Chapter 8

Linear Algebraic Equations and Matrices

Numerical Methods

Fall 2019

Overview (1)

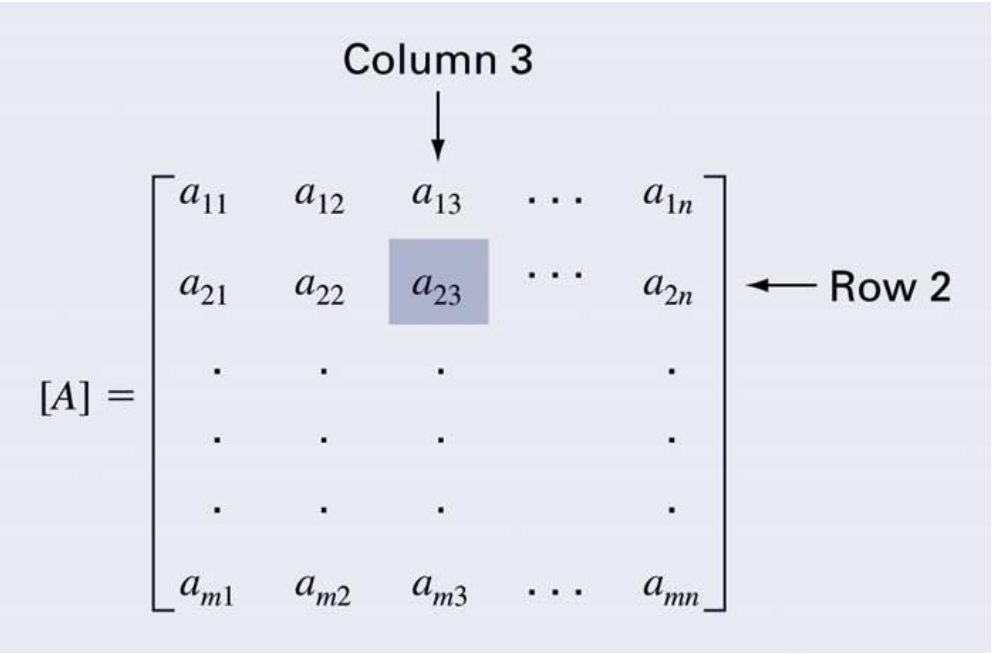
- A set of linear equations
  - k_1, k_2, k_3, mi, g are known
  - what are x_1, x_2, x_3?

$$
\begin{aligned}
(k_1 + k_2)x_1 \quad & - k_2 x_2 \quad & = m_1 g \\
-k_2 x_1 + (k_2 + k_3)x_2 \quad & - k_3 x_3 & = m_2 g \\
-k_3 x_2 + k_3 x_3 \quad & & = m_3 g
\end{aligned}
$$

  - This problem is to solving a system of three simultaneous equations for the three unknowns.
  - This equations can be expressed as Ax=b in matrix form.

Overview (2)

- A *matrix* consists of a rectangular array of elements represented by a single symbol (example: [*A*]).
- An individual entry of a matrix is an *element* (example: a_{23})



Overview (3)

- A horizontal set of elements is called a *row* and a vertical set of elements is called a *column*.
- The first subscript of an element indicates the row while the second indicates the column.
- The size of a matrix is given as *m* rows by *n* columns, or simply *m* by *n* (or *m* x *n*).
- 1 x *n* matrices are *row vectors*.
- *m* x 1 matrices are *column vectors.*

Special Matrices

- Matrices where $m = n$ are called *square matrices*.
- There are a number of special forms of square matrices:

Symmetric

$$[A] = \begin{bmatrix} 5 & 1 & 2 \\ 1 & 3 & 7 \\ 2 & 7 & 8 \end{bmatrix}$$

Diagonal

$$[A] = \begin{bmatrix} a_{11} & & \\ & a_{22} & \\ & & a_{33} \end{bmatrix}$$

Identity

$$[A] = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

Upper Triangular

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ & a_{22} & a_{23} \\ & & a_{33} \end{bmatrix}$$

