# Linear Systems

COS 323

# Linear Systems

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

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

# Linear Systems

- Solve Ax=b, where A is an $n \times n$ matrix and b is an $n \times 1$ column vector
- Can also talk about non-square systems where A is $m \times n$, b is $m \times 1$, and x is $n \times 1$
  - *Overdetermined* if $m > n$: "more equations than unknowns"
  - *Underdetermined* if $n > m$: "more unknowns than equations" Can look for best solution using least squares

# Singular Systems

- A is singular if some row is linear combination of other rows
- Singular systems can be underdetermined:

$$2x_1 + 3x_2 = 5$$
$$4x_1 + 6x_2 = 10$$

or inconsistent:

$$2x_1 + 3x_2 = 5$$
$$4x_1 + 6x_2 = 11$$

# Inverting a Matrix

- Usually not a good idea to compute $x=A^{-1}b$
  - Inefficient
  - Prone to roundoff error
- In fact, compute inverse using linear solver
  - Solve $Ax_i=b_i$ where $b_i$ are columns of identity, $x_i$ are columns of inverse
  - Many solvers can solve several R.H.S. at once

# Gauss-Jordan Elimination

- Fundamental operations:
  1. Replace one equation with linear combination of other equations
  2. Interchange two equations
  3. Re-label two variables
- Combine to reduce to trivial system
- Simplest variant only uses #1 operations, but get better stability by adding #2 (partial pivoting) or #2 and #3 (full pivoting)

# Gauss-Jordan Elimination

- Solve:

$$\begin{aligned} 2x_1 + 3x_2 &= 7 \\ 4x_1 + 5x_2 &= 13 \end{aligned}$$

- Only care about numbers – form "tableau" or "augmented matrix":

$$\left[\begin{array}{cc|c} 2 & 3 & 7 \\ 4 & 5 & 13 \end{array}\right]$$

# Gauss-Jordan Elimination

- Given:

$$\left[\begin{array}{cc|c} 2 & 3 & 7 \\ 4 & 5 & 13 \end{array}\right]$$

- Goal: reduce this to trivial system

$$\left[\begin{array}{cc|c} 1 & 0 & ? \\ 0 & 1 & ? \end{array}\right]$$

and read off answer from right column

# Gauss-Jordan Elimination

$$\left[\begin{array}{cc|c} 2 & 3 & 7 \\ 4 & 5 & 13 \end{array}\right]$$

- Basic operation 1: replace any row by linear combination with any other row
- Here, replace row1 with $^1/_2$ * row1 + 0 * row2

$$\left[\begin{array}{cc|c} 1 & \frac{3}{2} & \frac{7}{2} \\ 4 & 5 & 13 \end{array}\right]$$

# Gauss-Jordan Elimination

$$\left[\begin{array}{cc|c} 1 & \frac{3}{2} & \frac{7}{2} \\ 4 & 5 & 13 \end{array}\right]$$

- Replace row2 with row2 – 4 * row1

$$\left[\begin{array}{cc|c} 1 & \frac{3}{2} & \frac{7}{2} \\ 0 & -1 & -1 \end{array}\right]$$

- Negate row2

$$\left[\begin{array}{cc|c} 1 & \frac{3}{2} & \frac{7}{2} \\ 0 & 1 & 1 \end{array}\right]$$

# Gauss-Jordan Elimination

$$\left[\begin{array}{cc|c} 1 & 3/2 & 7/2 \\ 0 & 1 & 1 \end{array}\right]$$

- Replace row1 with row1 – $^3/_2$ * row2

$$\left[\begin{array}{cc|c} 1 & 0 & 2 \\ 0 & 1 & 1 \end{array}\right]$$

- Read off solution: $x_1 = 2,\ x_2 = 1$

# Gauss-Jordan Elimination

- For each row i:
  - Multiply row i by $1/a_{ii}$
  - For each other row j:
    - Add $-a_{ji}$ times row i to row j
- At the end, left part of matrix is identity, answer in right part
- Can solve any number of R.H.S. simultaneously

