Chapter 6: LINEAR SYSTEMS

FuSen F. Lin

Department of Computer Science & Engineering
National Taiwan Ocean University

Scientific Computing, Fall 2010

Linear Systems

Linear Systems

Linear Systems

Linear Systems

Linear Systems

Introduction

- The linear equation problem involves finding a vector $x \in \mathbb{R}^n$ so that $AX = b$, where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$ are nonsingular.
- This problem is at the heart of many problems in scientific computation. For example,
  - The vector of coefficients that define a polynomial interpolant is a solution to a linear system.
  - [illegible] system.
  - Most numerical techniques in optimization and differential equations involve repeated linear equation solving.
- It is extremely important to know how to solve this problem efficiently and accurately.

Introduction

- The linear equation problem involves finding a vector $x \in \mathbb{R}^n$ so that $AX = b$, where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are nonsingular.
- This problem is at the heart of many problems in scientific computation. For example,
  - The vector of coefficients that define a polynomial interpolant is a solution to a linear system.
  - The problem of spline interpolation led to a tridiagonal system.
  - Most numerical techniques in optimization and differential equations involve repeated linear equation solving.
- It is extremely important to know how to solve this problem efficiently and accurately.

Introduction

- The linear equation problem involves finding a vector $x \in \mathbb{R}^n$ so that $AX = b$, where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are nonsingular.
- This problem is at the heart of many problems in scientific computation. For example,
  - The vector of coefficients that define a polynomial interpolant is a solution to a linear system.
  - The problem of spline interpolation led to a tridiagonal system.
  - Most numerical techniques in optimization and differential equations involve repeated linear equation solving.
- It is extremely important to know how to solve this problem efficiently and accurately.

Introduction

- The linear equation problem involves finding a vector $x \in \mathbb{R}^n$ so that $AX = b$, where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are nonsingular.
- This problem is at the heart of many problems in scientific computation. For example,
  - The vector of coefficients that define a polynomial interpolant is a solution to a linear system.
  - The problem of spline interpolation led to a tridiagonal system.
  - Most numerical techniques in optimization and differential equations involve repeated linear equation solving.
- It is extremely important to know how to solve this problem efficiently and accurately.

Introduction

- The linear equation problem involves finding a vector $x \in \mathbb{R}^n$ so that $AX = b$, where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are nonsingular.
- This problem is at the heart of many problems in scientific computation. For example,
  - The vector of coefficients that define a polynomial interpolant is a solution to a linear system.
  - The problem of spline interpolation led to a tridiagonal system.
  - Most numerical techniques in optimization and differential equations involve repeated linear equation solving.
- It is extremely important to know how to solve this problem efficiently and accurately.

Introduction

- The linear equation problem involves finding a vector $x \in \mathbb{R}^n$ so that $AX = b$, where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are nonsingular.
- This problem is at the heart of many problems in scientific computation. For example,
  - The vector of coefficients that define a polynomial interpolant is a solution to a linear system.
  - The problem of spline interpolation led to a tridiagonal system.
  - Most numerical techniques in optimization and differential equations involve repeated linear equation solving.
- It is extremely important to know how to solve this problem efficiently and accurately.

main Topics

- It well-known that the process of Gaussian elimination is related to the factorization of matrix, $A = LU$, where L is lower triangular and U is upper triangular.
- We first discuss triangular, tridiagonal, and Hessenberg systems, and then arrive the general algorithm with/without pivoting.
- A discussion of permutation matrices and how they can be manipulated in MATLAB.
- Finally, we explore the issue of linear system sensitivity and identify the important role that the condition number plays.

main Topics

- It well-known that the process of Gaussian elimination is related to the factorization of matrix, $A = LU$, where L is lower triangular and U is upper triangular.
- We first discuss triangular, tridiagonal, and Hessenberg systems, and then arrive the general algorithm with/without pivoting.
- A discussion of permutation matrices and how they can be manipulated in MATLAB.
- Finally, we explore the issue of linear system sensitivity and identify the important role that the condition number plays.

main Topics

- It well-known that the process of Gaussian elimination is related to the factorization of matrix, $A = LU$, where L is lower triangular and U is upper triangular.
- We first discuss triangular, tridiagonal, and Hessenberg systems, and then arrive the general algorithm with/without pivoting.
- A discussion of permutation matrices and how they can be manipulated in MATLAB.
- Finally, we explore the issue of linear system sensitivity and identify the important role that the condition number plays.

main Topics

- It well-known that the process of Gaussian elimination is related to the factorization of matrix, $A = LU$, where L is lower triangular and U is upper triangular.
- We first discuss triangular, tridiagonal, and Hessenberg systems, and then arrive the general algorithm with/without pivoting.
- A discussion of permutation matrices and how they can be manipulated in MATLAB.
- Finally, we explore the issue of linear system sensitivity and identify the important role that the condition number plays.

