## 1.204 Lecture 20

**Linear systems:**
**Gaussian elimination**
**LU decomposition**

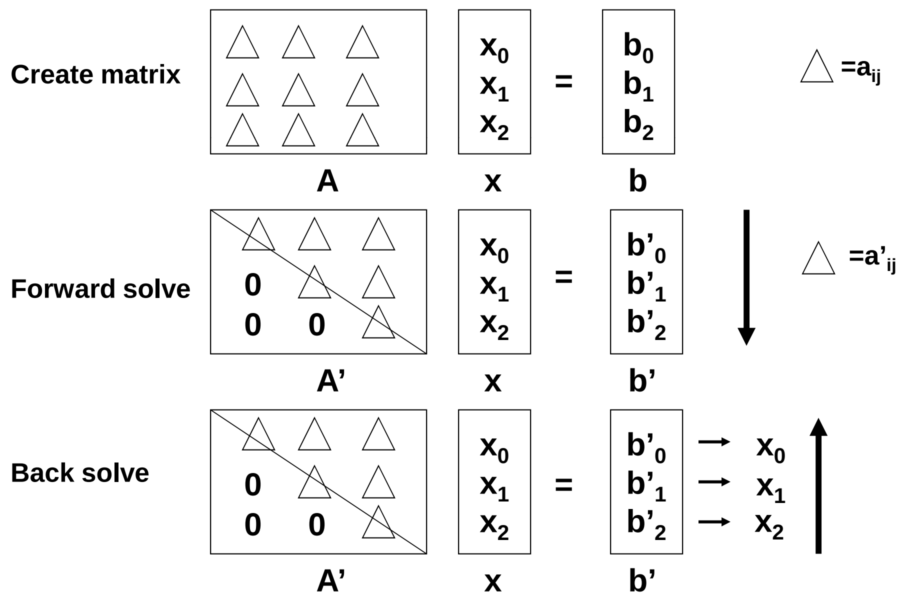
## Systems of Linear Equations

$$
\begin{aligned}
3x_0 + x_1 - 2x_2 &= 5 \\
2x_0 + 4x_1 + 3x_2 &= 35 \\
x_0 - 3x_1 &= -5
\end{aligned}
$$

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

$$
\begin{array}{ccccc}
A & & x & = & b \\
3 \times 3 & & 3 \times 1 & & 3 \times 1
\end{array}
$$

## Algorithm to Solve Linear System




## Gaussian Elimination: Forward Solve

$$Q = \left|\begin{array}{ccc|c|} 3 & 1 & -2 & 5 \\ 2 & 4 & 3 & 35 \\ 1 & -3 & 0 & -5 \\ \multicolumn{3}{c}{A} & \multicolumn{1}{c}{b} \end{array}\right.$$

**Form Q for convenience**
**Do elementary row ops:**
- **Multiply rows**
- **Add/subtract rows**

**Make column 0 have zeros below diagonal**

$$\begin{array}{r} \text{Pivot}= 2/3 \\ \text{Pivot}= 1/3 \\ \\ \end{array} \left|\begin{array}{ccc|c|} 3 & 1 & -2 & 5 \\ 0 & 10/3 & 13/3 & 95/3 \\ 0 & -10/3 & 2/3 & -20/3 \end{array}\right. \begin{array}{l} \\ \text{Row 1'= row 1 - (2/3) row 0} \\ \text{Row 2'= row 2 - (1/3) row 0} \end{array}$$

**Make column 1 have zeros below diagonal**

$$\begin{array}{r} \\ \text{Pivot}= -1 \longrightarrow \\ \\ \end{array} \left|\begin{array}{ccc|c|} 3 & 1 & -2 & 5 \\ 0 & 10/3 & 13/3 & 95/3 \\ 0 & 0 & 15/3 & 75/3 \end{array}\right. \begin{array}{l} \\ \\ \text{Row 2''= row 2' + 1 * row 1} \end{array}$$

## 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 |

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

| | | | |
|---|---|---|---|
| 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 |