# Pivoting

- Consider this system:

$$\left[\begin{array}{cc|c} 0 & 1 & 2 \\ 2 & 3 & 8 \end{array}\right]$$

- Immediately run into problem: algorithm wants us to divide by zero!
- More subtle version:

$$\left[\begin{array}{cc|c} 0.001 & 1 & 2 \\ 2 & 3 & 8 \end{array}\right]$$

# Pivoting

- Conclusion: small diagonal elements bad
- Remedy: swap in larger element from somewhere else

# Partial Pivoting

$$\left[\begin{array}{cc|c} 0 & 1 & 2 \\ 2 & 3 & 8 \end{array}\right]$$

- Swap rows 1 and 2:

$$\left[\begin{array}{cc|c} 2 & 3 & 8 \\ 0 & 1 & 2 \end{array}\right]$$

- Now continue:

$$\left[\begin{array}{cc|c} 1 & 3/2 & 4 \\ 0 & 1 & 2 \end{array}\right] \qquad \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 2 \end{array}\right]$$

# Full Pivoting

$$\left[\begin{array}{cc|c} 0 & 1 & 2 \\ 2 & 3 & 8 \end{array}\right]$$

- Swap largest element onto diagonal by swapping rows 1 and 2 and columns 1 and 2:

$$\left[\begin{array}{cc|c} 3 & 2 & 8 \\ 1 & 0 & 2 \end{array}\right]$$

- Critical: when swapping columns, must remember to swap results!

# Full Pivoting

$$\left[\begin{array}{cc|c} 3 & 2 & 8 \\ 1 & 0 & 2 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 1 & 2/3 & 8/3 \\ 0 & -2/3 & -2/3 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 1 & 0 & 2 \\ 0 & 1 & 1 \end{array}\right]$$

* Swap results 1 and 2

- Full pivoting more stable, but only slightly

# Operation Count

- For one R.H.S., how many operations?
- For each of n rows:
  - Do n times:
    - For each of n+1 columns:
      - One add, one multiply
- Total = $n^3+n^2$ multiplies, same # of adds
- Asymptotic behavior: when n is large, dominated by $n^3$

# Faster Algorithms

- Our goal is an algorithm that does this in $^1/_3\, n^3$ operations, and does not require all R.H.S. to be known at beginning
- Before we see that, let's look at a few special cases that are even faster

# Tridiagonal Systems

- Common special case:

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & 0 & 0 & \cdots & b_1 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & b_2 \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & b_3 \\ 0 & 0 & a_{43} & a_{44} & \cdots & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right]$$

- Only main diagonal + 1 above and 1 below

# Solving Tridiagonal Systems

- When solving using Gauss-Jordan:
  - Constant # of multiplies/adds in each row
  - Each row only affects 2 others

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & 0 & 0 & \cdots & b_1 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & b_2 \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & b_3 \\ 0 & 0 & a_{43} & a_{44} & \cdots & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right]$$

# Running Time

- 2n loops, 4 multiply/adds per loop (assuming correct bookkeeping)
- This running time has a fundamentally different dependence on n: linear instead of cubic
  - Can say that tridiagonal algorithm is $O(n)$ while Gauss-Jordan is $O(n^3)$

# Big-O Notation

- Informally, $O(n^3)$ means that the dominant term for large n is cubic
- More precisely, there exist a c and $n_0$ such that

  $$\text{running time} \leq c\, n^3$$

  if

  $$n > n_0$$

- This type of *asymptotic analysis* is often used to characterize different algorithms

# Triangular Systems

- Another special case: A is lower-triangular

$$\left[\begin{array}{ccccc|c} a_{11} & 0 & 0 & 0 & \cdots & b_1 \\ a_{21} & a_{22} & 0 & 0 & \cdots & b_2 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right]$$

# Triangular Systems

- Solve by forward substitution

$$\left[\begin{array}{ccccc|c} a_{11} & 0 & 0 & 0 & \cdots & b_1 \\ a_{21} & a_{22} & 0 & 0 & \cdots & b_2 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right]$$

$$x_1 = \frac{b_1}{a_{11}}$$

# Triangular Systems

- Solve by forward substitution

$$\left[\begin{array}{ccccc|c} a_{11} & 0 & 0 & 0 & \cdots & b_1 \\ a_{21} & a_{22} & 0 & 0 & \cdots & b_2 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right]$$

$$x_2 = \frac{b_2 - a_{21}x_1}{a_{22}}$$

# Triangular Systems

- Solve by forward substitution

$$\left[\begin{array}{ccccc|c} a_{11} & 0 & 0 & 0 & \cdots & b_1 \\ a_{21} & a_{22} & 0 & 0 & \cdots & b_2 \\ a_{31} & a_{32} & a_{33} & 0 & \cdots & b_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & b_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right]$$