Triangular Problems

- We first consider the linear system $AX = b$, where the coefficient matrix A is triangular.
- For example, a 3-by-3 lower triangular case,

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- The unknowns can easily be determined as follows:

$$\begin{aligned} x_1 &= b_1/l_{11} \\ x_2 &= (b_2 - l_{21}x_1)/l_{22} \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33} \end{aligned}$$

- This algorithm is called **forward substitution**. Notice that the process requires $\det(A) \neq 0$.

Triangular Problems

- We first consider the linear system $AX = b$, where the coefficient matrix A is triangular.
- For example, a 3-by-3 lower triangular case,

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- The unknowns can easily be determined as follows:

$$\begin{aligned} x_1 &= b_1/l_{11} \\ x_2 &= (b_2 - l_{21}x_1)/l_{22} \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33} \end{aligned}$$

- This algorithm is called **forward substitution**. Notice that the process requires $\det(A) \neq 0$.

Triangular Problems

- We first consider the linear system $AX = b$, where the coefficient matrix A is triangular.
- For example, a 3-by-3 lower triangular case,

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- The unknowns can easily be determined as follows:

$$\begin{aligned} x_1 &= b_1/l_{11} \\ x_2 &= (b_2 - l_{21}x_1)/l_{22} \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33} \end{aligned}$$

- This algorithm is called **forward substitution**. Notice that the process requires $\det(A) \neq 0$.

Triangular Problems

- We first consider the linear system $AX = b$, where the coefficient matrix A is triangular.
- For example, a 3-by-3 lower triangular case,

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- The unknowns can easily be determined as follows:

$$\begin{aligned} x_1 &= b_1/l_{11} \\ x_2 &= (b_2 - l_{21}x_1)/l_{22} \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33} \end{aligned}$$

- This algorithm is called **forward substitution**. Notice that the process requires $\det(A) \neq 0$.

Forward Substitution (1)

- In general, if the linear system $LX = b$, where L is lower triangular, then the solution x_i can be obtained by solving the ith equation

$$l_{i1}x_1 + \cdots + l_{ii}x_i = b_i$$

$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j \right) / l_{ii} \quad \text{for} \quad i = 1 : n.$$

- This is evaluated for $i = 1 : n$, then we have the codes of scalar-level.
- Note that the j-loop effectively subtracts the inner product

$$\sum_{j=1}^{i-1} l_{ij}x_j = L(i, 1 : i-1) * x(1 : i-1)$$

from b_i, so we have the codes of vector-level.

Forward Substitution (1)

- In general, if the linear system $LX = b$, where L is lower triangular, then the solution x_i can be obtained by solving the ith equation
$$l_{i1}x_1 + \cdots + l_{ii}x_i = b_i$$
$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j \right) / l_{ii} \quad \text{for} \;\; i = 1 : n.$$
- This is evaluated for $i = 1 : n$, then we have the codes of scalar-level.
- Note that the j-loop effectively subtracts the inner product
$$\sum_{j=1}^{i-1} l_{ij}x_j = L(i, 1 : i-1) * x(1 : i-1)$$
from b_i, so we have the codes of vector-level.

Forward Substitution (1)

- In general, if the linear system $LX = b$, where L is lower triangular, then the solution x_i can be obtained by solving the ith equation
$$l_{i1}x_1 + \cdots + l_{ii}x_i = b_i$$
$$x_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j \right) / l_{ii} \quad \text{for} \;\; i = 1 : n.$$
- This is evaluated for $i = 1 : n$, then we have the codes of scalar-level.
- Note that the j-loop effectively subtracts the inner product
$$\sum_{j=1}^{i-1} l_{ij}x_j = L(i, 1 : i-1) * x(1 : i-1)$$
from b_i, so we have the codes of vector-level.

