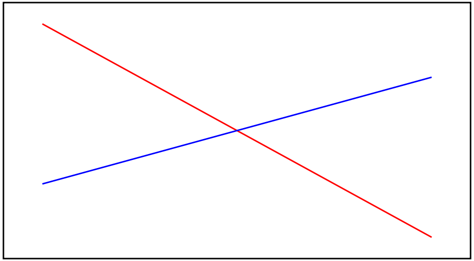
Linear Systems

Szymon Rusinkiewicz
COS 302, Fall 2020

PRINCETON UNIVERSITY

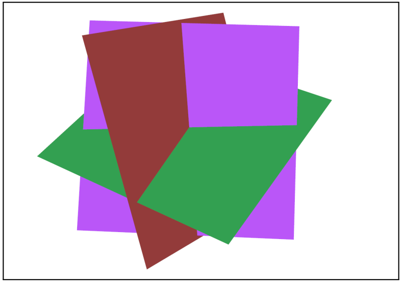
Linear Systems

- Simultaneously satisfy a set of *linear* equations
- In 2D, a linear equation in 2 variables defines a line
  - 2 equations *might* intersect in 1 point, giving a unique solution



Linear Systems

- Simultaneously satisfy a set of *linear* equations
- In 3D, a linear equation in 3 variables defines a plane
  - 3 equations *might* intersect in 1 point, giving a unique solution



Applications of Linear Systems

- Regression (“fitting a model to data”)
- Simulation (e.g., mass-spring systems)
- Analysis (e.g., how much stress is there in a beam in a building or bridge)
- Subroutine in other numerical algorithms (e.g., Newton’s method for optimization or implicit Euler method for differential equations)

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 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $n \times n$ matrix and $\boldsymbol{b}$ is an $n \times 1$ column vector
- Can also talk about non-square systems where $\boldsymbol{A}$ is $m \times n$, $\boldsymbol{b}$ is $m \times 1$, and $\boldsymbol{x}$ is $n \times 1$
  - Usually *overdetermined* if $m > n$: "more constraints than unknowns"
    (Can look for "best" solution using *least squares*.)
  - *Underdetermined* if $n > m$: "more unknowns than constraints"
    (Can compute all solutions, as in textbook, but it is more common to look for "best" solution using *regularization*.)

Singular Systems

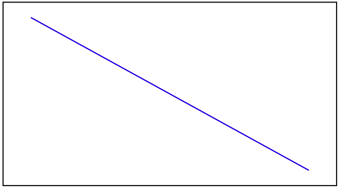
- A square matrix A is *singular* if some row is linear combination of other rows
- Singular systems might have infinitely many solutions:

$$\begin{aligned} 2x_1 + 3x_2 &= 5 \\ 4x_1 + 6x_2 &= 10 \end{aligned}$$

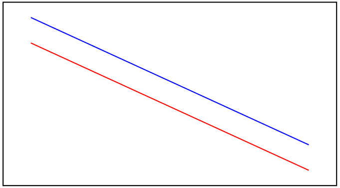
or no solutions:

$$\begin{aligned} 2x_1 + 3x_2 &= 5 \\ 4x_1 + 6x_2 &= 11 \end{aligned}$$

Singular Systems

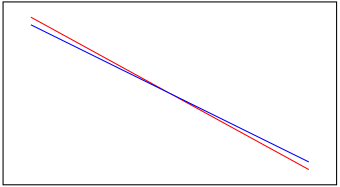


Singular with infinite solutions



Singular with no solutions

Near-Singular Systems



Near-singular or ill-conditioned: noise in inputs, or roundoff error in computation, may result in large changes to solution

Solving Linear Systems

- Seemingly the most direct way to solve a well-determined, square system is to use the matrix *inverse*:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} &= \boldsymbol{b} \\ \boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{A}^{-1}\boldsymbol{b} \\ \boldsymbol{x} &= \boldsymbol{A}^{-1}\boldsymbol{b} \end{aligned}$$

Notes:

1. The inverse of a square matrix need not exist.
   But if it does, it is unique and has the property $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$.
2. Matrix multiplication is associative, so $\boldsymbol{A}^{-1}(\boldsymbol{A}\boldsymbol{x}) = (\boldsymbol{A}^{-1}\boldsymbol{A})\boldsymbol{x} = \boldsymbol{I}\boldsymbol{x} = \boldsymbol{x}$.
3. Matrix multiplication is not commutative, so we were careful to multiply both $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{b}$ on the *left* by $\boldsymbol{A}^{-1}$.