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}$$

# Triangular Systems

- If A is upper triangular, solve by backsubstitution

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & b_1 \\ 0 & a_{22} & a_{23} & a_{24} & a_{25} & b_2 \\ 0 & 0 & a_{33} & a_{34} & a_{35} & b_3 \\ 0 & 0 & 0 & a_{44} & a_{45} & b_4 \\ 0 & 0 & 0 & 0 & a_{55} & b_5 \end{array}\right]$$

$$x_5 = \frac{b_5}{a_{55}}$$

# Triangular Systems

- If A is upper triangular, solve by backsubstitution

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & b_1 \\ 0 & a_{22} & a_{23} & a_{24} & a_{25} & b_2 \\ 0 & 0 & a_{33} & a_{34} & a_{35} & b_3 \\ 0 & 0 & 0 & a_{44} & a_{45} & b_4 \\ 0 & 0 & 0 & 0 & a_{55} & b_5 \end{array}\right]$$

$$x_4 = \frac{b_4 - a_{45}x_5}{a_{44}}$$

# Triangular Systems

- Both of these special cases can be solved in $O(n^2)$ time
- This motivates a factorization approach to solving arbitrary systems:
  - Find a way of writing A as LU, where L and U are both triangular
  - $Ax=b \quad \Rightarrow \quad LUx=b \quad \Rightarrow \quad Ly=b \quad \Rightarrow \quad Ux=y$
  - Time for factoring matrix dominates computation

# Cholesky Decomposition

- For symmetric matrices, choose $U=L^T$
- Perform decomposition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

- $Ax=b \quad \Rightarrow \quad LL^Tx=b \quad \Rightarrow \quad Ly=b \quad \Rightarrow \quad L^Tx=y$

# Cholesky Decomposition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

$$l_{11}^{\ 2} = a_{11} \quad \Rightarrow \quad l_{11} = \sqrt{a_{11}}$$

$$l_{11} l_{21} = a_{12} \quad \Rightarrow \quad l_{21} = \frac{a_{12}}{l_{11}}$$

$$l_{11} l_{31} = a_{13} \quad \Rightarrow \quad l_{31} = \frac{a_{13}}{l_{11}}$$

$$l_{21}^{\ 2} + l_{22}^{\ 2} = a_{22} \quad \Rightarrow \quad l_{22} = \sqrt{a_{22} - l_{21}^{\ 2}}$$

$$l_{21} l_{31} + l_{22} l_{32} = a_{23} \quad \Rightarrow \quad l_{32} = \frac{a_{23} - l_{21} l_{31}}{l_{22}}$$

# Cholesky Decomposition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

$$l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}}{l_{ii}}$$

# Cholesky Decomposition

- This fails if it requires taking square root of a negative number
- Need another condition on A: positive definite

For any v, $v^T A v > 0$

(Equivalently, all positive eigenvalues)

# Cholesky Decomposition

- Running time turns out to be $^{1}/_{6}n^{3}$
  - Still cubic, but much lower constant
- Result: this is preferred method for solving symmetric positive definite systems

# LU Decomposition

- Again, factor A into LU, where
  L is lower triangular and U is upper triangular

$$
\begin{gathered}
Ax = b \\
LUx = b \\
Ly = b \\
Ux = y
\end{gathered}
$$

- Last 2 steps in $O(n^2)$ time, so total time dominated by decomposition

# Doolittle Factorization

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- More unknowns than equations!
- Let all $l_{ii}=1$
  (Could also take all $u_{ii}=1$ – Crout's method)

