UNIVERSITY AT ALBANY
STATE UNIVERSITY OF NEW YORK

CSI 401 (Fall 2025)

Numerical Methods

Lecture 19: Course Review

Chong Liu

Department of Computer Science

Nov 24, 2025

Why learn Numerical Methods?

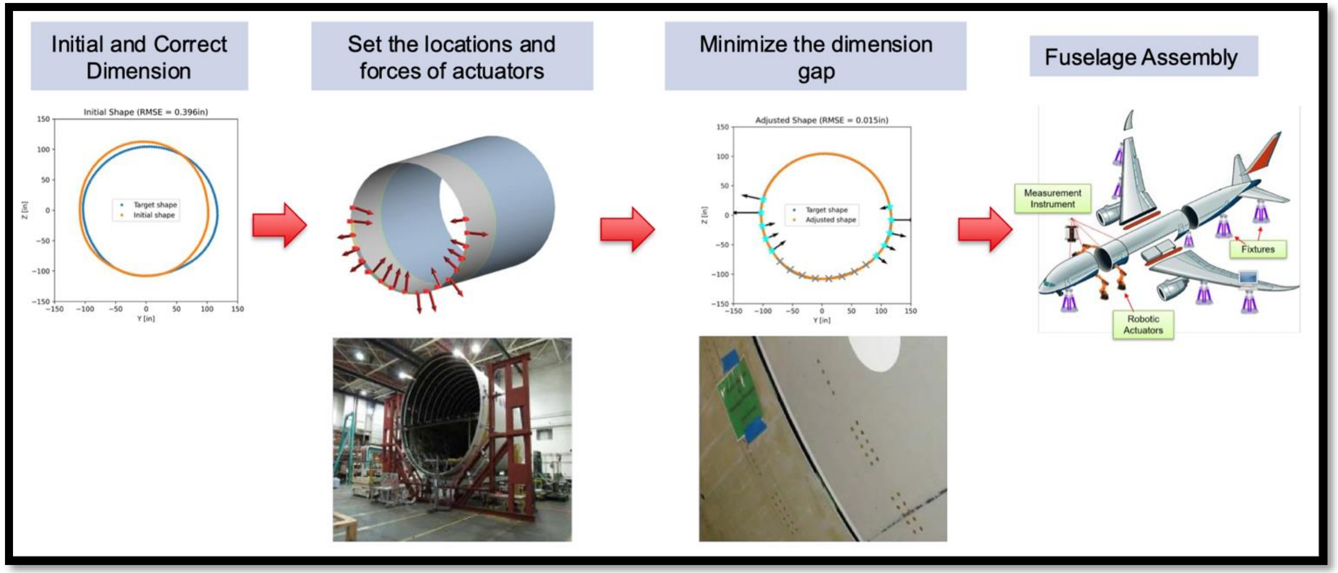
- Motivated by
  - most real-world problems in science, engineering, and data science **cannot** be solved exactly using **closed-form solutions**.
  - For example,
    - Many equations, such as nonlinear systems, high-dimensional integrals, and differential equations, either lack analytical solutions or are too complex to solve by hand.
- Advantages:
  - Providing systematic algorithms to approximate these solutions with controllable accuracy and efficiency
  - Bridging the gap between mathematical theory and computer implementation
  - Enabling the simulation and prediction of complex problems—such as climate modeling, structural design, or machine learning
  - Ensuring stability, convergence, and error control!

Topics of Numerical Methods covered

- Source of numerical errors
- Asymptotic notations and floating point arithmetic
- Review:
  - Linear algebra, Python, Matlab, LaTeX
- Linear systems:
  - Direct linear equations solvers
  - Eigenvalues and eigenvectors
  - Iterative linear equations solvers
  - Conditioning of linear equations
- Numerical interpolation
  - Data fitting and regression problems
- Nonlinear equations solvers
- Optimization methods
- Numerical integration and differentiation

Sources of numerical errors

- Data are always noisy
  - Discussion: Any example in your mind?
  - Aircraft fuselage design



- Computers can only handle discrete data
  - Think about data structure: string, array, tree, …

- No measuring device is perfect
  - Discussion: Any example in your mind?
  - Exact rainfall of Albany, NY in July 2025? No way!

Types of errors

- Discretization error: we can only deal with values of a function at finitely many points. For example, a very simple way to do numerical differentiation for a function f is to use a finite difference formula:

$$\hat{f}(x) = \frac{f(x+h) - f(x)}{h}. \tag{2.1}$$

  Here, the parameter h is some small number. It cannot be 0, so this introduces discretization error. Recall that the definition of the derivative of a function at a point x is

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{2.2}$$

  Later in the course, we'll considered better methods than this.

- Convergence error: in which we, say, truncate a power series expansion, stop an iterative algorithm after finitely many iterations, etc.

- Rounding error: This arises because computers have only finite precision. We can only store a finite amount of data in any given machine. Interestingly, in numerical differentiation, there is a tradeoff between discretization error and rounding error (since we cannot make h infinitely small), and this leads to some optimal choice of h! So multiple types of error can play an important role simultaneously in some problems.

One way is to take the absolute difference

- Definition of **absolute error**:
- $|u - v|$

- It's so simple, but it has some problems:
  - Suppose we try to figure out how much air passes through a Boeing 737 engine per minute during the flight.
  - The true answer is 120,000 pounds.
  - Your solution shows 119,000 pounds, so absolute error is 1,000 pounds.

  - Discussion: You missed 1,000 pounds? Are you doing a good job?

Another way to measure error

- Definition of relative error:
- $\left|\frac{u-v}{u}\right|$
  - note u is the true number

- Discussion: What’s the relative error in previous example?
  - Suppose we try to figure out how much air passes through a Boeing 737 engine per minute during the flight.
  - The true answer is 120,000 pounds.
  - Your solution shows 119,000 pounds.

Asymptotic notations

- Used to compare the growth of two functions $f(x)$ and $g(x)$ as x tends to some limit point x_0.