Lower Triangular

$$[A] = \begin{bmatrix} a_{11} & & \\ a_{21} & a_{22} & \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Banded

$$[A] = \begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}$$

Matrix Operations

- Matrix addition and subtraction are performed by adding or subtracting the corresponding elements. This requires that the two matrices be the same size.

$$C = A + B \quad => \quad c_{ij} = a_{ij} + b_{ij}$$

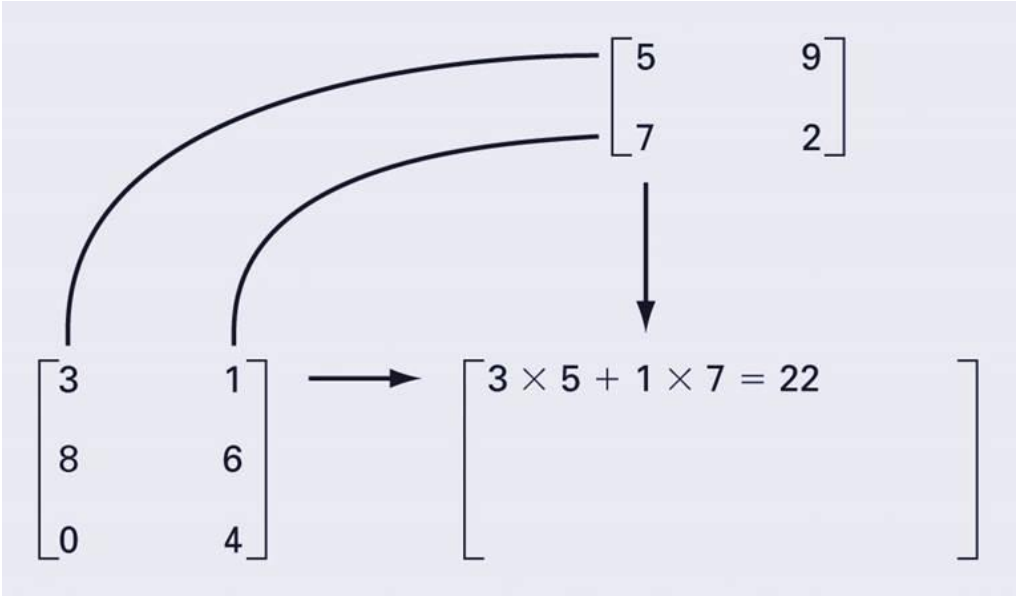
- Scalar matrix multiplication is performed by multiplying each element by the same scalar.

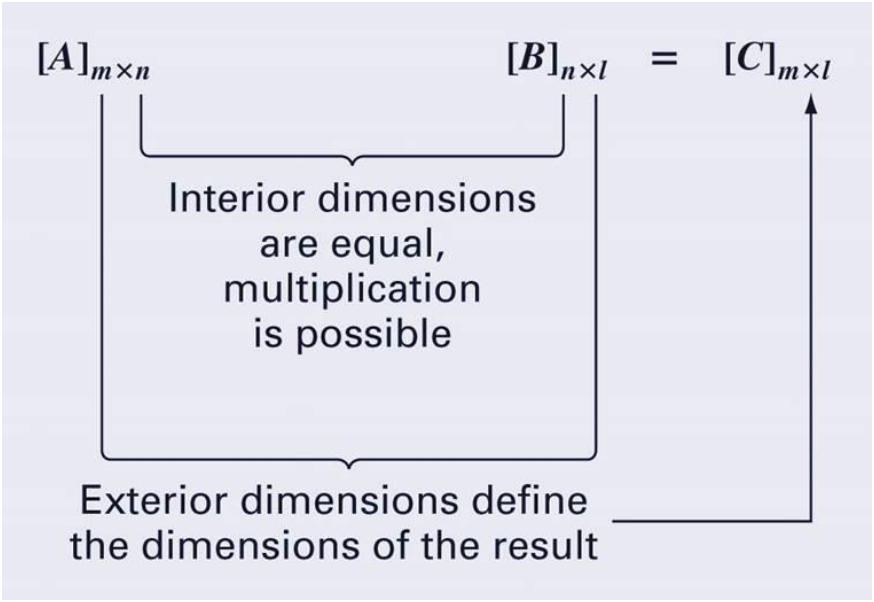
$$[D] = g[A] = \begin{bmatrix} ga_{11} & ga_{12} & ga_{13} \\ ga_{21} & ga_{22} & ga_{23} \\ ga_{31} & ga_{32} & ga_{33} \end{bmatrix}$$