Row 1'= row 1 - (2/0) row 0

Exchange rows: put largest pivot element in row:

| | | | |
|---|---|---|---|
| 2 | 4 | 3 | 35 |
| 0 | 1 | -2 | 5 |
| 1 | -3 | 0 | -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 class Gauss {
  public static double[] gaussian(double[][] a, double[] b) {
    int n = a.length;                // Number of unknowns
    double[][] q = new double[n][n + 1];

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)    // Form q matrix
        q[i][j]= a[i][j];
      q[i][n]= b[i];
    }
    forward_solve(q);                // Do Gaussian elimination
    back_solve(q);                   // Perform back substitution

    double[] x= new double[n];       // Extract column n of q,
    for (int i = 0; i < n; i++)      // which contains the solution x
    x[i]= q[i][n];
    return x;
  }
```

# Forward Solve

```
private static void forward_solve(double[][] q) {
  int n = q.length;
  int m= q[0].length;

  for (int i = 0; i < n; i++) {   // Find row w/max element in this
    int maxRow = i;               // column, at or below diagonal
    for (int k = i + 1; k < n; k++)
      if (Math.abs(q[k][i]) > Math.abs(q[maxRow][i]))
        maxRow = k;

    if (maxRow != i)              // If row not current row, swap
      for (int j = i; j < m; j++) {
        double t = q[i][j];
        q[i][j]= q[maxRow][j];
        q[maxRow][j]= t;
      }

    for (int j = i + 1; j < n; j++) {    // Calculate pivot ratio
      double pivot = q[j][i] / q[i][i];
      for (int k = i; k < m; k++)     // Pivot operation itself
        q[j][k] -= q[i][k] * pivot;
    }
  }
}
```

## Back Substitution

```
private static void back_solve(double[][] q) {
  int n = q.length;
  int m= q[0].length;
  for (int p= n; p < m; p++) {              // Loop over p columns
    for (int j = n - 1; j >= 0; j--) {      // Start at last row
      double t = 0.0;                       // t- temporary
      for (int k = j + 1; k < n; k++)
        t += q[j][k] * q[k][p];
      q[j][p]= (q[j][p] - t) / q[j][j];
    }
  }
}
```

## Variations

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

$$\left|\begin{array}{rrr|r|r|r} 3 & 1 & -2 & 5 & 7 & 87 \\ 2 & 4 & 3 & 35 & 75 & -1 \\ 1 & -3 & 0 & -5 & 38 & 52 \end{array}\right|$$

**Matrix inversion (rarely done in practice)**

$$\underbrace{\left|\begin{array}{rrr|rrr} 3 & 1 & -2 & 1 & 0 & 0 \\ 2 & 4 & 3 & 0 & 1 & 0 \\ 1 & -3 & 0 & 0 & 0 & 1 \end{array}\right|}_{Q} \longrightarrow \left|\begin{array}{ccc|ccc} \# & \# & \# & @ & @ & @ \\ 0 & \# & \# & @ & @ & @ \\ 0 & 0 & \# & @ & @ & @ \end{array}\right|$$

A I A$^{-1}$

**A • ? = I**

**? = A$^{-1}$**

## Invert

```
public static double[][] invert(double[][] a) {
  int n = a.length;                   // Number of unknowns
  double[][] q = new double[n][n+n];

  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)       // Form q matrix
      q[i][j]= a[i][j];

  // Form identity matrix in right half of q
  for (int i = 0; i < n; i++)
    q[i][n+i]= 1.0;

  forward_solve(q);            // Do Gaussian elimination
  back_solve(q);               // Perform back substitution

