LAPACK Working Notes:

LAPACK Working Note #1: J. Demmel and J. Dongarra and J. Du Croz and A. Greenbaum and S. Hammarling and D. Sorensen, *Prospectus for the Development of a Linear Algebra Library for High-Performance Computers,* Argonne National Laboratory, ANL-MCS-TM-97, September, 1987.

LAPACK Working Note #2: J. Dongarra and S. Hammarling and D. Sorensen, *Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations,* Argonne National Laboratory, ANL-MCS-TM-99, September, 1987.

LAPACK Working Note #3: J. Demmel and W. Kahan, *Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy,* Argonne National Laboratory, ANL-MCS-TM-110, February, 1988.

LAPACK Working Note #4: J. Demmel, J. Du Croz, S. Hammarling and D. Sorensen, *Guides for the Design of Symmetric Eigenroutines, SVD, and Iterative Refinement and Condition Estimation for Linear Systems,* Argonne National Laboratory, ANL-MCS-TM-111, February, 1988.

LAPACK Working Note #5: C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling and D. Sorensen *Provisional Contents,* Argonne National Laboratory, ANL-88-38, September, 1988.

LAPACK Working Note #6: O. Brewer, J. Dongarra and D. Sorensen *Tools to Aid in the Analysis of Memory Access Patterns for Fortran Programs,* Argonne National Laboratory, ANL-MCS-TM-119, June, 1988.

References

[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

[10]

[11]

[12]

[13]

[14]

[15]

[16]

[17]

[18]

[19]

[20]

[21]

[22]

[23]

[24]

- driver - means that these are driver routines. We are likely to provide equivalent functionality.
- Note A - we plan to use orthogonal transformations throughout, not elementary transformations.

SPOFA	SPOTRF	factors a symmetric positive definite matrix.
SPOSL	SPOTRS	solves the symmetric positive definite system $Ax = b$ using the factors computed by SPOCO or SPOFA.
SPPCO	SPPTRF	factors a symmetric positive definite
	SPPCON	matrix stored in packed form and estimates the condition of the matrix.
SPPDI	SPPTRI	computes the determinant and inverse of a symmetric positive definite matrix using the factors computed by SPPCO or SPPFA .
SPPFA	SPPTRF	factors a symmetric positive definite matrix stored in packed form.
SPPSL	SPPTRS	solves the symmetric positive definite system $Ax = b$ using the factors computed by SPPCO or SPPFA.
SPTSL	SPTSOL	given a positive definite tridiagonal matrix and a right hand side will find the solution.
SQRDC	SGEQRF	uses Householder transformations to compute the QR
	or	factorization of an n by p matrix X. column pivoting
	SGEQRP	based on the 2-norms of the reduced columns may be performed at the users option.
SQRSL	SGEQRS	applies the output of SQRDC to compute coordinate transformations, projections, and least squares solutions.
SSICO	SSYTRF	factors a symmetric matrix by elimination
	SSYCON	with symmetric pivoting and estimates the condition of the matrix.
SSIDI	SSYTRI	computes the determinant, inertia and inverse of a symmetric matrix using the factors from SSIFA.
SSIFA	SSYTRF	factors a symmetric matrix by elimination with symmetric pivoting.
SSISL	SSYTRS	solves the symmetric system $Ax = b$ using the factors computed by SSIFA.
SSPCO	SSPTRF	factors a symmetric matrix stored in
	SSPCON	packed form by elimination with symmetric pivoting and estimates the condition of the matrix.
SSPDI	SSPTRI	computes the determinant, inertia and inverse of a symmetric matrix using the factors from SSPFA, where the matrix is stored in packed form.
SSPFA	SSPTRF	factors a symmetric matrix stored in packed form by elimination with symmetric pivoting.
SSPSL	SSPTRS	solves the symmetric system $Ax = b$ using the factors computed by SSPFA.
SSVDC	driver	is a subroutine to reduce a n by p matrix X by orthogonal transformations u and v to diagonal form.
STRCO	STRCON	estimates the condition of a triangular matrix.
STRDI	STRTRI	computes the determinant and inverse of a triangular matrix.
STRSL	STRTRS	solves systems of the form $Tx = b$ or $T^T x = b$ where T is a triangular matrix of order n.

LINPACK	LAPACK	Function
SCHDC	SSYTRF	computes the Cholesky decomposition of a positive definite matrix. a pivoting option allows the user to estimate the condition of a positive definite matrix or determine the rank of a positive semidefinite matrix.
SCHDD	SPOTRU	downdates an augmented Cholesky decomposition or the triangular factor of an augmented qr decomposition.
SCHEX	SPOTRX	updates the Cholesky factorization
SCHUD	SPOTRU	updates an augmented Cholesky decomposition of the triangular part of an augmented qr decomposition.
SGBCO	SGBTRF	factors a band matrix by Gaussian
	SGBCON	elimination and estimates the condition of the matrix.
SGBDI	SGBTRI	computes the determinant of a band matrix using the factors computed by SGBCO or SGBFA. if the inverse is needed, use SGBSL n times.
SGBFA	SGBTRF	factors a band matrix by elimination.
SGBSL	SGBTRS	solves the band system $Ax = b$ or $A^T x = b$ using the factors computed by SGBCO or SGBFA.
SGECO	SGETRF	factors a matrix by Gaussian elimination
	SGECON	and estimates the condition of the matrix.
SGEDI	SGETRI	computes the determinant and inverse of a matrix using the factors computed by SGECO or SGEFA.
SGEFA	SGETRF	factors a matrix by Gaussian elimination.
SGESL	SGETRS	solves the system $Ax = b$ or $A^T x = b$ using the factors computed by SGECO or SGEFA.
SGTSL	SGTSOL	given a general tridiagonal matrix and a right hand side will find the solution.
SPBCO	SPBTRF	factors a symmetric positive definite
	SPBCON	matrix stored in band form and estimates the condition of the matrix.
SPBDI		computes the determinant of a symmetric positive definite band matrix using the factors computed by SPBCO or SPBFA.
SPBFA	SPBTRF	factors a symmetric positive definite matrix stored in band form.
SPBSL	SPBTRS	solves the symmetric positive definite band system $Ax = b$ using the factors computed by SPBCO or SPBFA.
SPOCO	SPOTRF	factors a symmetric positive definite
	SPOCON	matrix and estimates the condition of the matrix.
SPODI	SPOTRI	computes the determinant and inverse of a certain symmetric positive definite matrix using the factors computed by SPOCO, SPOFA or SQRDC.

SVD	driver	Compute the singular value decomposition
TINVIT	SSTEIN	Compute the eigenvectors corresponding to given eigenvalues of a symmetric tridiagonal matrix, using inverse iteration
TQL1	SSTEQR	Compute all eigenvalues using the QL algorithm
TQL2	SSTEQR or SSTEDC	Compute all eigenvalues and eigenvectors using the QL method; if the eigenpairs of a symmetric matrix are desired, input the similarity transformation computed by TRED2
TQLRAT	SSTEQR	Determine all eigenvalues of a symmetric tridiagonal by the rational QL method
TRBAK1	SORMUL	Forms the eigenvectors of a real symmetric matrix from the eigenvectors of that symmetric tridiagonal matrix determined by TRED1
TRBAK3	SORMUL	Forms the eigenvectors of a real symmetric matrix from the eigenvectors of that symmetric tridiagonal matrix determined by TRED3
TRED1	SSYTRD	Reduce to symmetric tridiagonal form using Householder transformations
TRED2	SSYTRD and SORMUL	Reduce to symmetric tridiagonal form using Householder transformations; the similarity transformation that yields the tridiagonal form is also constructed
TRED3	SSPTRD	Reduce to symmetric tridiagonal form using Householder transformations; input matrix stored in packed form
TRIDIB	SSTEBM	Compute those eigenvalues between specified indices using the Sturm sequence property
TSTURM	SSTEBM and SSTEIN	Compute those eigenvalues in a specified interval using the Sturm sequence property; the corresponding eigenvectors are computed using the inverse iteration

