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 $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$, … (and correspondingly producing multiple solutions $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, …)
- 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.

<table>
<thead>
<tr>
<th>Routine Group</th>
<th>Data Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cblas_?gemm</td>
<td>s, d, c, z</td>
<td>Computes a matrix-matrix product with general matrices.</td>
</tr>
<tr>
<td>cblas_?hemm</td>
<td>c, z</td>
<td>Computes a matrix-matrix product where one input matrix is Hermitian.</td>
</tr>
<tr>
<td>cblas_?herk</td>
<td>c, z</td>
<td>Performs a Hermitian rank-k update.</td>
</tr>
<tr>
<td>cblas_?her2k</td>
<td>c, z</td>
<td>Performs a Hermitian rank-2k update.</td>
</tr>
<tr>
<td>cblas_?symm</td>
<td>s, d, c, z</td>
<td>Computes a matrix-matrix product where one input matrix is symmetric.</td>
</tr>
<tr>
<td>cblas_?syrk</td>
<td>s, d, c, z</td>
<td>Performs a symmetric rank-k update.</td>
</tr>
<tr>
<td>cblas_?syr2k</td>
<td>s, d, c, z</td>
<td>Performs a symmetric rank-2k update.</td>
</tr>
<tr>
<td>cblas_?trmm</td>
<td>s, d, c, z</td>
<td>Computes a matrix-matrix product where one input matrix is triangular.</td>
</tr>
<tr>
<td>cblas_?trsm</td>
<td>s, d, c, z</td>
<td>Solves a triangular matrix equation.</td>
</tr>
</tbody>
</table>
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 iterative 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 \textbf{dense} 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 \textbf{sparse} 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)
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.

<table>
<thead>
<tr>
<th>Matrix type, storage scheme</th>
<th>Factorize matrix</th>
<th>Equilibrate matrix</th>
<th>Solve system</th>
<th>Condition number</th>
<th>Estimate error</th>
<th>Invert matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>?geeque</td>
<td></td>
<td></td>
<td>?gerfsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?gbeque</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Matrix Factorization: LAPACK Computational Routines

Prev: LAPACK Linear Equation Routines
Next: Matrix Factorization: LAPACK Computational Routines
## 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.

<table>
<thead>
<tr>
<th>Matrix type, storage scheme</th>
<th>Factorize matrix</th>
<th>Equilibrate matrix</th>
<th>Solve system</th>
<th>Condition number</th>
<th>Estimate error</th>
<th>Invert matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>?geequub</td>
<td></td>
<td></td>
<td>?gerfsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?gbequub</td>
<td></td>
<td></td>
<td>?gbtrfsx</td>
<td></td>
</tr>
<tr>
<td>diagonally dominant tridiagonal</td>
<td>?dttrfb</td>
<td></td>
<td>?dttrs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?poequub</td>
<td></td>
<td></td>
<td>?porfsx</td>
<td></td>
</tr>
<tr>
<td>symmetric positive-definite, RFP storage</td>
<td>?pftrf</td>
<td></td>
<td>?pftrs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Positive-definite, band</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positive-definite, tridiagonal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indefinite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indefinite, packed storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangular</td>
<td></td>
<td></td>
<td>?trtrs</td>
<td>?trcon</td>
<td>?trrfs</td>
<td>?trtri</td>
</tr>
<tr>
<td>Triangular, RFP storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>?tftri</td>
</tr>
<tr>
<td>Triangular band</td>
<td></td>
<td></td>
<td>?tbtrs</td>
<td>?tbcon</td>
<td>?tbrfs</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Matrix type, storage scheme</th>
<th>Factorize matrix</th>
<th>Equilibrate matrix</th>
<th>Solve system</th>
<th>Condition number</th>
<th>Estimate error</th>
<th>Invert matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>?geequb</td>
<td></td>
<td></td>
<td>?gerfsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?gbequb</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diagonally dominant tridiagonal</td>
<td>?dttrfb</td>
<td></td>
<td>?dttrs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?poequb</td>
<td></td>
<td></td>
<td>?porfsx</td>
<td></td>
</tr>
<tr>
<td>symmetric positive-definite, RFP storage</td>
<td>?pftrf</td>
<td></td>
<td>?pftrs</td>
<td></td>
<td></td>
<td>?pftri</td>
</tr>
</tbody>
</table>
General matrices

\[ A = PLU \]