Forward Substitution

```
  % scalar-level: double-loop

 for i=1:n,
    x(i) = b(i);
    for j=1:i-1,
       x(i) =  x(i) - L(i,j)*x(j);
    end
    x(i) = x(i)/L(i,i);
 end

% vector-level
 x(1) = b(1)/L(1,1);
 for i = 2:n,
    x(i) = (b(i)-L(i,1:i-1)*x(1:i-1)) / L(i,i);
 end
```

Forward Substitution (2)

- Since the computation of x_i involves about $2i$ flops, the entire process requires about $2(1 + 2 + \cdots + n) \approx n^2$ flops.
- The forward substitution algorithm that we have derived is row oriented (vector-level). At each stage an inner product must be computed that involves part of a row of L and the previously computed portion of x.
- A column-oriented version that features the saxpy operation can be also obtained. Consider the $n = 3$ case once again. Once x_1 is resolved, the equation 1 can be removed and it leaves us a reduced 2-by-2 lower triangular system.

Forward Substitution (2)

- Since the computation of x_i involves about $2i$ flops, the entire process requires about $2(1+2+\cdots+n) \approx n^2$ flops.
- The forward substitution algorithm that we have derived is row oriented (vector-level). At each stage an inner product must be computed that involves part of a row of L and the previously computed portion of x.
- A column-oriented version that features the saxpy operation can be also obtained. Consider the $n = 3$ case once again. Once x_1 is resolved, the equation 1 can be removed and it leaves us a reduced 2-by-2 lower triangular system.

Forward Substitution (2)

- Since the computation of x_i involves about $2i$ flops, the entire process requires about $2(1 + 2 + \cdots + n) \approx n^2$ flops.
- The forward substitution algorithm that we have derived is row oriented (vector-level). At each stage an inner product must be computed that involves part of a row of L and the previously computed portion of x.
- A column-oriented version that features the saxpy operation can be also obtained. Consider the $n = 3$ case once again. Once x_1 is resolved, the equation 1 can be removed and it leaves us a reduced 2-by-2 lower triangular system.

Forward Substitution (3)

- For example, to solve

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 5 & 0 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix},$$

  we find that $x_1 = 6/2 = 3$ and then deal with the 2-by-2 system

$$\begin{bmatrix} 5 & 0 \\ 8 & 9 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} - 3 \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} -1 \\ -16 \end{bmatrix}$$

- This implies that $x_2 = -1/5$. The system reduced to

$$8x_3 = -16 - 9(-1/5)$$

  from which we conclude that $x_3 = -71/40$.

Forward Substitution (3)

- For example, to solve

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 5 & 0 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix},$$

we find that $x_1 = 6/2 = 3$ and then deal with the 2-by-2 system

$$\begin{bmatrix} 5 & 0 \\ 8 & 9 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} - 3 \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} -1 \\ -16 \end{bmatrix}$$

- This implies that $x_2 = -1/5$. The system reduced to

$$8x_3 = -16 - 9(-1/5)$$

from which we conclude that $x_3 = -71/40$.

Forward Substitution (4)

- In general, at the jth step we solve for x_j and then remove it from equations $j+1$ through n. At the start, $x_1 = b_1/l_{11}$ and equations 2 through n transform to

$$\begin{bmatrix} l_{22} & 0 & \cdots & 0 \\ l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_2 - x_1 l_{21} \\ b_3 - x_1 l_{31} \\ \vdots \\ b_n - x_1 l_{n1} \end{bmatrix} = b(2:n) - x_1 L(2:n,1).$$

- In general the jth step computes $x_j = b_j/l_{jj}$ and then performs the saxpy update

$$b(j+1:n) \longleftarrow b(j+1:n) - x_j L(j+1:n,j).$$

Thus, we obtain the *function* **LTriSol**.

- The upper triangular case is analogous. The only difference is that the unknowns are resolved in reverse order. See the *function* **UTriSol**. This algorithm is called *backward substitution*.

Forward Substitution (4)

- In general, at the jth step we solve for x_j and then remove it from equations $j+1$ through n. At the start, $x_1 = b_1/l_{11}$ and equations 2 through n transform to

$$\begin{bmatrix} l_{22} & 0 & \cdots & 0 \\ l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_2 - x_1 l_{21} \\ b_3 - x_1 l_{31} \\ \vdots \\ b_n - x_1 l_{n1} \end{bmatrix} = b(2:n) - x_1 L(2:n,1).$$

- In general the jth step computes $x_j = b_j/l_{jj}$ and then performs the saxpy update

$$b(j+1:n) \longleftarrow b(j+1:n) - x_j L(j+1:n,j).$$

Thus, we obtain the *function* **LTriSol**.

- The upper triangular case is analogous. The only difference is that the unknowns are resolved in reverse order. See the *function* **UTriSol**. This algorithm is called *backward substitution*.