# Doolittle Factorization

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$u_{11} = a_{11}$$

$$l_{21}u_{11} = a_{21} \quad \Rightarrow \quad l_{21} = \frac{a_{21}}{u_{11}}$$

$$l_{31}u_{11} = a_{31} \quad \Rightarrow \quad l_{31} = \frac{a_{31}}{u_{11}}$$

$$u_{12} = a_{12}$$

$$l_{21}u_{12} + u_{22} = a_{22} \quad \Rightarrow \quad u_{22} = a_{22} - l_{21}u_{12}$$

$$l_{31}u_{12} + l_{32}u_{22} = a_{32} \quad \Rightarrow \quad l_{32} = \frac{a_{32} - l_{31}u_{12}}{u_{22}}$$

# Doolittle Factorization

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- For i = 1..n
  - For j = 1..i

$$u_{ji} = a_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki}$$

  - For j = i+1..n

$$l_{ji} = \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ki}}{u_{ii}}$$

# Doolittle Factorization

- Interesting note: # of outputs = # of inputs, algorithm only refers to elements not output yet
  - Can do this in-place!
  - Algorithm replaces A with matrix of l and u values, 1s are implied

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ l_{21} & u_{22} & u_{23} \\ l_{31} & l_{32} & u_{33} \end{bmatrix}$$

  - Resulting matrix must be interpreted in a special way: not a regular matrix
  - Can rewrite forward/backsubstitution routines to use this "packed" l-u matrix

# LU Decomposition

- Running time is $^1/_3 n^3$
  - Only a factor of 2 slower than symmetric case
  - This is the preferred general method for solving linear equations
- Pivoting very important
  - Partial pivoting is sufficient, and widely implemented
  - LU with pivoting can succeed even if matrix is singular (!) (but back/forward substitution fails…)

# Running Time – Is $O(n^3)$ the Limit?

- How fast is matrix multiplication?

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

- 8 multiples, 4 adds, right?
  (In general $n^3$ multiplies and $n^2(n-1)$ adds…)

# Running Time – Is O(n3) the Limit?

- Strassen's method [1969]

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Volker Strassen

$$M_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$M_2 = (a_{21} + a_{22})b_{11}$$
$$M_3 = a_{11}(b_{11} - b_{22})$$
$$M_4 = a_{22}(b_{21} - b_{11})$$
$$M_5 = (a_{11} + a_{12})b_{22}$$
$$M_6 = (a_{21} - a_{11})(b_{11} + b_{12})$$
$$M_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$
$$c_{11} = M_1 + M_4 - M_5 + M_7$$
$$c_{12} = M_3 + M_5$$
$$c_{21} = M_2 + M_4$$
$$c_{22} = M_1 - M_2 + M_3 + M_6$$

# Running Time – Is O(n3) the Limit?

- Strassen's method [1969]

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

- Uses only 7 multiplies (and a whole bunch of adds)
- Can be applied recursively!

$$M_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$M_2 = (a_{21} + a_{22})b_{11}$$
$$M_3 = a_{11}(b_{11} - b_{22})$$
$$M_4 = a_{22}(b_{21} - b_{11})$$
$$M_5 = (a_{11} + a_{12})b_{22}$$
$$M_6 = (a_{21} - a_{11})(b_{11} + b_{12})$$
$$M_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$
$$c_{11} = M_1 + M_4 - M_5 + M_7$$
$$c_{12} = M_3 + M_5$$
$$c_{21} = M_2 + M_4$$
$$c_{22} = M_1 - M_2 + M_3 + M_6$$

# Running Time – Is $O(n^3)$ the Limit?

- Recursive application for 4 half-size submatrices needs 7 half-size matrix multiplies
- Asymptotic running time is $O(n^{\log_2 7}) \approx O(n^{2.8})$
  - Only worth it for large *n*, because of big constant factors (all those additions…)
  - Still, practically useful for n > hundreds or thousands
- Current state of the art: Coppersmith-Winograd algorithm achieves $O(n^{2.376...})$
  - Not used in practice

# Running Time – Is $O(n^3)$ the Limit?

- Similar sub-cubic algorithms for inverse, determinant, LU, etc.
  - Most "cubic" linear-algebra problems aren't!

- Major open question: what is the limit?
  - Hypothesis: $O(n^2)$ or $O(n^2 \log n)$