Dense Matrix Solvers - LAPACK routines
Introduction to Parallel Sparse Direct solvers

Introduction

BLAS Level 3 routines provide convenient optimized operations that involve operations between multiple matrices, e.g.
- Multiply two matrices (GEMM)
- Add a (special) product of two matrices to a third matrix (rank-k update)
- Solve a triangular system $\mathbf{LX} = \mathbf{B}$ *where* $\mathbf{X}$ *&* $\mathbf{B}$ *are matrices (not vectors)*

So far, we haven't seen routines to solve linear systems of equations (with the exception of the triangular system solve; but that's a special case)

Typical mode of use for routines we will examine:
- We may want to solve systems of the form $\mathbf{Ax} = \mathbf{b}$ *where the matrix* $\mathbf{A}$ *is not necessarily an "easy" one to handle (i.e. a triangular matrix)*
- For many applications we may need to solve several systems like the one above with the same matrix $\mathbf{A}$ *but for different right-hand-sides* $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \ldots$ *(and correspondingly producing multiple solutions* $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$*)*
- In many cases we can afford a more expensive "one-time" pre-computation for the sake of accelerating the solution of subsequent problems.

BLAS Level 3 Routines

BLAS Level 3 routines perform matrix-matrix operations. The following table lists the BLAS Level 3 routine groups and the data types associated with them.

BLAS Level 3 Routine Groups and Their Data Types		
Routine Group	**Data Types**	**Description**
`cblas_?gemm`	s, d, c, z	Computes a matrix-matrix product with general matrices.
`cblas_?hemm`	c, z	Computes a matrix-matrix product where one input matrix is Hermitian.
`cblas_?herk`	c, z	Performs a Hermitian rank-k update.
`cblas_?her2k`	c, z	Performs a Hermitian rank-2k update.
`cblas_?symm`	s, d, c, z	Computes a matrix-matrix product where one input matrix is symmetric.
`cblas_?syrk`	s, d, c, z	Performs a symmetric rank-k update.
`cblas_?syr2k`	s, d, c, z	Performs a symmetric rank-2k update.
`cblas_?trmm`	s, d, c, z	Computes a matrix-matrix product where one input matrix is triangular.
`cblas_?trsm`	s, d, c, z	Solves a triangular matrix equation.

Introduction

*Solvers for linear systems (**$Ax = b$**) come in two (main) flavors:*
- <u>Iterative</u> solvers (e.g. Conjugate Gradients) that <u>converge</u> to the solution after a number of iterations (hopefully not too many …)
- <u>Direct</u> solvers produce the solution without iteration, by following a set algorithm that does not involve progressive "improvement" of a guess (e.g. Gauss Elimination, or Forward/Backward substitution when applicable)

Pros/Cons of <u>iterative</u> methods :
+ Relatively easy to set-up, as a general rule they don't require much pre-computation to be used
+ Some can be used without building the matrix explicitly (as in our earlier use of Conjugate Gradients)
- They may require many, many iterations to converge ("pre-conditioners" help, but are difficult to design)
- Some matrices can be particularly bad for them (it can be common that you need to use double-precision computation to barely get single-precision accurate results)

Introduction

*Solvers for linear systems (**Ax = b**) come in two (main) flavors:*
- Iterative solvers (e.g. Conjugate Gradients) that converge to the solution after a number of iterations (hopefully not too many …)
- Direct solvers produce the solution without iteration, by following a set algorithm that does not involve progressive "improvement" of a guess (e.g. Gauss Elimination, or Forward/Backward substitution when applicable)

Pros/Cons of direct methods :
+ No need to worry about how many iterations it will take (they "just" work …)
+ In many cases they are capable of computing solutions to higher accuracy, even for some "bad/problematic" matrices
- They require significant amounts of computation (up to $O(N^3)$) for systems with N equations and N unknowns)
- Pre-computation often required, which is only amortized if solving for many right-hand sides
??? Parallel Potential: Relatively easy to leverage for dense systems, much more challenging for sparse systems.

Solving a dense or a sparse matrix?