- Discussion:
  - $f(x) = x^2, g(x) = x$. Which notation shall we use?

To do this, we look at the absolute value of the ratio of the two:

$$\left|\frac{f(x)}{g(x)}\right|. \tag{2.6}$$

The behavior of this can be one of three different things as $x \to x_0$:

- $$\left|\frac{f(x)}{g(x)}\right| \to 0. \tag{2.7}$$

  In this case, we say that $f(x)$ is asymptotically negligible compared to $g(x)$. We also say that $f(x) = o(g(x))$ (i.e., "$f(x)$ is small 'oh' of $g(x)$") as $x \to x_0$.

- $$\left|\frac{f(x)}{g(x)}\right| \tag{2.8}$$

  converges to a positive constant or oscillates but stays bounded.

- $$\left|\frac{f(x)}{g(x)}\right| \to \infty. \tag{2.9}$$

  In this case, we say that $f(x)$ is asymptotically dominant compared to $g(x)$. This implies that $g(x) = o(f(x))$.

Asymptotic notations

- Additionally, we say that $f(x) = O(g(x))$ as $x \to x_0$ if there is some positive constant C such that
- $\left|\frac{f(x)}{g(x)}\right| \leqslant C$.

- Thus, the $O(\cdot)$ notation means that $f(x)$ is **asymptotically upper bounded** by $g(x)$.
- We also say that $f(x) = \Omega(g(x))$ if $g(x) = O(f(x))$.
- We say that $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

Properties of asymptotic notations

Theorem 3.2 (Properties of asymptotic notations). *Let $C > 0$ be some positive constant, and let $x_0 \in \mathbb{R} \cup \{\pm\infty\}$. Then, for any function $g(x)$, as $x \to x_0$,*

$$C \cdot O(g(x)) = O(g(x)) \tag{3.1}$$
$$C \cdot \Theta(g(x)) = \Theta(g(x)) \tag{3.2}$$
$$C \cdot \Omega(g(x)) = \Omega(g(x)) \tag{3.3}$$
$$C \cdot o(g(x)) = o(g(x)). \tag{3.4}$$

Additionally, for any $f(x)$

$$f(x)O(g(x)) = O(f(x)g(x)). \tag{3.5}$$

- The above theorem allows us to simplify expressions asymptotically. E.g., as $x \to \infty$
- $4e^x(\sin(x) + 5) + 3x = \Theta(e^x(\sin(x) + 5)) = \Theta(e^x)$,
- where the first equality is because $x = o(e^x \sin(x))$, and the second equality is because $0 \leqslant |\sin(x)| \leqslant 1$.
- Note that all of this can be verified by looking at ratios of functions, as in the definition of the notation.

Properties of polynomials

Corollary 3.4. *As $x \to \infty$, for any fixed k, nonzero constant c_k, and constants (possibly 0) c_j for $j \in \{0, 1, ..., k-1\}$,*

$$P(x) = c_k x^k + c_{k-1} x^{k-1} + ... + c_1 x + c_0 = \Theta(x^k). \tag{3.8}$$

As $x \to 0$, if j is the smallest number for which $c_j \neq 0$,

$$P(x) = \Theta(x^j). \tag{3.9}$$

Let us consider the following question: suppose $f(x) = \Theta(g(x))$. Is it true in general that $f(x) - g(x) = \Theta(g(x))$? **NO**. For instance,

$$f(x) = 3x, g(x) = 3x + 5 \implies f(x) - g(x) = -5 = o(g(x)). \tag{3.10}$$

Machine arithmetic - Decimal expansion

- Take 316.1415 for example:

$$316.1415 = 3 \cdot 10^2 + 1 \cdot 10^1 + 6 \cdot 10^0 + 1 \cdot 10^{-1} + 4 \cdot 10^{-2} + 1 \cdot 10^{-3} + 5 \cdot 10^{-4}.$$

- Any real number x can be written as

$$x = \pm \sum_{j=-\infty}^{\infty} d_j \cdot 10^j$$

- In-class exercise: Decimal expansions for (1) -2, (2) π.

Machine arithmetic - Binary expansion

- Similar to decimal expansion, every real number x has a binary (i.e., base B = 2) expansion:

$$x = \pm \sum_{j=-\infty}^{\infty} b_j \cdot 2^j$$

- In class exercise: consider a number $x = -(1011.01)_2$, what's the binary expansion of it?

$$x = -(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2})$$

Decimal to binary conversion

- Every number has a decimal and a binary expansion. Given a decimal expansion for a number x, how do we determine its binary expansion?

- We set $y = x$ and repeatedly do the following:
- 1. Compute the maximum integer j such that $y \geqslant 2^j$.
- 2. Output j.
- 3. Compute $y = y - 2^j$ and go to step 1.
- The algorithm terminates when $y = 0$.

Scientific notation

- Our ultimate goal: come up with a reasonable binary representation of numbers, suitable for storage and manipulation on a computer.
    - Why not just store the binary expansion? The trouble with this is that large numbers can take up a lot more space than smaller numbers, even if they don't have many nonzero digits.
- For instance, consider the following very large number (Avogadro's constant) that arises in chemistry:

$$602000000000000000000000$$

- How can we store this number in a compact way?

Scientific notation

Recall how scientific notation works. In decimal, we can write any real number other than 0 as

$$x = \pm m \times 10^E, \tag{5.12}$$

for a unique **mantissa** m and exponent E, with $1 \leqslant m < 10$ and E some integer. For example, consider the number 314.159. In scientific notation, this is written as

$$3.14159 \times 10^2. \tag{5.13}$$

In the same fashion, a number can be written in base 2 scientific notation: it takes the form

$$x = \pm m \times 2^E, \tag{5.14}$$

where this time $1 \leqslant m < 2$. For instance, consider the number 3.25. We converted this to binary to get $(11.01)_2$. In scientific notation, this becomes

$$(1.101)_2 \times 2^1. \tag{5.15}$$

- In-class exercise: scientific notations of 4125, 40.125, 4.125

How data are stored? Floating point system

b_{sign}	$b^{mant}_{n_M-1} b^{mant}_{n_M-2} \ldots b^{mant}_{1} b^{mant}_{0}$	$b^{exp}_{n_E-1} b^{exp}_{n_E-2} \ldots b^{exp}_{1} b^{exp}_{0}$

Here, there is a single bit giving the sign of the number (0 for negative, 1 for positive). Next is the mantissa, stored as an n_M-bit number (usually 52 bits). Finally, the exponent is stored as an n_E-bit number (usually 11 bits). For a nonzero number, the mantissa is not stored directly: since it is between 1 and 2, the binary expansion always begins with a 1. This is redundant, so we **do not** explicitly store it in the floating point representation. It is simply assumed to be there, leading to the so-called **hidden bit representation**. The number 0 has a special representation as all 0s.