RATQR	SSTEBM	Determine extreme eigenvalues of a symmetric tridiagonal matrix using the QR method with Newton corrections
REBAK	(STRSM)	Given the eigenvectors of the symmetric matrix output by REDUC or REDUC2, compute the eigenvectors corresponding to the original generalized eigenproblem
REBAKB	(STRMM)	Given the eigenvectors of the symmetric matrix output by REDUC2, compute the eigenvectors corresponding to the original eigenproblem $ABx = \lambda x$
REDUC	SSYGST	Reduce the symmetric generalized eigenproblem $Ax = \lambda Bx$, where B is positive definite to the standard symmetric eigenproblem using the Cholesky factorization of B
REDUC2	SSYGST	Reduce the eigenvalue problem $ABx = \lambda x$, where both A and B are symmetric and either A or B is positive definite to the standard symmetric eigenproblem using the Cholesky factorization
RG	driver	Compute eigenvalues and optionally eigenvectors of a real general matrix (driver routine)
RGG	driver	Compute eigenvalues and optionally eigenvectors of a real general generalized system $Ax = \lambda Bx$ (driver routine)
RS	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric matrix (driver routine)
RSB	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric band matrix (driver routine)
RSG	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric generalized system $Ax = \lambda Bx$, where A is symmetric and B is positive definite (driver routine)
RSGAB	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric generalized system $ABx = \lambda x$, where A is symmetric and B is positive definite (driver routine)
RSGBA	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric generalized system $BAx = \lambda x$, where A is symmetric and B is positive definite (driver routine)
RSM	driver	Compute some eigenvalues and optionally eigenvectors of a real symmetric matrix (driver routine)
RSP	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric matrix stored in packed form(driver routine)
RST	driver	Compute eigenvalues and optionally eigenvectors of a real symmetric tridiagonal matrix (driver routine)
RT	driver	Compute eigenvalues and optionally eigenvectors of a real tridiagonal matrix for which $a_{i,i+1}a_{i+1,i} \geq 0$ for every i (driver routine)

HTRIB3	CUNMUL	Given eigenvectors of the real symmetric tridiagonal matrix output by HTRID3, compute the corresponding eigenvectors of the original matrix
HTRIBK	CUNMUL	Given eigenvectors of the real symmetric tridiagonal matrix output by HTRIDI, compute the corresponding eigenvectors of the original matrix
HTRID3	CHPTRD	Reduce to symmetric tridiagonal matrix using Householder matrices; input matrix stored in packed form
HTRIDI	CHETRD	Reduce to symmetric tridiagonal matrix using Householder matrices
IMTQL1	SSTEQR or SSTEDC	Compute eigenvalues using the implicit QL method
IMTQL2	SSTEQR or SSTEDC	Compute the eigenvalues and eigenvectors using the implicit QL method;
IMTQLV	SSTEQR	Compute eigenvalues using implicit QL method while preserving the input matrix
INVIT	SHSEIN	Compute eigenvectors corresponding to given eigenvalues of an upper Hessenberg matrix, using inverse iteration
MINFIT	driver	For the linear system $Ax = b$, compute the singular value decomposition $A = QSP^T$ and the vector Q^Tb
ORTBAK	SORMUL	Given eigenvectors of the upper Hessenberg matrix output by ORTHES, compute the corresponding eigenvectors of the original matrix
ORTHES	SGEHRD	Reduce to upper Hessenberg form using Householder matrices
ORTRAN	SORGEN	Use the output of ORTHES to construct the similarity transformation that generates the upper Hessenberg form
QZHES	SGEQRF SORMUL SGGHRD	Reduce the generalized eigenproblem to standard form, where one matrix is upper Hessenberg and the other matrix is upper triangular
QZIT + QZVAL	SHGEQR	Given the generalized eigenproblem $Ax = \lambda Bx$ where A is upper Hessenberg and B is upper triangular, reduce A to quasi-upper triangular form using the QZ algorithm and compute the eigenvalues for the generalized eigenproblem $Ax = \lambda Bx$, where A is quasi-upper triangular and B is upper triangle
QZVEC	STGEVC (STRMM)	Given the eigenvalues for the generalized eigenproblem $Ax = \lambda Bx$, where A is quasi-upper triangular and B is upper triangular, compute the corresponding eigenvectors

6 Appendix B

6.1 LINPACK and EISPACK Counterparts

EISPACK	LAPACK	Function
BAKVEC		Invert the balancing made by FIGI ()
BALANC	SGEBAL	Apply balancing transformations
BALBAK	SGEBAK	Invert the balancing transformation made by BALANC (SGEBAL)
BANDR	SSBTRD	Reduce to symmetric tridiagonal form
BANDV		Given approximate eigenvalues of a band matrix, use inverse iteration to obtain corresponding eigenvectors
BISECT	SSTEBM	Determine eigenvalues of a symmetric tridiagonal matrix that lie on specified interval using Sturm sequences
BQR		Determine some eigenvalues using the QR method
CBABK2	CGEBAK	Invert the balancing transformation made by CBAL (CGEBAL)
CBAL	CGEBAL	Apply balancing transformations
CG	driver	Compute eigenvalues and optionally eigenvectors of a complex general matrix (driver routine)
CH	driver	Compute eigenvalues and optionally eigenvectors of a complex Hermitian matrix (driver routine)
CINVIT	CHSEIN	Given approximate eigenvalues, use inverse iteration to obtain corresponding eigenvectors
COMBAK	Note A	Given eigenvectors of upper Hessenberg matrix computed by COMHES (), compute corresponding eigenvectors of the original matrix.
COMHES	Note A	Reduce to upper Hessenberg form using elimination
COMLR	Note A	Compute all eigenvalues using modified LR algorithm
COMLR2	Note A	Compute all eigenvalues and eigenvectors using modified LR method
COMQR	CHSEQR	Compute all eigenvalues using QR algorithm
COMQR2	CUNGEN CHSEQR CTREVC (CTRMM)	Compute all eigenvalues and eigenvectors using QR algorithm
CORTB	CUNMUL	Given eigenvectors of upper Hessenberg matrix computed by CORTH, compute corresponding eigenvectors of original matrix
CORTH	CGEHRD	Reduce to upper Hessenberg form using Householder matrices
ELMBAK	Note A	Given eigenvectors of the upper Hessenberg matrix output by ELMHES, compute corresponding eigenvectors of original matrix
ELMHES	Note A	Reduce to upper Hessenberg form using elimination
ELTRAN	Note A	Use the output of ELMHES to construct the similarity transformation that generates the upper Hessenberg form
FIGI		Use a balancing transformation to symmetrize a nonsymmetric tridiagonal matrix for which $a_{i,i+1}a_{i+1,i} \geq 0$ for every i
FIGI2		Similar to FIGI except that the balancing transformation is also output
HQR	SHSEQR	Compute all eigenvalues using the implicit QR method
HQR2	SHSEQR STREVC (STRMM)	Compute all eigenvalues and eigenvalues and eigenvectors using the implicit QR method

```
            CALL SGEMM( 'No transpose', 'Transpose',
     $                  N - J - JB + 1, JB, J - 1,
     $                  -ONE, A( J + JB, 1 ), LDA, A( J, 1 ), LDA,
     $                  ONE, A( J + JB, J ), LDA)
*
*           Compute subdiagonal block of L.
*
            CALL STRSM( 'Right', 'Lower', 'Transpose', 'Non-unit',
     $                  N - J - JB + 1, JB, ONE, A( J, J ), LDA,
     $                  A( J + JB, J ), LDA )
   20    CONTINUE
         ENDIF
         GO TO 40
*
   30 CONTINUE
         INFO = INFO + J - 1
*
   40 CONTINUE
         RETURN
*
*     End of SPOTRF
         END
```

```
*
            CALL SPOTF2( 'Upper', JB, A( J, J ), LDA, INFO )
            IF( INFO.NE.0 ) GO TO 30
*
*           Update superdiagonal block.
*
            CALL SGEMM( 'Transpose', 'No Transpose',
     $                  JB, N - J - JB + 1, J - 1,
     $                  -ONE, A( 1, J ), LDA, A( 1, J + JB ), LDA,
     $                  ONE, A( J, J + JB ), LDA)
*
*           Compute superdiagonal block of U.
*
            CALL STRSM( 'Left', 'Upper', 'Transpose', 'Non-unit',
     $                  JB, N - J - JB + 1, ONE, A( J, J ), LDA,
     $                  A( J, J + JB ), LDA )
   10    CONTINUE
      ELSE
*
*        Compute the Cholesky factorization of a symmetric matrix
*        stored in the lower part of the array.
*
         DO 20 J = 1, N, NB
            JB = MIN( NB, N - J + 1 )
*
*           Update diagonal block.
*
            CALL SSYRK( 'Lower', 'No transpose', JB, J - 1,
     $                  -ONE, A( J, 1 ), LDA, ONE, A( J, J ), LDA )
*
*           Factorize diagonal block and test for
*           non-positive-definiteness.
*
            CALL SPOTF2( 'Lower', JB, A( J, J ), LDA, INFO )
            IF( INFO.NE.0 ) GO TO 30
*
*           Update subdiagonal block.
*
```

```
      INFO = 0
*
*     Quick return if possible.
*
      IF( N.EQ.0 ) RETURN
      IF( ( .NOT.LSAME( UPLO  , 'U' ) ).AND.
     $    ( .NOT.LSAME( UPLO  , 'L' ) )      )THEN
         INFO = -1
      ELSE IF( N.LT.0 )THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) )THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 )THEN
         CALL XERBLA( 'SPOTRF', -INFO )
         RETURN
      END IF
*
*     Determine the block size for this environment.
*
      CALL ENVIR( 'Get', NB )
      IF( NB.EQ.1 ) NB = N
*
      IF( LSAME( UPLO, 'U' ) )THEN
*
*        Compute the Cholesky factorization of a symmetric matrix
*        stored in the upper part of the array.
*
         DO 10 J = 1, N, NB
            JB = MIN( NB, N - J + 1 )
*
*           Update diagonal block.
*
            CALL SSYRK( 'Upper' , 'Transpose', JB, J - 1,
     $                  -ONE, A( 1, J ), LDA, ONE, A( J, J ), LDA )
*
*           Factorize diagonal block and test for
*           non-positive-definiteness.
```

```
*           On entry, A specifies the array which contains the matrix
*           being factored.
*           On exit, the array A is overwritten by the
*           Cholesky factorization. The factorization can be written as
*           either A = L*L' where L is a lower triangular matrix
*           or as A = U'*U where U is an upper triangular matrix.
*
*  LDA    - INTEGER.
*           On entry, LDA specifies the first dimension of A as declared
*           in the calling (sub) program.
*           LDA must be at least  max( 1, N ).
*           Unchanged on exit.
*
*  INFO   - INTEGER.
*           On exit, a value of 0 indicates a normal return.
*           A positive value K indicates that the leading minor of
*           order K is not positive definite, which is an error
*           condition that causes the subroutine to end.
*           A negative value, say -K, indicates the K-th argument has an
*           illegal value.
*
*     .. Parameters ..
      REAL               ONE
      PARAMETER          ( ONE = 1.0E+0 )
*     .. Local scalars ..
      INTEGER            J, JB, NB
*     .. External functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*     .. External subroutines ..
      EXTERNAL           ENVIR, SGEMM, SPOTF2, SSYRK, STRSM, XERBLA
*     .. Intrinsic functions ..
      INTRINSIC          MAX, MIN
*     ..
*     .. Executable Statements ..
*
*     Test the input parameters.
*
```