*Systems (**Ax = b**) where the matrix **A** is <u>dense</u> offer the most direct opportunity for accelerated parallel computation*
- Support in MKL provided through "LAPACK routines" (we will see next)
- Generally, matrices are given in row-major/column-major format
(with some modestly-space-saving variants we will discuss …)
- Parallel optimizations follow the pattern we have seen in GEMM-style operations (blocking, targeted vectorization, cache optimization)

*Systems (**Ax = b**) where the matrix **A** is <u>sparse</u> offer the most highest impact, since they can scale to many millions of equations relatively easily (as opposed to dense methods that are rarely used beyond ~50,000 equations/unknowns)*
- Support in MKL provided through the PARDISO library
(will visit briefly today, in more detail next lecture)
- Generally, matrices are given in CSR/CSC compressed format
- Parallel optimizations use highly advanced ideas and concepts,
with great degree of sophistication both in theory and parallel programming
(we will attempt to appreciate at least the "spirit" of such optimizations)

Developer Reference for Intel® Math Kernel Library - C

Submitted March 30, 2020

LAPACK Linear Equation Computational Routines

Table "Computational Routines for Systems of Equations with Real Matrices" lists the LAPACK computational routines for factorizing, equilibrating, and inverting *real* matrices, estimating their condition numbers, solving systems of equations with real matrices, refining the solution, and estimating its error. Table "Computational Routines for Systems of Equations with Complex Matrices" lists similar routines for *complex* matrices.

Computational Routines for Systems of Equations with Real Matrices

Matrix type, storage scheme	Factorize matrix	Equilibrate matrix	Solve system	Condition number	Estimate error	Invert matrix
general	?getrf	?geequ, ?geequb	?getrs	?gecon	?gerfs, ?gerfsx	?getri
general band	?gbtrf	?gbequ, ?gbequb	?gbtrs	?gbcon	?gbrfs, ?gbrfsx	
general tridiagonal	?gttrf		?gttrs	?gtcon	?gtrfs	

LAPACK Linear Equation Computational Routines

Table "Computational Routines for Systems of Equations with Real Matrices" lists the LAPACK computational routines for factorizing, equilibrating, and inverting *real* matrices, estimating their condition numbers, solving systems of equations with real matrices, refining the solution, and estimating its error. Table "Computational Routines for Systems of Equations with Complex Matrices" lists similar routines for *complex* matrices.

Computational Routines for Systems of Equations with Real Matrices

Matrix type, storage scheme	Factorize matrix	Equilibrate matrix	Solve system	Condition number	Estimate error	Invert matrix
general	?getrf	?geequ, ?geequb	?getrs	?gecon	?gerfs, ?gerfsx	?getri
general band	?gbtrf	?gbequ, ?gbequb	?gbtrs	?gbcon	?gbrfs, ?gbrfsx	
general tridiagonal	?gttrf		?gttrs	?gtcon	?gtrfs	
diagonally dominant tridiagonal	?dttrfb		?dttrsb			
symmetric positive-definite	?potrf	?poequ, ?poequb	?potrs	?pocon	?porfs, ?porfsx	?potri
symmetric positive-definite, packed storage	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
symmetric positive-definite, RFP storage	?pftrf		?pftrs			?pftri
symmetric positive-definite, band	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	

symmetric positive-definite, band	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	
symmetric positive-definite, tridiagonal	?pttrf		?pttrs	?ptcon	?ptrfs	
symmetric indefinite	?sytrf ?sytrf_rk ?sytrf_aa	?syequb	?sytrs ?sytrs2 ?sytrs3 ?sytrs_aa	?sycon ?sycon_3	?syrfs, ?syrfsx	?sytri ?sytri2 ?sytri2x ?sytri_3
symmetric indefinite, packed storage	?sptrf mkl_?spffrt2, mkl_?spffrtx		?sptrs	?spcon	?sprfs	?sptri
triangular			?trtrs	?trcon	?trrfs	?trtri
triangular, packed storage			?tptrs	?tpcon	?tprfs	?tptri
triangular, RFP storage						?tftri
triangular band			?tbtrs	?tbcon	?tbrfs	

LAPACK Linear Equation Computational Routines

Table "Computational Routines for Systems of Equations with Real Matrices" lists the LAPACK computational routines for factorizing, equilibrating, and inverting *real* matrices, estimating their condition numbers, solving systems of equations with real matrices, refining the solution, and estimating its error. Table "Computational Routines for Systems of Equations with Complex Matrices" lists similar routines for *complex* matrices.

