

# 1.00 Lecture 22

## November 1, 2005

### Systems of Linear Equations

### Systems of Linear Equations

$$3x_0 + x_1 - 2x_2 = 5$$

$$2x_0 + 4x_1 + 3x_2 = 35$$

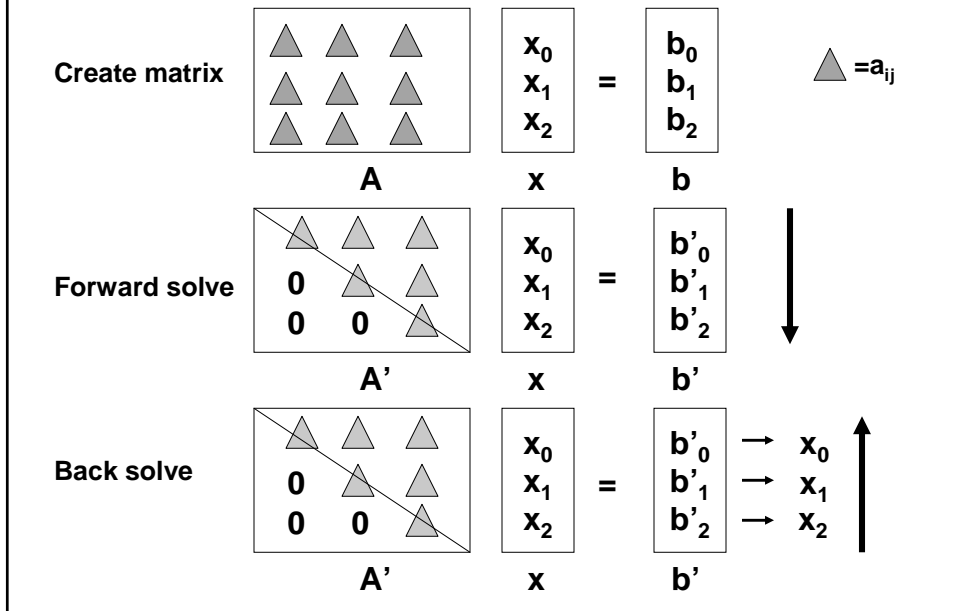
$$x_0 - 3x_1 = -5$$

$$\begin{array}{ccc|ccc} 3 & 1 & -2 & x_0 & & 5 \\ 2 & 4 & 3 & x_1 & = & 35 \\ 1 & -3 & 0 & x_2 & & -5 \end{array}$$

$$A \quad x = b$$

$$3 \ x \ 3 \quad 3 \ x \ 1 \quad 3 \ x \ 1$$

# Algorithm to Solve Linear System



## Gaussian Elimination: Forward Solve

<b>Q=</b>	3	1	-2	5	Form Q for convenience Do elementary row ops: Multiply rows Add/subtract rows
	2	4	3	35	
	1	-3	0	-5	
		<b>A</b>		<b>b</b>	

**Make column 0 have zeros below diagonal**

Pivot= 2/3 →	3	1	-2	5	Row 1' = row 1 - (2/3) row 0 Row 2' = row 2 - (1/3) row 0
Pivot= 1/3 →	0	10/3	13/3	95/3	
	0	-10/3	2/3	-20/3	

**Make column 1 have zeros below diagonal**

Pivot= 1 →	3	1	-2	5	Row 2'' = row 2' + 1 * row 1
	0	10/3	13/3	95/3	
	0	0	15/3	75/3	

## Gaussian Elimination: Back Solve

3	1	-2	5
0	10/3	13/3	95/3
0	0	15/3	75/3

$$(15/3)x_2 = (75/3)$$

$$x_2 = 5$$

3	1	-2	5
0	10/3	13/3	95/3
0	0	15/3	75/3

$$(10/3)x_1 + (13/3)*5 = (95/3) \quad x_1 = 3$$

3	1	-2	5
0	10/3	13/3	95/3
0	0	15/3	75/3

$$3x_0 + 1*3 - 2*5 = 5$$

$$x_0 = 4$$

## A Complication

0	1	-2	5
2	4	3	35
1	-3	0	-5

$$\text{Row 1}' = \text{row 1} - (2/0) \text{row 0}$$

Exchange rows: put largest pivot element in row:

2	4	3	35
1	-3	0	-5
0	1	-2	5

Do this as we process each column.

If there is no nonzero element in a column,  
matrix is not full rank.

