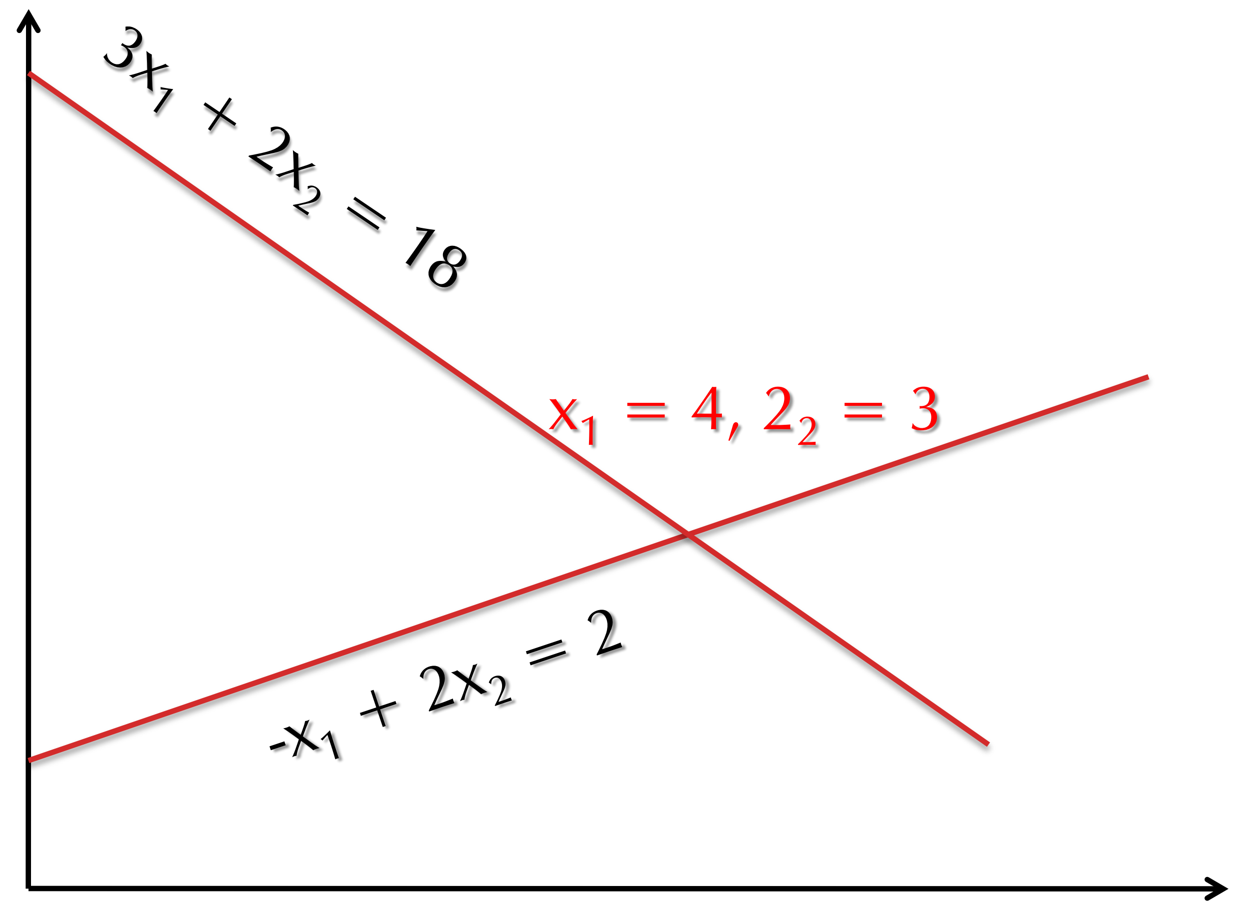
Linear Systems

COS 323

Solving Linear Systems of Equations

- Define linear system
- Singularities in linear systems
- Gaussian Elimination: A general purpose method
  - Naïve Gauss
  - Gauss with pivoting
  - Asymptotic analysis
- Triangular systems and LU decomposition
- Special matrices and algorithms:
  - Symmetric positive definite: Cholesky decomposition
  - Tridiagonal matrices
- Singularity detection and condition numbers