Matrix Multiplication

- The elements in the matrix [C] that results from multiplying matrices [A] and [B] are calculated using:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$





Matrix Inverse and Transpose

- The *inverse* of a square, nonsingular matrix [*A*] is that matrix which, when multiplied by [*A*], yields the identity matrix.
  - $[A][A]^{-1}=[A]^{-1}[A]=[I]$
  - In Matlab: **`inv(A)`**
- The *transpose* of a matrix involves transforming its rows into columns and its columns into rows.
  - $(a_{ij})^{\mathrm{T}}=a_{ji}$
  - In Matlab: **`A'`**

MATLAB Matrix Manipulations

```
>> A = [1 5 6;7 4 2;-3 6 7]

A =
     1     5     6
     7     4     2
    -3     6     7

>> A'

ans =
     1     7    -3
     5     4     6
     6     2     7

>> x = [8 6 9];
>> y = [-5 8 1];
>> z = [4 8 2];
>> B = [x; y; z]

B =
     8     6     9
    -5     8     1
     4     8     2
```

```
>> C = A+B

C =
     9    11    15
     2    12     3
     1    14     9

>> A.*B

ans =
     8    30    54
   -35    32     2
   -12    48    14

>> AI = inv(A)

AI =
    0.2462    0.0154   -0.2154
   -0.8462    0.3846    0.6154
    0.8308   -0.3231   -0.4769
```

MATLAB Matrix Manipulations

```
>> I = eye(3)

I =

     1     0     0
     0     1     0
     0     0     1

>> Aug = [A I]

Aug =

     1     5     6     1     0     0
     7     4     2     0     1     0
    -3     6     7     0     0     1

>> [n,m] = size(Aug)

n =

     3

m =

     6
```

Representing Linear Algebra

- Matrices provide a concise notation for representing and solving simultaneous linear equations:

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

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

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

Solving With MATLAB

- MATLAB provides two direct ways to solve systems of linear algebraic equations [A]{x}={b}:
  - Left-division

    ```
    x = A\b
    ```

    A can be a rectangle matrix
  - Matrix inversion

    ```
    x = inv(A)*b
    ```

    When A is a square matrix
- The matrix inverse is less efficient than left-division and also only works for square, non-singular systems.

Solving With MATLAB

- Solve the following linear system of equation

$$\begin{bmatrix} 150 & -100 & 0 \\ -100 & 150 & -50 \\ 0 & -50 & 50 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 588.6 \\ 686.7 \\ 784.8 \end{Bmatrix}$$

```
>> K = [150 -100 0;-100 150 -50;0 -50 50]

K =
   150  -100     0
  -100   150   -50
     0   -50    50

>> mg = [588.6; 686.7; 784.8]

mg =
  588.6000
  686.7000
  784.8000
>> x = K\mg

x =
   41.2020
   55.9170
   71.6130
```