Inverses and Linear Systems

- In fact, using $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, and computing the inverse, is usually a bad idea:
  - Inefficient
  - Prone to roundoff error
- Linear solver algorithms
  - *Direct:* nested loops over matrix, get solution at end
  - *Iterative:* get approximate answer, then each iteration improves it
- In fact compute inverse using linear solver
  - Solve $\boldsymbol{A}\boldsymbol{x}_i = \boldsymbol{b}_i$, where $\boldsymbol{b}_i$ are columns of identity, $\boldsymbol{x}_i$ are columns of inverse
  - Many solvers can solve several Right Hand Sides (RHS) at once

Gauss-Jordan or Gaussian Elimination

- Simple-to-understand *direct* solver (though not used in practice)
- Transforms matrix to *reduced row-echelon form*[*], while simultaneously manipulating one or more right-hand side(s)

  [*]First nonzero entry in each row is 1, it's to the right of the 1 in the row above, and it's the only nonzero entry in that column.
- Fundamental operations:
  1. Replace one row with linear combination of it and other rows
  2. Interchange two rows
  3. Re-label two variables (interchange two columns)
- Simplest variant uses only #1 operations but numerical stability improved by adding #2 (partial pivoting) or both #2 and #3 (full pivoting).

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":

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

- Could have multiple right-hand sides (e.g. for computing an inverse)

Gaussian Elimination

- Given:

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

- Goal: reduce this to the following form:

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

and read off answer from right column

Gaussian Elimination

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

- Basic operation: replace any row by linear combination with any other row
- Here, replace first row with $1/2$ times first row plus 0 times second row:

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

Gaussian Elimination

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

- Replace second row with -4 times first row plus 1 times second row:

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

- Negate second row:

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

Gaussian Elimination

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

- Add $-3/2$ times second row to first row:

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

- …aaaaand we're done. This is in reduced row-echelon form!

Gaussian Elimination Analysis

- 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
- Innermost loop executed $n(n-1)$ times, and requires $n+1$ additions and multiplications
  - Asymptotic behavior: when n is large, that's about $2n$ arithmetic operations in inner loop, or about $2n^3$ total
- Can solve any number of RHS at once (but must be known ahead of time)

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:

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

- Small diagonal ("pivot") elements bad!
  - 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:

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

Real-World Linear Solvers

- In practice, partial pivoting widely implemented
- To speed things up, don't go all the way to reduced row-echelon form
  - Also, it would be nice to be able to specify a new $\boldsymbol{b}$ *after* the expensive computation on A has been done...

Triangular Systems

- Special case: *lower-triangular system*

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

Triangular Systems

- Solve by forward substitution

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

$$x_1 = \frac{b_1}{a_{11}} \quad x_2 = \frac{b_2 - a_{21}x_1}{a_{22}} \quad x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \quad \cdots$$

Triangular Systems

- Similarly, *upper triangular* systems solved by back-substitution

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & a_{14} & b_1 \\ 0 & a_{22} & a_{23} & a_{24} & b_2 \\ 0 & 0 & a_{33} & a_{34} & b_3 \\ 0 & 0 & 0 & a_{44} & b_4 \end{array}\right]$$

$$x_4 = \frac{b_4}{a_{44}} \qquad x_3 = \frac{b_3 - a_{34}x_4}{a_{33}} \qquad x_2 = \frac{b_2 - a_{24}x_4 - a_{23}x_3}{a_{22}} \qquad \ldots$$

LU Decomposition

- Both special cases can be solved in time $\sim n^2$
- This motives a factorization approach:
  - Find a way of writing $\boldsymbol{A}$ as $\boldsymbol{LU}$, where $\boldsymbol{L}$ is lower-triangular and $\boldsymbol{U}$ is upper-triangular
  - $\boldsymbol{Ax} = \boldsymbol{b} \implies \boldsymbol{LUx} = \boldsymbol{b} \implies \boldsymbol{Ly} = \boldsymbol{b} \implies \boldsymbol{Ux} = \boldsymbol{y}$
  - Time to factor matrix dominates computation
  - Turns out to be faster than Gaussian elimination (but still cubic in n)
  - Can solve for new $\boldsymbol{b}$ at any time after factorization
- Real-world, general-purpose linear solvers (such as `numpy.linalg.solve`) use LU Decomposition with partial pivoting
  - ...but for big ($n \sim$ thousands or millions) problems, approximate iterative methods are common