Graphical interpretation



Linear Systems

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

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

Linear Systems

- **Solve for x** given 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"

    Can look for best solution using **least squares**
  - *Underdetermined* if $n > m$:
    "more unknowns than equations"

Singular Systems

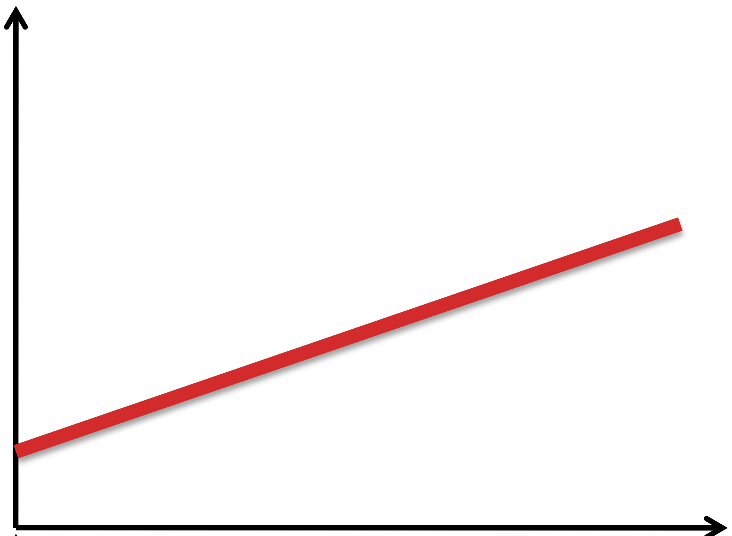
- A is *singular* if some row is a linear combination of other rows
- Singular systems can be underdetermined:

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

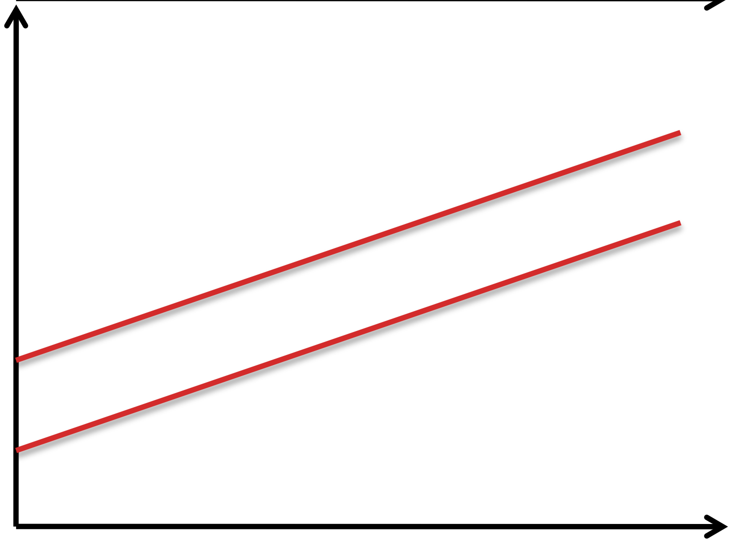
or inconsistent:

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

Graphical Interpretation

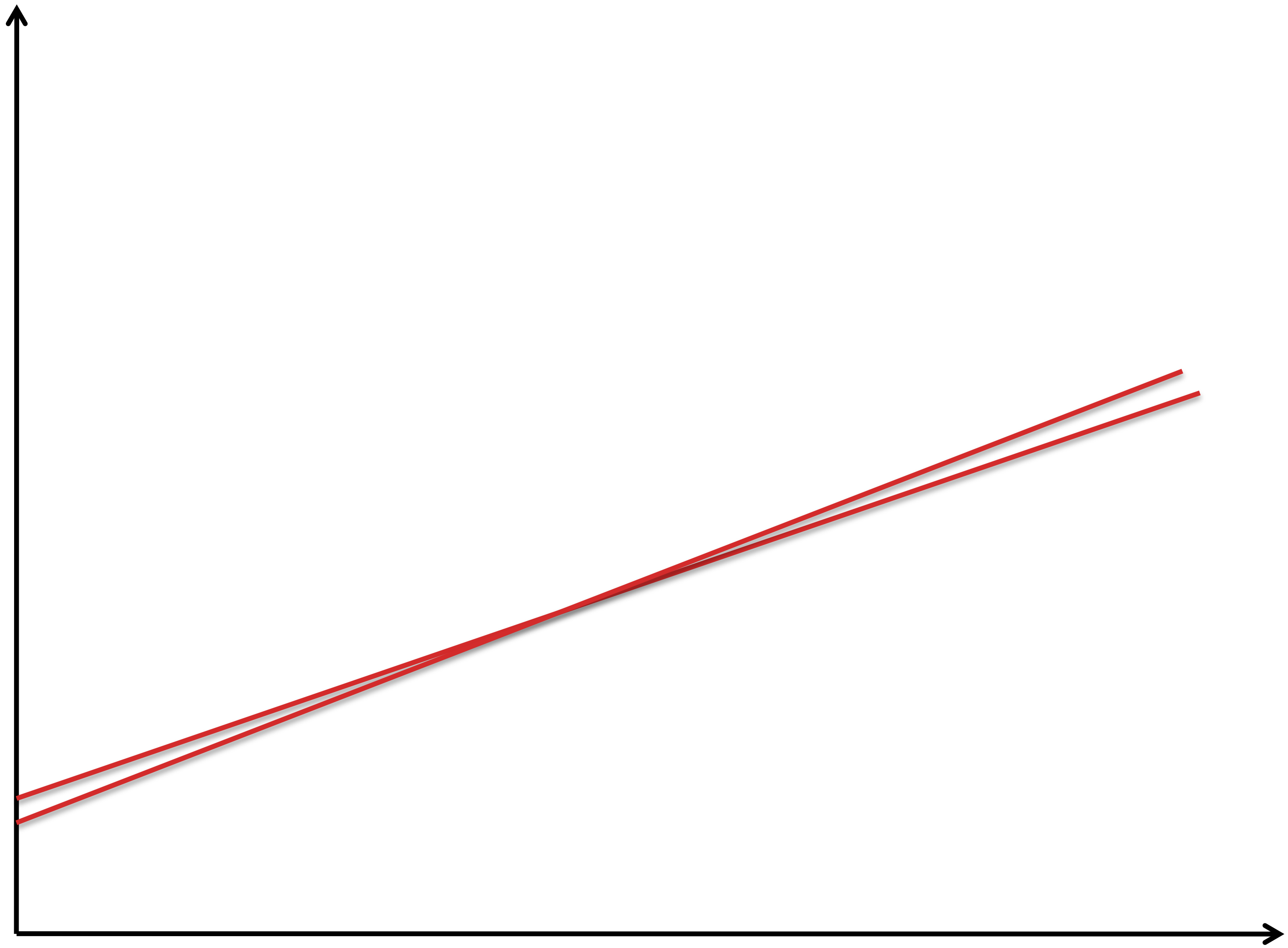


Singular with infinite solutions



Singular with no solution

Near-Singular or Ill-Conditioned



Why not just invert A?

- $x=A^{-1}b$
  - BUT: Inefficient
  - Prone to roundoff error
- In fact, compute inverse using linear solver

Solve by hand…

$$3x_1 + 2x_2 = 18$$

$$-x_1 + 2x_2 = 2$$

$$0x_1 + 8x_2 = 24 \rightarrow x_2 = 3$$

$$-x_1 + 2 * 3 = 2 \rightarrow x_1 = 4$$

Gaussian 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 (identity)
  - Alternative: triangular system + *back-substitution*
- Simplest variant only uses #1 operations, but get better stability by adding #2 (partial pivoting) or #2 & #3 (full pivoting)

"Naïve" Gaussian 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":

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

"Naïve" Gaussian Elimination

- Given:

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

- 1) Elimination: reduce this to system of form

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

- 2) Back-substitution: Solve for x_2, then "plug in" to solve for x_1

"Naïve" Gaussian Elimination: Forward elimination stage

$$\left[\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{array}\right] = \left[\begin{array}{cc|c} 2 & 3 & 7 \\ 4 & 5 & 13 \end{array}\right]$$

- 1. Define $f = a_{21}/a_{11}$ (here, $f = 2$)
- 2. Replace 2^{nd} row r_2 with $r_2 - (f * r_1)$