5.2 SPOTRF

```
      SUBROUTINE SPOTRF( UPLO, N, A, LDA, INFO )
*
*  -- LAPACK routine --
*     Argonne National Laboratory
*     September 14, 1988
*
*     .. Scalar arguments ..
      CHARACTER*1        UPLO
      INTEGER            N, LDA, INFO
*
*     .. Array arguments ..
      REAL               A( LDA, * )
*
*  Purpose
*  =======
*
*     SPOTRF computes the Cholesky factorization of a symmetric
*     positive definite matrix A.
*     This is the Level 3 BLAS version of the algorithm, reducing NB
*     columns at a time.
*
*  Arguments
*  =========
*
*  UPLO   - CHARACTER*1.
*           On entry, UPLO specifies whether the upper or lower
*           triangular part of the symmetric matrix A is stored.
*              UPLO = 'U' or 'u'   The upper triangle of A is stored.
*              UPLO = 'L' or 'l'   The lower triangle of A is stored.
*           Unchanged on exit.
*
*  N      - INTEGER.
*           On entry, N specifies the number of columns of the matrix
*           A . N must be at least zero.
*           Unchanged on exit.
*
*  A      - REAL             array of DIMENSION ( LDA, N ).
```

```
   40 CONTINUE
      RETURN
*
*     End of SGETRF
      END
```

```
            IF( IP.NE.I )
     $         CALL SSWAP( JB, A( I, J ), LDA, A( IP, J ), LDA )
   10    CONTINUE
*
*        Compute superdiagonal block of U.
*
         CALL STRSM( 'Left', 'Lower', 'No transpose', 'Unit', J - 1,
     $               JB, ONE, A, LDA, A( 1, J ), LDA )
*
*        Update diagonal and subdiagonal blocks.
*
         CALL SGEMM( 'No transpose', 'No transpose', M - J + 1, JB,
     $               J - 1, -ONE, A( J, 1 ), LDA, A( 1, J ), LDA, ONE,
     $               A( J, J ), LDA )
*
*        Factorize diagonal and subdiagonal blocks and test for exact
*        singularity.
*
         CALL SGETF2( M - J + 1, JB, A( J, J ), LDA, IPIV( J ), INFO )
         DO 20 I = J, J + JB - 1
            IPIV( I ) = J - 1 + IPIV( I )
   20    CONTINUE
         IF( INFO.EQ.0 ) THEN
*
*           Apply interchanges to previous blocks.
*
            DO 30 I = J, J + JB - 1
               IP = IPIV( I )
               IF( IP.NE.I )
     $            CALL SSWAP( J - 1, A( I, 1 ), LDA, A( IP, 1 ), LDA )
   30       CONTINUE
         ELSE
*
*           If INFO is not zero, a zero pivot was found in SGETF2.
*           Correct the index returned from SGETF2 and go on.
*
            INFO = INFO + J - 1
         ENDIF
```

```
      EXTERNAL           ENVIR, SGEMM, SGETF2, SSWAP, STRSM, XERBLA
*     .. Intrinsic functions ..
      INTRINSIC          MAX, MIN
*     ..
*     .. Executable Statements ..
*     Gaussian elimination with partial pivoting
*
*     Test the input parameters.
*
      INFO = 0
*
*     Quick return if possible.
*
      IF( M.EQ.0 .OR. N.EQ.0 ) RETURN
      IF( M.LT.0 )THEN
         INFO = -1
      ELSE IF( N.LT.0 )THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, M ) )THEN
         INFO = -4
      END IF
      IF( INFO.NE.0 )THEN
         CALL XERBLA( 'SGETRF', -INFO )
         RETURN
      END IF
*
*     Determine the block size for this environment.
*
      CALL ENVIR( 'Get', NB )
      IF( NB.EQ.1 ) NB = N
*
      DO 40 J = 1, N, NB
         JB = MIN( N - J + 1, NB )
*
*        Apply previous interchanges to current block.
*
         DO 10 I = 1, J - 1
            IP = IPIV( I )
```

```
*  N      - INTEGER.
*           On entry, N specifies the number of columns of the matrix
*           A . N must be at least zero.
*           Unchanged on exit.
*
*  A      - REAL             array of DIMENSION ( LDA, N ).
*           On entry, A specifies the array which contains the matrix
*           being factored.
*           On exit, the array A is overwritten by the
*           LU factorization. The factorization can be written as
*           A = L*U where L is a product of permutation and unit lower
*           triangular matrices and U is an upper triangular matrix.
*
*  LDA    - INTEGER.
*           On entry, LDA specifies the first dimension of A as declared
*           in the calling (sub) program.
*           LDA must be at least  max( 1, M ).
*           Unchanged on exit.
*
*  IPIV   - INTEGER           array of DIMENSION ( M ).
*           On exit, the array IPIV contains the pivot indices.
*
*  INFO   - INTEGER.
*           On exit, a value of 0 indicates a normal return; a positive
*           value, say K, indicates that U(K,K) = 0.0 exactly.
*           This is not an error condition for this subroutine, but it
*           does indicate that SGETRS or SGETRI will divide by zero
*           if called. Use routine SGECON for a reliable indication of
*           singularity.
*           A negative value, say -K, indicates the Kth argument has an
*           illegal value.
*
*     .. Parameters ..
      REAL               ONE
      PARAMETER          ( ONE = 1.0E+0 )
*     .. Local scalars ..
      INTEGER            I, IP, J, JB, NB
*     .. External subroutines ..
```

5 Appendix A

We include here prototype code for two LAPACK routines. This code is intended primarily to show the typical style and structure of LAPACK routines. Other variants of these particular routines are possible, and we make no claim that the variants illustrated here give the best performance.

In addition to Level 3 BLAS, each routine calls an unblocked version of the same algorithm (subroutines SGETF2 and SPOTF2).