# Gaussian Elimination

```
public static void gaussian(Matrix a, Matrix b, Matrix x) {
    int i, j, n;
    n = a.getNumRows(); // Number of unknowns
    Matrix q = new Matrix(n, n+1);
    for (i=0; i < n; i++) {
        for (j=0; j < n; j++) // Form q matrix
            // q(i,j) = a(i,j)
            q.setElement(i, j, a.getElement(i, j));
        // q(i,n) = b(i,0)
        q.setElement(i, n, b.getElement(i, 0));
    }
    forward_solve(q); // Do Gaussian elimination
    back_solve(q); // Perform back substitution

    for (i=0; i < n; i++)
        // x(i,0) = q(i,n)
        x.setElement(i, 0, q.getElement(i, n));
}
```

```
private static void forward_solve(Matrix q) {
    int i, j, k, maxr, n;
    double t, pivot;
    n = q.getNumRows();

    for (i=0; i < n; i++) { // Find row w/max element in this
        maxr = i; // column, at or below diagonal
        for (j=i+1; j < n; j++)
            if (Math.abs(q.getElement(j, i)) >
                Math.abs(q.getElement(maxr, i)))
                maxr = j;
        if (maxr != i) // If row not current row, swap
            for (k=i; k <= n; k++)
                { t = q.getElement(i, k); // t = q(i, k)
                  // q(i, k) = q(maxr, k)
                  q.setElement(i, k, q.getElement(maxr, k));
                  q.setElement(maxr, k, t); // q(maxr, k) = t
                }
        for (j=i+1; j < n; j++) // Calculate pivot ratio
            { pivot = q.getElement(j, i) / q.getElement(i, i);
              for (k=n; k >= i; k--)
                  q.setElement(j, k, q.getElement(j, k) -
                    q.getElement(i, k) * pivot);
              // q(j, k) -= q(i, k) * pivot; Update row j below diag
            }
    }
}
```

## Back Substitution

```
private static void back_solve(Matrix q)
{
    int j, k, n;
    double t; // t- temporary
    n= q.getNumRows();

    for (j=n-1; j >=0; j--) // Start at last row
    {
        t= 0.0;
        for (k= j+1; k < n; k++) // t += q(j,k)* q(k,n)
            t += q.getElement(j, k)* q.getElement(k, n);
        q.setElement(j, n,
            (q.getElement(j, n) -t)/q.getElement(j, j));
        // q(j, n)= (q(j, n) -t)/q(j, j);
    }
}
```

## Main Program

```
import javax.swing.*;

public class GaussMain {
    public static void main(String[] args) {
        int i, j;
        double term;
        String input= JOptionPane.showInputDialog
            ("Enter number of unknowns");
        int n= Integer.parseInt(input);
        // Create matrices a, b, x
        Matrix a= new Matrix(n, n);
        Matrix b= new Matrix(n, 1);
        Matrix x= new Matrix(n, 1);
        // Enter matrix a
        for (i = 0; i < a.getNumRows(); i++)
            for (j = 0; j < a.getNumCols(); j++) {
                input= JOptionPane.showInputDialog
                    ("Enter a["+i+"]["+j+"]");
                term= Double.parseDouble(input);
                a.setElement(i, j, term);
            }
    }
}
```

## Main Program, p.2

```

// Enter vector b as 1-column matrix
for (i = 0; i < b.getNumRows(); i++) {
    input= JOptionPane.showInputDialog
        ("Enter b["+i+"]");
    term= Double.parseDouble(input);
    b.setElement(i, 0, term);
}
gaussian(a, b, x);
System.out.println("Matrix a:");
a.print();
System.out.println("Vector b:");
b.print();
System.out.println("Solution vector x:");
x.print();
}

```

## Variations

Multiple right hand sides: augment Q, solve all eqns at once

$$\begin{array}{ccc|c|c|c}
 3 & 1 & -2 & 5 & 7 & 87 \\
 2 & 4 & 3 & 35 & 75 & -1 \\
 1 & -3 & 0 & -5 & 38 & 52
 \end{array}$$

Matrix inversion (rarely done in practice)

$$\begin{array}{ccc|ccc}
 3 & 1 & -2 & 1 & 0 & 0 \\
 2 & 4 & 3 & 0 & 1 & 0 \\
 1 & -3 & 0 & 0 & 0 & 1 \\
 \hline
 \text{A} & & & \text{I} & & \\
 \hline
 \text{Q} & & & & & 
 \end{array}
 \longrightarrow
 \begin{array}{ccc|ccc}
 \# & \# & \# & @ & @ & @ \\
 0 & \# & \# & @ & @ & @ \\
 0 & 0 & \# & @ & @ & @ \\
 \hline
 & & & \text{A}^{-1} & & 
 \end{array}$$

