Math 541 - Numerical Analysis

Lecture Notes – Linear Algebra: Part A

Joseph M. Mahaffy,
⟨jmahaffy@mail.sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://jmahaffy.sdsu.edu

Spring 2018

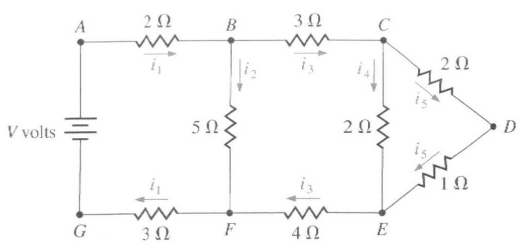
Outline

Applications

Applications and Matrices: Widely used in many fields

Kirchhoff's Law: Matrices used to find currents in an electric circuit

- At any node in an electrical circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node
- The directed sum of the electrical potential differences (voltage) around any closed network is zero



Kirchhoff's Law

Kirchhoff's Law applied to the circuit above gives the ***system of equations***

$$\begin{aligned} 5i_1 + 5i_2 &= V \\ i_3 - i_4 - i_5 &= 0 \\ 2i_4 - 3i_5 &= 0 \\ i_1 - i_2 - i_3 &= 0 \\ 5i_2 - 7i_3 - 2i_4 &= 0 \end{aligned}$$

or

$$AI = B$$

with $I = [i_1, i_2, i_3, i_4, i_5]^T$, $B = [V, 0, 0, 0, 0]^T$, and

$$A = \begin{pmatrix} 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 2 & -3 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 5 & -7 & -2 & 0 \end{pmatrix}$$

Kirchhoff's Law – MatLab Solution

From above, we want to solve $AI = b$ or

$$\begin{pmatrix} 5 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 2 & -3 \\ 1 & -1 & -1 & 0 & 0 \\ 0 & 5 & -7 & -2 & 0 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{pmatrix} = \begin{pmatrix} V \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

If $V = 1.5$, then **MatLab** gives the solution

$$\begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{pmatrix} = \begin{pmatrix} 0.185047 \\ 0.114953 \\ 0.070093 \\ 0.042056 \\ 0.028037 \end{pmatrix}$$

MatLab Solution (Easy) – $AI = b$

There are multiple ways where **MatLab** solves the above system

$$AI = b$$

- `A\b`
- `inv(A)*b`
- `linsolve(A,b)`
- `rref([A,b])`

- Are all of these calculations the same?
- Which methods are more efficient and why?
- How does MatLab perform these calculations and what problems arise?

Linear System

Linear System: Operations to simplify

$$\begin{aligned} E_1 &: \quad a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ E_2 &: \quad a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ &\vdots \\ E_n &: \quad a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{aligned}$$

- E_i can be multiplied by a nonzero constant λ with the resulting equation used in place of E_i (Denoted $(\lambda E_i) \to (E_i)$.)
- E_j can be multiplied by any constant λ and added to E_i with the resulting equation used in place of E_i (Denoted $(E_i + \lambda E_j) \to (E_i)$.)
- E_i and E_j can be transposed in order. (Denoted $(E_i) \longleftrightarrow (E_j)$.)

Solving $Ax = b$

Let A be an $n \times n$ matrix and x and b be $n \times 1$ vectors.

Consider the **system of linear equations** given by

$$Ax = b$$

The solution set x satisfies one of the following:

1. The system has a ***single unique solution***
2. The system has ***infinitely many solutions***
3. The system has ***no solution***

Note that the system has a ***unique solution*** if and only if $\det(A) \neq 0$ or equivalently A is ***nonsingular*** (it has an ***inverse***)

Gaussian Elimination

Elimination Process: We want to describe the step-by-step process to solve

$$
\begin{aligned}
E_1: &\quad a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = a_{1,n+1} \\
E_2: &\quad a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = a_{2,n+1} \\
&\quad \vdots \\
E_n: &\quad a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = a_{n,n+1}
\end{aligned}
$$

Begin by creating the **augmented matrix** $A = [a_{ij}]$ for $1 \le i \le n$ and $1 \le j \le n+1$

We desire a **programmable process** for creating an equivalent system with an ***upper triangular matrix***, which is then readily solved by ***backward substitution***

Gaussian Elimination

We take the original $n \times n$ ***linear system*** and create the **augmented matrix** $A = [a_{ij}]$ for $1 \le i \le n$ and $1 \le j \le n+1$

Algorithm (Gaussian Elimination)