Computational Routines for Systems of Equations with Real Matrices

Matrix type, storage scheme	Factorize matrix	Equilibrate matrix	Solve system	Condition number	Estimate error	Invert matrix
general	?getrf	?geequ, ?geequb	?getrs	?gecon	?gerfs, ?gerfsx	?getri
general band	?gbtrf	?gbequ, ?gbequb	?gbtrs	?gbcon	?gbrfs, ?gbrfsx	
general tridiagonal	?gttrf		?gttrs	?gtcon	?gtrfs	
diagonally dominant tridiagonal	?dttrfb		?dttrsb			
symmetric positive-definite	?potrf	?poequ, ?poequb	?potrs	?pocon	?porfs, ?porfsx	?potri
symmetric positive-definite, packed storage	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
symmetric positive-definite, RFP storage	?pftrf		?pftrs			?pftri
symmetric positive-definite, band	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	

General matrices

$$\mathbf{A} = \mathbf{PLU}$$

Factorization Stage: Compute LU Factorization (or LU "decomposition")

General matrices

Factorization Stage: Compute LU Factorization (or LU "decomposition")

$$\mathbf{A} = \mathbf{PLU}$$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ l_{31} & l_{32} & 1 & \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \quad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & u_{22} & u_{23} & u_{24} \\ & & u_{33} & u_{34} \\ & & & u_{44} \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix}$$

General matrices

Factorization Stage: Compute LU Factorization (or LU “decomposition”)

$$\mathbf{A} = \mathbf{P}\mathbf{L}\mathbf{U}$$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ l_{31} & l_{32} & 1 & \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \quad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & u_{22} & u_{23} & u_{24} \\ & & u_{33} & u_{34} \\ & & & u_{44} \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix}$$

Lower-triangular factor (“unitary” along the diagonal)

Upper-triangular factor

Permutation matrix

General matrices

Factorization Stage: Compute LU Factorization (or LU "decomposition")

$$\mathbf{A} = \mathbf{PLU}$$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$\mathbf{A}_{\text{after}} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & u_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & u_{44} \end{pmatrix}$$

Both factors returned in-place overwriting the input matrix

$$\mathbf{P} = \begin{pmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix}$$

Returned as a vector (encoding the permutation)

?getrf

Computes the LU factorization of a general m-by-n matrix.

Syntax

```
lapack_int LAPACKE_sgetrf (int matrix_layout , lapack_int m , lapack_int n , float * a ,
lapack_int lda , lapack_int * ipiv );
```

```
lapack_int LAPACKE_dgetrf (int matrix_layout , lapack_int m , lapack_int n , double * a ,
lapack_int lda , lapack_int * ipiv );
```

Description

The routine computes the LU factorization of a general m-by-n matrix A as

$$A = P*L*U,$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$) and U is upper triangular (upper trapezoidal if $m < n$). The routine uses partial pivoting, with row interchanges.

NOTE
This routine supports the Progress Routine feature. See Progress Function for details.

Input Parameters

matrix_layout	Specifies whether matrix storage layout is row major (`LAPACK_ROW_MAJOR`) or column major (`LAPACK_COL_MAJOR`).
m	The number of rows in the matrix A ($m \geq 0$).
n	The number of columns in A; $n \geq 0$.
a	Array, size at least max(1, *lda***n*) for column-major layout or max(1, *lda***m*) for row-major layout. Contains the matrix A.
lda	The leading dimension of array *a*, which must be at least max(1, *m*) for column-major layout or max(1, *n*) for row-major layout.

Output Parameters

a	Overwritten by L and U. The unit diagonal elements of L are not stored.
ipiv	Array, size at least $\max(1, \min(m, n))$. Contains the pivot indices; for $1 \leq i \leq \min(m, n)$, row i was interchanged with row $ipiv(i)$.

General matrices

Computational complexity : $O(N^3)$ arithmetic operations, $O(N^2)$ memory
- Uses very similar opportunities for parallelization as GEMM (e.g. blocking)
- Starts being compute-bound for matrix sizes exceeding approximately 1000

General matrices

Computational complexity : O(N^3) arithmetic operations, O(N^2) memory
- Uses very similar opportunities for parallelization as GEMM (e.g. blocking)
- Starts being compute-bound for matrix sizes exceeding approximately 1000

docs.nvidia.com/cuda/cusolver/index.html

DEVELOPER ZONE CUDA TOOLKIT DOCUMENTATION

Search

The GPU path of the cuSolver library assumes data is already in the device memory. It is the responsibility of the developer to allocate memory and to copy data between GPU memory and CPU memory using standard CUDA runtime API routines, such as `cudaMalloc()`, `cudaFree()`, `cudaMemcpy()`, and `cudaMemcpyAsync()`.

cuSolverMg is GPU-accelerated ScaLAPACK. By now, cuSolverMg supports 1-D column block cyclic layout and provides symmetric eigenvalue solver.

