

## Linear Systems of Equations

---

- The method of moment formulation leads to a linear system of equations of the form:

$$Ax = b$$

- $A$  is a square dense matrix (i.e., fully populated, and is typically complex valued)
  - $x$  is the unknown solution vector
  - $b$  is the known “forcing vector”, which is known and due to the driving source
- In general the solution for  $x$  is found from the inverse of  $A$ :
$$x = A^{-1}b$$
- We need to review basic numerical linear algebra to understand the fundamentals of solving the linear system of equations to understand the efficiency of various solution techniques as well as potential errors that may result from a chosen method
- Excellent reference: “Matrix Computations,” Gene H. Golub and Charles F. Van Loan, John Hopkins University Press, Baltimore, MD. 1989.

## Invertible Matrix

---

- The  $N \times N$  matrix  $A$  is non-singular if:
  - The inverse of  $A$  ( $A^{-1}$ ) exists
  - $\text{Det}(A) \neq 0$
  - $\text{Rank}(A) = N$
  - For any vector  $x \neq 0$ ,  $Ax \neq 0$ .
- If  $A$  is non-singular, then  $Ax = b$  has a unique solution for any  $b$
- A singularity in  $A$  can occur if any two rows (or columns) can be related through a simple linear relationship.
  - Example:
$$A = \begin{bmatrix} 2 & 6 \\ 3 & 9 \end{bmatrix}$$
- A matrix is nearly singular (or highly ill-conditioned) if any two rows (or columns) are closely related through a linear relationship.

## Some Basic Definitions: Vector Norm

---

- Vector Norm:

- Similar to the magnitude of a scalar, we define the norm of a vector

- $p$ -norm:

$$\|x\|_p = \left( \sum_{i=1}^N |x_i|^p \right)^{\frac{1}{p}}$$

- Typically used:

- 1-norm  $\|x\|_1 = \sum_{i=1}^N |x_i|$

- 2-norm  $\|x\|_2 = \sqrt{\sum_{i=1}^N |x_i|^2}$

- $\infty$ -norm:  $\|x\|_\infty = \max |x_i|$

- In general for a vector  $x$  in  $R^n$ :

$$\|x\|_1 \geq \|x\|_2 \geq \|x\|_\infty$$

## Properties of Vector Norm

---

- For any vector  $p$ -norm:

$$\|x\| > 0 \text{ if } x \neq 0$$

$$\|\alpha x\| = |\alpha| \|x\| \text{ for any scalar } \alpha$$

$$\|x + y\| \leq \|x\| + \|y\| \text{ (triangle inequality)}$$

$$\left| \|x\| - \|y\| \right| \leq \|x - y\|$$

## Matrix Norms

---

- The norm of a matrix measures the maximum “stretching” the matrix does to any vector for a given vector norm

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

- The matrix norm resulting from a vector 1-norm is the maximum absolute column sum:

$$\|A\|_1 = \max_j \sum_{i=1}^N |a_{i,j}|$$

- The matrix norm resulting from a vector infinite-norm is the maximum absolute row sum:

$$\|A\|_\infty = \max_i \sum_{j=1}^N |a_{i,j}|$$

## Properties of Matrix Norm

---

- For any matrix norm:

$$\|A\| > 0 \text{ if } A \neq 0$$

$$\|\alpha A\| = |\alpha| \|A\| \text{ for any scalar } \alpha$$

$$\|A + B\| \leq \|A\| + \|B\|$$

$$\|Ax\| \leq \|A\| \|x\|$$

$$\|AB\| \leq \|A\| \|B\|$$

## Condition Number of a Matrix

---

- The condition number of a matrix is defined as:

$$\text{cond}(A) = \|A\| \|A^{-1}\|$$

- Based on the definition of matrix norm:  $\|A\| \|A^{-1}\| = \left( \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right) \left( \min_{x \neq 0} \frac{\|Ax\|}{\|x\|} \right)^{-1}$
- Therefore, the condition number is the ratio of the maximum stretching of a vector to the maximum shrinking of a vector
- Defining the eigenvectors and eigenvalues of  $A$ :  $Ax = \lambda x$

– Condition number can also be identified as:  $\text{cond}(A) = \lambda_{\max} / \lambda_{\min}$

- Two extremes:

–  $A$  is the identity matrix  $I$ :  $\text{cond}(I) = \|I\| \|I^{-1}\| = 1$

–  $A$  is a singular matrix:  $\text{cond}(A) = \|A\| \|A^{-1}\| = \infty$

- In general, for very large condition numbers,  $A$  is nearly singular
  - The error in a matrix solution will then be amplified!