- Discussion: what's the number in [0|0100000...|...0000011]?
- Solution: -(1+0.25)*8=-10.

Linear systems (linear equations)

- An example of linear systems
  - Any linear system can always be rewritten in matrix form

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

- More generally, $Ax = b$
  - A is an $m \times n$ matrix
  - x is an n-dimensional vector
  - b is an m-dimensional vector
- Problem: given A and b, how can you solve x?

Gaussian elimination

- Rewrite the problem in augmented matrix form
  - Original augmented matrix and manipulated augmented matrix

$$\begin{bmatrix} 1 & 1 & 1 & 7 \\ 3 & 2 & 1 & 11 \\ 4 & -2 & 2 & 8 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 1 & 7 \\ 0 & -1 & -2 & -10 \\ 0 & 0 & 10 & 40 \end{bmatrix}$$

- Elementary row operations:
  - 2^{nd} line = 2^{nd} line – 3 * 1^{st} line [0 -1 -2 -10], done!
  - 3^{rd} line = 3^{rd} line – 4 * 1^{st} line [0 -6 -2 -20]
  - Discussion: how to proceed?
  - 3^{rd} line = 3^{rd} line – 6 * 2^{nd} line [0 0 10 40], done!

Gaussian elimination

- Key idea:
  - Use elementary row operations to make A become a right-triangular matrix
  - So that you can sequentially solve the linear systems bottom-up!

$$\left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1k} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2k} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{kk} & b'_k \end{array}\right].$$

Gauss-Jordan Elimination: Beyond Gaussian Elimination

- Consider this linear system:

$$\begin{aligned} x - 2y + 3z &= 9 \\ -x + 3y \quad &= -4 \\ 2x - 5y + 5z &= 17 \end{aligned}$$

- Yes! It’s Gauss-Jordan Elimination.
  - Key idea:
    - Use elementary row operations to make A become an identity matrix
    - So that you can directly read the results!

$$\begin{bmatrix} 1 & -2 & 3 & 9 \\ -1 & 3 & 0 & -4 \\ 2 & -5 & 5 & 17 \end{bmatrix} \xrightarrow[\text{Gaussian elimination}]{\text{Elementary row operations}} \begin{bmatrix} 1 & -2 & 3 & 9 \\ 0 & 1 & 3 & 5 \\ 0 & 0 & 1 & 2 \end{bmatrix} \xrightarrow[\text{Gauss-Jordan elimination}]{\text{Elementary row operations}} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

LU decomposition

- $A = LU$
  - L is a lower triangular matrix, U is a upper triangular matrix
  - Note this decomposition may **not** be unique

- In-class exercise:
  - Find an LU decomposition of $A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & -6 & 0 \\ -2 & 7 & 2 \end{bmatrix}$

  - Solutions:
    - $u_{11} = a_{11} = 2,\ u_{12} = a_{12} = 1,\ u_{13} = a_{13} = 1$
    - $\ell_{21} = a_{21}/u_{11} = 4/2 = 2,\ \ \ell_{31} = a_{31}/u_{11} = -2/2 = -1$
    - $u_{22} = a_{22} - \ell_{21}u_{12} = -6 - 2 \cdot 1 = -8$
    - $u_{23} = a_{23} - \ell_{21}u_{13} = 0 - 2 \cdot 1 = -2$
    - $\ell_{32} = (a_{32} - \ell_{31}u_{12})/u_{22} = (7 - (-1) \cdot 1)/(-8) = 8/(-8) = -1$
    - $u_{33} = a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23} = 2 - (-1) \cdot 1 - (-1) \cdot (-2) = 2 + 1 - 2 = 1$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -8 & -2 \\ 0 & 0 & 1 \end{bmatrix}.$$

Partial pivoting prevents this issue

- How does partial pivoting work?
  - Swap rows to make pivot have the largest absolute value in its column.

$$\left(\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \xrightarrow{R_1 \leftrightarrow R_2} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 10^{-20} & 1 & 1 \end{array}\right) \xrightarrow{R_2 \leftarrow R_2 - 10^{-20} R_1} \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 - 10^{-20} & 1 - 2 \cdot 10^{-20} \end{array}\right)$$

$$x_2 = \frac{1 - 2 \cdot 10^{-20}}{1 - 10^{-20}} \approx 1 \pm 10^{-20}$$

$$x_1 + x_2 = 2 \implies x_1 = 1 \pm 10^{-20}.$$

- Why does it work?
  - Avoid dividing by tiny numbers, reduces relative error, and makes LU numerically stable for most matrices.