  double[][] x= new double[n][n];     // Extract R half of q
  for (int i = 0; i < n; i++)         // which contains inverse
    for (int j= 0; j < n; j++)
      x[i][j]= q[i][j+n];
  return x;
}
// Method multiply() in download
// Example use in GaussTest in download
```

## LU decomposition

- We can write matrix A as the product of two matrices L and U:

$$\begin{bmatrix} l_{00} & 0 & 0 & 0 \\ l_{10} & l_{11} & 0 & 0 \\ l_{20} & l_{21} & l_{22} & 0 \\ l_{30} & l_{31} & l_{32} & l_{33} \end{bmatrix} \cdot \begin{bmatrix} u_{00} & u_{01} & u_{02} & u_{03} \\ 0 & u_{11} & u_{12} & u_{13} \\ 0 & 0 & u_{22} & u_{23} \\ 0 & 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- We can solve $A \cdot x = (L \cdot U) \cdot x = L \cdot (U \cdot x) = b$

  by first solving for a vector y $L \cdot y = b$

  and then solving $U \cdot x = y$

Why? Solving each is trivial: forward, back substitution

## Why and How

- This is perhaps twice as fast as Gaussian elimination (count steps)
- L and U do not depend on b, so we can solve as many right hand sides as we wish
- How: Crout's method
  – We can decompose matrix A into matrices L and U by arranging the equations in a given order
  – The rearrangement is subtle; we don't cover it in class since you'll never need to implement or modify it
- Java implementation is on the Web site, based on Press et al, ***Numerical Recipes***

## Class LU

- Constructor: `LU(double[][] a)`
  – Stores LU decomposition in a single matrix
    - All $l_{ii}$ = 1.0 in matrix L
    - We store all u elements and all non-diagonal l elements in LU
- Methods:
  – `public double[] solve(double[] b)`
  – `public double[][] solve(double[][] b)`
  – `public double[][] inverse()`
  – `public double determinant()`
  – `public double[] improve(double[] b, double[] x)`
- See download for code and LUTest class for examples of usage
  – You can use it as a 'black box'
  – Use this in preference to class Gauss

## Other linear system algorithms

- Banded matrices
- Sparse matrices
- Singular value decompositions (SVD)
  - Should be used in least squares computations
- Cholesky decomposition ($A = L L^T$)
  - Square, symmetric, positive definite matrices
  - Used in econometrics
- And others…
  - Almost all are based on pivot operations

## Linear system model: Rail performance

- Compute running time, including delays, for trains on a single track railroad
  - Traffic in both directions: east and west
  - Three types of train (6 classes of train, including direction)

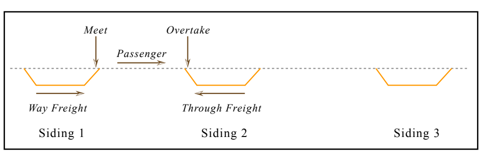



Figure by MIT OpenCourseWare.

| Class | Description | Velocity |
|---|---|---|
| 0 | WB way freight | -25 |
| 1 | WB thru freight | -50 |
| 2 | WB passenger | -80 |
| 3 | EB passenger | 80 |
| 4 | EB thru freight | 50 |
| 5 | EB way freight | 25 |

From E.R. Petersen

## Rail performance, p.2

- **Priorities used to model meets and overtakes**
  - Meets occur when trains travel in opposite directions and one must take siding and wait until other passes
  - Overtakes occur when trains traveling in same direction interact, and slower one takes siding to let faster one go by
  - Assume all sidings are long enough
- **We want to model the running time for each class of train over a segment of railroad, as a function of:**
  - Number of trains of each class (type, direction)
  - Speed of trains
  - Number of sidings
  - Priorities, and other, less important variables
- Our linear model will give nonlinear performance behavior!