## More Properties of Condition Number

---

$$\text{cond}(A) \geq 1$$

$$\text{cond}(\alpha A) = \text{cond}(A)$$

$$\text{cond}(I) = 1$$

Define a diagonal matrix  $D$  :

$$\text{cond}(D) = \frac{\max |d_i|}{\min |d_i|}$$

- Finally, computing the condition number is challenging as it requires either estimating the eigenvalues of  $A$  or the norm of  $A^{-1}$ . There are numerical routines to do so in LA-PACK (Linear Algebra Package). We will discuss these routines at a later date. The numerical methods used to perform this calculation are interesting, but beyond the scope of this course.

## Solution of Linear Systems of Equations

---

- Gaussian Elimination
- LU-Factorization
- Pivoting
- Affects of Condition number

## Gaussian Elimination

---

- In Gaussian Elimination, we transform the linear system of equations into a “Triangular” matrix
  - Performed via a series of transformations that successively zero’s out entries of the matrix
- The resulting linear system is solved using “backward” substitution
- Given the matrix equation:

$$Ax = b \Rightarrow$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- Divide row 1 by  $a_{1,1}$  and then multiply the row by  $-a_{1,2}$  and add to row 2, and multiply by  $-a_{1,3}$  and add to row 3, leads to:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,2} - a_{1,2} \frac{a_{2,1}}{a_{1,1}} & a_{2,3} - a_{1,3} \frac{a_{2,1}}{a_{1,1}} \\ 0 & a_{3,2} - a_{1,2} \frac{a_{3,1}}{a_{1,1}} & a_{3,3} - a_{1,3} \frac{a_{3,1}}{a_{1,1}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - b_1 \frac{a_{2,1}}{a_{1,1}} \\ b_3 - b_1 \frac{a_{3,1}}{a_{1,1}} \end{bmatrix}$$

## Matrix Transformation

---

- This can be defined via a matrix transformation:

$$M_1Ax = M_1b$$

- Where,

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -a_{2,1}/a_{1,1} & 1 & 0 \\ -a_{3,1}/a_{1,1} & 0 & 1 \end{bmatrix}$$

- Next, define  $M_2$  as below, where  $\tilde{a}_{3,2}$  is from the product above.

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\tilde{a}_{3,2}/a_{2,2} & 1 \end{bmatrix}$$

- Then the final transformation leads to a triangular matrix:

$$M_2M_1Ax = M_2M_1b$$

## Triangular Matrix

---

- The triangular matrix is of the form:

$$M_2 M_1 A = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & u_{3,3} \end{bmatrix}, \quad M_2 M_1 b = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \end{bmatrix}$$

- For a general  $N \times N$  square matrix:

$$M_k = \begin{bmatrix} 1 & \cdots & & 0 & \cdots & 0 \\ \vdots & \ddots & 0 & \vdots & & \vdots \\ & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -\frac{\tilde{a}_{k+1,k}}{\tilde{a}_{k,k}} & 0 & \cdots & 0 \\ \vdots & & \vdots & 0 & \ddots & \\ 0 & \cdots & -\frac{\tilde{a}_{N,k}}{\tilde{a}_{k,k}} & 0 & \cdots & 1 \end{bmatrix} \quad M_{N-1} \cdots M_1 A = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,N} \\ 0 & u_{2,2} & \cdots & u_{2,N} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & u_{N,N} \end{bmatrix}$$

## Back-Substitution

---

- The transformations result in an upper-triangular matrix
- This can be solved using “back substitution”:

```
do  $j = N, 1, -1$   
   $x_j = \tilde{b}_j$   
  do  $i = j + 1, N$   
     $x_j = x_j - u_{j,i}x_i$   
  enddo  
   $x_j = x_j / u_{j,j}$   
enddo
```

- Floating-point Operations:
  - Gaussian Elimination:  $O(N^3)$
  - Backward Substitution:  $O(N^2)$

## Limitations

---

- Gaussian elimination operators on the matrix and the forcing vector
  - $b$  must be known *a priori*
    - This may not be desirable, as we may want to solve later for currently unknown forcing vectors (fairly common)
- During the elimination process, we can encounter a zero diagonal element due to cancellation
  - This does not necessarily imply that the matrix is singular
  - It is the order of operations that can lead to this problem
  - Solution: pivoting
- For certain classes of matrices, pivoting will *not* be needed:
  - *Diagonally dominant matrices*
$$\left( \sum_{\substack{i=1 \\ i \neq j}}^n |a_{i,j}| \right) < |a_{j,j}|$$
  - *Symmetric, positive definite matrices*  $A = A^T$  and  $x^T A x > 0$  for all  $x \neq 0$
- In computational EM, our matrices do not have such nice properties!