5.1 SGETRF

```
      SUBROUTINE SGETRF( M, N, A, LDA, IPIV, INFO )
*
*  -- LAPACK routine --
*     Argonne National Laboratory
*     September 14, 1988
*
*     .. Scalar arguments ..
      INTEGER            M, N, LDA, INFO
*     .. Array arguments ..
      INTEGER            IPIV( * )
      REAL               A( LDA, * )
*
*  Purpose
*  =======
*
*     SGETRF computes the LU factorization of a general m-by-n
*     matrix A, using partial pivoting with row interchanges.
*     This is the Level 3 BLAS version of the algorithm, reducing NB
*     columns at a time.
*
*  Arguments
*  =========
*
*  M      - INTEGER.
*           On entry, M specifies the number of rows of the matrix
*           A .  M must be at least zero.
*           Unchanged on exit.
*
```

to record entry points and addresses as there is in the general SCHEDULE mechanism. Moreover, all of the code will be in Fortran. Since there will on the order of three or four possibilities for different subroutines executing in parallel within a given LAPACK algorithm, a simple examination of cases will suffice to decide which subroutine should be executed with respect to a given process descriptor.

Preferably a loop-based mechanism will be employed to get the generic SCHEDULE work subroutines executing in parallel. Critical sections will be constructed from whatever synchronization primitive is available on the given machine. A simple lock mechanism ("lockon" and "lockoff") are sufficient for this purpose but other equivalent mechanisms might be used in their place. In keeping with the discussion of workspace in Section 5.4, there will be no use of named common as was done in SCHEDULE. Instead the required shared work space will be passed as parameters and shared through calling sequences as needed.

4.8 Mixed Language Programming

LAPACK will be coded in Fortran 77 and designed to be called from Fortran 77 programs. However, we hope to gain experience of calling LAPACK from other programming languages, for example C or Ada, and to be able to give advice about it in the final LAPACK documentation.

Some LAPACK routines will also require access to values related to the range of real numbers on the machine, in order to avoid overflow or underflow by suitable scaling. These values, BIG (the largest "safe" number in the machine) and TINY (the smallest positive "safe" number in the machine) will be made available by a numeric enquiry function. Like the relative machine precision, TINY can be computed in a reliable machine independent way, but BIG can not. Instead, the portable version will return a conservative value such as 1.0E+35 for BIG that is safe for most known machines. This value could be modified to correspond to whatever machine is being used. The only disadvantage of using a smaller value than the machine could permit will be that scaling is performed somewhat more often than strictly necessary. In addition to the relative machine precision, TINY and BIG, the radix will also be made available, computed in a machine independent way.

4.7 Provision for Parallel Execution

The loop-based aspect of parallelism is generally straightforward. Many of the vendors currently give adequate support to the concept of loop-based parallelism. We envision invoking this within the Level 3 BLAS and perhaps also within the Level 2 BLAS. We are following the activities of the Parallel Computing Forum [23] which has been formed by computer vendors, software developers, national laboratories, and universities to exchange technical information and to document agreements on constructs for programming parallel applications for shared memory parallel processors. The Forum is planning to issue a draft proposal for parallel Fortran constructs by the end of the summer.

In all the cases we are aware of when loop-based parallelism is invoked at a higher level, subsequent invocations at a lower level of a nested loop are either negated or properly queued to ensure a correct parallel execution as long as the machine specific loop-based mechanisms are used. Therefore, we do not expect to suffer from the problems associated with a user invoking parallelism at a level that is above a call to an LAPACK routine that depends upon BLAS that also invoke parallelism.

Several of the algorithms we intend to implement will require more than loop-based parallelism. These algorithms will rely upon the simplified SCHEDULE [24] mechanism to invoke parallelism. We refer the reader to [24] for terminology and ideas used in the remainder of this section. The simplifications to SCHEDULE will include a reduction in the layers of subroutine calls between the act of placing a process descriptor in the computational graph and its subsequent execution. It will also replace the data structures that were constructed for general use with ones that are specific to the application. This will reduce overhead involved with special cases and error checking that is necessary in the general case but not in the specific algorithms that will arise in LAPACK. There will be generic "work" routines specific to each algorithm which will receive information required to identify and invoke a process by decoding integer arguments. There will not be any need

block size. This raises a difficulty since the block size will vary in different environments, and in any case will not be known by the user. Our proposal in such cases is to advise the user to supply a work array of length lw, say, where lw is also passed as an argument to the routine, and lw is as large as is convenient; the routine can then compute $nbmax = lw/n$, and use $nbmax$ as an upper limit on the block size. Thus the block size used may be less than optimal if insufficient workspace has been provided, but our present evidence indicates that speed is comparatively insensitive to variations in block size over quite a wide range on either side of the optimum.

4.5 Array Arguments

All array arguments will be declared as assumed-size arrays (last dimension *),e.g.

```
REAL A(LDA,*), W(*)
```

This has two advantages over declarations as adjustable arrays such as:

```
REAL A(LDA,N), W(N)
```

- The routines can be called with $N = 0$, without contravening the Fortran 77 standard.
- For 2-dimensional arrays, the corresponding actual argument can be an arbitrary element of a 2-dimensional array in the calling program, again without contravening the standard.

There is one restriction of standard Fortran which we prefer not to observe. It does not affect the way in which LAPACK routines are called, but does affect the code within LAPACK routines, when lower level routines such as the Level 3 BLAS are called. The standard requires that if a 2-dimensional array is declared as A(LDA,*), then the actual array passed must be at least LDA elements long. This implies a contravention of the standard if the actual argument is an element of the last column of the array in the calling program, say A(I,N) with I > 1. We know of only one compiler which is capable of detecting this contravention. Rather than introduce special code to handle such cases, we assume that the lower level routines will be compiled without these checks being performed.

4.6 Numerical machine-dependencies

Many LAPACK routines will require the value of the relative machine precision. We propose to make this available through an enquiry function where the value can be computed in a reliable portable manner (or if an installer desires, a specific value can be programmed in). We prefer explicit reference to an enquiry function, rather than attempting to compute the value in-line wherever it is needed, or relying on tests such as TEST .NE. TEST + DELTA.

routine will be designed so that the block size can be specified by the calling program, and the effects of varying the block size can be studied.

On many machine architectures (for example, most scalar machines and some single processor vector machines), block algorithms offer no advantage over an unblocked version. A block algorithm executed with block size equal to 1 would have the same structure as the unblocked version of the same algorithm, but would be inefficient because it would use calls to Level 3 BLAS, where calls to Level 1 or 2 BLAS would be sufficient. For some block algorithms (for example, SGETRF and SPOTRF in Appendix A), setting the block size to n (or greater), where n is the order of the matrix, has the effect of forcing execution of the unblocked version (the whole matrix is treated as a single block). But for other block algorithms (for example, those described in Working Note #2), this is not so. In order to have a consistent convention, we shall ensure (by special code) that a block size of 1 forces execution of an efficient unblocked version of each block algorithm.

Hitherto we have envisaged routines working with a fixed block size (though this will vary from one installation to another, possibly also from one routine to another). A more sophisticated strategy is to allow the block size to vary dynamically during the course of the algorithm - for example, allowing the block size to increase in order to keep roughly constant the size of submatrices passed to Level 3 BLAS matrices. We plan to investigate whether or not dynamic blocking would offer significant benefits in performance. It seems that implementing it would involve hardly any extra complication in the code, but a more elaborate procedure would be required to determine a good dynamic blocking strategy for each machine [11].

4.4 Workspace

Many LAPACK routines will require workspace. We do not think that the mechanism for automatic workspace allocation devised by Fox, Hall and Schryer [22] for the Port library is suitable for LAPACK. It involves the use of a shared labeled COMMON block, which is likely to cause difficulties on multi-tasking machines, and requires non-standard Fortran usage if the user wishes to use more than the default amount of workspace.

Therefore work arrays will need to be passed as arguments to LAPACK routines. Since shortage of memory is not likely to be a serious constraint on the machines for which LAPACK is primarily targeted, we think it reasonable for a routine to require workspace equivalent to several vectors of length n, where n is the order of the matrix; and we may design routines to use more than the minimum possible amount of workspace if this significantly improves their performance. However, we shall avoid workspace of $O(n^2)$ elements, unless absolutely necessary.

A number of routines implementing block algorithms will require workspace sufficient to hold one block of columns of the matrix, that is, workspace of size $n \times nb$, where nb is the

```
CALL SPOTRF ('Upper triangle', . . . )
```

The significant initial character may be in upper or lower case.

It will be permissible for the problem dimensions to be passed as zero, in which case the computation (or part of it) will be skipped. (See also section 5.5.) Negative dimensions will be regarded as an error.

Each 2-dimensional array argument will be immediately followed in the argument-list by its leading dimension, whose name will have the form LD<array-name>.

All documented routines will have a diagnostic argument INFO. (See next section.)