Chapter 9

Gauss Elimination

Numerical Methods
Fall 2019

Objectives

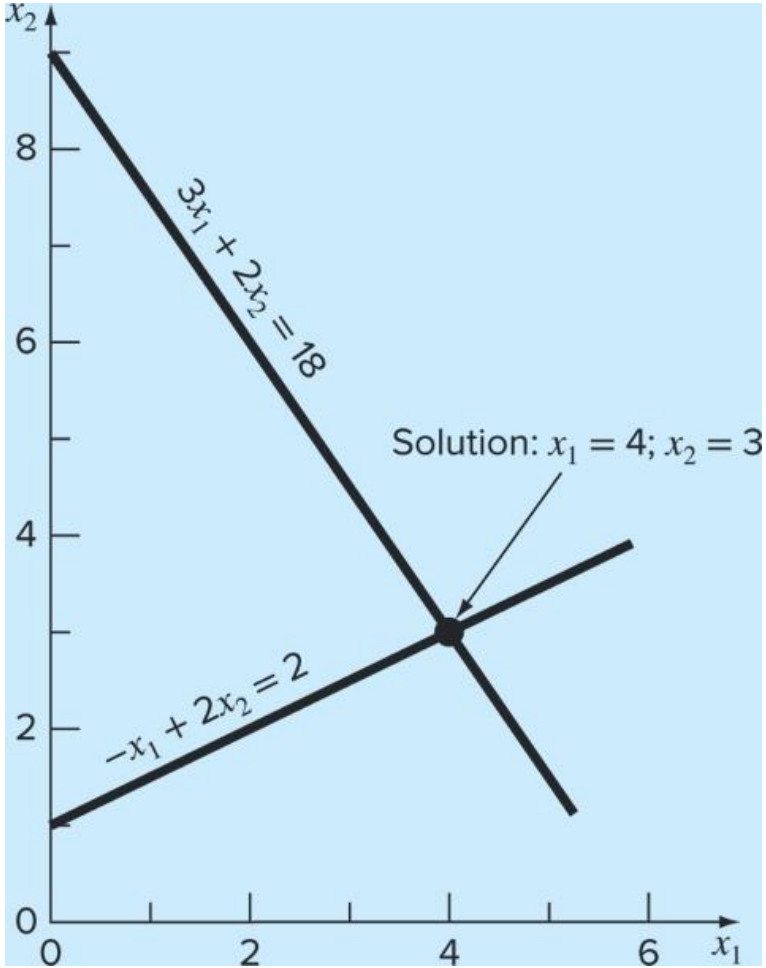
- Solving systems of linear algebraic equations using MATLAB

```
x = A\b
x = inv(A)*b
```

- Chapters 9 and 10 provide background on how such solutions are obtained.

Graphical Method (1)

- For small sets of simultaneous equations, graphing them and determining the location of the intercept provides a solution.



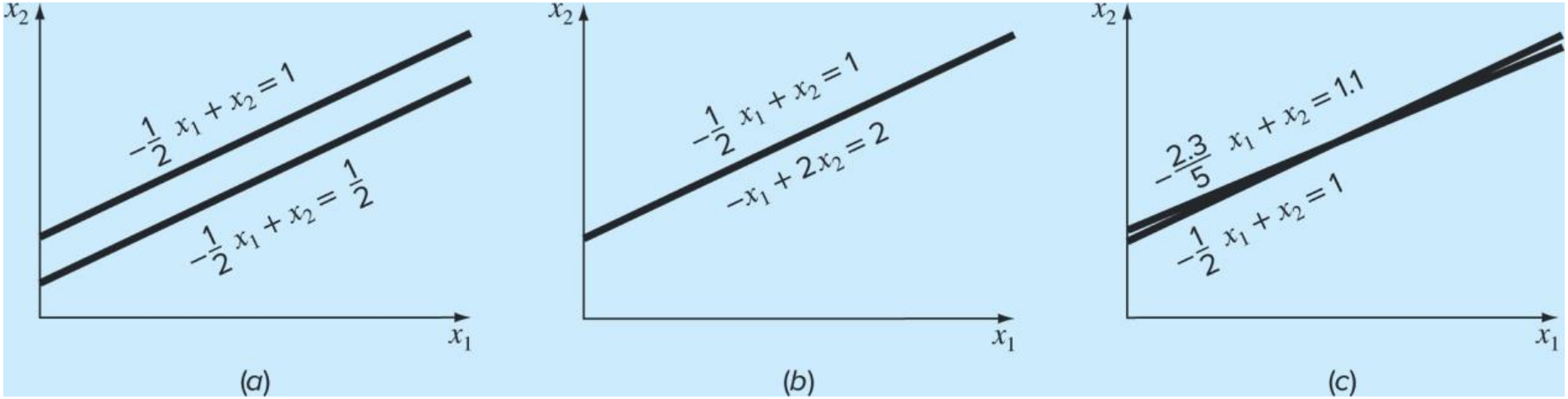
Graphical Method, 2

- Graphing the equations can also show systems where:

a) No solution exists

b) Infinite solutions exist

c) System is ill-conditioned



SINGULAR

ILL-CONDITIONED

Determinants

- The *determinant* $D=|A|$ of a matrix is formed from the coefficients of [A].
- Determinants for small matrices are:

1×1 $\quad |a_{11}| = a_{11}$

2×2 $\quad \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$

3×3 $\quad \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$

- Determinants for matrices larger than 3 x 3 can be very complicated.

Cramer's Rule

- *Cramer's Rule* states that each unknown in a system of linear algebraic equations may be expressed as a fraction of two determinants with denominator D and with the numerator obtained from D by replacing the column of coefficients of the unknown in question by the constants b_1, b_2, …, b_n.

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

$$D = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

determinant

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{D}$$

Cramer's Rule Example

- Find x_2 in the following system of equations:

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$
$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

- Find the determinant D

$$D = \begin{vmatrix} 0.3 & 0.52 & 1 \\ 0.5 & 1 & 1.9 \\ 0.1 & 0.3 & 0.5 \end{vmatrix} = 0.3 \begin{vmatrix} 1 & 1.9 \\ 0.3 & 0.5 \end{vmatrix} - 0.52 \begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} + 1 \begin{vmatrix} 0.5 & 1 \\ 0.1 & 0.4 \end{vmatrix}$$
$$= -0.0022$$

- Find determinant D_2 by replacing D's second column with b

$$D_2 = \begin{vmatrix} 0.3 & -0.01 & 1 \\ 0.5 & 0.67 & 1.9 \\ 0.1 & -0.44 & 0.5 \end{vmatrix} = 0.3 \begin{vmatrix} 0.67 & 1.9 \\ -0.44 & 0.5 \end{vmatrix} - 0.01 \begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} + 1 \begin{vmatrix} 0.5 & 0.67 \\ 0.1 & -0.44 \end{vmatrix}$$
$$= -0.0649$$

- Divide

$$x_2 = \frac{D_2}{D} = \frac{0.0649}{-0.0022} = -29.5$$

Cramer's Rule Example

```
>> A=[0.3 0.52 1;0.5 1 1.9;0.1 0.3 0.5];
>> D=det(A)

D =
   -0.0022
```