Factorization Stage: Compute LU
Factorization (or LU “decomposition”)
## General matrices

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

\[ A = PLU \]

\[
L = \begin{pmatrix}
1 &  &  & \\
 & l_{21} & 1 & \\
 & l_{31} & l_{32} & 1 \\
 & l_{41} & l_{42} & l_{43} & 1
\end{pmatrix}
\]

\[
U = \begin{pmatrix}
 &  &  & \\
 &  & u_{11} & u_{12} & u_{13} & u_{14} \\
 &  & u_{22} & u_{23} & u_{24} \\
 &  & u_{33} & u_{34} \\
 & u_{44} &
\end{pmatrix}
\]

\[
P = \begin{pmatrix}
1 &  &  &  \\
 &  &  & 1 \\
 &  &  & 1 \\
1 &  &  & 1
\end{pmatrix}
\]

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

\[ A = PLU \]

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

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

Lower-triangular factor ("unitary" along the diagonal)

Upper-triangular factor

Permutation matrix
General matrices

\[ A = PLU \]

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

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

\[ P = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \]

Factorization Stage: Compute LU

Factorization (or LU “decomposition”)

Both factors returned in-place overwriting the input matrix

Returned as a vector (encoding the permutation)
?getrf

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

Syntax

\[
\text{lapack_int LAPACKE_sgetrf} (\text{int } \text{matrix_layout}, \text{lapack_int } m, \text{lapack_int } n, \text{float } * a, \\
\text{lapack_int } lda, \text{lapack_int } * \text{ipiv});
\]

\[
\text{lapack_int LAPACKE_dgetrf} (\text{int } \text{matrix_layout}, \text{lapack_int } m, \text{lapack_int } n, \text{double } * a, \\
\text{lapack_int } lda, \text{lapack_int } * \text{ipiv});
\]

Description

The routine computes the \(LU\) factorization of a general \(m\text{-by-}n\) matrix \(A\) as

\[
A = P*LU,
\]

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix_layout</td>
<td>Specifies whether matrix storage layout is row major (LAPACK_ROW_MAJOR) or column major (LAPACK_COL_MAJOR).</td>
</tr>
<tr>
<td>m</td>
<td>The number of rows in the matrix $A$ ($m \geq 0$).</td>
</tr>
<tr>
<td>n</td>
<td>The number of columns in $A$; $n \geq 0$.</td>
</tr>
<tr>
<td>a</td>
<td>Array, size at least max($1, lda \times n$) for column-major layout or max($1, lda \times m$) for row-major layout. Contains the matrix $A$.</td>
</tr>
<tr>
<td>lda</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Output Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Overwritten by $L$ and $U$. The unit diagonal elements of $L$ are not stored.</td>
</tr>
<tr>
<td>ipiv</td>
<td>Array, size at least $\text{max}(1, \text{min}(m, n))$. Contains the pivot indices; for $1 \leq i \leq \text{min}(m, n)$, row $i$ was interchanged with row $\text{ipiv}(i)$.</td>
</tr>
</tbody>
</table>
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
General matrices

\[ A = PLU \quad \text{Ax} = PLUx = b \]
General matrices

\[ A = PLU \]

Solve Stage: Use LU decomposition in triangular substitution steps

\[ Ax = PLUx = b \]

\[ Pw = b \]

Solve for \( w \) by permuting elements of \( b \)
General matrices

\[ A = PLU \]

Solve Stage: Use LU decomposition in triangular substitution steps

\[ Ax = PLUx = b \]

\[ Pw = b \]

\[ Lz = w \]

Solve for \( z \) using forward substitution
General matrices

\[ A = PLU \quad Ax = PLUx = b \]

\[ Pw = b \]

\[ Lz = w \]

\[ Ux = z \]

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:

<table>
<thead>
<tr>
<th>Equation</th>
<th>trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A^T X = B$</td>
<td>'T'</td>
</tr>
<tr>
<td>$A^H X = B$</td>
<td>'C'    (for complex matrices only)</td>
</tr>
</tbody>
</table>