4.2 Error-handling

The diagnostic argument INFO will indicate the success or failure of the computation:

- INFO = 0: successful termination
- INFO < 0: illegal value of argument - no computation performed
- INFO > 0: failure in the course of computation

All documented routines will check that input arguments such as N, LDA, have permitted values, even if the same checks are repeated by lower level routines. This means that any error-message can name the routine that the user called, rather than a lower level routine that he may be unaware of.

If an illegal value of the i-th argument is detected, the routine will first call an error-handling routine XERBLA and then set INFO = $-i$. XERBLA has the same specification as in the Level 2 and Level 3 BLAS: its 1st argument is the name of the calling routine, and its 2nd argument is the number of the argument with an illegal value. The model implementation of XERBLA prints a message and stops, but this is open to modification by installers.

We do not propose to call any error-handling routine such as XERBLA before an exit with INFO > 0.

4.3 Choice of Block Size

Routines which implement block algorithms will need to obtain a value for the block size from an enquiry routine. Determining optimal, or near optimal, block sizes for different environments is a major research topic for the LAPACK project. The optimum is likely to depend on several factors, such as the architecture of the machine, the dimensions of the problem, and the current state of the system (for example, the cache/local memory size or the number of processors available). In the preliminary phase of testing, the enquiry

3.3 Questions for the Community

For convenience we summarize here those questions on which we would particularly welcome feedback:

- Should we provide the facility to work with either the upper or the lower triangle of a symmetric matrix (see 3.1.3)? If the answer is yes, should we provide the same facility in routines for the symmetric eigenvalue problem (see 3.2.1)?
- Should we provide a backward Cholesky factorization instead of, or as optional alternative to, the usual Cholesky factorization, if it is significantly faster (see 3.1.3)?
- Have we provided sufficient facilities for computing or updating QR and related factorizations (see 3.1.13)?
- Should we provide routines for systems of equations with other kinds of special structure, for example; block tridiagonal, almost block diagonal ("staircase"), arrow head? Our feeling at this state is that we should not, or at least that we should postpone any work on them. In some cases (e.g. for symmetric positive-definite arrowhead systems), it may be possible for us to illustrate how routines to solve them can be built out of other LAPACK components.

4 Aspects of Software Design

4.1 Design of Calling Sequences

Arguments of an LAPACK routine will appear in the following order:

- arguments specifying options
- problem dimensions
- array or scalar arguments defining the input data; some of them may be overwritten by results
- other array or scalar arguments returning results
- work arrays (and associated array dimensions)
- diagnostic argument INFO

The examples in Appendix A illustrate what this ordering implies in practice.

Arguments specifying options will usually be CHARACTER*1 arguments, as in the Level 2 and Level 3 BLAS. They have the advantage that a longer character string can be passed as the actual argument, making the calling program more readable, e.g.

3.2.5 Unsymmetric generalized eigenproblems

The routines in this group deal with square matrix pencils (A,B) in which B is upper triangular; a QR-factorization of B can be used to achieve this form initially.

SGGHRD	reduce a pencil (A,B) to one in which A is upper Hessenberg
SHGEQR	all or part of generalized Schur factorization of a matrix pencil (A,B) in which A is upper Hessenberg
STGEVC	eigenvectors of a pencil (A,B) in which A is upper quasi-triangular
SHGEIN	selected eigenvectors of a matrix pencil (A,B) in which A is upper Hessenberg, by inverse iteration
SGGBAL	balance a matrix pencil
SGGBAK	backtransform eigenvectors to those of a pencil balanced by SGGBAL
STGSEN	computes or estimates condition numbers associated with a single deflating subspace
STGSNA	computes or estimates condition numbers associated with each generalized eigenvalue-eigenvector pair.
STGSYL	solve triangular generalized Sylvester equation
STGEXC	exchange adjacent diagonal elements or blocks of a pencil (A,B) in which A is upper quasi-triangular

Notes:

- a prototype for STGEXC is the subroutine EXCHQZ of Van Dooren [20].
- for SGGBAL see Ward [21].
- STGSYL will solve the equation $AX + YB = C, DX + YE = F$ when A, B, C and D are upper triangular or quasi-triangular. This routine will be needed by STGSEN and STGSNA.

3.2.6 Generalized singular value problems

STGSJA	all or part of generalized SVD of a pair of triangular matrices; generalized singular values, and optionally vectors, using Jacobi's method

Notes:

- to compute the GSVD of a pair of rectangular matrices, it is assumed that STGSJA will be proceeded by a call of SGGQRP (see 3.1.14).
- STGSJA will take triangular A and B and return orthogonal U, V, Q, triangular R, and diagonal C and S, such that UAQ=CR, VBQ=SR. R is overwritten on top of A. It requires workspace for extra copies of both A and B.

3.2.3 Singular value problems

SGEBRD	reduce a rectangular matrix to upper bidiagonal form
STRBRD	reduce an upper triangular matrix to upper bidiagonal form
SGBBRD	reduce a band matrix to upper bidiagonal form
SBDSQR	all or part of singular value factorization of upper bidiagonal matrix, by QR algorithm
SBDSDC	singular value factorization of upper bidiagonal matrix using a divide-and-conquer algorithm
SBDSIN	selected singular vectors of upper bidiagonal matrix, by inverse iteration
SBDSBM	selected singular values of upper bidiagonal matrix, by bisection/multisection
SBDSVU	singular values and vectors of rank-1 update of upper bidiagonal matrix

Notes:

- For reduction to bidiagonal form, two paths are provided: either direct reduction by SGEBRD, or QR-factorization by SGEQRF followed by reduction of the upper triangular factor by STRBRD.
- A block algorithm for SGEBRD is discussed in Working Note #2.
- Algorithms and related issues concerning SBDSVF, SBDSDC, SBDSIN and SBDSBM are discussed in Working Note #3 and in Chapter 2 of Working Note #4.
- SGBBRD will reduce a band matrix to bidiagonal form while preserving the band structure, using sequences of plane rotations in a similar manner to SSBTRD (which corresponds to the EISPACK routine BANDR).

3.2.4 Symmetric-definite generalized eigenproblems

SSYGST	reduce problem to standard form
SSPGST	as SSYGST using packed storage
SSBGST	as SSYGST for band matrices
SDBEBM	Szyld's bisection/Rayleigh quotient algorithm for band matrices.

Notes:

- SSBGST will be based on the algorithm of Crawford [19].
- to backtransform eigenvectors of the standard problem to those of the generalized problem use STRSM after reduction by SSYGST, or STPSV after reduction by SSPGST.

3.2.2 Unsymmetric eigenvalue problems

SGEHRD	reduce unsymmetric matrix to upper Hessenberg form
SHSEQR	all or part of Schur factorization of upper Hessenberg matrix
STREVC	eigenvectors of upper quasi-triangular matrix
SHSEIN	selected eigenvectors of upper Hessenberg matrix, by inverse iteration
SGEBAL	balance an unsymmetric matrix
SGEBAK	backtransform eigenvectors to those of the matrix balanced by SGEBAL
STRSEN	computes or estimates condition numbers associated with a single invariant subspace
STRSNA	computes or estimates condition numbers associated with all eigenvalue-eigenvector pairs
STRSYL	solve quasi-triangular Sylvester equation
STREXC	exchange adjacent diagonal elements or blocks of upper quasi-triangular matrix

Notes:

- a block algorithm for SGEHRD is described in Working Note #2.
- block QR methods are being investigated for SHSEQR
- STREVC will have options to compute either left or right eigenvectors
- a prototype for STREXC is the algorithm of Stewart [14]. See also Ng and Parlett [15].
- STRSEN will require the user to specify the eigenvalues which define the desired invariant subspace. We expect to base this routine on the methods of Van Loan [16].
- STRSNA will be based on the algorithm of Chan, Feldman and Parlett [17] for computing the sensitivities of the eigenvalues, and on the methods of Van Loan [16] for estimating the condition numbers of the eigenvectors.
- STRSYL will solve the equation $AX + XB = C$ when A and B are both upper triangular or quasi-triangular. This routine will be needed by STRSEN and STRSNA. Block algorithms are being investigated by Kågström [18].

Notes:

- The routines for unsymmetric problems allow the Schur factorization to be computed, with a separate routine for computing eigenvectors of the triangular Schur factor.
- For backtransformation of eigenvectors, either the Level 3 BLAS routines STRMM or STRSM or the routine SORMUL can be called as appropriate; hence no routines specifically for backtransformation have been proposed (except after balancing). But would users prefer the calls to be packaged into specific back-transformation routines?