- *For $i = 1, ..., n-1$, do the next* **3 steps**
  1. *Let p be the smallest integer with $i \le p \le n$ and $a_{pi} \ne 0$. If no integer p can be found, then* **OUTPUT:** ***no unique solution exists*** *and* **STOP**
  2. *If $p \ne i$, then perform $(E_p) \longleftrightarrow (E_i)$ (**pivoting**)*
  3. *For $j = i+1, ..., n$ do the following:*
     1. *Set $m_{ji} = a_{ji}/a_{ii}$*
     2. *Perform $(E_j - m_{ji}E_i) \to (E_j)$ (producing a leading zero element in Row j)*
  4. *If $a_{nn} = 0$, then* **OUTPUT:** ***no unique solution exists*** *and* **STOP**

Back Substitution

The previous **algorithm** produces an ***augmented matrix*** with the first n columns creating an ***upper triangular matrix***, $U = [u_{ij}]$

Algorithm (Back Substitution)

- *Set $x_n = u_{n,n+1}/u_{nn}$*
- *For $i = n-1, ..., 1$ set*

$$x_i = \frac{1}{u_{ii}} \left[u_{i,n+1} - \sum_{j=i+1}^{n} u_{ij}x_j \right]$$

- **OUTPUT** $(x_1, ..., x_n)$

Gaussian Elimination Operations

The previous **algorithms** solve

$$Ax = b$$

There were numerous **Multiplications/divisions** and **Additions/subtractions** in the **Gaussian elimination** and **back substitution**

These calculations are readily counted

- **Multiplications/divisions** total

$$\frac{n^3}{3} + n^2 - \frac{n}{3}$$

- **Additions/subtractions** total

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$$

which means that **arithmetic operations** are proportional to n^3, the ***dimension of the system***

Partial Pivoting

After **pivoting** our **Algorithm** uses the new **pivot element** to produce **0** below in the remaining rows

The operation is

$$m_{ji} = a_{ji}/a_{ii}$$

If a_{ii} is small compared to a_{ji}, then $m_{ji} \gg 1$, which can introduce significant ***round-off error***

Further computations compound the original error

In addition, the **back substitution** using the small a_{ii} also introduces more error, which means that the ***round-off error*** dominates the calculations

Pivoting Strategy: Row exchanges are done to reduce ***round-off error***

Partial Pivoting – Example

Example: Consider the following system of equations:

$$\begin{aligned} E_1: &\quad 0.003000x_1 + 59.14x_2 = 59.17 \\ E_2: &\quad 5.291x_1 - 6.130x_2 = 46.78 \end{aligned}$$

Apply ***Gaussian elimination*** to this system with 4-digit arithmetic with rounding and compare to the exact solution, which is $x_1 = 10.00$ and $x_2 = 1.000$

Solution: The first ***pivot*** element is $a_{11} = 0.003000$, which is small, and its multiplier is

$$m_{21} = \frac{5.291}{0.003000} = 1763.66\bar{6}$$

rounding to $m_{21} = 1764$, which is large

Partial Pivoting – Example

Example (cont): Performing $(E_2 - m_{21}E_1) \to (E_2)$ with appropriate rounding gives

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17 \\ -104300x_2 &\approx -104400 \end{aligned}$$

while the exact system is

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17 \\ -104309.37\bar{6}x_2 &= -104309.37\bar{6} \end{aligned}$$

The disparity in $m_{21}a_{13}$ and a_{23} has introduced ***round-off error***, but it has not been propagated

Partial Pivoting – Example

Example (cont): ***Back substitution*** yields

$$x_2 \approx 1.001,$$

which is close to the actual value $x_2 = 1.000$

However, the small ***pivot*** $a_{11} = 0.003000$ gives

$$x_1 = \frac{59.17 - (59.14)(1.001)}{0.003000} = -10.00,$$

while the actual value is $x_1 = 10.00$

The ***round-off error*** comes from the small error of 0.001 multiplied by

$$\frac{59.14}{0.003000} \approx 20000$$

For this system is is very easy to see where the error occurs and propagates, but it becomes much harder in larger systems

Partial Pivoting

Partial Pivoting: To avoid the difficulty in the previous **Example**, we select the largest magnitude element in the column below the diagonal and perform a ***pivoting*** with this row

Specifically, determine the smallest $p \geq k$ such that

$$|a_{pk}| = \max_{k \leq i \leq n} |a_{ik}|$$

and perform $(E_k) \longleftrightarrow (E_p)$

If this is done on the previous **Example**, then the 4-digit rounding answer agrees with the exact answer

Gaussian Elimination with Partial Pivoting

To perform **Gaussian Elimination with Partial Pivoting**, we use the previous **Gaussian Elimination** and **Back substitution algorithms** with the replacement of the first step by the following:

