LAPACK summary

Prefixes

Each routine has a prefix, denoted by a hypen - in this guide, made of 3 letters *xyy*, where *x* is the data type, and *yy* is the matrix type.

Fortran data type	C data type	Abbreviation
real	float	s
double precision	double	d
complex	float complex	c
complex*16	double complex	z

Matrix type	full	packed	RFP	banded	tridiag	generalized problem
general	ge			gb	gt	gg
symmetric	sy	sp	sf	sb	st	
Hermitian	he	hp	hf	hb		
SPD / HPD	po	pp	pf	pb	pt	
triangular	tr	tp	tf	tb		tg
upper Hessenberg	hs					hg
trapezoidal	tz					
orthogonal	or	op				
unitary	un	up				
diagonal					di	
bidiagonal					bd	

Storage

Full matrices are stored column-wise (so-called "Fortran" order). For symmetric, Hermition, and triangular matrices, elements below/above the diagonal (for upper/lower respectively) are not accessed. Similarly for upper Hessenberg, elements below the subdiagonal are not accessed.

Packed storage for triangular or symmetric matrices is by columns. For example, a 3×3 upper triangular matrix is stored as

$$\boldsymbol{U} = \begin{bmatrix} 1 & 2 & 4 \\ \cdot & 3 & 5 \\ \cdot & \cdot & 6 \end{bmatrix} \Longrightarrow [\ \underbrace{1}_{\text{column 1}}\ \ \underbrace{2\ 3}_{\text{column 2}}\ \ \underbrace{4\ 5\ 6}_{\text{column 3}}\ ]$$

and a 3×3 lower triangular matrix is stored as

$$\boldsymbol{L} = \begin{bmatrix} 1 & \cdot & \cdot \\ 2 & 4 & \cdot \\ 3 & 5 & 6 \end{bmatrix} \Longrightarrow [\ \underbrace{1\ 2\ 3}_{\text{column 1}}\ \ \underbrace{4\ 5}_{\text{column 2}}\ \ \underbrace{6}_{\text{column 3}}\ ]$$

Rectangular full packed (RFP) for triangular or symmetric matrices reuses fast routines for full matrices, but with half the storage and avoiding the inefficient indexing of packed storage. It divides the matrix into 4 quadrants, transposes one of the triangles and packs it next to the other triangle; see LAPACK Working Note 199. For example,

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{11} & \cdot \\ \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{bmatrix} \Longrightarrow \begin{bmatrix} \boldsymbol{L}_{11} \text{ and } \boldsymbol{L}_{22}^T \\ \boldsymbol{L}_{21} \end{bmatrix}$$

$$\boldsymbol{L} = \left[\begin{array}{ccc|ccc} a_{11} & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{12} & a_{22} & \cdot & \cdot & \cdot & \cdot \\ a_{13} & a_{23} & a_{33} & \cdot & \cdot & \cdot \\ \hline a_{14} & a_{24} & a_{34} & \mathbf{a_{44}} & \cdot & \cdot \\ a_{15} & a_{25} & a_{35} & \mathbf{a_{45}} & \mathbf{a_{55}} & \cdot \\ a_{16} & a_{26} & a_{36} & \mathbf{a_{46}} & \mathbf{a_{56}} & \mathbf{a_{66}} \end{array}\right] \Longrightarrow \left[\begin{array}{ccc} \mathbf{a_{44}} & \mathbf{a_{45}} & \mathbf{a_{46}} \\ a_{11} & \mathbf{a_{55}} & \mathbf{a_{56}} \\ a_{12} & a_{22} & \mathbf{a_{66}} \\ a_{13} & a_{23} & a_{33} \\ \hline a_{14} & a_{24} & a_{34} \\ a_{15} & a_{25} & a_{35} \\ a_{16} & a_{26} & a_{36} \end{array}\right]$$

There are 4 versions depending on whether n is even or odd and whether the upper or lower triangle is stored. Conversion routines are provided.

Banded storage puts columns of the matrix in corresponding columns of the array, and diagonals in rows of the array, for example:

$$\begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} & \cdot & \cdot \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdot \\ \cdot & a_{42} & a_{43} & a_{44} & a_{45} \\ \cdot & \cdot & a_{53} & a_{54} & a_{55} \end{bmatrix} \Longrightarrow \begin{array}{l} \begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{bmatrix} \end{array} \begin{array}{l} \text{1st diagonal} \\ \text{2nd (main) diagonal} \\ \text{3rd diagonal} \\ \text{4th diagonal} \end{array}$$

Bi- and tri-diagonal matrices are stored as 2 or 3 vectors of length n and $n-1$.

BLAS and LAPACK guides are available from http://web.eecs.utk.edu/~mgates3/
Copyright ©2007–2011 by Mark Gates. Updated September 6, 2011.
You may freely copy and modify this document under the Creative Commons Attribution license. When distributing, please include the above link and copyright for this original document.

Drivers

Drivers are higher level routines that solve an entire problem.

Linear system, solve $Ax = b$.
`-sv` — solve
`-svx` — expert; also $A^T x = b$ or $A^H x = b$, condition number, error bounds, scaling
Matrix types [General ge, gb, gt; SPD po, pp, pb, pt; Symmetric sy, sp, he, hp]

Linear least squares, minimize $\|b - Ax\|_2$.
`-ls` — full rank, $\text{rank}(A) = \min(m, n)$, uses QR.
`-lsy` — rank deficient, uses complete orthogonal factorization.
`-lsd` — rank deficient, uses SVD.
Matrix types [General ge]

Generalized linear least squares.
Minimize $\|c - Ax\|_2$ subject to $Bx = d$.
`-lse` — B full row rank, matrix $\begin{bmatrix} A \\ B \end{bmatrix}$ full col rank.