3.2.1 Symmetric eigenvalue problems

SSYTRD	reduce symmetric matrix to tridiagonal form
SSPTRD	reduce symmetric matrix in packed storage to tridiagonal form
SSBTRD	reduce symmetric band matrix to tridiagonal form
SSTEQR	all eigenvalues and optionally all eigenvectors of symmetric tridiagonal matrix, using QR algorithm
SSTEDC	all eigenvalues and eigenvectors of symmetric tridiagonal matrix, using a divide-and-conquer algorithm
SSTEIN	selected eigenvectors of symmetric tridiagonal matrix, by inverse iteration
SSTEBM	selected eigenvalues of symmetric tridiagonal matrix, by bisection/multisection
SSTEVU	eigenvalues and eigenvectors of rank-1 update of symmetric tridiagonal matrix
SSBEBM	eigenvalues of symmetric banded matrix using Szyld's algorithm

Notes:

- A block algorithm for SSYTRD is described in Working Note #2
- We are considering the possibility of allowing SSYTRD, SSPTRD and SSBTRD to work with either the upper or lower triangle of the symmetric matrix, according to the value of an option argument UPLO.
- Issues concerned with the choice of method for SSTEQR, SSTEDC and SSTEBM are discussed in chapter 1 of Working Note #4. Resolution of those issues may possibly result in a different structure of routines from that proposed here.
- SSTEIN is intended for computing eigenvectors by inverse iteration corresponding to eigenvalues which have already been computed by SSTEQR or SSTEBM
- to form the orthogonal matrix used for the reduction in SSYTRD, use SORGEN.
- to back-transform eigenvectors computed by SSTEQR or SSTEDC or SSTEIN to those of an original symmetric matrix, use SORMUL.

- SGERQF is intended primarily for factorizing an m-by-n matrix with $m \leq n$ as $[0 : R]Q$ where R is upper triangular. This is needed in some applications in nonlinear optimization, and also as a first step in computing the SVD of an m-by-n matrix with $m \leq n$. A block algorithm analogous to that for SGEQRF can be used.

- STZRQF will factorize an m-by-n upper trapezoidal matrix with $m \leq n$ as $[R : 0]Q$ where R is upper triangular. This is needed to compute the complete orthogonal factorization of a rank-deficient matrix and hence to obtain the minimum-norm solution of rank-deficient linear least squares problems (see Lawson and Hanson [12] for details).

- SGEQRS may provide only the straightforward solution of a full-rank linear least squares problem, that is, not necessarily all of the functions provided by the LINPACK routine SQRSL. Other functions provided by SQRSL can be obtained by separate calls of SORMUL.

- SORMUL will have options to compute QB, Q^TB, BQ or BQ^T for given B (overwriting the result on B).

- SORGEN will allow the factor Q in a QR factorization to be formed explicitly.

- Both SORMUL and SORGEN can use block algorithms.

- SGEQRU will perform a low-rank update of a QR factorization, i.e. $QR + \alpha XY^T = \tilde{Q}\tilde{R}$.

- Note that other updates of QR factorization can be obtained from the routines SPOTRU and SPOTRX (see 3.1.3).

3.1.14 Generalized QR Factorization

SGGQRP	generalized QR factorization of a pair of rectangular matrices (pivoting is necessary)

Notes:

- SGGQRP will compute a generalized QR factorization as defined by Paige [13].

3.2 Eigenvalue Problems

This section of LAPACK is concerned with computing eigenvalues and eigenvectors, singular values and singular vectors, of standard and generalized problems. It covers almost all the facilities of EISPACK as well as many new ones, with the routines being organized more systematically than in EISPACK.

2. perform Cholesky factorization and solve linear equations
3. solve linear equations using the factorization from a previous call

The LINPACK routine SPTSL only performs option 2 but the other options can be provided at little extra cost in complexity.

- SMTSOL is envisaged as implementing the same algorithms as SPTSOL, but allowing vectorization over the systems of equations. This requirement is common when solving P.D.E.'s.

3.1.13 QR factorization and related routines

SGEQRF	QR factorization of a rectangular matrix without pivoting
SGEQRP	as SGEQRF but with column interchanges
SGERQF	RQ factorization of a rectangular matrix
STZRQF	RQ factorization of an upper trapezoidal matrix
SGEQRS	solve linear least squares problem after factorization by SGEQRF or SGEQPR
SORGEN	generate leading columns of an orthogonal matrix which is defined as a product of Householder matrices
SORMUL	multiply a rectangular matrix by an orthogonal matrix which is defined as a product of Householder matrices
SGEQRU	rank-k update of a QR factorization

Notes:

- Block algorithms for SGEQRF have been described by Bischof and Van Loan [8], Walker [9], and Schreiber and van Loan [10].
- Two distinct routines SGEQRP and SGEQRF are proposed: one with, and one without, the facility for column interchanges. The argument list for SGEQRF will be a good deal simpler than that of SGEQRP. SGEQRF is envisaged as a module that will primarily be used as a component in algorithms such as the singular value decomposition and the generalized eigenvalue problem, whereas SGEQRP will be more useful for solving linear least squares problems.
- It is not possible to implement a block algorithm for SGEQRP if arbitrary column interchanges are to be allowed. However, we will investigate the possibility of using a safeguarded local pivoting strategy proposed by Bischof [11]. using interchanges only within the current block provided that this is acceptable. We will consider providing an option to specify either global pivoting or this local pivoting.

3.1.10 Triangular band matrices

STBTRS	solve systems of linear equations
STBRFS	compute error bound
STBCON	estimate condition number

Notes (see also 3.1.8 where relevant):

- These routines will use the same storage scheme as the TB routines in the Level 2 BLAS.

3.1.11 General tridiagonal matrices

SGTSOL	Solve linear equations
SGTTRF	LU-factorization with row interchanges
SGTTRS	solve linear equations after factorization by SGTTRF
SGTRFS	refine solution computed by SGTTRF, with optional error bounds
SGTCON	estimate (or compute?) condition number
SGTEQU	equilibrate matrix

Notes:

- SGTSOL is similar to the LINPACK routine SGTSL: it solves the system of equations directly and does not save full details of the factorization; it is more economical in storage and speed than successive calls to SGTTRF and SGTTRS.
- SGTCON may use Higham's results on computing condition numbers of tridiagonal matrices [5].

3.1.12 Symmetric positive-definite tridiagonal matrices

SPTSOL	solve linear equations
SPTTRF	Cholesky factorization
SPTTRS	solve linear equations after factorization by SPTTRF
SPTRFS	refine solution computed by SPTTRF, with optional error bounds
SPTCON	estimate (or compute?) condition number
SPTEQU	equilibrate matrix
SMTSOL	as SPTSOL but for multiple systems of equations each with its own right hand side

Notes (see also 3.1.11 where relevant):

- SPTSOL will have options to:

  1. perform Cholesky factorization

3.1.7 Symmetric indefinite matrices, packed storage

SSPTRF	Bunch-Parlett factorization
SSPTRS	solve linear equations, after factorization by SSPTRF
SSPTRI	compute inverse, after factorization by SSPTRF
SSPRFS	refine solution computed by SSPTRS, with optional error bounds
SSPCON	estimate condition number, after factorization by SSPTRF
SSPTRU	low-rank update of a Bunch-Parlett factorization
SSPEQU	equilibrate matrix

Notes: See 3.1.6 and 3.1.4 where relevant.

3.1.8 Triangular matrices

STRTRS	solve linear equations
STRTRI	compute inverse
STRRFS	compute error bound
STRCON	estimate condition number

Notes:

- These routines will handle either an upper or a lower triangular matrix, according to the value of an option argument UPLO.
- STRTRS will be little more than an interface to the Level 3 BLAS routine STRSM with the addition of a test for singularity.

3.1.9 Triangular matrices, packed storage

STPTRS	solve systems of linear equations
STPTRI	compute inverse
STPRFS	compute error bound
STPCON	estimate condition number

Notes (see also 3.1.8 where relevant):

- These routines will use the same packed storage scheme as the Level 2 BLAS, namely packing by column.

3.1.5 Symmetric positive-definite band matrices

SPBTRF	Cholesky-factorization
SPBTRS	solution of linear equations, after factorization by SPBTRF
SPBRFS	refine solution computed by SPBTRS, with optional error bounds
SPBCON	estimate condition number, after factorization by SPBTRF
SPBEQU	equilibrate matrix

Notes (See also 3.1.3 where relevant):

- The routines in this group use the same storage scheme when UPLO = 'U' as the PB routines in LINPACK, with the obvious extension when UPLO = 'L'.