## LU-Factorization

---

- Recognizing certain properties, we can factorize the matrix  $A$  into the product of lower and upper triangular matrices

$$A = LU = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ l_{N,1} & \cdots & l_{N,N-1} & 1 \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,N} \\ 0 & u_{2,2} & \cdots & u_{2,N} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & u_{N,N} \end{bmatrix}$$

- Using the transformations of Gaussian Elimination, we can define:

$$M_{N-1} \cdots M_1 A = U$$

$$(M_{N-1} \cdots M_1)^{-1} = L$$

$$\therefore (M_{N-1} \cdots M_1)^{-1} (M_{N-1} \cdots M_1 A) = LU = A$$

- Advantages:
  - Having factorized  $A$  into  $L$  and  $U$ , we can solve the linear system for any number of forcing functions using “forward/backward” substitution

## Forward/Backward Substitution

---

- Solving:  $LUx = b$
- Solve as:  
 $Ly = b$  using forward substitution  
 $Ux = y$  using backward substitution
- We defined the backward substitution algorithm earlier
- Forward substitution algorithm (Note  $l_{i,i} = 1$ ):

```
do  $j = 1, N$   
   $x_j = b_j$   
  do  $i = 1, j - 1$   
     $x_j = x_j - l_{j,i}x_i$   
  enddo  
enddo
```

- Note that no transformations are made on  $b$

## Efficient LU-Factorization Algorithm

---

- The LU-factorization of a matrix can be performed in a computationally efficient manner using a triple-nested loop.
- The general *ijk* algorithm has the form:

```
do ...
  do ...
    do ...
      
$$a_{i,j} = a_{i,j} - (a_{i,k} / a_{k,k}) a_{k,j}$$

    enddo
  enddo
enddo
```

- The order of looping through *i*, *j*, and *k* is arbitrary.
  - The choice of the order affects the efficiency of the algorithm for a given computer architecture.

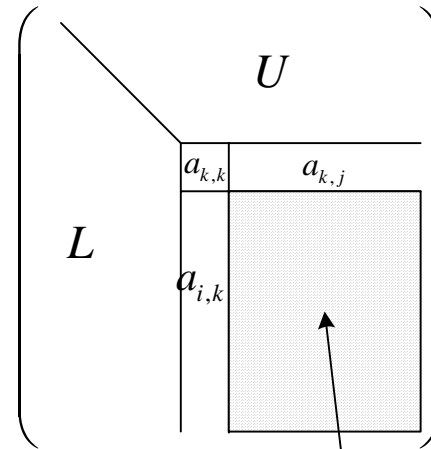
## The *kji* - Algorithm

- One of the simplest representations of LU-factorization is the *kji* algorithm

```

do k = 1, N - 1
  do l = k + 1, N
     $a_{l,k} = a_{l,k} / a_{k,k}$ 
  enddo
  do j = k + 1, N
    do i = k + 1, N
       $a_{i,j} = a_{i,j} - a_{i,k} a_{k,j}$ 
    enddo
  enddo
enddo

```



$$a_{i,j} = a_{i,j} - a_{i,k} a_{k,j}$$

Performed via the “SAXPY” operation  
( $z = a x + y$  - BLAS routine)

- Computation requires the outer product of the active row and column to transform the shaded block