Algorithm (Gaussian Elimination with Partial Pivoting)

1. *Find the smallest $p \geq k$ such that*

$$|a_{pk}| = \max_{k \leq i \leq n} |a_{ik}|.$$

If $|a_{pk}| = 0$, then **OUTPUT:** ***no unique solution exists*** *and* **STOP**

Gaussian Elimination with Partial Pivoting

This **Gaussian Elimination with Partial Pivoting** procedure is relatively easy to code and provides a "reasonably" stable algorithm for solving $Ax = b$

Further improvements with additional costs that are $\mathcal{O}(n^3)$ can be accomplished by **pivoting** both **rows** and **columns**

This strategy is recommended for systems where accuracy is essential and the additional execution time is justified (roughly doubles the execution time)

Example

Example: Consider the following system:

$$\begin{aligned} 10x_1 - 7x_2 &= 7 \\ -3x_1 + 2x_2 + 6x_3 &= 4 \\ 5x_1 - x_2 + 5x_3 &= 6 \end{aligned}$$

This is written as the ***matrix equation***

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}$$

The first step is accomplished by adding 0.3 times the first equation to the second equation and subtracting 0.5 times the first equation from the third equation:

$$(0.3R_1 + R_2) \to (R_2) \qquad \text{and} \qquad (-0.5R_1 + R_3) \to (R_3)$$

Example

Example: This operation is the ***first pivot***

$$(0.3R_1 + R_2) \to (R_2) \qquad \text{and} \qquad (-0.5R_1 + R_3) \to (R_3)$$

Resulting in

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix}$$

The ***second pivot*** could perform the operation $(25R_2 + R_3) \to (R_3)$, but in general, we select the largest coefficient and perform a ***pivoting*** (minimizing ***roundoff error***), which in this case, is

$$(R_3) \longleftrightarrow (R_2)$$

resulting in

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.1 \end{pmatrix}$$

Applications
Gaussian Elimination
LU Factorization
Example
General LU Factorization
MatLab Program for solving $Ax = b$

Example

Example: Now the ***second pivot*** is 2.5, and x_2 is eliminated from the third equation by

$$(0.04R_2 + R_3) \to (R_3)$$

Resulting in

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix}$$

This produces an **Upper Triangular Matrix**

The solution is obtained by ***back substitution***, so

$$6.2x_3 = 6.2 \qquad \text{or} \qquad x_3 = 1$$

The next equation is

$$2.5x_2 + 5(1) = 2.5 \qquad \text{or} \qquad x_2 = -1$$

The final equation is

$$10x_1 - 7(-1) = 7 \qquad \text{or} \qquad x_1 = 0$$

Applications
Gaussian Elimination
LU Factorization
Example
General LU Factorization
MatLab Program for solving $Ax = b$

Example

Example: This set of operations can be compactly written in matrix notation

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & -0.04 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} \qquad P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

where U is the final coefficient matrix, L contains the multipliers used in the elimination, and P describes all the pivoting

With these matrices,

$$LU = PA,$$

which means the original coefficient matrix is expressed in terms of products of matrices with simpler structures

Applications
Gaussian Elimination
LU Factorization
Example
General LU Factorization
MatLab Program for solving $Ax = b$

LU Factorization Example

Example Reviewed: Return to the steps of **Gaussian Elimination** in previous example, starting with

$$(0.3R_1 + R_2) \to (R_2)$$

This can be written

$$M_1A = \begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 5 & -1 & 5 \end{pmatrix}$$

Similarly, $(-0.5R_1 + R_3) \to (R_3)$ can be written

$$M_2(M_1A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 5 & -1 & 5 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix}$$

LU Factorization Example

Example Reviewed: Exchanging rows uses a ***permutation matrix***, P_{23}

$$(R_2) \longleftrightarrow (R_3)$$

This can be written

$$P_{23}(M_2M_1A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix}$$

Similarly, $(0.04R_2 + R_3) \to (R_3)$ can be written

$$M_3(P_{23}M_2M_1A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0.04 & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix} = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix}$$

Thus,

$$U = M_3P_{23}M_2M_1A$$

LU Factorization Example

Example Reviewed: We are solving

$$Ax = b,$$

so

$$M_3P_{23}M_2M_1Ax = M_3P_{23}M_2M_1b \qquad \text{or} \qquad Ux = y,$$

which is easily solved by ***back substitution***
This implies that

$$U = M_3P_{23}M_2M_1A \qquad \text{or} \qquad A = M_1^{-1}M_2^{-1}P_{23}^{-1}M_3^{-1}U = L_1L_2P_{23}^{-1}L_3U$$