Here, replace r_2 with $r_2 - 2 * r_1$

$$\left[\begin{array}{cc|c} a_{11} & a_{12} & b_1 \\ 0 & a'_{22} & b'_2 \end{array}\right] = \left[\begin{array}{cc|c} 2 & 3 & 7 \\ 0 & -1 & -1 \end{array}\right]$$

Forward elimination pseudocode

```
for k=1 to n-1 {  // Loop over all rows
    for i=(k+1) to n { // Loop over all rows beneath kth
        factor_ik ← a_ik / a_kk
        for j = k to n { // Loop over elements in the row
            a_ij ← a_ij – factor_ik * a_kj   // Update element
        }
    }
}
```

Outcome of forward elimination

$$
\begin{array}{ccccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \mathrm{K} & + & a_{1n}x_n & = & b_1 \\
 & & a'_{22}x_2 & + & a'_{23}x_3 & + & \mathrm{K} & + & a'_{2n}x_n & = & b'_2 \\
 & & & & a''_{33}x_3 & + & \mathrm{K} & + & a''_{3n}x_n & = & b''_3 \\
 & & & & & \cdot & & & & & \cdot \\
 & & & & & & \cdot & & & & \cdot \\
 & & & & & & & \cdot & & & \cdot \\
 & & & & & & & & a_{nn}^{(n-1)}x_n & = & b_n^{(n-1)}
\end{array}
$$

Back-substitution Pseudocode

$x_n = b_n / a_{nn}$

for i = (n-1) to 1 ***descending*** {

  sum $\leftarrow b_i$

  for j = (i+1) to n {

    sum $\leftarrow$ sum – a_{ij} * x_j

  }

  $x_i \leftarrow$ sum / a_{ii}

}

Questions?

What could go wrong?

```
for k=1 to n-1 {  // Loop over all rows
    for i=(k+1) to n { // Loop over all rows beneath kth
        factor_ik ← a_ik / a_kk
        for j = k to n { // Loop over elements in the row
            a_ij ← a_ij – factor_ik * a_kj   // Update element
        }
    }
}
```

What could go wrong?

$x_n = b_n / a_{nn}$

for $i = (n\text{-}1)$ to 1 ***descending*** {

  sum $\leftarrow b_i$

  for $j = (i+1)$ to n {

    sum $\leftarrow$ sum $- a_{ij} * x_j$

  }

  $x_i \leftarrow$ sum $/ a_{ii}$

}

Small pivot element example

$$0.0003x_1 + 3.0000x_2 = 2.0001$$

$$1.0000x_1 + 1.0000x_2 = 1.0000$$

After pivot, equation 2 becomes

$$-9999x_2 = -6666$$

Solve for $x_2 = 2/3$

Solve for $x_1 = (2.0001 - 3\,(2/3))\,/\,.0003$

$\rightarrow x_1 =$ -3.33 or 0.0000 or 0.330000

(depending on # digits used to represent 2/3)

Partial Pivoting

Swap rows to pivot on largest element possible (i.e., put large numbers in the diagonal):

$$0.0003x_1 + 3.0000x_2 = 2.0001$$

$$1.0000x_1 + 1.0000x_2 = 1.0000$$

becomes

$$1.0000x_1 + 1.0000x_2 = 1.0000$$

$$0.0003x_1 + 3.0000x_2 = 2.0001$$

Partial pivot applied

$$1.0000x_1 + 1.0000x_2 = 1.0000$$

$$0.0003x_1 + 3.0000x_2 = 2.0001$$

Factor = .0003/1.0000, so Equation 2 becomes

$$2.9997\ x_2 = -1.9998$$

Solve for $x_2 = 2/3$

Solve for $x_1 = (1.0000 - 1 * (2/3)) / 1.0$

$\rightarrow x_1 = 0.333$ or 0.3333 or 0.333333

(depending on # digits used to represent 2/3)

Full Pivoting

- Swap largest element onto diagonal by swapping **rows and columns**
- More stable, but only slightly

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

Questions on Gaussian Elimination?

Complexity of Gaussian Elimination

- Forward elimination:

  $2/3 * n^3 + O(n^2)$

  (triple for-loops yield n^3)

- Back substitution:

  $n^2 + O(n)$

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

LU Decomposition

Triangular Systems are nice!

