Chapter 10

LU Factorization

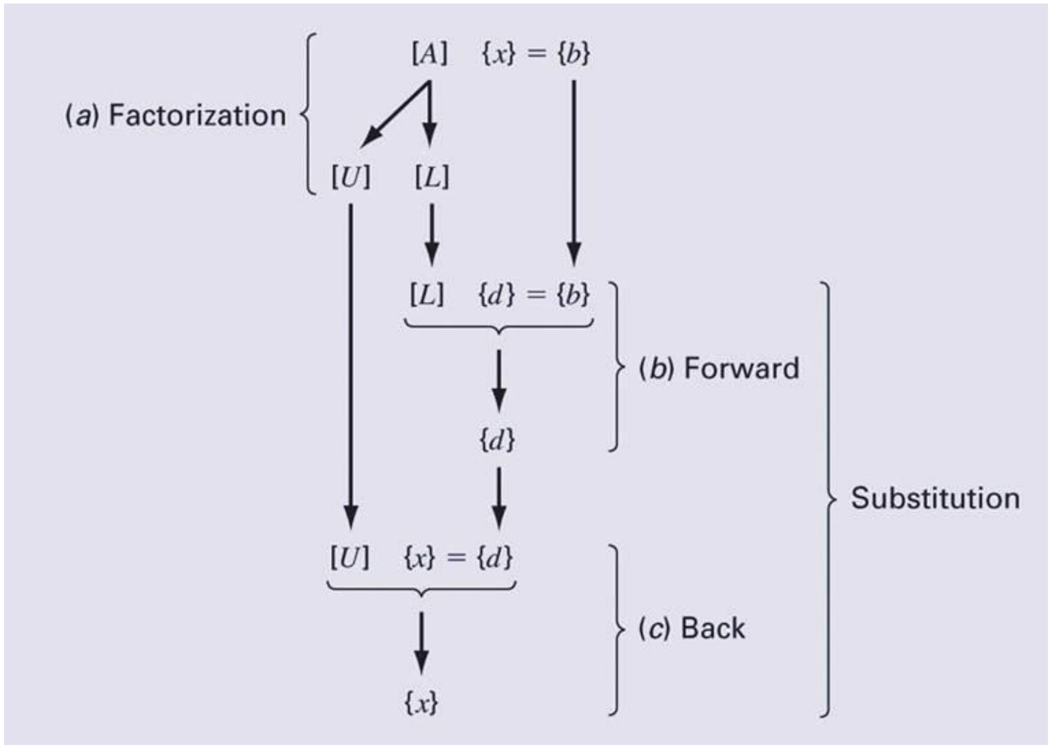
Numerical Methods
Fall 2019

LU Factorization (1)

- Recall that the forward–elimination step of Gauss elimination comprises the bulk of the computational effort.
- *LU* factorization methods separate the time–consuming elimination of the matrix [*A*] from the manipulations of the right–hand–side [*b*].
- Once [*A*] has been factored (or decomposed), multiple right–hand–side vectors can be evaluated in an efficient manner.

LU Factorization (2)

- *LU* factorization involves two steps:
  - Factorization to decompose the [*A*] matrix into a product of a lower triangular matrix [*L*] and an upper triangular matrix [*U*]. [*L*] has 1 for each entry on the diagonal.
  - Substitution to solve for {*x*}
- Gauss elimination can be implemented using *LU* factorization



Gauss *LU* Factorization (3)

$$[A]\{x\} - \{b\} = 0$$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix}$$

$$[U]\{x\} - \{d\} = 0$$

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}$$

Gauss Elimination as *LU* Factorization (1)

- [*A*]{*x*}={*b*} can be rewritten as [*L*][*U*]{*x*}={*b*} using *LU* factorization.
- The *LU* factorization algorithm requires the same total flops as for Gauss elimination.
- The main advantage is once [*A*] is decomposed, the same [*L*] and [*U*] can be used for multiple {*b*} vectors.
- MATLAB's lu() function can be used to generate the [*L*] and [*U*] matrices:

  [*L*, *U*] = lu(*A*)

Gauss Elimination as *LU* Factorization (2)