## The *jki* - Algorithm

- The *jki* version is better suited for RISC processors
  - Total operations:  $2N^3/3$  ( $A$  is Real),  $8N^3/3$  ( $A$  is Complex)
  - When completed,  $a_{i,j}$  houses  $L$  for  $i > j$  and  $U$  for  $i \leq j$

do  $j = 1, N$

Solve  $\{L(1:j-1, 1:j-1)U(1:j-1, j) = A(1:j-1, j)\}$

do  $k = 1, j-1$

do  $i = k+1, j-1$

$$a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$$

enddo

enddo

$\{v(j:N) = A(j:N, j) - L(j:N, 1:j-1)U(1:j-1, j)\}$

do  $k = 1, j-1$

do  $i = j, N$

$$a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$$

enddo

enddo

$\{L(j+1:N, j) = v(j+1:N)/v(j)\}$

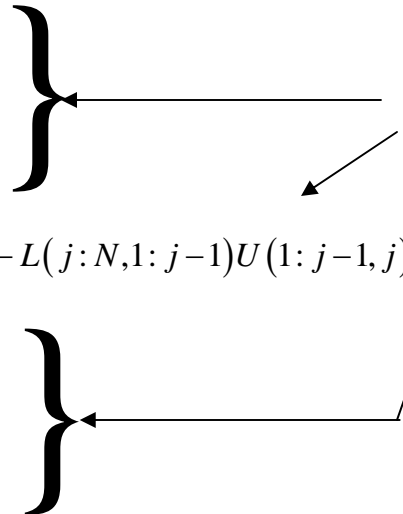
do  $l = j+1, N$

$$a_{l,j} = a_{l,j}/a_{j,j}$$

enddo

enddo

Performed using a "GAXPY" operation  
(  $z = Ax + y$  )



## Pivoting

---

- As found earlier, Gaussian Elimination or LU factorization can result in zero-diagonal elements, which leads to a catastrophic error
- Near-zero diagonal elements can lead to significant rounding error
- Pivoting reorders rows and columns prior to each transformation to move the maximum element to the diagonal
  - Pivoting is done through the product with a “permutation” matrix
- Example:

$$A = \begin{bmatrix} .0001 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10,000 & 1 \end{bmatrix} \begin{bmatrix} .0001 & 1 \\ 0 & -9999 \end{bmatrix} = LU$$

- Define the permutation matrix:

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- Then

$$PA = \begin{bmatrix} 1 & 1 \\ .0001 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ .0001 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0.9999 \end{bmatrix} = LU$$

## Permutation Matrix

---

- A permutation matrix is the identity matrix with its rows reordered.

– e.g.:

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Permutation matrices are orthogonal. Therefore:  $P^{-1} = P^T$
- Permutation vectors are used for pivoting
- Partial Pivoting: Exchange two rows to move the largest column entry to the diagonal
  - Thus, the permutation vector is defined by permuting only two rows of the identity matrix

## Example of Partial Pivoting

---

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix}$$

$$\text{Define: } P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \Rightarrow P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 2 & 4 & -2 \\ 3 & 7 & 10 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix} \Rightarrow M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & -2 & 2 \\ 0 & 8 & 16 \end{bmatrix}$$

$$\text{Define: } P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \Rightarrow P_2 M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & -2 & 2 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/4 & 1 \end{bmatrix} \Rightarrow M_2 P_2 M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix} = U$$

$$L = (M_2 P_2 M_1 P_1)^{-1}$$

## Error in Solution

---

Let  $\hat{L}$  and  $\hat{U}$  be the computed factors of  $A$  using partial pivoting. Then, let  $\hat{y}$  be the computed solution to  $\hat{L}y = b$ , and  $\hat{x}$  be the computed solution to  $\hat{U}x = \hat{y}$ . Then,  $\hat{x}$  satisfies  $(A + E)\hat{x} = b$ , with

$$\|E\|_{\infty} \leq 8n^3 \rho \|A\|_{\infty} \varepsilon_{\text{mach}}$$

where  $\rho$  is the growth factor defined by:

$$\rho = \max_{i,j,k} \frac{|\hat{a}_{i,j}^{(k)}|}{\|A\|_{\infty}}$$

and  $\hat{A}^{(k)}$  is the computed version of the matrix:

$$A^{(k)} = M_k P_k \cdots M_1 P_1 A$$

Without pivoting  $\rho$  can become unbounded. With pivoting,  $\rho$  can be as large as  $2^{n-1}$ , but it is typically much smaller ( $\sim 10$ ).