$Ax=b$   
 $x= A^{-1} b$

## Exercise

- **Experiment with the linear systems application. Download GEI i m. j ava:**
  - **In your web browser go to course web site and download Lecture22JavaFile.zip. Unzip it.**
  - **Create a new project for Lecture 22 and put the files you downloaded into it.**
  - **Compile and run GEI i m. j ava.**

## Linear Systems Application

- **Application's functionalities:**
  - Setup: constructs a square matrix of dimension N.
  - Undo: Reverses the previous action (you can only undo once).
  - Save: saves the current matrix.
  - Load: loads the saved matrix.
  - Demo: loads a pre-saved matrix which happens to be the 3x3 matrix example that you saw in the previous slides.
  - Quit: exits the application
  - Pivot on Selection: First, you need to select a cell by clicking on it. Then click on "Pivot on Selection" to reduce to 0 the cells below the selection.

## Linear Systems Application

- **The Application's functionalities (cont'd):**
  - **Back Subst. on Selection:** You need to select a cell by clicking on it. In order to successfully perform a back substitution, the selected cell should be the only non-zero cell in its row (except for the "b cell"). If this is not the case, the value of the selected variable would be considered unknown and you cannot back substitute.
  - **Divide on selection:** divides the entire row by the value of the selected cell.
  - **Solve:** directly solves the matrix by reducing it to a "row echelon form".
  - **Swap:** swaps the first named row with the second.
  - **Commit:** multiply a row by a constant or replace the first named row by a combination with the second.

## Hands On

- **Experiment with the following 3 matrices:**
  - **The 3x3 matrix example that you saw in the previous slides. Click on "Demo" to load it.**

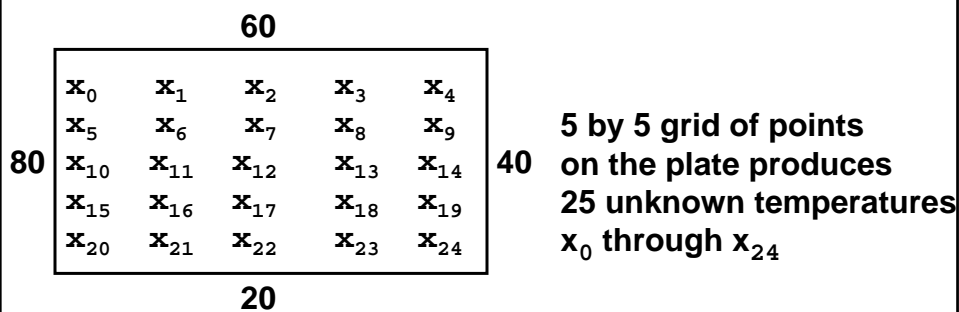
$$- \begin{array}{ccc|c} 4 & 6 & -3 & 10 \\ 2 & 5 & 9 & 12 \\ 8 & 8 & -27 & 6 \end{array}$$

$$- \begin{array}{cccc|c} 12 & -13.5 & 3 & 0.5 & 2.75 \\ 8 & -9 & 4 & 2.5 & 3.5 \\ 3 & 6 & 1.5 & 2 & 4.25 \\ 2 & 1.5 & 4 & 12 & 6 \end{array}$$

## Using Matrices

- A common pattern in engineering, scientific and other analytical software:
  - Problem generator (model, assemble matrix)
    - Customized to specific application (e.g. heat transfer)
    - Use matrix multiplication, addition, etc.
  - Problem solution (system of simultaneous linear equations)
    - Usually “canned”: either from library or written by you for a library
  - Output generator (present result in understandable format)
    - Customized to specific application (often with graphics, etc.)