Another example of partial pivoting

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 10^{-20} & 4 \\ 7 & -20 & 5 \end{pmatrix} \xrightarrow{R_1 \leftrightarrow R_3} \begin{pmatrix} 7 & -20 & 5 \\ 1 & 10^{-20} & 4 \\ 2 & 1 & 3 \end{pmatrix} \xrightarrow{R_2 \leftarrow R_2 - (1/7)R_1, R_3 \leftarrow R_3 - (2/7)R_1} \begin{pmatrix} 7 & -20 & 5 \\ 0 & 2.857 & 3.286 \\ 0 & 6.714 & 1.5714 \end{pmatrix} \tag{11.18}$$

$$\xrightarrow{R_2 \leftrightarrow R_3} \begin{pmatrix} 7 & -20 & 5 \\ 0 & 6.714 & 1.5714 \\ 0 & 2.857 & 3.286 \end{pmatrix} \xrightarrow{R_3 \leftarrow R_3 - (2.857/6.714)R_2} \begin{pmatrix} 7 & -20 & 5 \\ 0 & 6.714 & 1.5714 \\ 0 & 0 & 2.6173 \end{pmatrix} \tag{11.19}$$

Eigenvalues and eigenvectors

- Definition:
  - For an $n \times n$ matrix A, an eigenvector v of A is a nonzero vector such that there exists some $\lambda \in R$ satisfying
  - $Av = \lambda v.$

- Discussion:
  - What's the dimension of v?
  - Is Av a matrix or a vector?
  - Is λv a vector or a real number?

Goal: Find eigenvectors and eigenvalues of a square matrix A

- Discussion: How to find eigenvalues by hand?
  - Reviewed in Lecture 3.

- Work with the characteristic equation for the eigenvalues λ:

$$Av = \lambda v \Leftrightarrow Av - \lambda v = 0 \Leftrightarrow Av - \lambda I v = 0 \Leftrightarrow (A - \lambda I)v = 0.$$

- But v is a nonzero vector, so $\det(A - \lambda I) = 0$.

Power method: Computing the eigenvalue of largest modulus and its corresponding eigenvector

$$|\lambda_1| > |\lambda_2| \geqslant \ldots \geqslant |\lambda_n|.$$

- Works for diagonalizable matrix only. All symmetric matrices are diagonalizable.

- Algorithm:
  - Start with an initial nonzero vector $w^{(0)}$
  - Run in K iterations

$$w^{(k+1)} = \frac{Aw^{(k)}}{\|Aw^{(k)}\|_2}.$$

  - Then your final $w^{(K)} \approx v_1$
  - And $\lambda_1 = Av_1/v_1$

Summary

- Power method is to used to calculate eigenvalue and eigenvector of a matrix
  - In an iterative way
    - Stopping criterion: number of iterations, relative error
  - Works for diagonalizable matrices only
    - All symmetric matrices are diagonalizable
  - Only finds the eigenvalue of the largest absolute value and its associated eigenvector
    - HW2 also requires you to find the second largest eigenvalue. How?

An example of D, L, U decomposition

- Linear system:

$$\begin{cases} 4x_1 + x_2 + 2x_3 = 4, \\ x_1 + 3x_2 + x_3 = 5, \\ 2x_1 + x_2 + 5x_3 = 6. \end{cases}$$

- Then,

$$A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 1 & 5 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$D = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Jacobi method

- After DLU decomposition, we have
  - $(D + L + U)x = b$
- Rearranging gives:
  - $Dx = b - (L + U)x$
- Jacobi iteration updates:
  - $x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)})$.
  - Or equivalently:
    - $x^{(k+1)} = D^{-1}b + \underbrace{(-D^{-1}(L + U))}_{=: T_J} x^{(k)}$.
- So in compact form:
  - $x^{(k+1)} = T_J x^{(k)} + c, where\ c = D^{-1}b$.

Gauss-Seidel method

- After DLU decomposition, we have
  - $(D + L + U)x = b$
- Rearranging gives:
  - $(D + L)x = b - Ux$
- Gass-Seidel iteration updates:
  - $(D + L)x^{(k+1)} = b - Ux^{(k)}$
  - Formally, $x^{(k+1)} = T_{GS}x^{(k)} + c_{GS}$
    - where $T_{GS} = -(D + L)^{-1}U, c_{GS} = (D + L)^{-1}b$

Summary of Jacobi & Gauss-Seidel method

- Both use the L, U, D decomposition
  - $A = D + L + U,$
    - D: diagonal of A
    - L: strictly lower triangular part of A
    - U: strictly upper triangular part of A
- Both are guaranteed to converge if A is symmetric positive definite (SPD).
  - SPD: $A = A^T$ and $x^T Ax > 0$ if any $x \neq 0$.

Matrix rank of A ($m \times n$ matrix)

- Definition:
  - Maximal number of linearly independent columns or maximal number of linearly independent rows.

- Two examples: Find ranks using Gaussian Elimination.

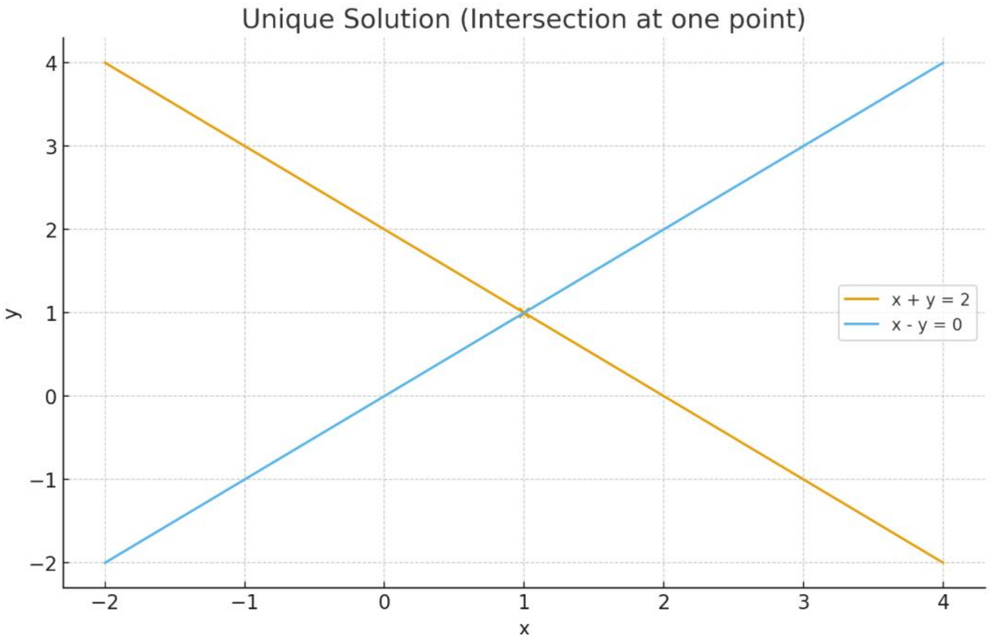
$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 0 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 0 & 1 & 1 \end{bmatrix} \xrightarrow{R_2 \leftarrow R_2 - 2R_1} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