Note: The cuSolver library requires hardware with a CUDA compute capability (CC) of at least 2.0 or higher. Please see the *CUDA C++ Programming Guide* for a list of the compute capabilities corresponding to all NVIDIA GPUs.

cuSolverDN: Dense LAPACK

The cuSolverDN library was designed to solve dense linear systems of the form

$$A x = b$$

where the coefficient matrix $A \in R^{nxn}$, right-hand-side vector $b \in R^n$ and solution vector $x \in R^n$

The cuSolverDN library provides QR factorization and LU with partial pivoting to handle a general matrix `A`, which may be non-symmetric. Cholesky factorization is also provided for symmetric/Hermitian matrices. For symmetric indefinite matrices, we provide Bunch-Kaufman (LDL) factorization.

The cuSolverDN library also provides a helpful bidiagonalization routine and singular value decomposition (SVD).

The cuSolverDN library targets computationally-intensive and popular routines in LAPACK, and provides an API compatible with LAPACK. The user can accelerate these time-consuming routines with cuSolverDN and keep others in LAPACK without a major change to existing code.

cuSolverSP: Sparse LAPACK

The cuSolverSP library was mainly designed to a solve sparse linear system

$$A x = b$$

General matrices

Solve Stage: Use LU decomposition in triangular substitution steps

$$\mathbf{A} = \mathbf{PLU}$$

$$\mathbf{Ax} = \mathbf{PLUx} = \mathbf{b}$$

General matrices

Solve Stage: Use LU decomposition in triangular substitution steps

$$\mathbf{A} = \mathbf{PLU}$$

$$\mathbf{Ax} = \mathbf{P}\underline{\mathbf{LUx}} = \mathbf{b}$$

$$\mathbf{Pw} = \mathbf{b}$$

*Solve for **w** by permuting elements of **b***

General matrices

Solve Stage: Use LU decomposition in triangular substitution steps

$$\mathbf{A} = \mathbf{PLU} \qquad \mathbf{Ax} = \mathbf{PL}\underline{\mathbf{Ux}} = \mathbf{b}$$

$$\mathbf{Pw} = \mathbf{b}$$

$$\mathbf{Lz} = \mathbf{w}$$

Solve for ***z*** *using forward substitution*

General matrices

Solve Stage: Use LU decomposition in triangular substitution steps

$$\mathbf{A} = \mathbf{PLU}$$

$$\mathbf{Ax} = \mathbf{PLUx} = \mathbf{b}$$

$$\mathbf{Pw} = \mathbf{b}$$

$$\mathbf{Lz} = \mathbf{w}$$

$$\mathbf{Ux} = \mathbf{z}$$

Solve for ***x*** *using backward substitution*

?getrs

Solves a system of linear equations with an LU-factored square coefficient matrix, with multiple right-hand sides.

Syntax

```
lapack_int LAPACKE_sgetrs (int matrix_layout , char trans , lapack_int n , lapack_int nrhs , const float * a , lapack_int lda , const lapack_int * ipiv , float * b , lapack_int ldb );
```

```
lapack_int LAPACKE_dgetrs (int matrix_layout , char trans , lapack_int n , lapack_int nrhs , const double * a , lapack_int lda , const lapack_int * ipiv , double * b , lapack_int ldb );
```

Description

The routine solves for X the following systems of linear equations:

$A*X = B$	if *trans*='N',
$A^T*X = B$	if *trans*='T',
$A^H*X = B$	if *trans*='C' (for complex matrices only).

Before calling this routine, you must call `?getrf` to compute the LU factorization of A.