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 $x$)

$x^T A x > 0$
Symmetric, positive definite matrices

\[
A = LL^T
\]

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

LU decomposition is replaced by “Cholesky” factorization.

\[
A \text{ is a symmetric matrix}
\]
Symmetric, positive definite matrices

\[ A = LL^T \]

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

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

LU decomposition is replaced by “Cholesky” factorization

Only a single lower-triangular factor (used twice, via transpose)
Symmetric, positive definite matrices

\[ A = LL^T \]

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

\[
L = \begin{pmatrix}
  l_{11} \\
  l_{21} \\
  l_{31} \\
  l_{41}
\end{pmatrix}
\]

\[
A_{\text{after}} = \begin{pmatrix}
  l_{11} & a_{21} & a_{31} & a_{41} \\
  l_{21} & l_{22} & a_{32} & a_{42} \\
  l_{31} & l_{32} & l_{33} & a_{43} \\
  l_{41} & l_{42} & l_{43} & l_{44}
\end{pmatrix}
\]

LU decomposition is replaced by “Cholesky” factorization

The factor \( L \) is returned in-place
potrf

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

Syntax

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

```c
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 $A$:

<table>
<thead>
<tr>
<th>$A = U^T U$ for real data, $A = U^H U$ for complex data</th>
<th>if $\text{uplo}='U'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = L^T L$ for real data, $A = L^H L$ for complex data</td>
<td>if $\text{uplo}='L'$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Matrix type, storage scheme</th>
<th>Factorize matrix</th>
<th>Equilibrate matrix</th>
<th>Solve system</th>
<th>Condition number</th>
<th>Estimate error</th>
<th>Invert matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>?geequb</td>
<td></td>
<td></td>
<td>?gerfx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?gbequb</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diagonally dominant tridiagonal</td>
<td>?dttrfb</td>
<td>?dttrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?poequb</td>
<td></td>
<td></td>
<td>?porfx</td>
<td></td>
</tr>
<tr>
<td>symmetric positive-definite, RFP storage</td>
<td>?pftrf</td>
<td></td>
<td>?pftrs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compact storage (RFP)

\[
A = \begin{pmatrix}
  a_{11} & a_{21} & a_{31} & a_{41} \\
  a_{21} & a_{22} & a_{32} & a_{42} \\
  a_{31} & a_{32} & a_{33} & a_{43} \\
  a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}
\]
Compact storage (RFP)

\[
A = \begin{pmatrix}
\alpha_{11} & \alpha_{21} & \alpha_{31} & \alpha_{41} \\
\alpha_{21} & \alpha_{22} & \alpha_{32} & \alpha_{42} \\
\alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{43} \\
\alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44}
\end{pmatrix}
\]
Compact storage (RFP)

\[
A = \begin{pmatrix}
    a_{11} & a_{21} & a_{31} & a_{41} \\
    a_{21} & a_{22} & a_{32} & a_{42} \\
    a_{31} & a_{32} & a_{33} & a_{43} \\
    a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}
\]
Compact storage (RFP)

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

\[
A_{\text{RFP}} = \begin{pmatrix}
    a_{44} & a_{43} \\
    a_{11} & a_{33} \\
    a_{21} & a_{22} \\
    a_{31} & a_{32} \\
    a_{41} & a_{42}
\end{pmatrix}
\]
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.

<table>
<thead>
<tr>
<th>Matrix type, storage scheme</th>
<th>Factorize matrix</th>
<th>Equilibrate matrix</th>
<th>Solve system</th>
<th>Condition number</th>
<th>Estimate error</th>
<th>Invert matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>?geequb</td>
<td></td>
<td></td>
<td>?gerfsx</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?gbequb</td>
<td></td>
<td></td>
<td>?gbrfsx</td>
<td></td>
</tr>
<tr>
<td>diagonally dominant tridiagonal</td>
<td>?dtrfb</td>
<td></td>
<td>?dttrs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>?poequb</td>
<td></td>
<td></td>
<td>?porfsx</td>
<td></td>
</tr>
<tr>
<td>symmetric positive-definite, RFP storage</td>
<td>?pftrf</td>
<td></td>
<td>?pftrs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>