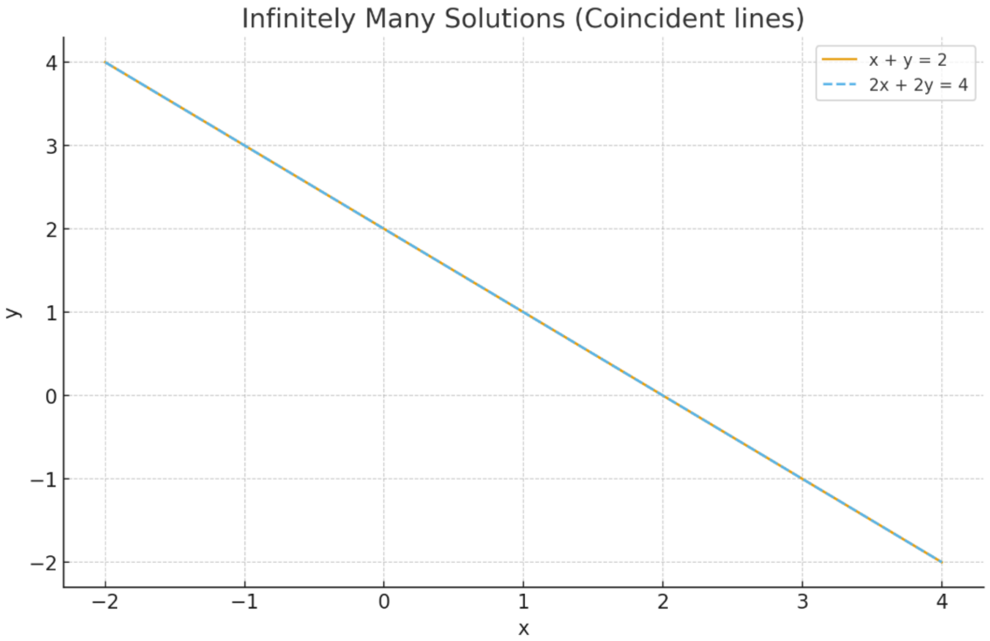
- Rank is number of pivots after Gaussian Elimination.

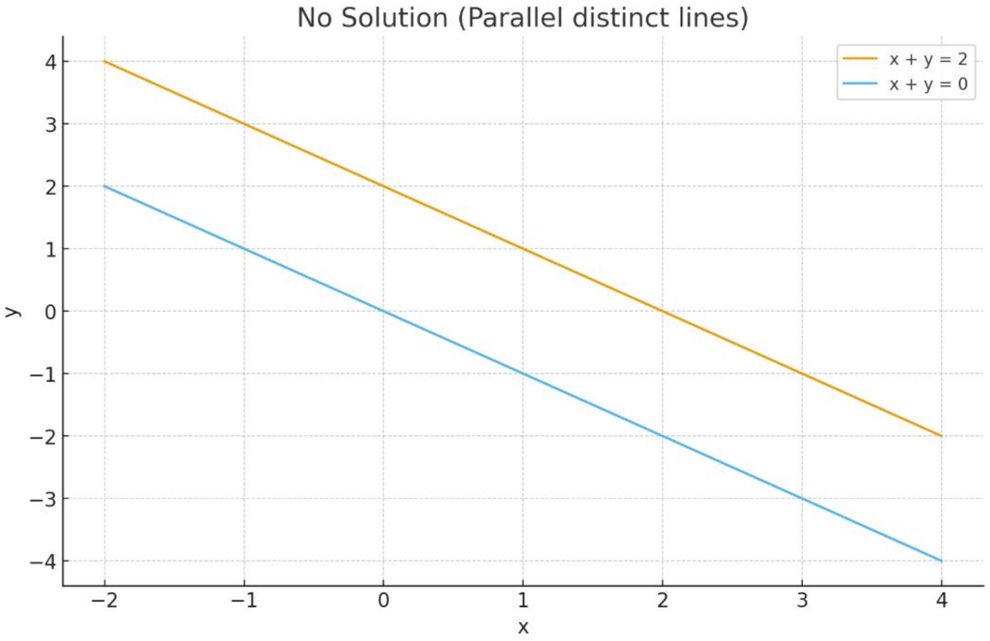
Summary of Conditions for Solutions of a Linear System $A\mathrm{x} = \mathrm{b}$

Case	Rank Condition	Number of Solutions	Geometric Interpretation
No Solution	$\text{rank}(A) < \text{rank}([A \mid \mathbf{b}])$	None (inconsistent system)	Hyperplanes do not intersect (contradictory equations)
Unique Solution	$\text{rank}(A) = \text{rank}([A \mid \mathbf{b}]) = n$	Exactly **one**	Hyperplanes intersect at a single point
Infinitely Many Solutions	$\text{rank}(A) = \text{rank}([A \mid \mathbf{b}]) < n$	Infinitely many	Hyperplanes intersect along a line, plane, or higher-dimensional subspace