## Delay matrix D

- **Matrix D gives average delay for each interaction (meet or overtake) between two classes of train**
  - We will then multiply this by the expected number of interactions, to get total delay
- **Delay matrix D has coefficients $D_{ij}$:**

$$D_{ij} = p_{ij} S_i + \frac{60 p_{ij}^2}{2(b+1)} \left| \frac{d}{v_i} - \frac{d}{v_j} \right|$$

  - $D_{ij}$= Expected delay to train i due to train j, in minutes
  - $S_i$= Time to take siding for train i, in minutes (5 minutes)
  - $p_{ij}$= Relative number of times train i waits for train j ($0 <= p_{ij} <= 1$)
  - b= Number of sidings (19)
  - D= Distance of railroad segment being modeled, in miles (400 mi)
  - $v_i$= Free running velocity for train i, in miles per hour

# Delay matrix D

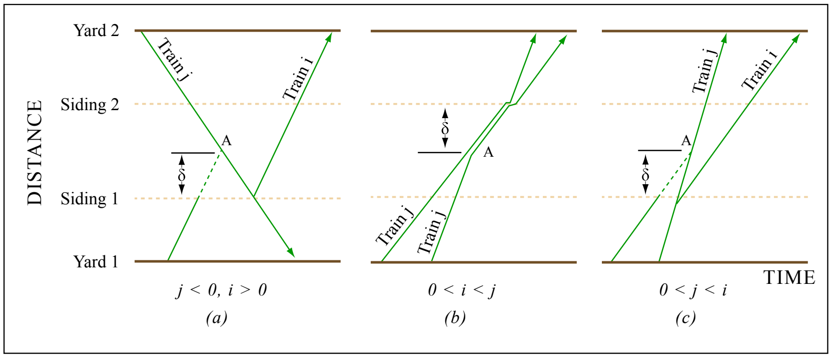



Figure by MIT OpenCourseWare.

# Probability, delay matrices

| **Prob(delay)** | WB way | WB thru | WB pass | EB pass | EB thru | EB way |
|---|---|---|---|---|---|---|
| WB way | 0 | 0.7 | 0.9 | 1 | 0.7 | 0.5 |
| WB thru | 0.3 | 0 | 0.7 | 0.7 | 0.5 | 0.3 |
| WB pass | 0.1 | 0.3 | 0 | 0.5 | 0.3 | 0 |
| EB pass | 0 | 0.3 | 0.5 | 0 | 0.3 | 0.1 |
| EB thru | 0.3 | 0.5 | 0.7 | 0.7 | 0 | 0.3 |
| EB way | 0.5 | 0.7 | 1 | 0.9 | 0.7 | 0 |

| **Delay** | WB way | WB thru | WB pass | EB pass | EB thru | EB way |
|---|---|---|---|---|---|---|
| WB way | 0 | 9.4 | 17.9 | 36.5 | 21.1 | 14.5 |
| WB thru | 2.6 | 0 | 5.7 | 13.1 | 8.5 | 4.7 |
| WB pass | 0.7 | 1.9 | 0 | 6.3 | 3.3 | 0 |
| EB pass | 0 | 3.3 | 6.3 | 0 | 1.9 | 0.7 |
| EB thru | 4.7 | 8.5 | 13.1 | 5.7 | 0 | 2.6 |
| EB way | 14.5 | 21.1 | 36.5 | 17.9 | 9.4 | 0 |

# Derivation of linear system

- **Let**
  - $W_i$= average time for train of class i, including delays
  - $T_i$= free running time for train of class I (input)
  - $D_{ij}$= delays due to meets and overtakes to train of class I due to trains of class j
  - $M_{ij}$= number of meets and overtakes between train of class I and trains of class j
- **Average time for train to travel across segment:**

$$W_i = T_i + \sum_j (D_{ij} \cdot M_{ij})$$

- **Interactions between train i and trains of class j:**

$$M_{ij} = N_j (W_j + W_i) / 1440$$

  - $M_{ij}$= number of trains/day in class j times fraction of day that train of class i can interact with trains of class j
    - If EB train takes 12 hours (720 minutes) to cover line, as does WB, it will meet every WB train that operates that day
    - If EB train took only 6 hours, it would have half the interactions

# Derivation of linear system, p.2

$$W_i = T_i + \sum_i (D_{ij} \cdot M_{ij}) \qquad M_{ij} = N_j (W_j + W_i) / 1440$$

- **The rest is algebra, to write the two equations above in the form AW= T, with W the unknown ("x")**

$$W_i = T_i + \sum_j \frac{D_{ij} \cdot N_j}{1440} (W_j + W_i)$$

Collect Wi terms on left side of equation :

$$W_i - \sum_j \frac{D_{ij} \cdot N_j}{1440} (W_j + W_i) = T_i$$

Rearrange terms to separate Wi and Wj terms :

$$(1 - \sum_j \frac{D_{ij} \cdot N_j}{1440}) \cdot W_i - \sum_j \frac{D_{ij} \cdot N_j}{1440} \cdot W_j = T_i$$

The Wi coefficient is the diagonal; the Wj coefficients are the off - diagonal elements in matrix A

$$\begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} \cdot \begin{vmatrix} w_0 \\ w_1 \\ w_2 \end{vmatrix} = \begin{vmatrix} t_0 \\ t_1 \\ t_2 \end{vmatrix}$$

- **There is a +/- sign convention handled by a matrix C (multiplies $D_{ij}$):**
  - c[i][j]= -1 if j<i<=2 or (3 <=i< j), 0 otherwise (with 6 train classes)

## Data members, constructor