- To solve [*A*]{*x*}={*b*}, first decompose [*A*] to get [*L*][*U*]{*x*}={*b*}
- Set up and solve [*L*]{*d*}={*b*}, where {*d*} can be found using *forward* substitution.
- Set up and solve [*U*]{*x*}={*d*}, where {*x*} can be found using *backward* substitution.
- In MATLAB:

```
>> [L, U] = lu(A)
>> d = L\b
>> x = U\d
```

Gauss Elimination as *LU* Factorization (3)

- Upper triangular matrix [U]

$$[U] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix}$$

Ax=b $\Rightarrow$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

The first step in Gauss elimination is to multiply row 1 by the factor [recall Eq. (9.9)]

$$f_{21} = \frac{a_{21}}{a_{11}}$$

and subtract the result from the second row to eliminate a_{21}. Similarly, row 1 is multiplied by

$$f_{31} = \frac{a_{31}}{a_{11}}$$

and the result subtracted from the third row to eliminate a_{31}. The final step is to multiply the modified second row by

$$f_{32} = \frac{a'_{32}}{a'_{22}}$$

Gauss Elimination as *LU* Factorization (4)

- [*A*] matrix can therefore be written as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ f_{21} & a'_{22} & a'_{23} \\ f_{31} & f_{32} & a''_{33} \end{bmatrix}$$

$$[A] \to [L][U]$$

where

$$[U] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix}$$

and

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ f_{21} & 1 & 0 \\ f_{31} & f_{32} & 1 \end{bmatrix}$$

Example 10.1, 10.2

LU Factorization MATLAB

```
>> A = [3 -.1 -.2;.1 7 -.3;.3 -.2 10];
>> b = [7.85; -19.3; 71.4];
```

Next, the *LU* factorization can be computed with

```
>> [L,U] = lu(A)

L =
    1.0000         0         0
    0.0333    1.0000         0
    0.1000   -0.0271    1.0000

U =
    3.0000   -0.1000   -0.2000
         0    7.0033   -0.2933
         0         0   10.0120
```

```
>> d = L\b

d =
     7.8500
   -19.5617
    70.0843
```

And then use this result to compute the solution

```
>> x = U\d

x =

     3.0000
    -2.5000
     7.0000
```

Cholesky Factorization

- Symmetric matrix occurs commonly in both mathematical and engineering/science problems, and there are special solution techniques available for such systems.

$$[A] = [A]^T$$

- The *Cholesky factorization* is one of the most popular of these techniques, and is based on the fact that a symmetric matrix can be decomposed as $[A] = [U]^T[U]$, where T stands for transpose.
- The rest of the process is similar to *LU* decomposition and Gauss elimination, except only one matrix, $[U]$, needs to be stored.

Cholesky Factorization

- Solution

  $[A] = [U]^T[U]$

  $[U]^T \{d\} = \{b\}$

  $[U] \{x\} = \{d\}$

- The factorization can be generated efficiently by recurrence relations. For the *i*–th row:

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2}$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj}}{u_{ii}} \qquad \text{for } j = i+1, \ldots, n$$

➡ Example 10.5

Cholesky Factorization MATLAB

- MATLAB can perform a Cholesky factorization with the built-in `chol` command:

```
>> U = chol(A)
```

- MATLAB's left division operator \ examines the system to see which method will most efficiently solve the problem.
- This includes trying banded solvers, back and forward substitutions, Cholesky factorization for symmetric systems. If these do not work and the system is square, Gauss elimination with partial pivoting is used.

Example 10.6

Cholesky Factorization MATLAB

- Example 10.6

$$[A] = \begin{bmatrix} 6 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix}$$

```
>> A = [6 15 55; 15 55 225; 55 225 979];
>> b = [sum(A(1,:)); sum(A(2,:)); sum(A(3,:))]

   b =
            76
           295
          1259

>> U = chol(A)

U =
    2.4495    6.1237   22.4537
         0    4.1833   20.9165
         0         0    6.1101
```

Cholesky Factorization MATLAB

To generate the solution, we first compute

```
>> d = U'\b

d =
   31.0269
   25.0998
    6.1101
```