The error in  $\hat{x}$  can also be estimated to be on the order of:  $\text{cond}(A) \varepsilon_{\text{mach}}$

## Example

---

- Using 3-digit arithmetic, consider the following solution :

$$A = \begin{bmatrix} .151 & 1.22 \\ .303 & 2.44 \end{bmatrix} \quad b = \begin{bmatrix} -0.1 \\ 0.25 \end{bmatrix}$$

$$\text{cond}(A) = 5500$$

$$M_1 A = \begin{bmatrix} .151 & 1.22 \\ 0 & 0 \end{bmatrix} \Rightarrow x = \begin{bmatrix} \infty \\ \infty \end{bmatrix}$$

with pivoting:

$$P_1 A = \begin{bmatrix} .303 & 2.44 \\ .151 & 1.22 \end{bmatrix}, \quad M_1 P_1 A = \begin{bmatrix} .303 & 2.44 \\ 0 & 0.00488 \end{bmatrix}, \quad M_1 P_1 b = \begin{bmatrix} 0.25 \\ -0.225 \end{bmatrix}$$

$$x = \begin{bmatrix} 370 \\ -46.1 \end{bmatrix}$$

Exact solution:

$$x = \begin{bmatrix} 450 \\ -55.778688524\dots \end{bmatrix}$$

Not even 1 digit of precision!

$$\text{Note } \text{cond}(A) \cdot \varepsilon_{\text{mach}} = 5500 \cdot 10^{-3} = 5.5$$

## Example

---

- Repeat using 10-digit arithmetic:

$$A = \begin{bmatrix} .151 & 1.22 \\ .303 & 2.44 \end{bmatrix} \quad b = \begin{bmatrix} -0.1 \\ 0.25 \end{bmatrix}$$

with pivoting:

$$P_1 A = \begin{bmatrix} .303 & 2.44 \\ .151 & 1.22 \end{bmatrix}, \quad M_1 P_1 A = \begin{bmatrix} .303 & 2.44 \\ 0 & -.0040260264 \end{bmatrix}, \quad M_1 P_1 b = \begin{bmatrix} 0.25 \\ -0.2245874587 \end{bmatrix}$$

$$x = \begin{bmatrix} 450.0000046 \\ -55.77868910 \end{bmatrix}$$

Exact solution:

$$x = \begin{bmatrix} 450 \\ -55.778688524589\dots \end{bmatrix}$$

Solution has about 7 digits of precision. Note  $\text{cond}(A) \cdot \epsilon_{\text{mach}} \approx 10^{-7}$

## LA-PACK

---

- LA-PACK (“Linear Algebra Package”) is a public domain set of subroutines available to perform linear-algebra operations numerically.
- Users-manual found online at: <http://www.netlib.org/lapack/lug/>
- Most computer systems with tuned compilers have math-libraries you link with during compilation to the lapack libraries as well as the processor tuned BLAS libraries.
  - For example on the UK superdome cluster (sdx) compile with `-llapack` and `-lveclib`

# Linear System of Equations

**Table 2.7:** Computational routines for linear equations

Type of matrix and storage scheme	Operation	Single precision		Double precision	
		real	complex	real	complex
general	factorize	SGETRF	CGETRF	DGETRF	ZGETRF
	solve using factorization	SGETRS	CGETRS	DGETRS	ZGETRS
	estimate condition number	SGECON	CGECON	DGECON	ZGECON
	error bounds for solution	SGERFS	CGERFS	DGERFS	ZGERFS
	invert using factorization	SGETRI	CGETRI	DGETRI	ZGETRI
	equilibrate	SGEQU	CGEQU	DGEQU	ZGEQU
general	factorize	SGBTRF	CGBTRF	DGBTRF	ZGBTRF
band	solve using factorization	SGBTRS	CGBTRS	DGBTRS	ZGBTRS
	estimate condition number	SGBCON	CGBCON	DGBCON	ZGBCON
	error bounds for solution	SGBRFS	CGBRFS	DGBRFS	ZGBRFS
	equilibrate	SGBEQU	CGBEQU	DGBEQU	ZGBEQU
general	factorize	SGTTRF	CGTTRF	DGTTRF	ZGTTRF
tridiagonal	solve using factorization	SGTTRS	CGTTRS	DGTTRS	ZGTTRS
	estimate condition number	SGTCON	CGTCON	DGTCON	ZGTCON
	error bounds for solution	SGTRFS	CGTRFS	DGTRFS	ZGTRFS
symmetric/Hermitian	factorize	SPOTRF	CPOTRF	DPOTRF	ZPOTRF
positive definite	solve using factorization	SPOTRS	CPOTRS	DPOTRS	ZPOTRS
	estimate condition number	SPOCON	CPOCON	DPOCON	ZPOCON
	error bounds for solution	SPORFS	CPORFS	DPORFS	ZPORFS
	invert using factorization	SPOTRI	CPOTRI	DPOTRI	ZPOTRI
	equilibrate	SPOEQU	CPOEQU	DPOEQU	ZPOEQU
symmetric/Hermitian	factorize	SPPTRF	CPPTRF	DPPTRF	ZPPTRF
positive definite	solve using factorization	SPPTRS	CPPTRS	DPPTRS	ZPPTRS