Minimize $\|y\|_2$ subject to $d = Ax + By$.
`-glm` — A full col rank, matrix $\begin{bmatrix} A & B \end{bmatrix}$ full row rank.
Matrix types [General gg]

SVD singular value decomposition, $A = U\Sigma V^H$
`-svd` — singular values, [left, right vectors]
`-sdd` — divide-and-conquer; faster but more memory
Matrix types [General ge]

Generalized SVD, $A = U\Sigma_1 Q^T$ **and** $B = V\Sigma_2 Q^T$
`-svd` — singular values, [left, right vectors]
Matrix types [General gg]

Eigenvalues, solve $Ax = \lambda x$.
Symmetric
`-ev` — all eigenvalues, [eigenvectors]
`-evx` — expert; also subset
`-evd` — divide-and-conquer; faster but more memory
`-evr` — relative robust; fastest and least memory
Matrix types [Symmetric sy, sp, sb, st, he, hp, hb]

Nonsymmetric
`-ev` — eigenvalues, [left, right eigenvectors]
`-evx` — expert; also balance matrix, condition numbers
`-es` — Schur factorization
`-esx` — expert; also condition numbers
Matrix types [General ge]

Generalized eigenvalue, solve $Ax = \lambda Bx$
Symmetric, B SPD
`-gv` — all eigenvalues, [eigenvectors]
`-gvx` — expert; also subset
`-gvd` — divide-and-conquer, faster but more memory
Matrix types [Symmetric sy, sp, sb, he, hp, hb]

Nonsymmetric
`-ev` — eigenvalues, [left, right eigenvectors]
`-evx` — expert; also balance matrix, condition numbers
`-es` — Schur factorization
`-esx` — expert; also condition numbers
Matrix types [General gg]

Conversion routines

`-tfttp` —	RFP	to	Packed
`-tfttr` —	RFP	to	Triangular full
`-trttp` —	Triangular full	to	Packed
`-trttf` —	Triangular full	to	RFP
`-tpttr` —	Packed	to	Triangular full
`-tpttf` —	Packed	to	RFP

Computational routines

Computational routines perform one step of solving the problem. Drivers call a sequence of computational routines.

Triangular factorization

`-trf` — factorize: General LU, Cholesky LL^T, tridiag LDL^T, sym. indefinite LDL^T
`-trs` — solve using factorization
`-con` — condition number estimate
`-rfs` — error bounds, iterative refinement
`-tri` — inverse (not for band)
`-equ` — equilibrate A (not for tridiag, symmetric indefinite, triangular)
Matrix types [General ge, gb, gt; SPD po, pp, pf, pb, pt; Symmetric sy, sp, he, hp,
Triangular tr, tp, tf, tb]

Orthogonal factorization

`-qp3` — QR factorization, with pivoting
`-qrf` — QR factorization
`-rqf` — RQ factorization
`-qlf` — QL factorization
`-lqf` — LQ factorization
`-tzrqf` — RQ factorization
`-tzrzf` — RZ trapezoidal factorization
Matrix types [General ge, some Trapezoidal tz]

`-gqr` — generate Q after `-qrf`
`-grq` — generate Q after `-rqf`
`-gql` — generate Q after `-qlf`
`-glq` — generate Q after `-lqf`
`-mqr` — multiply by Q after `-qrf`
`-mrq` — multiply by Q after `-rqf`
`-mql` — multiply by Q after `-qlf`
`-mlq` — multiply by Q after `-lqf`
`-mrz` — multiply by Q after `-tzrzf`
Matrix types [Orthogonal or, un]

Generalized orthogonal factorization

`-qrf` — QR of A, then RQ of $Q^T B$
`-rqf` — RQ of A, then QR of BQ^T
Matrix types [General gg]

Eigenvalue

`-trd` — tridiagonal reduction
Matrix types [Symmetric sy, he, sp, hp]

`-gtr` — generate matrix after `-trd`
`-mtr` — multiply matrix after `-trd`
Matrix types [Orthogonal or, op, un, up]

Symmetric tridiagonal eigensolvers
`-eqr` — using implicitly shifted QR
`-pteqr` — using Cholesky and bidiagonal QR
`-erf` — using square-root free QR
`-edc` — using divide-and-conquer
`-egr` — using relatively robust representation
`-ebz` — eigenvalues using bisection
`-ein` — eigenvectors using inverse iteration
Matrix types [Symmetric tridiag st, one SPD pt]

Nonsymmetric
`-hrd` — Hessenberg reduction
`-bal` — balance
`-bak` — back transforming
Matrix types [General ge]

`-ghr` — generate matrix after `-hrd`
`-mhr` — multiply matrix after `-hrd`
Matrix types [Orthogonal or, un]

`-eqr` — Schur factorization
`-ein` — eigenvectors using inverse iteration
Matrix types [upper Hessenberg hs]

`-evc` — eigenvectors
`-exc` — reorder Schur factorization
`-syl` — Sylvester equation
`-sna` — condition numbers
`-sen` — condition numbers of eigenvalue cluster/subspace
Matrix types [Triangular tr]