```
public class RailDelay {
   private int n;      // Number of train classes
   private int d;      // Distance in miles
   private int b;      // Number of sidings
   private double[][] p;  // Probability of delay in interaction
   private double[] s;    // Time to take siding, by train class
   private double[] v;   // Velocity by train class
   private double[][] c; // Matrix that indicates if interaction
   // is pass or meet, by train classes involved
   private int[] nTrain; // Number of trains by class, per day

   public RailDelay(String filename) {
        // Constructor reads inputs from file, sets data members
   }
```

## getDelay()

```
public double[] getDelay() {
  double[][] dm= new double[n][n];   // Delay matrix D
  for (int i= 0; i < n; i++)
    for (int j= 0; j < n; j++) {
      dm[i][j]= p[i][j]*s[i] + 60.0*p[i][j]*p[i][j] *
               Math.abs(d/v[i]- d/v[j])/(2.0*(b+1)); }
  double[] t= new double[n];         // Free running time T
  for (int i= 0; i < n; i++)
    t[i]= d*60.0/v[i];
  double[][] a= new double[n][n];    // Total delay matrix A
  for (int i= 0; i < n; i++) {
    double delay= 0.0;
    for (int j= 0; j < n; j++) {
      if (i != j) {
        a[i][j]= dm[i][j]*nTrain[j]*c[i][j]/1440;
        delay += a[i][j];
    } }
    a[i][i]= 1.0 - delay; }
  double[] w= Gauss.gaussian(a, t);
  return w;
}
```

## main(), sample output

```
    public static void main(String[] args) {
        RailDelay r= new RailDelay("src/linear/rail.txt");
        double[] w= r.getDelay();     // Gets output w
        int n= w.length;
        System.out.println("i Act time Free time");
        for (int i= 0; i < n; i++)
                System.out.printf("%d %8.1f %8.1f \n",i,
                     Math.abs(w[i]), Math.abs(r.getFreeTime(i)));
        System.out.println();
    }
// Sample output: 3 wayfreight, 4 thru freight, 2 passenger
               i Act time Free time
               0   1265.2    960.0
               1    544.9    480.0
               2    315.8    300.0
               3    315.8    300.0
               4    544.9    480.0
               5   1265.2    960.0
```

## Rail performance estimate

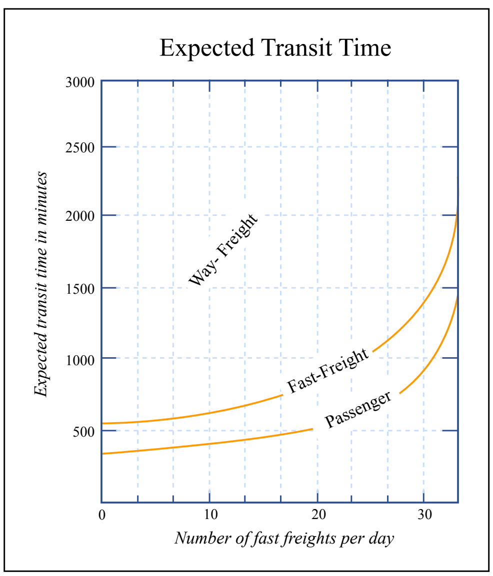



Figure by MIT OpenCourseWare.

## Summary

- Linear models are a reasonable starting point in many cases to understand complex systems
  - Writing down equations to model a system analytically or through solving linear or nonlinear systems is often a viable option
  - Linear models can produce nonlinear behavior
  - In the rail example, this is more intuitive (for some of us) and more robust than simulation
- I used essentially the rail analysis in a Vermont Act 250 expert witness case
  - Traffic impacts on a neighborhood from a large development
  - Narrow road with parking on both sides
  - Number of “meets” between cars would increase very sharply
  - Project application was denied
- We’ll use linear systems as a “subproblem” in next lecture

1.204 Computer Algorithms in Systems Engineering
Spring 2010