Cramer's rule can be applied to compute x_1 as in

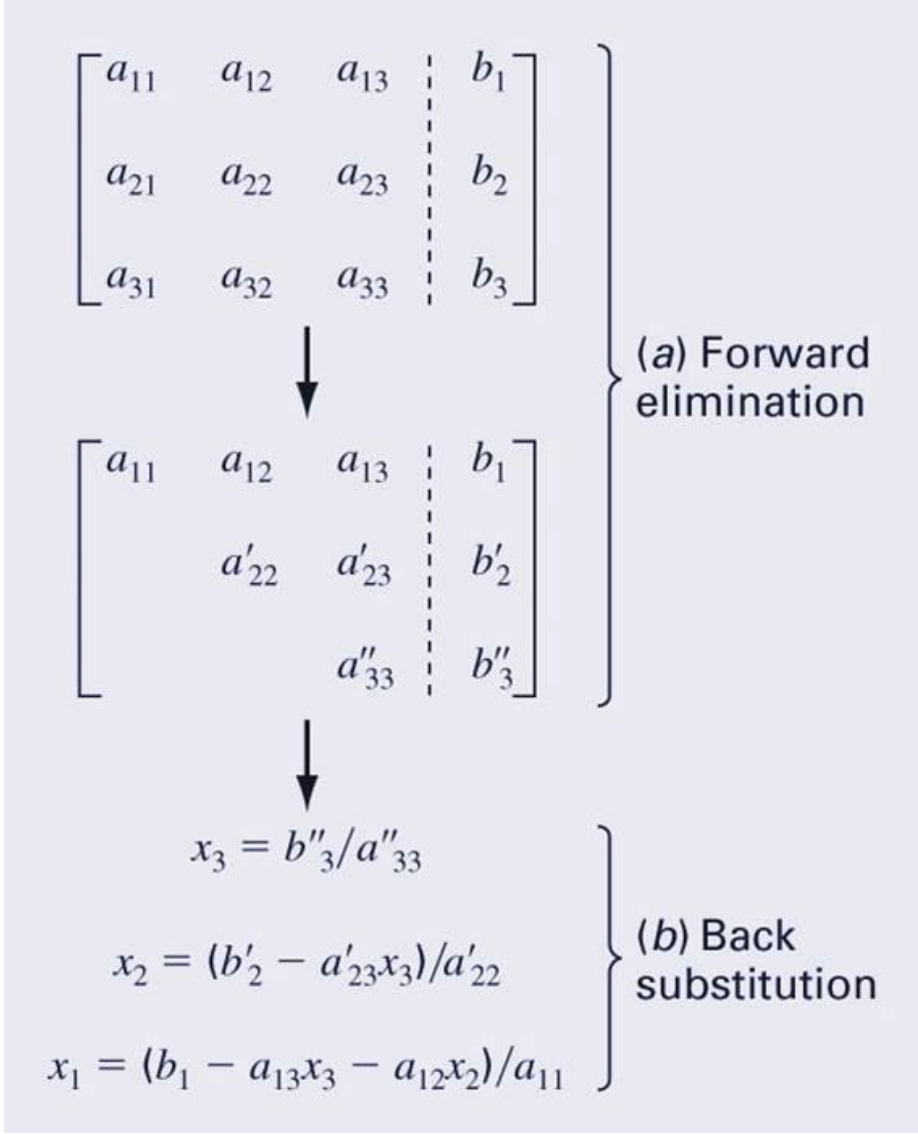
```
>> A(:,1)=[-0.01;0.67;-0.44]

A =
   -0.0100    0.5200    1.0000
    0.6700    1.0000    1.9000
   -0.4400    0.3000    0.5000

>> x1=det(A)/D

x1 =
  -14.9000
```

Naïve Gauss Elimination (1)

- For larger systems, Cramer's Rule can become unwieldy.
- Instead, a sequential process of removing unknowns from equations using *forward elimination* followed by *back substitution* may be used—this is Gauss elimination.
- "Naïve" Gauss elimination simply means the process does not check for potential problems resulting from division by zero.

Naïve Gauss Elimination (2)

- **Forward elimination**
  - Starting with the first row, add or subtract multiples of that row to eliminate the first coefficient from the second row and beyond.
  - Continue this process with the second row to remove the second coefficient from the third row and beyond.
  - Stop when an upper triangular matrix remains.
- **Back substitution**
  - Starting with the *last* row, solve for the unknown, then substitute that value into the next highest row.
  - Because of the upper-triangular nature of the matrix, each row will contain only one more unknown.



Naïve Gauss Elimination (3)

- to solve a general set of *n* equations

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (1)$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \quad (2)$$
$$\cdots\cdots$$
$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \quad (3)$$

multiply the above equation (1) by a_{21}/a_{11} to give

$$a_{21}x_1 + \frac{a_{21}}{a_{11}}a_{12}x_2 + \frac{a_{21}}{a_{11}}a_{13}x_3 + \cdots + \frac{a_{21}}{a_{11}}a_{1n}x_n = \frac{a_{21}}{a_{11}}b_1$$

This equation can be subtracted from Eq. (2) to give

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \cdots + \left(a_{2n} - \frac{a_{21}}{a_{11}}a_{1n}\right)x_n = b_2 - \frac{a_{21}}{a_{11}}b_1$$

$$a'_{22}x_2 + \cdots + a'_{2n}x_n = b'_2$$

$$a'_{32}x_2 + a'_{33}x_3 + \cdots + a'_{3n}x_n = b'_3$$

$$\vdots \qquad\qquad \vdots$$

$$a'_{n2}x_2 + a'_{n3}x_3 + \cdots + a'_{nn}x_n = b'_n$$

Naïve Gauss Elimination (4)