Geometric view of these three systems

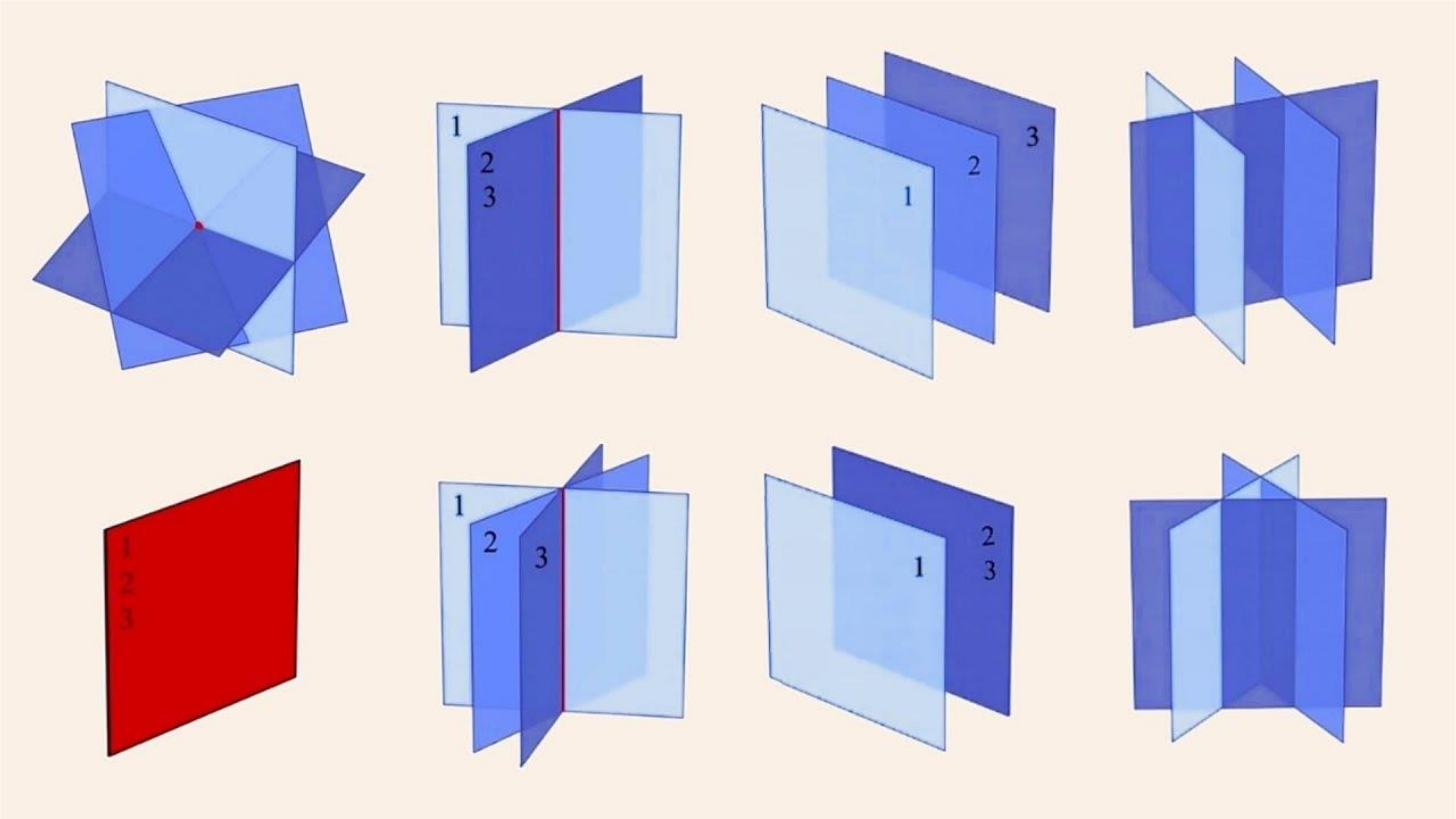






Geometric view of solutions (3-d case)

- Discussion:
  - Which figure shows a *unique* solution?
  - Which figure shows *infinitely many* solutions?
  - Which figure shows *no* solution?



Case study: Housing price

- Suppose we would like to build a model predicting house prices.
  - The model takes features of a house as inputs, and outputs predicted price.

- Discussion:
  - What are the factors (features) of a house that affects its price?

- For example,
  - 8 features:

```
- MedInc        median income in block group
- HouseAge      median house age in block group
- AveRooms      average number of rooms per household
- AveBedrms     average number of bedrooms per household
- Population    block group population
- AveOccup      average number of household members
- Latitude      block group latitude
- Longitude     block group longitude
```

  - 1 label: house price

Linear model

- Take input feature vector
  - $\text{Price}(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \cdots$
  - x_1: median income
  - x_2: median house age
  - x_3: average number of rooms
  - x_4: average number of bedrooms
  - …

- Label space is the real number space R

Linear model

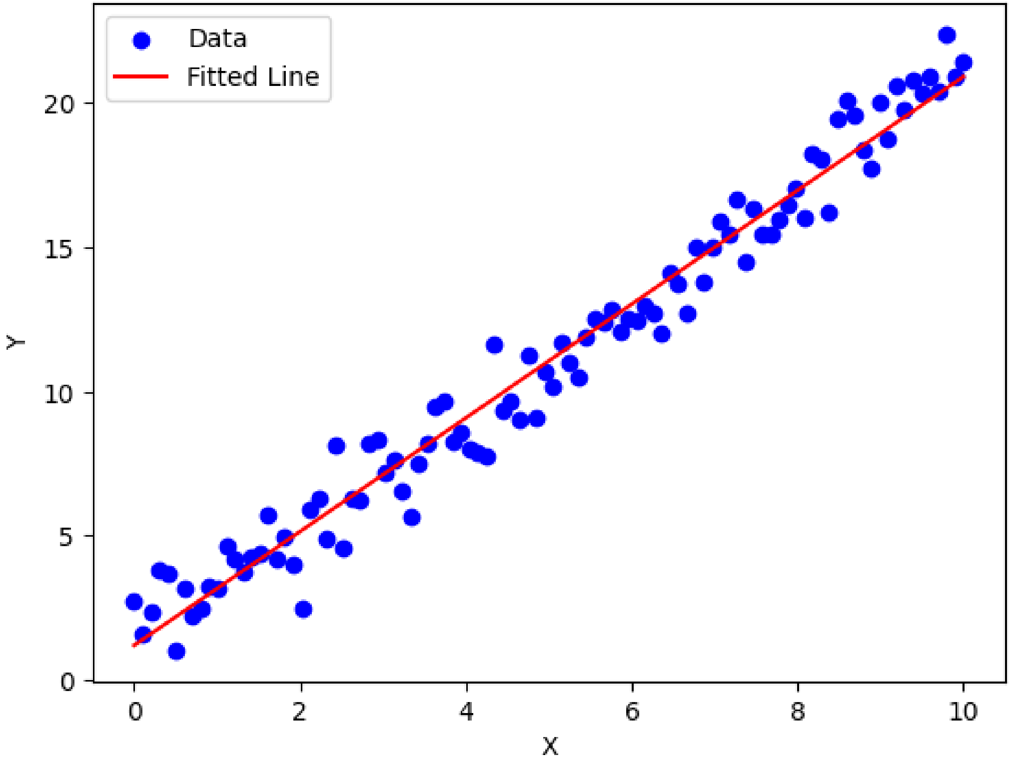
- In vector form:
  - $\text{Price}(x) = x^{\mathrm{T}}w$
  - $x = [x_1, x_2, \ldots, x_8]$: feature vector
  - $w = [w_1, w_2, \ldots, x_8]$: parameter vector