- 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
  - $Ax=b \Rightarrow LUx=b \Rightarrow Ld=b \Rightarrow Ux=d$
  - Time for **factoring matrix** dominates computation

Solving Ax = b with LU Decomposition of A

$$Ax = b$$

$$LU$$

$$Ld = b$$

$$d$$

$$Ux = d$$

$$x$$

A = LU

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

- More unknowns than equations!
- Let all $l_{ii}=1$ (Doolittle's method)

  or let all $u_{ii}=1$ (Crout's method)

Doolittle Factorization for LU Decomposition

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

- U is result of forward elimination step of Gauss
- L elements are the factors computed in forward elimination!
  - e.g. $l_{21} = f_{21} = a_{21} / a_{11}$ and $l_{32} = f_{32} = a'_{32} / a'_{22}$

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 of A, not b
- 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 $^2/_3 n^3$
  - Independent of RHS, each of which requires $O(n^2)$ back/forward substitution
  - 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…)

Matrix Inversion using LU

- LU depend only on A, not on b
- Re-use L & U for multiple values of b
  - i.e., repeat back-substitution
- How to compute A^{-1}?

  $AA^{-1} = \mathbf{I}$ **(n×n identity matrix),** e.g.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\rightarrow$ Use LU decomposition with

$$b_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Questions on LU Decomposition?

Working with Special Matrices

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 Gaussian elimination:
  - 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 is $O(n^3)$
- In general, a banded system of bandwith w requires $O(wn)$ storage and $O(w^2n)$ computations.

Symmetric matrices: Cholesky Decomposition

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

  $(A = LL^T)$

- Perform decomposition

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

- $Ax = b \quad \Rightarrow \quad LL^Tx = b \quad \Rightarrow \quad Ld = b \quad \Rightarrow \quad L^Tx = d$

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

  i.e., For any v, $v^T A v > 0$

  (Equivalently, all positive eigenvalues)

Cholesky Decomposition

- Running time turns out to be $^1/_6 n^3$ multiplications + $^1/_6 n^3$ additions
  - Still cubic, but lower constant
  - Half as much computation & storage as LU
- Result: this is preferred method for solving symmetric positive definite systems

Running time revisited

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 $O(n^3)$ 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 $O(n^3)$ 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?