Forward Substitution (4)

- In general, at the jth step we solve for x_j and then remove it from equations $j+1$ through n. At the start, $x_1 = b_1/l_{11}$ and equations 2 through n transform to

$$\begin{bmatrix} l_{22} & 0 & \cdots & 0 \\ l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_2 - x_1 l_{21} \\ b_3 - x_1 l_{31} \\ \vdots \\ b_n - x_1 l_{n1} \end{bmatrix} = b(2:n) - x_1 L(2:n,1).$$

- In general the jth step computes $x_j = b_j/l_{jj}$ and then performs the saxpy update

$$b(j+1:n) \longleftarrow b(j+1:n) - x_j L(j+1:n,j).$$

Thus, we obtain the *function* **LTriSol**.

- The upper triangular case is analogous. The only difference is that the unknowns are resolved in reverse order. See the *function* **UTriSol**. This algorithm is called *backward substitution*.

Banded and Hessenberg Problems (1)

- Before we embark on the solvers for general linear systems, We first look at the two special cases, the Banded and Hessenberg problems, where the matrix of coefficients already have lots of zeros.
- In the spline interpolation problem of Sec.3.3, we have to solve the tridiagonal linear system, whose matrix of coefficients looks like this:

$$A = \begin{bmatrix} x & x & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & 0 \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

- Here, the x-notation represents a nonzero scalar in the matrix.

Banded and Hessenberg Problems (1)

- Before we embark on the solvers for general linear systems, We first look at the two special cases, the Banded and Hessenberg problems, where the matrix of coefficients already have lots of zeros.
- In the spline interpolation problem of Sec.3.3, we have to solve the tridiagonal linear system, whose matrix of coefficients looks like this:

$$A = \begin{bmatrix} x & x & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & 0 \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

- Here, the x-notation represents a nonzero scalar in the matrix.

Banded and Hessenberg Problems (1)

- Before we embark on the solvers for general linear systems, We first look at the two special cases, the Banded and Hessenberg problems, where the matrix of coefficients already have lots of zeros.
- In the spline interpolation problem of Sec.3.3, we have to solve the tridiagonal linear system, whose matrix of coefficients looks like this:

$$A = \begin{bmatrix} x & x & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & 0 \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

- Here, the x-notation represents a nonzero scalar in the matrix.

Banded and Hessenberg Problems (2)

- The second family of specialized problems is Hessenberg Problems. For example, an upper Hessenberg matrix has lower bandwith 1 looks like this:

$$A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

- Upper Hessenberg linear systems arise in many applications where the eigenvalues and eigenvectors of a matrix are required.
- For these two specially structured linear systems, we set out to show how the matrix of coefficients can be factored into a product $A = LU$, where L is lower triangular and U is upper triangular.

Banded and Hessenberg Problems (2)

- The second family of specialized problems is Hessenberg Problems. For example, an upper Hessenberg matrix has lower bandwith 1 looks like this:

$$A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

- Upper Hessenberg linear systems arise in many applications where the eigenvalues and eigenvectors of a matrix are required.
- For these two specially structured linear systems, we set out to show how the matrix of coefficients can be factored into a product $A = LU$, where L is lower triangular and U is upper triangular.

Banded and Hessenberg Problems (2)

- The second family of specialized problems is Hessenberg Problems. For example, an upper Hessenberg matrix has lower bandwith 1 looks like this:

$$A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

- Upper Hessenberg linear systems arise in many applications where the eigenvalues and eigenvectors of a matrix are required.
- For these two specially structured linear systems, we set out to show how the matrix of coefficients can be factored into a product $A = LU$, where L is lower triangular and U is upper triangular.

Banded and Hessenberg Problems (3)

- If such a factorization is available, then the solution ot $A\mathbf{x} = \mathbf{b}$ follows from a pair of triangular system solves:

$$\left\{ \begin{array}{l} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{array} \right\} \Longrightarrow A\mathbf{x} = (LU)\mathbf{x} = L(U\mathbf{b}) = L\mathbf{y} = \mathbf{b}$$

- In terms of the MATLAB functions **LTriSol** and **UTriSol**,

$$\begin{aligned} y &= \text{LTriSol}(L, b); \\ x &= \text{UTriSol}(U, y); \end{aligned}$$

  For tridiagonal and Hessenberg systems, the computation of L and U is easier to explain than for general matrices.