3.1.6 Symmetric indefinite matrices

SSYTRF	Bunch - Parlett factorization
SSYTRS	solve linear equations, after factorization by SSYTRF
SSYTRI	compute inverse, after factorization by SSYTRF
SSTRFS	refine solution computed by SSYTRS, with optional error bounds
SSYCON	estimate condition number, after factorization by SSYTRF
SSYTRU	low-rank update of a Bunch-Parlett factorization
SSYEQU	equilibrate matrix

Notes:

- Because of the need for diagonal pivoting in the Bunch-Parlett factorization it does not seem to be possible to develop a block algorithm for SSYTRF, however there is scope for using Level 2 BLAS.
- We are investigating the possibility of combining into the single routine SSYTRF the functions of the LINPACK routines SSIFA and SCHDC.
- Here also we are considering the possibility of working with either the upper or lower triangle.
- SSYTRU corresponds to the routine SPOTRU, but applied to a symmetric indefinite factorization.

- Prototype code for one possible variant of SPOTRF is presented in Appendix A.
- For SPOTRF, just as for SGETRF, more than one variant of the block algorithm can be derived. We will investigate the performance of different variants.
- SPOTRU corresponds to the LINPACK routines SCHUD and SCHDD, with the difference that it allows a rank-k modification with $k \geq 1$. In downdating, the modification may have to be performed as a sequence of rank-1 downdates to maintain stability. Note that these matrices can also be regarded as updating the triangular factor of a QR factorization.
- SPOTRX corresponds to the LINPACK routine SCHEX. It also can be regarded as updating the triangular factor of a QR factorization and allows such factorizations to be updated by the addition or deletion of a column.

3.1.4 Symmetric positive-definite matrices in packed storage

SPPTRF	Cholesky factorization
SPPTRS	solve linear equations, after factorization by SPPTRF
SPPTRI	compute inverse, after factorization by SPPTRF
SPPRFS	refine solution computed by SPPTRS, with optional error bounds
SPPCON	estimate condition number, after factorization by SPPTRF
SPPTRU	low-rank update or downdate of a Cholesky factorization
SPPTRX	permute columns of a Cholesky factorization
SPPEQU	equilibrate matrix

Notes (See also 3.1.3 where relevant):

- The routines in this group will call only Level 2 BLAS, not Level 3, since the Level 3 BLAS do not cater for packed storage.
- The routines will use the same packed storage scheme as the Level 2 BLAS: that is, if UPLO = 'U', the upper triangle is packed sequentially by column (this is the convention used in LINPACK, and is equivalent to packing the lower triangle by rows); if UPLO = 'L', the lower triangle is packed sequentially by column (which is equivalent to packing the upper triangle by rows).

3.1.2 General band matrices

SGBTRF	LU-factorization with row interchanges
SGBTRS	solve linear equations, after factorization by SGBTRF
SGBRFS	refine solution computed by SGBTRF, with optional error bounds
SGBCON	estimate condition number, after factorization by SGBTRF
SGBEQU	equilibrate matrix

Notes (see also 3.1.1 where relevant):

- The routines in this group will use the same storage scheme as the GB routines in LINPACK, that is, diagonals of the matrix are stored in rows of the array, and columns of the matrix are stored in columns of the array.

3.1.3 Symmetric positive-definite matrices

SPOTRF	Cholesky factorization
SPOTRS	solve linear equations, after factorization by SPOTRF
SPOTRI	compute inverse, after factorization by SPOTRF
SPORFS	refine solution computed by SPOTRS, with optional error bounds
SPOCON	estimate condition number, after factorization by SPOTRF
SPOTRU	low-rank update or downdate of a Cholesky factorization
SPOTRX	permute columns of a Cholesky factorization
SPOEQU	equilibrate matrix

Notes:

- We are considering the possibility of providing all the routines in this chapter with an option parameter UPLO. If UPLO = 'U', the upper triangle of the symmetric matrix must be suplied and the matrix will be factorized as U^TU, as in LINPACK; if UPLO = 'L', the lower triangle will be supplied, and the matrix factorized as LL^T, as in EISPACK (routines REDUC and REDUC2). Would this additional flexibility be useful? or would it be an unwelcome complication?

- We are also considering the possibility of providing "backward" Cholesky factorizations UU^T and L^TL, as well as the conventional "forward" factorizations. It is possible that a backward factorization is significantly faster on some machines, and we plan to investigate this. If there is a significant advantage in performance, should we provide either of the backward factorizations as an optional alternative to the usual ones, or even instead of them? (The LINPACK routines SSIFA already uses a backward factorization for symmetric indefinite matrices.)

- Block algorithms for SPOTRF, SPOTRS and SPOTRI are straightforward to derive.

3.1.1 General matrices

SGETRF	LU-factorization with row interchanges
SGETRS	solve linear equations, after factorization by SGETRF
SGETRI	compute inverse, after factorization by SGETRF
SGERFS	refine solution computed by SGETRS, with optional error bounds
SGECON	estimate condition number, after factorization by SGETRF
SGEEQU	equilibrate matrix

Notes:

- Block algorithms for SGETRF, SGETRS and SGETRI are straightforward to derive.
- For SGETRF more than one variant of the block algorithm can be derived, by extending to block algorithms the analysis of Dongarra, Gustavson and Karp [6]. The performance of the different variants will be investigated.
- Prototype code for one possible variant of SGETRF is presented in Appendix A.
- SGETRF will factorize a rectangular matrix (so that the factor L may be trapezoidal). This additional flexibility is occasionally useful; also a blocked algorithm for SGETRF requires an unblocked version of the algorithm to factorize a rectangular submatrix. The other routines in this group work only with square matrices.
- SGETRS will solve $AX = B$ or $A^T X = B$
- For SGETRI two methods are possible: either to compute L^{-1} and U^{-1} by calls to STRTRI and then to form their product $U^{-1}L^{-1}$; or to compute U^{-1} by a call to STRTRI and then to solve for X the equation $XL = U^{-1}$. The latter method is used in the LINPACK routine SGEDI and is likely to be faster, but requires workspace for one block of columns.
- For SGEEQU we envisage options for row scaling, column scaling or row-and-column scaling. For the last option we are investigating the scaling algorithm of Curtis and Reid [7] and also cheaper alternatives.

3 Organization of Routines and Choice of Algorithms

In this section we describe the functions of the routines, and the interrelationships between them; and give notes on the choice of algorithms. For simplicity we describe the routines for **real, single precision matrices only** (routine names beginning with S).

For convenience we divide the routines into two sections

- routines associated with the solution of systems of linear equations including also some routines for solving linear least problems. These are centered on one of the standard non-iterative factorizations (LU, Cholesky, QR), and correspond roughly to LINPACK.
- routines associated with the solution of eigenvalue problems (including generalized problems and singular value problems). These are centered on an iterative method for computing eigenvalues, and correspond roughly to EISPACK.

The division is not clear cut: some routines in the first section also have a role in solving eigenvalue problems; and linear least squares problems may be solved either by routines in the first section (using QR factorization) or by routines in the second (using SVD).

3.1 Routines for Solving Linear Equations

This section of LAPACK is concerned with the solution of systems of linear equations $AX = B$. Similar groups of routines will be provided for different types of the matrix A, and these are described in the following subsections. The overall structure is similar to that of LINPACK.

This section also contains routines based on the QR factorization for solving linear least squares problems.

The following remarks apply to all groups of routines in this chapter:

- The routines for solving linear equations (TRS routines) will all handle multiple right hand sides (with the possible exception of routines for tridiagonal systems).
- For further discussion of the routines for iterative refinement and condition estimation (RFS and CON routines), see Chapter 3 of Working Note #4.
- The routines for condition estimation will all use Higham's version of Hager's method [5]. The CON routines mentioned here will all call Higham's algorithm SONEST to estimate the norm of A^{-1}.

	GE	GB	GG	HS	HG	TR	TG	SY	SP	SB	ST	BD	DB
HRD	x		x										
TRD								x	x	x			
BRD	x	x				x							
EQR				x	x						x		
EDC											x		
EIN				x	x						x		
EVC						x	x						
EBM										x	x		x
EVU											x		
SQR												x	
SDC												x	
SIN												x	
SBM												x	
SVU												x	
SJA							x						
SEN						x	x						
SNA						x	x						
SYL						x	x						
EXC						x	x						
BAL	x		x										
BAK	x		x										
GST								x	x	x			

The following tables indicate which combinations of the codes XX and YYY are envisaged. The first table covers routines which are primarily associated with the solution of systems of linear equations and are listed in Section 3.1. The second table covers routines which are primarily associated with eigenvalue problems and are listed in Section 3.2. (We do not list the complex counterparts of SY and SP.)