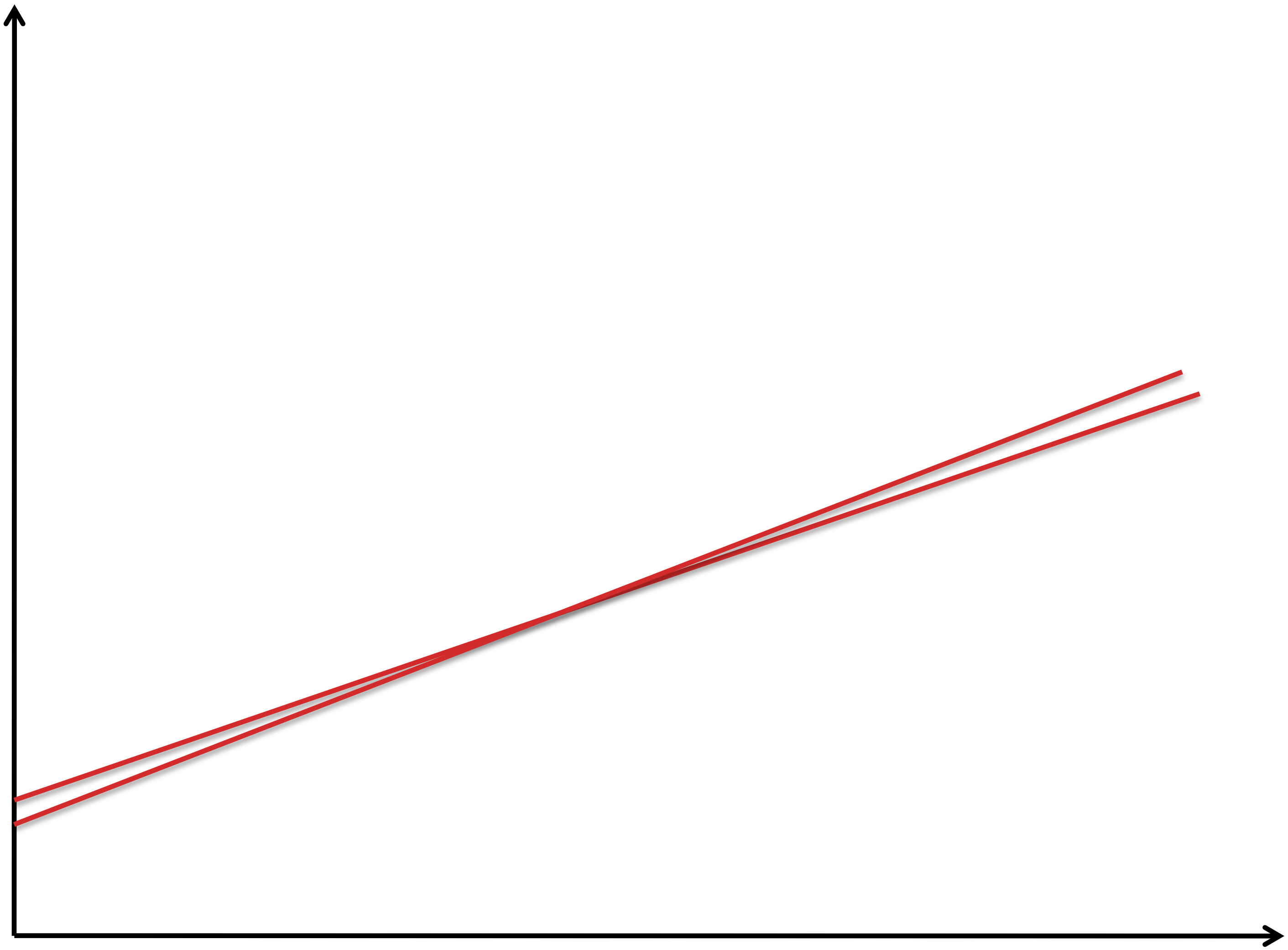
- Recursive application for 4 half-size submatrices needs 7 half-size matrix multiplies
- Asymptotic running time is $O(n^{\log_2 7}) \approx O(n^{2.8})$
  - Only worth it for large *n*, because of big constant factors (all those additions…)
  - Still, **practically useful** for n > hundreds or thousands
- 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 $O(n^2 \log n)$