However,

$$M_1^{-1} = L_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -0.3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

LU Factorization Example

Example Reviewed: Similarly, $M_2^{-1} = L_2$ and $M_3^{-1} = L_3$ with

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix} \qquad \text{and} \qquad L_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -0.04 & 1 \end{pmatrix}$$

The ***permutation matrix*** is its own inverse, so

$$P_{23} = P_{23}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad \text{or} \qquad P_{23} \cdot P_{23} = I$$

Consider

$$LP_{23} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 0 & 1 \\ l_{31} & 1 & 0 \end{pmatrix}$$

LU Factorization Example

Example Reviewed: Multiplying by the ***permutation matrix***

$$P_{23}LP_{23} = \begin{pmatrix} 1 & 0 & 0 \\ l_{31} & 1 & 0 \\ l_{21} & 0 & 1 \end{pmatrix}$$

Since $I = P_{23}^2$ and $P_{23}^{-1} = P_{23}$, we have

$$\begin{aligned} A &= P_{23}^2A = L_1L_2P_{23}L_3U \\ P_{23}A &= (P_{23}L_1L_2P_{23})L_3U \\ P_{23}A &= LU \end{aligned}$$

where

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & 0.04 & 1 \end{pmatrix}$$

LU Factorization Example

Example Reviewed: Multiplying by the ***permutation matrix***

$$P_{23}LP_{23} = \begin{pmatrix} 1 & 0 & 0 \\ l_{31} & 1 & 0 \\ l_{21} & 0 & 1 \end{pmatrix}$$

Since $I = P_{23}^2$ and $P_{23}^{-1} = P_{23}$, we have

$$\begin{aligned} A = P_{23}^2 A &= L_1 L_2 P_{23} L_3 U \\ P_{23} A &= (P_{23} L_1 L_2 P_{23}) L_3 U \\ P_{23} A &= LU \end{aligned}$$

where

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & 0.04 & 1 \end{pmatrix}$$

General LU Factorization

The previous **Example** was a 3×3 matrix, so how does this generalize for solving, $Ax = b$, with A $n \times n$?

The process, described in the **algorithm** earlier, can be accomplished with matrices as described above in the LU factorization:

$$PA = LU$$

1. Examine the **diagonal elements**, $k = 1..n$, successively
2. Find the **largest element in magnitude** below each of these diagonal elements and perform a **pivoting**
3. Use the **diagonal element** to **pivot** and **eliminate** all other elements below this diagonal element
4. Repeat the process until $k = n$

General LU Factorization

In ***LU* Factorization** from a **matrix** perspective we seek

$$PA = LU, \qquad \text{with} \quad P = P_{n-1}P_{n-2} \cdot ... \cdot P_2 P_1,$$

where P_k switches the k^{th} row with some row beneath it, selecting the largest element in the k^{th} column below in the transformed matrix

Recall that $P_k^{-1} = P_k$

Also, created **elimination matrices**, M_k, which perform row operations to eliminate elements in the k^{th} column below the diagonal element

The matrix M_k has ones on the diagonal, and subdiagonal elements are ≤ 1

Need to build a sequence of matrices P_k and M_k such that

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2} \cdot ... \cdot M_1 P_1 A = U,$$

where U is an ***upper diagonal matrix***

General LU Factorization

We need to create the appropriate ***lower diagonal matrix***, L, from our equation

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2} \cdot ... \cdot M_1 P_1 A = U.$$

Define matrices M'_k as follows:

$$\begin{aligned} M'_{n-1} &= M_{n-1} \\ M'_{n-2} &= P_{n-1}M_{n-2}P_{n-1}^{-1} \\ M'_{n-3} &= P_{n-1}P_{n-2}M_{n-3}P_{n-2}^{-1}P_{n-1}^{-1} \\ ... &= ... \\ M'_k &= P_{n-1} \cdots P_{k+1} M_k P_{k+1}^{-1} \cdots P_{n-1}^{-1}, \end{aligned}$$

where each M'_k has the same structure as M_k with the subdiagonal permuted

Minimal work shows

$$M_{n-1}P_{n-1} \cdots M_1 P_1 = M'_{n-1} \cdots M'_1 \cdot P_{n-1} \cdots P_1$$

