot A \cdot U^T$ and $U \cdot A \cdot U^*$ (conjugate transpose).

$$
\begin{array}{c|ccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
1 & x & x & x & x & x & x & & & & & & & & & \\
2 & & x & x & x & x & x & x & & & & & & & & \\
3 & & & x & x & x & x & x & x & \mathbf{c} & & & & & & \\
4 & & & \mathbf{d} & x & x & x & x & x & x & & & & & & \\
5 & & & & & x & x & x & x & x & x & & & & & \\
6 & & & & & & x & x & x & x & x & x & & & & \\
7 & & & & & & & x & x & x & x & x & x & & & \\
8 & & & & & & & & x & x & x & x & x & \mathbf{a} & & \\
9 & & & & & & & & \mathbf{b} & x & x & x & x & x & & \\
10 & & & & & & & & & & x & x & x & x & x & \\
11 & & & & & & & & & & & x & x & x & x & x \\
\ldots & & & & & & & & & & & \ldots & \ldots & \ldots & \ldots & \ldots
\end{array}
$$

Figure 1: Adding a diagonal to an upper triangular band matrix.

9.2 Generating Band Matrices using Givens Rotations

The method described in section 3 for generating a band matrix with specified singular values or eigenvalues requires $O(n^3)$ time and $O(n^2)$ storage, since it first generates a dense matrix and then uses Householder transformations to eliminate nonzero entries outside of the bandwidth, column by column or row by row. The other method that we use in the code adds successive off-diagonals by using Givens rotations; this method requires $O(n^2k)$ time to generate an $n \times n$ matrix of bandwidth k and no more storage than is needed to store the final matrix. We first describe the method as applied to an upper-triangular band matrix, and then describe how the method is extended to deal with symmetric, Hermitian, non-symmetric, and rectangular matrices.

To generate an upper-triangular band matrix using Givens rotations, we start with a diagonal matrix of singular values. We then apply a random Givens rotation to the first two rows; this makes the $(1,2)$ and $(2,1)$ entries non-zero, assuming the first two singular values are non-zero. We then apply a suitably chosen Givens rotation to the first two columns to make the $(2,1)$ entry zero; if the first two singular values differ, the $(1,2)$ entry will remain non-zero. We next apply a random Givens rotation to the second and third rows; this makes the $(2,3)$ and $(3,2)$ entries non-zero, so we rotate the second and third columns to make the $(3,2)$ entry zero but the $(1,3)$ entry non-zero, we rotate the first two rows to make this entry zero but the $(2,1)$ entry non-zero, which we eliminate by rotating the first two columns. Proceeding in this way, we fill in the first off-diagonal. The same procedure fills in the second, and so on. Figure 1 shows the rotations used when filling in the eighth entry on the fifth off-diagonal: **a** is the entry being filled in by the random Givens rotation, **b** is the unwanted entry filled in by the random rotation and eliminated by the next rotation, **c** is filled in when **b** is eliminated, **d** is filled in when **c** is eliminated, but eliminating **d** fills in no new entries.

A pseudo-code description of the method for generating an $n \times n$ upper-triangular matrix A of bandwidth k from a diagonal matrix of singular values is as follows:

For each off-diagonal $j = 1, \ldots, k$

For each entry $i = 1, \ldots, n-1$

- Apply a random Givens rotation to rows i and $i+1$. If $i \leq n-k$, this makes $A_{i,i+j}$ non-zero.
- Apply a Givens rotation to columns i and $i+1$ to eliminate $A_{i+1,i}$.
- Continue applying row and column rotations to "chase" the unwanted entry up to and off of the upper-left end of the matrix.

Note that we rotate through rows $n-1$ and n, even though no new entry on the upper diagonal is filled in thereby. This insures that the last two columns will be mixed, so that the last column need not be zero, even if the last singular value is zero.