## Heat Transfer Example



$$T = (T_{\text{left}} + T_{\text{right}} + T_{\text{up}} + T_{\text{down}}) / 4$$

Edge temperatures are known; interior temperatures are unknown  
This produces a 25 by 25 matrix of linear equations

## Heat Transfer, p.2

- Node 0:

$$x_0 = (80 + x_1 + 60 + x_5)/4 \quad 4x_0 - x_1 - x_5 = 140$$

- Node 6:

$$x_6 = (x_5 + x_7 + x_1 + x_{11})/4 \quad 4x_6 - x_5 - x_7 - x_1 - x_{11} = 0$$

- Interior node:

$$x_i = (x_{i-1} + x_{i+1} + x_{i-n} + x_{i+n})/4 \quad 4x_i - x_{i-1} - x_{i+1} - x_{i-n} - x_{i+n} = 0$$

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

## Heat Transfer, p.3

25

$$\begin{array}{c}
 \boxed{
 \begin{array}{cccc}
 a_{00} & a_{01} & a_{02} & \dots & a_{0,24} \\
 a_{10} & a_{11} & a_{12} & \dots & a_{1,24} \\
 a_{20} & a_{21} & a_{22} & \dots & a_{2,24} \\
 & & & \dots & \\
 a_{24,0} & a_{24,1} & & \dots & a_{24,24}
 \end{array}
 }
 \end{array}
 \begin{array}{c}
 \boxed{
 \begin{array}{c}
 x_0 \\
 x_1 \\
 x_2 \\
 \dots \\
 x_{24}
 \end{array}
 }
 \end{array}
 =
 \begin{array}{c}
 \boxed{
 \begin{array}{c}
 b_0 \\
 b_1 \\
 b_2 \\
 \dots \\
 b_{24}
 \end{array}
 }
 \end{array}$$

A

x

b

Contains 0, -1, 4  
coefficients in  
(simple) pattern

Known temperatures  
(often 0 but use edge  
temperatures when close)

25 unknown interior temperatures

## Heat Transfer, p.4

Solution:

			60			
	67	61	57	54	50	
	68	60	54	50	45	
80	66	56	50	45	42	40
	61	49	43	39	38	
	50	38	33	31	32	
			20			

- There is a `NumberFormatter` class in Java to control the number of decimal places, etc.
- Example uses ints for simplicity in the output; the temperatures are doubles, really.
- Could use color graphics to show gradient, etc.

## Heat Transfer Code, p.1

```
public class Heat {          // Problem generator
    public static void main(String[] args) { // Edge temps
        double Te= 40.0, Tn=60.0, Tw=80.0, Ts=20.0;
        final int col= 5;
        final int row= 5;
        final int n= col * row;
        Matrix a= new Matrix(n,n);
        for (int i=0; i < n; i++)
            for (int j=0; j < n; j++) {
                if (i==j) // Diagonal element
                    a.setElement(i, j, 4.0);
                else ...
                    // Complete this code: Set element to -1.0
                    // if node i next to node j
                    // Be careful at the edges (e.g. node 5 is
                    // not adjacent to node 4)
                else
                    a.setElement(i, j, 0.0);            }
    }
}
```

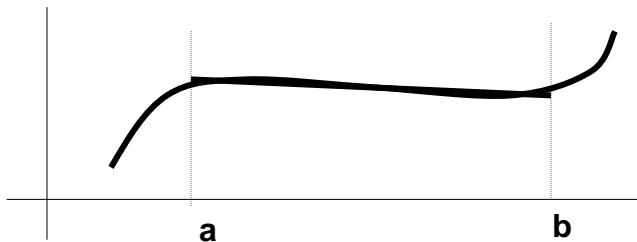
## Heat Transfer Code, p.2

```
Matrix x= new Matrix(n, 1);          // Unknown temps
for (int i=0; i < n; i++)
    x.setElement(i, 0, 0.0);        // Initialize to zeros
Matrix b= new Matrix(n, 1);          // Known temps
for (int i=0; i < n; i++) {
    b.setElement(i, 0, 0.0);
    if (i < col)                    // Next to north edge
        b.setElement(i, 0, b.getElement(i,0)+Tn);
    // Complete this code for the other edges; no 'elses'!
    // Always add the edge temperature to b
}
GaussMain.gaussian(a, b, x);        // Problem solution
System.out.println("Temperature grid:"); // Output generator
// Display the output in a 5 by 5 grid. Use ints.
}
}

// Download Heat2.java from the Web site and complete the code
// Also download Matrix.java and GaussMain.java
```

## Other Applications

- **Solve systems with 1,000s or millions of linear equations or inequalities**
  - Networks, mechanics, fluids, materials, economics
  - Often linearize systems in a defined range



- **Routines in this lecture are ok for a few hundred equations**
  - They aren't very good at picking up collinear systems. Check first-see Numerical Recipes
- **Otherwise, see Numerical Recipes**