And then use this result to compute the solution

```
>> x = U\d

x =
    1.0000
    1.0000
    1.0000
```

Chapter 11

Matrix Inverse and Condition

Numerical Methods
Fall 2019

Matrix Inverse (1)

- Recall that if a matrix [*A*] is square, there is another matrix $[A]^{-1}$, called the inverse of [*A*], for which $[A][A]^{-1}=[A]^{-1}[A]=[I]$
- The inverse can be computed in a column by column fashion by generating solutions with unit vectors as the right-hand-side constants:

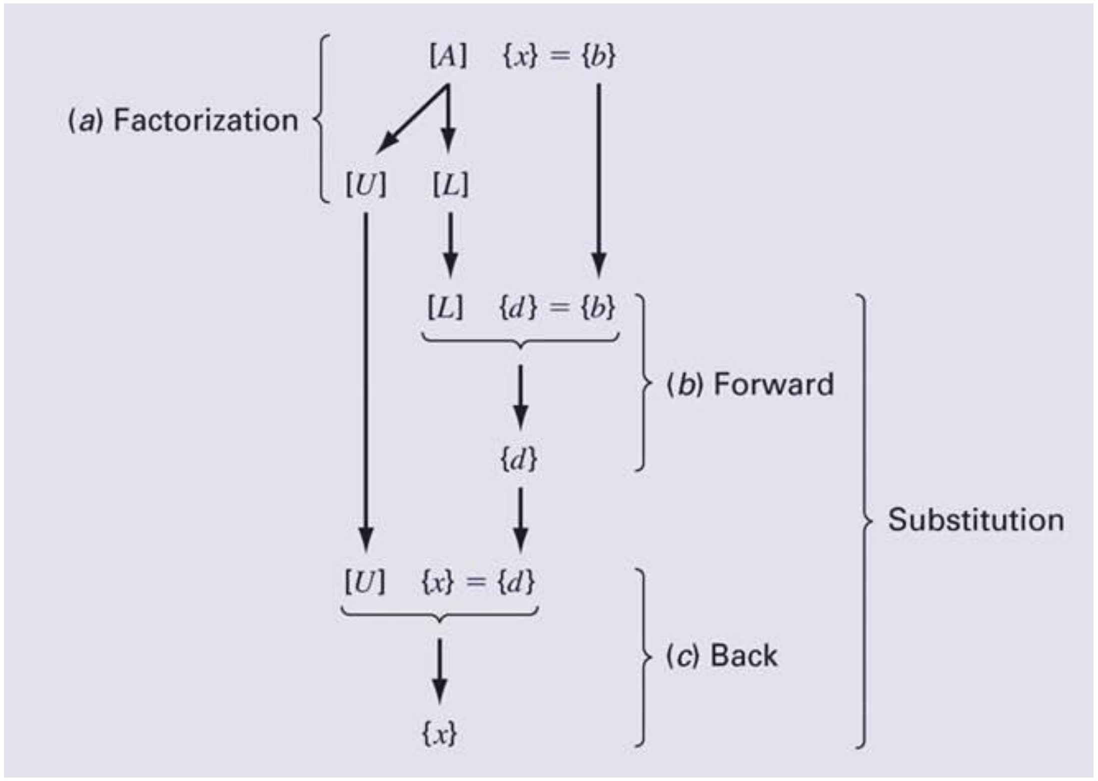
$$[A]\{x_1\} = \begin{Bmatrix}1\\0\\0\end{Bmatrix} \quad [A]\{x_2\} = \begin{Bmatrix}0\\1\\0\end{Bmatrix} \quad [A]\{x_3\} = \begin{Bmatrix}0\\0\\1\end{Bmatrix}$$

$$[A]^{-1} = [x_1 \quad x_2 \quad x_3]$$

Example 11.1

Matrix Inverse, 2

- Recall that *LU* factorization can be used to efficiently evaluate a system for multiple right-hand-side vectors—thus, it is ideal for evaluating the multiple unit vectors needed to compute the inverse.