- The procedure can be continued using the remaining equations.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n &= b'_2 \\
a''_{33}x_3 + \cdots + a''_{3n}x_n &= b''_3 \\
\ddots \qquad\qquad &\vdots \\
a_{nn}^{(n-1)}x_n &= b_n^{(n-1)}
\end{aligned}
$$

- Back substitution

$$
x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}
$$

$$
x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}} \qquad \text{for } i = n-1, n-2, \ldots, 1
$$

Naïve Gauss Elimination Program

```
function x = GaussNaive(A,b)
% GaussNaive: naive Gauss elimination
%   x = GaussNaive(A,b): Gauss elimination without pivoting.
% input:
%   A = coefficient matrix
%   b = right hand side vector
% output:
%   x = solution vector

[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
nb = n+1;
Aug = [A b];
% forward elimination
for k = 1:n-1
  for i = k+1:n
    factor = Aug(i,k)/Aug(k,k);
    Aug(i,k:nb) = Aug(i,k:nb)-factor*Aug(k,k:nb);
  end
end
% back substitution
x = zeros(n,1);
x(n) = Aug(n,nb)/Aug(n,n);
for i = n-1:-1:1
  x(i) = (Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
```

Gauss Program Efficiency

- The execution of Gauss elimination depends on the amount of *floating-point operations* (or flops). The flop count for an *n* x *n* system is:

	Forward elimination	$\frac{2n^3}{n} + O(n^2)$
+	Back substitution	$n^2 + O(n)$
	Total	$2n^3 + O(n^2)$

- Conclusions:
  - As the system gets larger, the computation time increases greatly.
  - Most of the effort is incurred in the elimination step.

Pivoting

- Problems arise with naïve Gauss elimination if a coefficient along the diagonal is 0 (problem: division by 0) or close to 0 (problem: round-off error)

$$\begin{aligned} 2x_2 + 3x_3 &= 8 \\ 4x_1 + 6x_2 + 7x_3 &= -3 \\ 2x_1 - 3x_2 + 6x_3 &= 5 \end{aligned}$$

- One way to combat these issues is to determine the coefficient with the largest absolute value in the column below the pivot element. The rows can then be switched so that the largest element is the pivot element. This is called *partial pivoting*.
- If the rows to the right of the pivot element are also checked and columns switched, this is called *complete pivoting*. But you do not gain much by doing this!

Partial Pivoting example

- Use Gauss elimination to solve

$$0.0003x_1 + 3.0000x_2 = 2.0001$$
$$1.0000x_1 + 1.0000x_2 = 1.0000$$

Solution. Multiplying the first equation by $1/(0.0003)$ yields

$$x_1 + 10{,}000x_2 = 6667$$

which can be used to eliminate x_1 from the second equation:

$$-9999x_2 = -6666$$

which can be solved for $x_2 = 2/3$

$$x_1 = \frac{2.0001 - 3(2/3)}{0.0003}$$

Significant Figures	x_2	x_1	Absolute Value of Percent Relative Error for x_1
3	0.667	−3.33	1099
4	0.6667	0.0000	100
5	0.66667	0.30000	10
6	0.666667	0.330000	1
7	0.6666667	0.3330000	0.1

Partial Pivoting example

- if the equations are solved in reverse order, the row with the larger pivot element is normalized.

$$1.0000x_1 + 1.0000x_2 = 1.0000$$
$$0.0003x_1 + 3.0000x_2 = 2.0001$$

$$x_2 = 2/3$$
$$x_1 = \frac{1 - (2/3)}{1}$$

Significant Figures	x_2	x_1	Absolute Value of Percent Relative Error for x_1
3	0.667	0.333	0.1
4	0.6667	0.3333	0.01
5	0.66667	0.33333	0.001
6	0.666667	0.333333	0.0001
7	0.6666667	0.3333333	0.0000