	GE	GB	GT	PO	PP	PB	PT	MT	SY	SP	TR	TP	TB	TZ	OR	GG
TRF	x	x	x	x	x	x	x		x	x						
TRS	x	x	x	x	x	x	x		x	x	x	x	x			
RFS	x	x	x	x	x	x	x		x	x	x	x	x			
TRI	x			x	x				x	x	x	x				
CON	x	x	x	x	x	x	x		x	x	x	x	x			
SOL			x				x	x								
TRU				x	x				x	x						
TRX				x	x											
EQU	x	x	x	x	x	x	x		x	x						
QRP	x															x
QRF	x															
RQF	x													x		
QRS	x															
QRU	x															
GEN															x	
MUL															x	

QRF	QR-factorization without pivoting
QRP	QR-factorization with pivoting
QRS	solution of linear least squares problems, following QR factorization
QRU	update QR-factorization
RFS	refine initial approximate solution returned by TRS routines, with optional error bound
RQF	RQ-factorization
SBM	compute selected singular values, by bisection/multisection
SDC	all singular values and vectors, using a divide-and-conquer algorithm
SEN	condition number (sensitivity) of an invariant subspace
SIN	selected singular vectors (assuming singular values are known), by inverse iteration
SJA	computes singular values and optionally singular values using Jacobi (needed by GSVD)
SNA	condition numbers (sensitivities) of all eigenvalue-eigenvector pairs
SOL	solution of linear equations
SQR	compute singular values and, optionally, singular vectors, using QR algorithm
SVU	rank-1 update of singular value decomposition
SYL	solve Sylvester's equation
TRD	reduction to symmetric tridiagonal form
TRF	triangular factorization (LU, Cholesky, etc)
TRI	compute inverse (based on triangular factorization)
TRS	solution of linear equations (based on triangular factorization)
TRU	update or downdate triangular factorization
TRX	exchange rows and columns in triangular factorization

BAK	back-transformation of eigenvectors after balancing
BAL	balance a matrix or matrices (for eigenvalue computation)
BRD	reduction to bidiagonal form
CON	estimate condition number
EBM	selected eigenvalues, by bisection/multisection
EDC	all eigenvalues and eigenvectors, using a divide and conquer algorithm
EIN	selected eigenvectors (assuming eigenvalues are known), by inverse iteration
EQR	all eigenvalues and, optionally, Schur factorization or eigenvectors, using QR algorithm
EQU	equilibrate matrix (for solving linear equations)
EVC	eigenvectors from Schur factorization
EVU	rank-1 update of eigenvalue decomposition
EXC	exchange eigenvalues (in Schur factorization)
GEN	generate a real orthogonal or complex unitary matrix (as a product of Householder matrices)
GST	reduce symmetric-definite generalized eigenvalue problem to standard form
HRD	reduction to upper Hessenberg form
MUL	multiply a matrix by real orthogonal or complex unitary matrix (by applying a product of Householder matrices)

PT symmetric or Hermitian positive definite tridiagonal

SB (real) symmetric band

SP (real) symmetric, packed storage

ST (real) symmetric tridiagonal

SY (real) symmetric

TB triangular band

TG triangular matrices, generalized problem

TP triangular, packed storage

TR triangular (or in some cases quasi-triangular)

TZ trapezoidal

UN (complex) unitary

The precise meaning of some of these codes may become clearer in the light of the list of proposed routines in Section 3.

The final letters YYY indicate the computation done by a particular subroutine. Again, Section 3 may make the meaning of some of the codes more clear.

Note that the last is not standard Fortran but is available in many Fortran systems on machines where double precision computation is usual.

The next two letters, XX, indicate the type of matrix (in some cases of matrices, or of most significant matrix). Most of these two-letter codes apply to both real and complex routines; a few apply specifically to one or the other, and this is indicated below.

BD bidiagonal

DB generalized banded symmetric or Hermitian positive definite

GB general band

GE general (i.e. unsymmetric, in some cases rectangular)

GG general matrices, generalized problem

GT general tridiagonal

HB (complex) Hermitian band

HE (complex) Hermitian

HG Hessenberg matrix, generalized problem

HP (complex) Hermitian, packed storage

HS Hessenberg

MT as PT but for multiple systems of equations

OR (real) orthogonal

PB symmetric or Hermitian positive definite band

PO symmetric or Hermitian positive definite

PP symmetric or Hermitian positive definite, packed storage

of algorithms. Section 4 discusses aspects of software design. Specimen versions of a few routines are presented in Appendix A. Appendix B shows how the functions of LINPACK and EISPACK routines would be covered (with a few exceptions) by LAPACK routines.

The contents of this working note are **provisional** and are likely to be modified to some extent in the light of comment and experience. We are publishing our plans at this stage in order to give people an early opportunity to offer suggestions, criticisms or even prototype software. Some questions on which we would particularly welcome feedback are listed at the end of Sections 3 and 4.

2 Naming Scheme

A subroutine naming scheme has been designed, similar in style to that used for LINPACK [2] and later for the Level 2 [3] and Level 3 BLAS [4]. The following principles influenced the design:

- the names should be as mnemonic and systematic as possible within the very tight constraints of standard Fortran 77 6-character names.
- the names should indicate the function of the routines rather than the algorithm used (except in a few cases where we plan to provide more than one algorithm for the same task).
- there should be no clashes with names already used in EISPACK, LINPACK or the BLAS.

We have tried to make the computational routines as modular as possible, more so than in either LINPACK or EISPACK. The reasons for this are:

- when the areas covered by LINPACK and EISPACK are combined, there is considerably greater scope for sharing common features.
- the routines in LAPACK, based on block algorithms, are likely to involve more complex code than LINPACK or EISPACK, and hence there are stronger reasons not to duplicate it.

Each subroutine name is a coded specification of the computation done by the subroutine. All names consist of six letters in the form TXXYYY. The first letter, T, indicates the matrix data type as follows:

S REAL
D DOUBLE PRECISION
C COMPLEX
Z COMPLEX*16 or DOUBLE COMPLEX (if available)

LAPACK Working Note #5 Provisional Contents

Chris Bischof, James Demmel, Jack Dongarra, Jeremy Du Croz,
Anne Greenbaum, Sven Hammarling, and Danny Sorensen

ABSTRACT

This note outlines the proposed computational routines in LAPACK. It describes a naming scheme for the routines, enumerates the individual routines, includes notes on the choice of algorithms and discusses aspects of software design. The contents of this note are provisional and may be modified in the light of comment and experience.

1 Overview

LAPACK is planned to be a collection of Fortran 77 subroutines for the analysis and solution of various systems of simultaneous linear algebraic equations, linear least-squares problems, and matrix eigenvalue problems.

The subroutines are intended to be transportable and efficient across a wide range of computing environments, with special emphasis on modern high-performance computers. For more about the background, motivation and design goals of LAPACK, see the Prospectus [1].

Our plan is that LAPACK should include two broad categories of routines (though the distinction may in some cases be blurred):

computational routines, each performing a distinct algorithmic task, such as performing an LU factorization, reducing a matrix to Hessenberg form, or computing the SVD of a bidiagonal matrix.

driver routines, each of which solves a complete problem, using a series of calls to computational routines and possibly some additional code, for example: solving a system of linear equations with one or many right hand sides, or computing all eigenvalues and optionally eigenvectors of a symmetric matrix.

Driver routines are provided in EISPACK (RG, RGG, RS and so on); the absence of such routines in LINPACK has often been criticized.

This working note dicusses the computational routines only. At this stage of the project we feel that it is more important to design and develop the computational routines.

Section 2 of this working note describes the naming scheme for the routines. Section 3 lists the proposed routines and contains notes on the structure of the routines and choice

ABSTRACT

This note outlines the provisional contents of LAPACK. It describes a naming scheme for the routines, enumerates the individual routines, and includes notes on the choice of algorithms and aspects of software design.

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

LAPACK Working Note # 5
Provisional Contents

**Chris Bischof, James Demmel, Jack Dongarra, Jeremy Du Croz,
Anne Greenbaum, Sven Hammarling, and Danny Sorensen**

Mathematics and Computer Science Division

ANL-88-38

March 16, 1992

Work supported in part by the National Science Foundation under grant No. NSF ASC-8715728, the Applied Mathematical Sciences subprogram of the Office of Energy Research of the U.S. Department of Energy under Contract W-31-109-Eng-38, and the Bergen Scientific Center IBM.