General LU Factorization

Thus,

$$\begin{aligned} M_{n-1}P_{n-1}M_{n-2}P_{n-2}\cdot \ldots \cdot M_1P_1A &= U \\ (M'_{n-1}\cdots M'_1)\cdot(P_{n-1}\cdots P_1)A &= U \\ PA &= LU, \end{aligned}$$

where

$$P = P_{n-1}\cdots P_1 \qquad \text{and} \qquad L = (M'_{n-1}\cdots M'_1)^{-1}$$

Since each M'_k is a unit ***lower diagonal matrix***, then the product $M'_{n-1}\cdots M'_1$ forms a unit ***lower diagonal matrix***, which by choice has all subdiagonal elements ≤ 1

The inverse $L = (M'_{n-1}\cdots M'_1)^{-1}$ is easily found by simply negating the subdiagonal entries, completing our **General LU Factorization**

MatLab Program for LU Factorization

Program by Cleve Moler for **LU Factorization** of a matrix A, which starts by finding the size of A

```
function [L,U,p] = lutx(A)
%LUTX  Triangular factorization, textbook version
%   [L,U,p] = lutx(A) produces a unit lower ...
    triangular matrix L,
%   an upper triangular matrix U, and a ...
    permutation vector p,
%   so that L*U = A(p,:)

%   Copyright 2014 Cleve Moler
%   Copyright 2014 The MathWorks, Inc.

[n,n] = size(A);
p = (1:n)';
```


MatLab Program for LU Factorization

Find the largest element for ***pivoting***

```
for k = 1:n-1

   % Find index of largest element below diagonal ...
       in k-th column
   [r,m] = max(abs(A(k:n,k)));
   m = m+k-1;

   % Skip elimination if column is zero
   if (A(m,k) ≠ 0)
```


MatLab Program for LU Factorization

Pivot about $\{a_{kk}\}$

```
      % Swap pivot row
      if (m ≠ k)
         A([k m],:) = A([m k],:);
         p([k m]) = p([m k]);
      end

      % Compute multipliers
      i = k+1:n;
      A(i,k) = A(i,k)/A(k,k);

      % Update the remainder of the matrix
      j = k+1:n;
      A(i,j) = A(i,j) - A(i,k)*A(k,j);
   end
end
```

MatLab Program for LU Factorization

Produce the output matrices, L and U

```
% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
```

Most of the time of execution is performed on the line
`A(i,j)= A(i,j)- A(i,k)*A(k,j);`

At the k^{th} step, matrix multiplications are performed to create zeros below the diagonal and an $(n-k)\times(n-k)$ submatrix in the lower right corner

This would require a double nested loop for a non-vector computer language

MatLab Program for Back Substitution

Back Substitution completes the solution of $Ax = b$

First check for special matrix forms

```
function x = bslashtx(A,b)
% BSLASHTX  Solve linear system (backslash)
% x = bslashtx(A,b) solves A*x = b

[n,n] = size(A);
if isequal(triu(A,1),zeros(n,n))
   % Lower triangular
   x = forward(A,b);
   return
elseif isequal(tril(A,-1),zeros(n,n))
   % Upper triangular
   x = backsubs(A,b);
   return
```


MatLab Program for Back Substitution

Continue special matrix forms

```
elseif isequal(A,A')
   [R,fail] = chol(A);
   if ¬fail
      % Positive definite
      y = forward(R',b);
      x = backsubs(R,y);
      return
   end
end
```


MatLab Program for Back Substitution

Use the previous LU **Factorization** to perform **Back Substitution**

```
% Triangular factorization
[L,U,p] = lutx(A);

% Permutation and forward elimination
y = forward(L,b(p));

% Back substitution
x = backsubs(U,y);
```

The program first calls the LU **Factorization**, then calls on two other subroutines to use the permutation, then **Back Substitute**

MatLab Program for Back Substitution

The permutation is performed by the line
`y = forward(L,b(p));`
with the code

```
function x = forward(L,x)
% FORWARD. Forward elimination.
% For lower triangular L, x = forward(L,b) solves ...
    L*x = b.
[n,n] = size(L);
x(1) = x(1)/L(1,1);
for k = 2:n
   j = 1:k-1;
   x(k) = (x(k) - L(k,j)*x(j))/L(k,k);
end
```


MatLab Program for Back Substitution

The **Back Substitution** is called in the line
`x = backsubs(U,y);`

```
function x = backsubs(U,x)
% BACKSUBS.  Back substitution.
% For upper triangular U, x = backsubs(U,b) ...
    solves U*x = b.
[n,n] = size(U);
x(n) = x(n)/U(n,n);
for k = n-1:-1:1
   j = k+1:n;
   x(k) = (x(k) - U(k,j)*x(j))/U(k,k);
end
```

This gives the value of x and completes the solution of $Ax = b$