- As long as we find a good w, we have a good linear model.

- Goal: Find a good w.

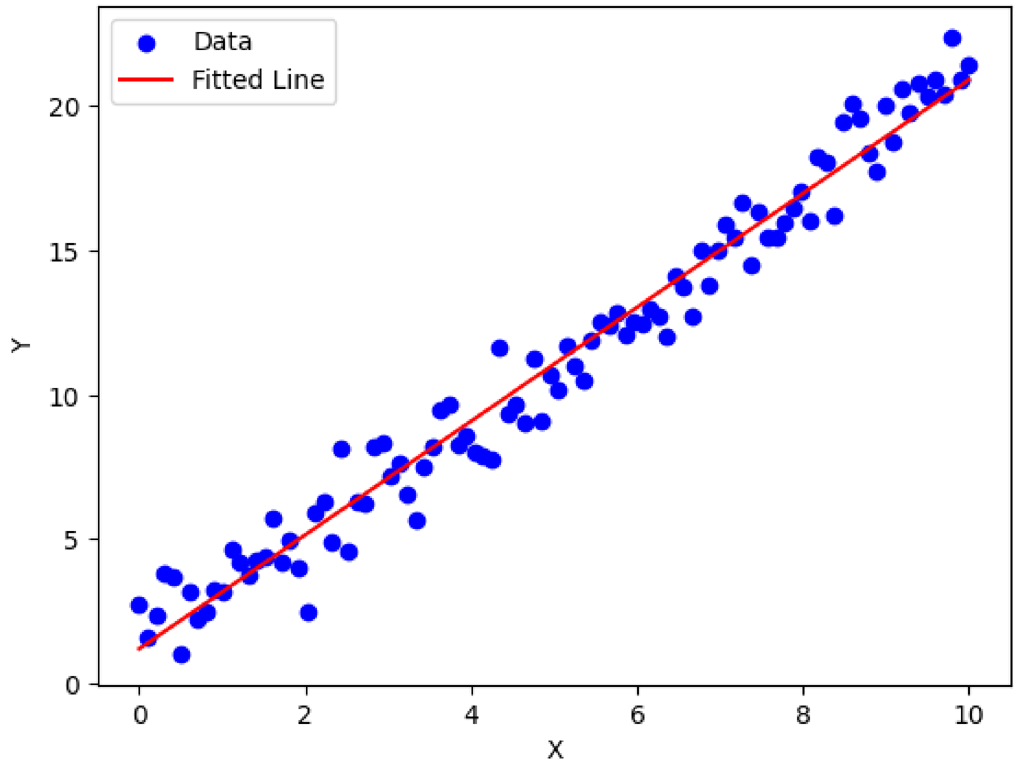
Considering conditions of linear systems

- In real-world applications, there are many challenges.
  - No solution
  - Noisy data
  - Overdetermined systems (most common case)
    - Fitting a hyperplane (a line in 2-d) to too many data points.
  - Right figure:
    - x is a feature of the house
    - y is the price.



Considering conditions of linear systems

- In real-world applications, there are many challenges.
  - No solution
  - Noisy data
  - Overdetermined systems (most common case)
    - Fitting a hyperplane (a line in 2-d) to too many data points.



- So our goal reduces to find the an approximate w that best describes the data!
- How?

The objective function for learning linear regression under square loss

- $\hat{w} = \text{argmin}_w \frac{1}{n} \sum_{i=1}^{n} (x_i^T w - y_i)^2 = \text{argmin}_w \|Xw - y\|_2^2$
  - aka: Ordinary Least Square (OLS)
- In-class exercise: solve this optimization problem by setting gradient of the objective function to 0.

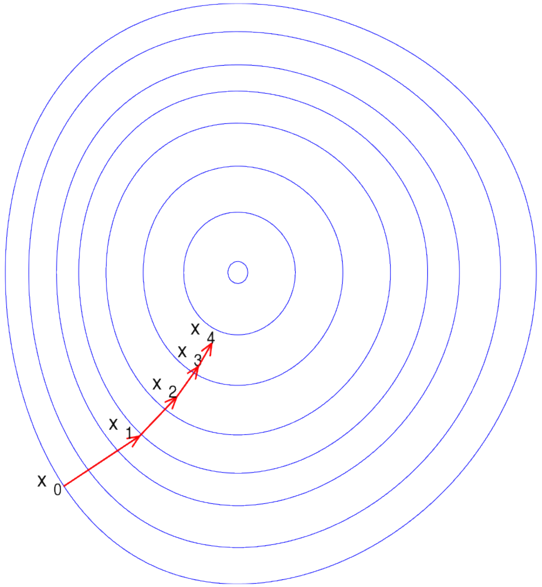
How do we optimize a continuously differentiable function in general?

- The problem: $\min_{\theta} f(\theta)$

- Discussion: How do you solve this optimization problem?