Stimulus–Response Computations

- Many systems can be modeled as a linear combination of equations, and thus written as a matrix equation:

$$[\text{Interactions}]\{\text{response}\} = \{\text{stimuli}\}$$

- The system response can thus be found using the matrix inverse.
- Each element of the inverse tells you the response of the *i* th unknown to a unit change of the *j* th forcing function. That is,

$\boldsymbol{a_{ij}^{-1}}$ = **the change of** $\boldsymbol{x_i}$ **due to a unit change of** $\boldsymbol{b_j}$

element of A^{-1}

Vector and Matrix Norms

- A *norm* is a real-valued function that provides a measure of the size or “length” of multi-component mathematical entities such as vectors and matrices.
- Vector norms and matrix norms may be computed differently.

Vector Norms

- For a vector $\{X\}$ of size *n*, the *p*-norm is:

$$\|X\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$

- Important examples of vector *p*-norms include:

$p = 1$: sum of the absolute values $\qquad \|X\|_1 = \sum_{i=1}^{n} |x_i|$

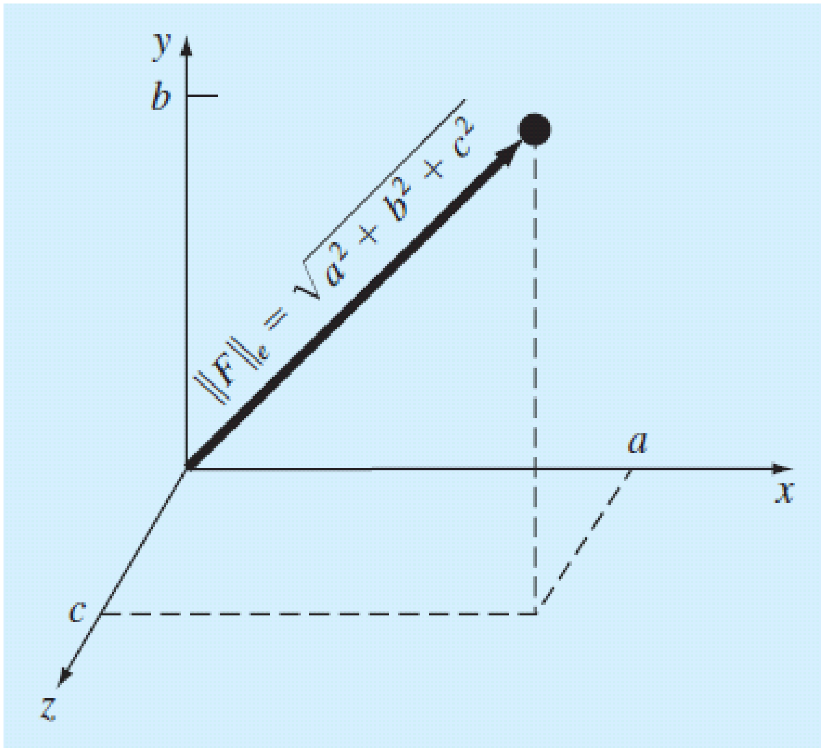
$p = 2$: Euclidian norm (length) $\qquad \|X\|_2 = \|X\|_e = \sqrt{\sum_{i=1}^{n} x_i^2}$

$p = \infty$: maximum $-$ magnitude $\qquad \|X\|_\infty = \max_{1 \le i \le n} |x_i|$

Vector Norms

- {F} = (a,b,c)

$$\|F\|_e = \sqrt{a^2 + b^2 + c^2}$$



Matrix Norms

- Common matrix norms for a matrix [*A*] include:

column - sum norm $$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$$

Frobenius norm $$\|A\|_f = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2}$$

row - sum norm $$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$$

spectral norm (2 norm) $$\|A\|_2 = (\mu_{\max})^{1/2}$$

- Note – $\mu_{\max}$ is the largest eigenvalue of $[A]^T[A]$.

MATLAB Commands

- MATLAB has built-in functions to compute both norms and condition numbers:
  - **`norm(`*`X`*`,`*`p`*`)`**
    - Compute the *p* norm of vector *X*, where *p* can be any number, `inf`, or `‘fro’` (for the Euclidean norm)
  - **`norm(A,p)`**
    - Compute a norm of matrix *A*, where *p* can be 1, 2, `inf`, or `‘fro’` (for the Frobenius norm)