Symmetric, positive definite matrices

Defining property
*(for any nonzero vector **x**)*

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$$

Symmetric, positive definite matrices

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

LU decomposition is replaced by "Cholesky" factorization

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

A *is a symmetric matrix*

Symmetric, positive definite matrices

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

LU decomposition is replaced by "Cholesky" factorization

$$\mathbf{A} \;=\; \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$\mathbf{L} \;=\; \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ l_{31} & l_{32} & l_{33} & \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

Only a single lower-triangular factor (used twice, via transpose)

Symmetric, positive definite matrices

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

LU decomposition is replaced by "Cholesky" factorization

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ l_{31} & l_{32} & l_{33} & \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \qquad \mathbf{A}_{\text{after}} = \begin{pmatrix} l_{11} & a_{21} & a_{31} & a_{41} \\ l_{21} & l_{22} & a_{32} & a_{42} \\ l_{31} & l_{32} & l_{33} & a_{43} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

*The factor **L** is returned in-place*

?potrf

Computes the Cholesky factorization of a symmetric (Hermitian) positive-definite matrix.

Syntax

```
lapack_int LAPACKE_spotrf (int matrix_layout , char uplo , lapack_int n , float * a ,
lapack_int lda );
```

```
lapack_int LAPACKE_dpotrf (int matrix_layout , char uplo , lapack_int n , double * a ,
lapack_int lda );
```

Description

The routine forms the Cholesky factorization of a symmetric positive-definite or, for complex data, Hermitian positive-definite matrix A:

$A = U^T * U$ for real data, $A = U^H * U$ for complex data	if `uplo='U'`
$A = L * L^T$ for real data, $A = L * L^H$ for complex data	if `uplo='L'`

where L is a lower triangular matrix and U is upper triangular.

LAPACK Linear Equation Computational Routines

Table "Computational Routines for Systems of Equations with Real Matrices" lists the LAPACK computational routines for factorizing, equilibrating, and inverting *real* matrices, estimating their condition numbers, solving systems of equations with real matrices, refining the solution, and estimating its error. Table "Computational Routines for Systems of Equations with Complex Matrices" lists similar routines for *complex* matrices.

Computational Routines for Systems of Equations with Real Matrices

Matrix type, storage scheme	Factorize matrix	Equilibrate matrix	Solve system	Condition number	Estimate error	Invert matrix
general	?getrf	?geequ, ?geequb	?getrs	?gecon	?gerfs, ?gerfsx	?getri
general band	?gbtrf	?gbequ, ?gbequb	?gbtrs	?gbcon	?gbrfs, ?gbrfsx	
general tridiagonal	?gttrf		?gttrs	?gtcon	?gtrfs	
diagonally dominant tridiagonal	?dttrfb		?dttrsb			
symmetric positive-definite	?potrf	?poequ, ?poequb	?potrs	?pocon	?porfs, ?porfsx	?potri
symmetric positive-definite, packed storage	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
symmetric positive-definite, RFP storage	?pftrf		?pftrs			?pftri
symmetric positive-definite, band	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	

Compact storage (RFP)

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Compact storage (RFP)

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Compact storage (RFP)

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Compact storage (RFP)

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{21} & a_{22} & a_{32} & a_{42} \\ a_{31} & a_{32} & a_{33} & a_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \quad \mathbf{A}_{\mathrm{RFP}} = \begin{pmatrix} a_{44} & a_{43} \\ a_{11} & a_{33} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{pmatrix}$$

LAPACK Linear Equation Computational Routines

Table "Computational Routines for Systems of Equations with Real Matrices" lists the LAPACK computational routines for factorizing, equilibrating, and inverting *real* matrices, estimating their condition numbers, solving systems of equations with real matrices, refining the solution, and estimating its error. Table "Computational Routines for Systems of Equations with Complex Matrices" lists similar routines for *complex* matrices.

Computational Routines for Systems of Equations with Real Matrices

Matrix type, storage scheme	Factorize matrix	Equilibrate matrix	Solve system	Condition number	Estimate error	Invert matrix
general	?getrf	?geequ, ?geequb	?getrs	?gecon	?gerfs, ?gerfsx	?getri
general band	?gbtrf	?gbequ, ?gbequb	?gbtrs	?gbcon	?gbrfs, ?gbrfsx	
general tridiagonal	?gttrf		?gttrs	?gtcon	?gtrfs	
diagonally dominant tridiagonal	?dttrfb		?dttrsb			
symmetric positive-definite	?potrf	?poequ, ?poequb	?potrs	?pocon	?porfs, ?porfsx	?potri
symmetric positive-definite, packed storage	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
symmetric positive-definite, RFP storage	$\boxed{\text{?pftrf}}$		$\boxed{\text{?pftrs}}$			?pftri
symmetric positive-definite, band	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	