Singularity and Condition Number

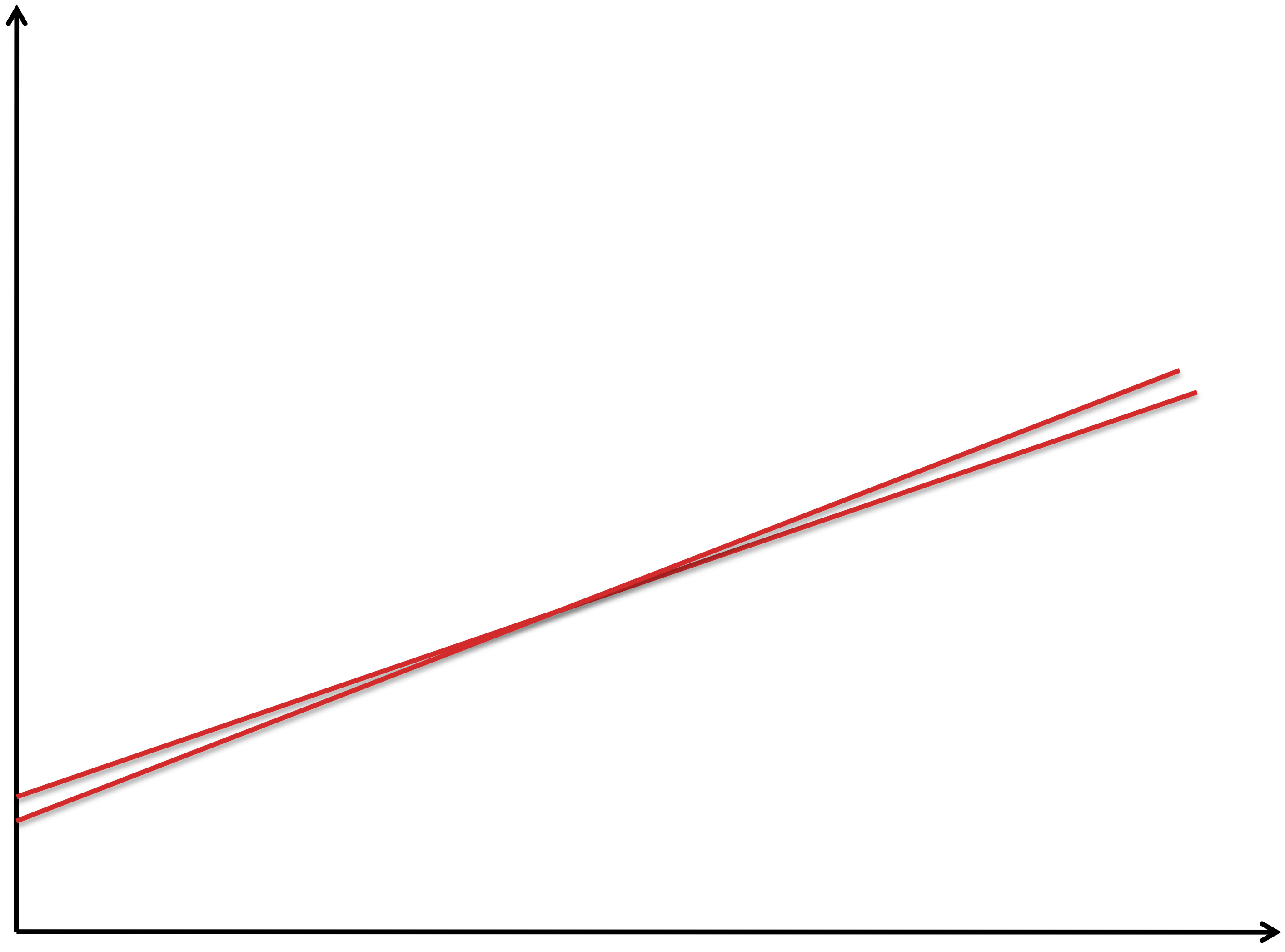
A near-singular system



Detecting singularity and near-singularity

- Graph it! (in 2 or 3 dimensions)
- Does $A A^{-1} = \mathbf{I}$ (identity) ?
- Does $(A^{-1})^{-1} = A$?
- Does $Ax = b$?
- Does $(A^{-1})_{c1} = (A^{-1})_{c2}$ for compilers c1, c2?
- Are any of LU diagonals (with pivoting) near-zero?

A near-singular system



Condition number

- Cond(A) is function of A
- Cond(A) ≥ 1, bigger is **bad**
- Measures how change in input is propogated to change in output

$$\frac{\| \Delta x \|}{\| x \|} \leq cond(A) \frac{\| \Delta A \|}{\| A \|}$$

- E.g., if cond(A) = 451 then can lose log(451)= 2.65 digits of accuracy in x, compared to precision of A

Computing condition number

- cond(A) = ||A|| ||A^{-1}||
- where ||M|| is a matrix norm

$$\|M\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|, \quad \|M\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$$

$$\|M\|_2 = (\lambda_{\max})^{1/2}$$ (using largest eigenvalue of A^{T}A)

- ||M||$_{inf}$ is often easiest to compute
- Different norms give different values, but similar order of magnitude

Useful Matlab functions

- \ : matrix division
  - e.g. x = A\b
- **cond**: matrix condition number
- **norm**: matrix or vector norm
- **chol** : Cholesky factorization
- **lu** : LU decomposition
- **inv**: inverse (don't use unless you really need the inverse!)
- **rank**: # of linearly independent rows or columns
- **det**: determinant
- **trace**: sum of diagonal elements
- **null**: null space

Other resources

- Heath interactive demos:
  - http://www.cse.illinois.edu/iem/linear_equations/gaussian_elimination/
  - http://www.cse.illinois.edu/iem/linear_equations/conditioning/
- http://www.math.ucsd.edu/~math20f/Spring/Lab2/Lab2.shtml
  - Good reading on how linear systems can be used in web recommendation (Page Rank) and economics (Leontief Models)