The method for generating an $m \times n$ upper trapezoidal (i.e., $m > n$) band matrix differs from the preceding only in that the column rotations must continue through column $\min(m+j, n)$, to insure that columns $m+1, \ldots, \min(m+k, n)$ are not always zero. The pseudo-code description is then:

For each off-diagonal $j = 1, \ldots, k$

For each entry $i = 1, \ldots, \min(m+j, n) - 1$

if $i < m$ **then**:

- Apply a random Givens rotation to rows i and $i+1$. If $i \leq n-k$, this makes $A_{i,i+j}$ non-zero.
- Apply a Givens rotation to columns i and $i+1$ to eliminate $A_{i+1,i}$.

else:

- Apply a random Givens rotation to columns i and $i+1$.

end **if**:

- Continue applying row and column rotations to "chase" the unwanted entry up to and off of the upper-left end of the matrix.

Lower-triangular and lower-trapezoidal band matrices are generated by the transpose of the method described so far. A general band matrix, $m \times n$, with upper bandwidth k and lower bandwidth ℓ is generated by:

(1) generating an upper-triangular/-trapezoidal matrix with the desired upper bandwidth, then

(2) adding lower diagonals. Adding the lower diagonals differs from the procedure for generating a lower-triangular/-trapezoidal matrix only in that after the random Givens rotation applied to columns i and $i+1$, the next Givens rotation, which eliminates the unwanted element in the upper triangle, is applied, not to rows i and $i+1$, but to rows $i-k$ and $i-k+1$.

Symmetric and Hermitian matrices are generated by the same method as for upper-triangular matrices, except that each Givens rotation must be applied from both the left and (if Hermitian, then conjugated) from the right. The lower triangle is not explicitly computed, since it is just a copy of the upper; if the lower triangle is desired, the code copies it from the upper. Some care must be taken with the first sub-diagonal, since there is a stage where the Givens rotation has been applied from one side but not yet from the other when one element on the sub-diagonal is not equal to the corresponding element on the super-diagonal.

The method as described so far generates each diagonal starting from the top left, and we therefore call it the "top-down" version; the code also includes a corresponding "bottom-up" version as well. The "top-down" version is used if the first eigen- or signular value is larger than the last, otherwise the "bottom-up" version is used. We originally believed that this choice of version would reduce the effect of rounding error on small eigen- and singular values when the original, diagonal matrix is graded, but experiments have not shown any clear, consistent difference. It does have the advantage that if the order of the singular or eigenvalues is reversed, and the same random seed is used to start off with each time, then the resulting generated matrix is the same as the originally generated matrix with the order of the rows and of the columns reversed.

9.3 Accuracy Limitations

Any option which requires multiplication by orthogonal, unitary or other dense matrices within the test suite will introduce rounding errors which may obscure tiny eigenvalues or singular values. In particular, if ϵ is the machine precision, then the true eigenvalues or singular values of the `A` computed by xLATMS will differ from their prescribed values by as much as $O(\epsilon\,\|\mathtt{A}\|)$ ($\|\mathtt{A}\|$ is the largest absolute eigenvalue or singular value). In particular, tiny eigenvalues or singular values may have large absolute errors. This effect is even more pronounced in the case of xLATME, since eigenvalues of a nonsymmetric matrix may be extremely sensitive to perturbations, as discussed in section 4. In fact, if the matrix is chosen to have all zero eigenvalues, `UPPER` = 'T' so there is just one Jordan block, and `SIM` = 'T' so that there is pre- and postmultiplication by dense matrices, the true eigenvalues may be as large as $O(\epsilon^{1/n})$, where n is the dimension. This sensitivity is inherent in the problem [6].

References

[1] J. Bunch, J. Dongarra, C. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, PA, 1979.

[2] J. Demmel. On condition numbers and the distance to the nearest ill-posed problem. *Numerische Mathematik*, 51(3):251–289, July 1987.

[3] J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. Prospectus for the development of a linear algebra library for high-performance computers. Mathematics and Computer Science Division Report ANL/MCS-TM-97, Argonne National Laboratory, Argonne, IL, September 1987.

[4] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A proposal for a set of level 3 basic linear algebra subprograms. *SIGNUM Newsletter*, 22(3):2–14, February 1987.

[5] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of fortran basic linear algebra subroutines. *ACM Transactions on Mathematical Software*, 14(1):1–17, March 1988.

[6] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 1983.

[7] G. W. Stewart. On efficient generation of random orthogonal matrices with an application to condition estimation. *SIAM Journal of Numerical Analysis*, 17(3):403–409, 1980.