Banded and Hessenberg Problems (3)

- If such a factorization is available, then the solution ot $A\mathbf{x} = \mathbf{b}$ follows from a pair of triangular system solves:

$$\left\{ \begin{array}{c} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{array} \right\} \quad \Longrightarrow \quad A\mathbf{x} = (LU)\mathbf{x} = L(U\mathbf{b}) = L\mathbf{y} = \mathbf{b}$$

- In terms of the MATLAB functions **LTriSol** and **UTriSol**,

$$\begin{aligned} y &= \text{LTriSol}(L, b); \\ x &= \text{UTriSol}(U, y); \end{aligned}$$

For tridiagonal and Hessenberg systems, the computation of L and U is easier to explain than for general matrices.

Tridiagonal Systems (1)

- Consider the 4-by-4 tridiagonal linear system:

$$\begin{bmatrix} d_1 & f_1 & 0 & 0 \\ e_2 & d_2 & f_2 & 0 \\ 0 & e_3 & d_3 & f_3 \\ 0 & 0 & e_4 & d_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

- One way to derive the *LU* factorization for a tridiagonal *A* is to equate entries in the equation:

$$\begin{bmatrix} d_1 & f_1 & 0 & 0 \\ e_2 & d_2 & f_2 & 0 \\ 0 & e_3 & d_3 & f_3 \\ 0 & 0 & e_4 & d_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 \\ 0 & l_3 & 1 & 0 \\ 0 & 0 & l_4 & 1 \end{bmatrix} \begin{bmatrix} u_1 & f_1 & 0 & 0 \\ 0 & u_2 & f_2 & 0 \\ 0 & 0 & u_3 & f_3 \\ 0 & 0 & 0 & u_4 \end{bmatrix}$$

- Doing this, see p.216. In general, for $i = 2 : n$ we have

$$\begin{aligned} (i, i-1): \quad e_i &= l_i u_{i-1} &&\Rightarrow\quad l_i = e_i / u_{i-1} \\ (i, i): \quad d_i &= l_i f_{i-1} + u_i &&\Rightarrow\quad u_i = d_i - l_i f_{i-1} \end{aligned}$$

which leads to the following procedure: See the *function* **TriDiLU** in page 217.

Tridiagonal Systems (1)

- Consider the 4-by-4 tridiagonal linear system:

$$\begin{bmatrix} d_1 & f_1 & 0 & 0 \\ e_2 & d_2 & f_2 & 0 \\ 0 & e_3 & d_3 & f_3 \\ 0 & 0 & e_4 & d_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

- One way to derive the LU factorization for a tridiagonal A is to equate entries in the equation:

$$\begin{bmatrix} d_1 & f_1 & 0 & 0 \\ e_2 & d_2 & f_2 & 0 \\ 0 & e_3 & d_3 & f_3 \\ 0 & 0 & e_4 & d_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 \\ 0 & l_3 & 1 & 0 \\ 0 & 0 & l_4 & 1 \end{bmatrix} \begin{bmatrix} u_1 & f_1 & 0 & 0 \\ 0 & u_2 & f_2 & 0 \\ 0 & 0 & u_3 & f_3 \\ 0 & 0 & 0 & u_4 \end{bmatrix}$$

- Doing this, see p.216. In general, for $i = 2 : n$ we have

$$\begin{aligned} (i, i-1):\quad e_i &= l_i u_{i-1} &&\Rightarrow\quad l_i = e_i / u_{i-1} \\ (i, i):\quad d_i &= l_i f_{i-1} + u_i &&\Rightarrow\quad u_i = d_i - l_i f_{i-1} \end{aligned}$$

which leads to the following procedure: See the *function* **TriDiLU** in page 217.

Tridiagonal Systems (1)

- Consider the 4-by-4 tridiagonal linear system:

$$\begin{bmatrix} d_1 & f_1 & 0 & 0 \\ e_2 & d_2 & f_2 & 0 \\ 0 & e_3 & d_3 & f_3 \\ 0 & 0 & e_4 & d_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

- One way to derive the *LU* factorization for a tridiagonal *A* is to equate entries in the equation:

$$\begin{bmatrix} d_1 & f_1 & 0 & 0 \\ e_2 & d_2 & f_2 & 0 \\ 0 & e_3 & d_3 & f_3 \\ 0 & 0 & e_4 & d_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 \\ 0 & l_3 & 1 & 0 \\ 0 & 0 & l_4 & 1 \end{bmatrix} \begin{bmatrix} u_1 & f_1 & 0 & 0 \\ 0 & u_2 & f_2 & 0 \\ 0 & 0 & u_3 & f_3 \\ 0 & 0 & 0 & u_4 \end{bmatrix}$$

- Doing this, see p.216. In general, for $i = 2 : n$ we have

$$\begin{aligned} (i, i-1): \quad e_i &= l_i u_{i-1} && \Rightarrow \quad l_i = e_i / u_{i-1} \\ (i, i): \quad d_i &= l_i f_{i-1} + u_i && \Rightarrow \quad u_i = d_i - l_i f_{i-1} \end{aligned}$$

which leads to the following procedure: See the *function* **TriDiLU** in page 217.