# CGETRF – Perform LU Factorization of Complex Matrix (SP)

```
xterm
CGETRF(3F)                                CGETRF(3F)
NAME
CGETRF - compute an LU factorization of a general M-by-N matrix A using
partial pivoting with row interchanges
SYNOPSIS
SUBROUTINE CGETRF( M, N, A, LDA, IPIV, INFO )
        INTEGER      INFO, LDA, M, N
        INTEGER      IPIV( * )
        COMPLEX      A( LDA, * )
PURPOSE
CGETRF computes an LU factorization of a general M-by-N matrix A using
partial pivoting with row interchanges.
The factorization has the form
  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).
This is the right-looking Level 3 BLAS version of the algorithm.
ARGUMENTS
M      (input) INTEGER
       The number of rows of the matrix A.  M >= 0.
N      (input) INTEGER
       The number of columns of the matrix A.  N >= 0.
A      (input/output) COMPLEX array, dimension (LDA,N)
       On entry, the M-by-N matrix to be factored.  On exit, the factors
       L and U from the factorization A = P*L*U; the unit diagonal
       elements of L are not stored.
LDA    (input) INTEGER
       The leading dimension of the array A.  LDA >= max(1,M).
IPIV   (output) INTEGER array, dimension (min(M,N))
       The pivot indices; for 1 <= i <= min(M,N), row i of the matrix
       was interchanged with row IPIV(i).
INFO   (output) INTEGER
       = 0: successful exit
       < 0: if INFO = -i, the i-th argument had an illegal value
       > 0: if INFO = i, U(i,i) is exactly zero. The factorization has
       been completed, but the factor U is exactly singular, and
       division by zero will occur if it is used to solve a system of
       equations.
Page 1
```

# CGETRS – Forward/Backward Solution Given LU Factorization

```
xterm
CGETRS(3F)                                CGETRS(3F)
NAME
CGETRS - solve a system of linear equations  $A * X = B$ ,  $A^{**T} * X = B$ , or
 $A^{**H} * X = B$  with a general N-by-N matrix A using the LU factorization
computed by CGETRF.
SYNOPSIS
SUBROUTINE CGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )

      CHARACTER          TRANS
      INTEGER            INFO, LDA, LDB, N, NRHS
      INTEGER            IPIV( * )
      COMPLEX            A( LDA, * ), B( LDB, * )

PURPOSE
CGETRS solves a system of linear equations
 $A * X = B$ ,  $A^{**T} * X = B$ , or  $A^{**H} * X = B$  with a general N-by-N
matrix A using the LU factorization computed by CGETRF.
ARGUMENTS
TRANS  (input) CHARACTER*1
        Specifies the form of the system of equations:
        = 'N':  $A * X = B$       (No transpose)
        = 'T':  $A^{**T} * X = B$  (Transpose)
        = 'C':  $A^{**H} * X = B$  (Conjugate transpose)

N      (input) INTEGER
        The order of the matrix A,  $N \geq 0$ .

NRHS   (input) INTEGER
        The number of right hand sides, i.e., the number of columns of
        the matrix B.  $NRHS \geq 0$ .

A      (input) COMPLEX array, dimension (LDA,N)
        The factors L and U from the factorization  $A = P * L * U$  as computed
        by CGETRF.

LDA    (input) INTEGER
        The leading dimension of the array A.  $LDA \geq \max(1,N)$ .

IPIV  (input) INTEGER array, dimension (N)
        The pivot indices from CGETRF; for  $1 \leq i \leq N$ , row i of the matrix
        was interchanged with row IPIV(i).

B      (input/output) COMPLEX array, dimension (LDB, NRHS)
        On entry, the right hand side matrix B. On exit, the solution
        matrix X.

Page 1
```