Partial Pivoting Program

```
function x = GaussPivot(A,b)
% GaussPivot: Gauss elimination pivoting
%   x = GaussPivot(A,b): Gauss elimination with pivoting.
% input:
%   A = coefficient matrix
%   b = right hand side vector
% output:
%   x = solution vector

[m,n]=size(A);
if m~=n, error('Matrix A must be square'); end
nb=n+1;
Aug=[A b];
% forward elimination
for k = 1:n-1
  % partial pivoting
  [big,i]=max(abs(Aug(k:n,k)));
  ipr=i+k-1;
  if ipr~=k
    Aug([k,ipr],:)=Aug([ipr,k],:);
  end
  for i = k+1:n
    factor=Aug(i,k)/Aug(k,k);
    Aug(i,k:nb)=Aug(i,k:nb)-factor*Aug(k,k:nb);
  end
end
% back substitution
x=zeros(n,1);
x(n)=Aug(n,nb)/Aug(n,n);
for i = n-1:-1:1
  x(i)=(Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
```

Tridiagonal Systems

- A *tridiagonal* system is a banded system with a bandwidth of 3:

$$\begin{bmatrix} f_1 & g_1 & & & & & & \\ e_2 & f_2 & g_2 & & & & & \\ & e_3 & f_3 & g_3 & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & e_{n-1} & f_{n-1} & g_{n-1} \\ & & & & & & e_n & f_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} r_1 \\ r_2 \\ r_3 \\ \cdot \\ \cdot \\ \cdot \\ r_{n-1} \\ r_n \end{Bmatrix}$$

- Tridiagonal systems can be solved using the same method as Gauss elimination, but with much less effort because most of the matrix elements are already 0.

Tridiagonal Systems

- Solve the following tridiagonal system

$$\begin{bmatrix} 2.04 & -1 & & \\ -1 & 2.04 & -1 & \\ & -1 & 2.04 & -1 \\ & & -1 & 2.04 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 40.8 \\ 0.8 \\ 0.8 \\ 200.8 \end{Bmatrix}$$

  - first step involves transforming the matrix to upper triangular form. This is done by multiplying the first equation by the factor e_2/f_1 and subtracting the result from the second equation

$$f_2 = f_2 - \frac{e_2}{f_1} g_1 = 2.04 - \frac{-1}{2.04}(-1) = 1.550$$

$$r_2 = r_2 - \frac{e_2}{f_1} r_1 = 0.8 - \frac{-1}{2.04}(40.8) = 20.8$$

$$\begin{bmatrix} 2.04 & -1 & & \\ & 1.550 & -1 & \\ & & 1.395 & -1 \\ & & & 1.323 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 40.8 \\ 20.8 \\ 14.221 \\ 210.996 \end{Bmatrix}$$

Tridiagonal Systems

- Now back substitution can be applied to generate the final solution

$$x_4 = \frac{r_4}{f_4} = \frac{210.996}{1.323} = 159.480$$

$$x_3 = \frac{r_3 - g_3 x_4}{f_3} = \frac{14.221 - (-1)159.480}{1.395} = 124.538$$

$$x_2 = \frac{r_2 - g_2 x_3}{f_2} = \frac{20.800 - (-1)124.538}{1.550} = 93.778$$

$$x_1 = \frac{r_1 - g_1 x_2}{f_1} = \frac{40.800 - (-1)93.778}{2.040} = 65.970$$

Tridiagonal System Solver

```
function x = Tridiag(e,f,g,r)
% Tridiag: Tridiagonal equation solver banded system
%   x = Tridiag(e,f,g,r): Tridiagonal system solver.
% input:
%   e = subdiagonal vector
%   f = diagonal vector
%   g = superdiagonal vector
%   r = right hand side vector
% output:
%   x = solution vector
n=length(f);
% forward elimination
for k = 2:n
  factor = e(k)/f(k-1);
  f(k) = f(k) - factor*g(k-1);
  r(k) = r(k) - factor*r(k-1);
end
% back substitution
x(n) = r(n)/f(n);
for k = n-1:-1:1
  x(k) = (r(k)-g(k)*x(k+1))/f(k);
end
```