Tridiagonal Systems (2)

- The above procedure requires $3n$ flops to carry out and is defined as long as $u_i (i = 1, ..., n-1)$ are nonzero.
- As mensioned previously, to solve we must solve

$$Ly = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 \\ 0 & l_3 & 1 & 0 \\ 0 & 0 & l_4 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \mathbf{b}$$

for **y**, and

$$U\mathbf{x} = \begin{bmatrix} u_1 & f_1 & 0 & 0 \\ 0 & u_2 & f_2 & 0 \\ 0 & 0 & u_3 & f_3 \\ 0 & 0 & 0 & u_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \mathbf{y}$$

for **x**.

- These bidiagonal systems can be solved very simply. See page 217 and the *functions* **LBiDiSol** and **UBiDiSol**.

Tridiagonal Systems (2)

- The above procedure requires $3n$ flops to carry out and is defined as long as $u_i(i = 1, ..., n-1)$ are nonzero.
- As mensioned previously, to solve we must solve

$$L\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 \\ 0 & l_3 & 1 & 0 \\ 0 & 0 & l_4 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \mathbf{b}$$

  for **y**, and

$$U\mathbf{x} = \begin{bmatrix} u_1 & f_1 & 0 & 0 \\ 0 & u_2 & f_2 & 0 \\ 0 & 0 & u_3 & f_3 \\ 0 & 0 & 0 & u_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \mathbf{y}$$

  for **x**.
- These bidiagonal systems can be solved very simply. See page 217 and the *functions* **LBiDiSol** and **UBiDiSol**.

Tridiagonal Systems (2)

- The above procedure requires $3n$ flops to carry out and is defined as long as $u_i (i = 1, ..., n-1)$ are nonzero.
- As mensioned previously, to solve we must solve

$$L\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_2 & 1 & 0 & 0 \\ 0 & l_3 & 1 & 0 \\ 0 & 0 & l_4 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \mathbf{b}$$

  for **y**, and

$$U\mathbf{x} = \begin{bmatrix} u_1 & f_1 & 0 & 0 \\ 0 & u_2 & f_2 & 0 \\ 0 & 0 & u_3 & f_3 \\ 0 & 0 & 0 & u_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \mathbf{y}$$

  for **x**.
- These bidiagonal systems can be solved very simply. See page 217 and the *functions* **LBiDiSol** and **UBiDiSol**.

Tridiagonal Systems (3)

- Summarizing the overall solution process, the script

$$\begin{aligned}[l, u] &= \mathrm{TriDiLU}(d, e, f);\\ y &= \mathrm{LBiDiSol}(l, b);\\ x &= \mathrm{UBiDiSol}(u, f, y);\end{aligned}$$

  solves the tridiagonal system $A\mathbf{x} = \mathbf{b}$, assuming that d, e, and f house the diagonal, subdiagonal, and superdiagonal of A.
- The amount of arithmetic required $3n$, $2n$, and $3n$ for each procedure above, respectively.

Tridiagonal Systems (3)

- Summarizing the overall solution process, the script

$$\begin{aligned}[l, u] &= \mathrm{TriDiLU}(d, e, f); \\ y &= \mathrm{LBiDiSol}(l, b); \\ x &= \mathrm{UBiDiSol}(u, f, y);\end{aligned}$$

  solves the tridiagonal system $A\mathbf{x} = \mathbf{b}$, assuming that d, e, and f house the diagonal, subdiagonal, and superdiagonal of A.

- The amount of arithmetic required $3n$, $2n$, and $3n$ for each procedure above, respectively.

Hessenberg Systems (1)

- We derived the algorithm for tridiagonal LU (factorization) by equating coefficients in $A = LU$. We could use the same strategy for Hessenberg LU. However, in anticipation of the general LU computation, we proceed in ”elimination terms”. See page 218 and **HessLU**.

$$A = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix} \begin{bmatrix} x \\ x \\ x \\ x \\ x \\ x \end{bmatrix} = \begin{bmatrix} x \\ x \\ x \\ x \\ x \\ x \end{bmatrix}$$