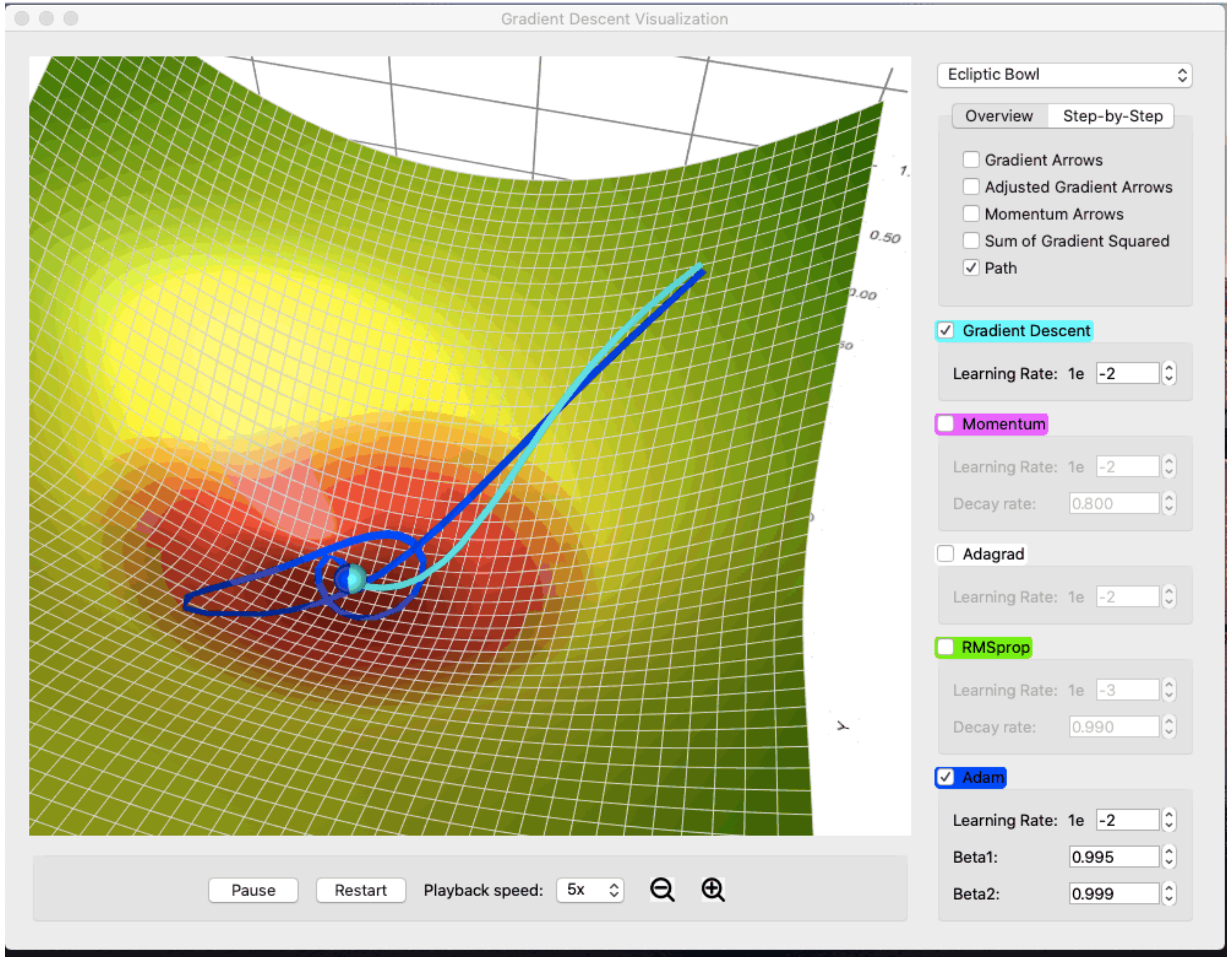
- Gradient descent in iterations

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$



Gradient Descent Demo in 2-D

- An excellent demo tool:
  - https://github.com/lilipads/gradient_descent_viz



Gradient descent for quadratic function

- $\min f(x) = x^2$

- Follow me on the first two examples:

1. Find x_4 given $x_0 = 2, \eta = 0.1$
2. Find x_4 given $x_0 = 2, \eta = 0.4$

- In-class exercise questions:

1. Find x_4 given $x_0 = 4, \eta = 0.4$
2. Find x_4 given $x_0 = 2, \eta = 1.5$ (what did you find?)

Back to linear regression: How to solve it using Gradient Descent?

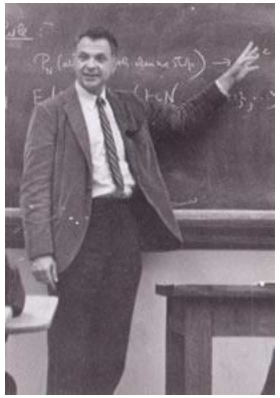
- $\hat{w} = \text{argmin}_w \frac{1}{n} \sum_{i=1}^{n} (x_i^T w - y_i)^2 = \text{argmin}_w ||Xw - y||_2^2$

- In-class exercise: Write the GD updating rule for solving w.
  - $w \leftarrow w - 2\eta X^T(Xw - y)$

Checkpoint

- Least square:
  - Heavily used in practice, due to
    - Large datasets (many data points)
    - Noisy data
    - No solution based on conditions of linear systems
- Linear regression
  - $\hat{w} = \mathrm{argmin}_w \frac{1}{n}\sum_{i=1}^{n}(x_i^T w - y_i)^2 = \mathrm{argmin}_w \|Xw - y\|_2^2$
  - Direct solver: $\hat{w} = (X^T X)^{-1} X^T y$
  - GD: $w \leftarrow w - 2\eta X^T(Xw - y)$

Stochastic Gradient Descent (Robbins-Monro 1951)



Herbert Robbins
1915 - 2001

- Gradient descent

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

- Stochastic gradient descent
  - Using a stochastic approximation of the gradient:

$$\theta_{t+1} = \theta_t - \eta_t \hat{\nabla} f(\theta_t)$$

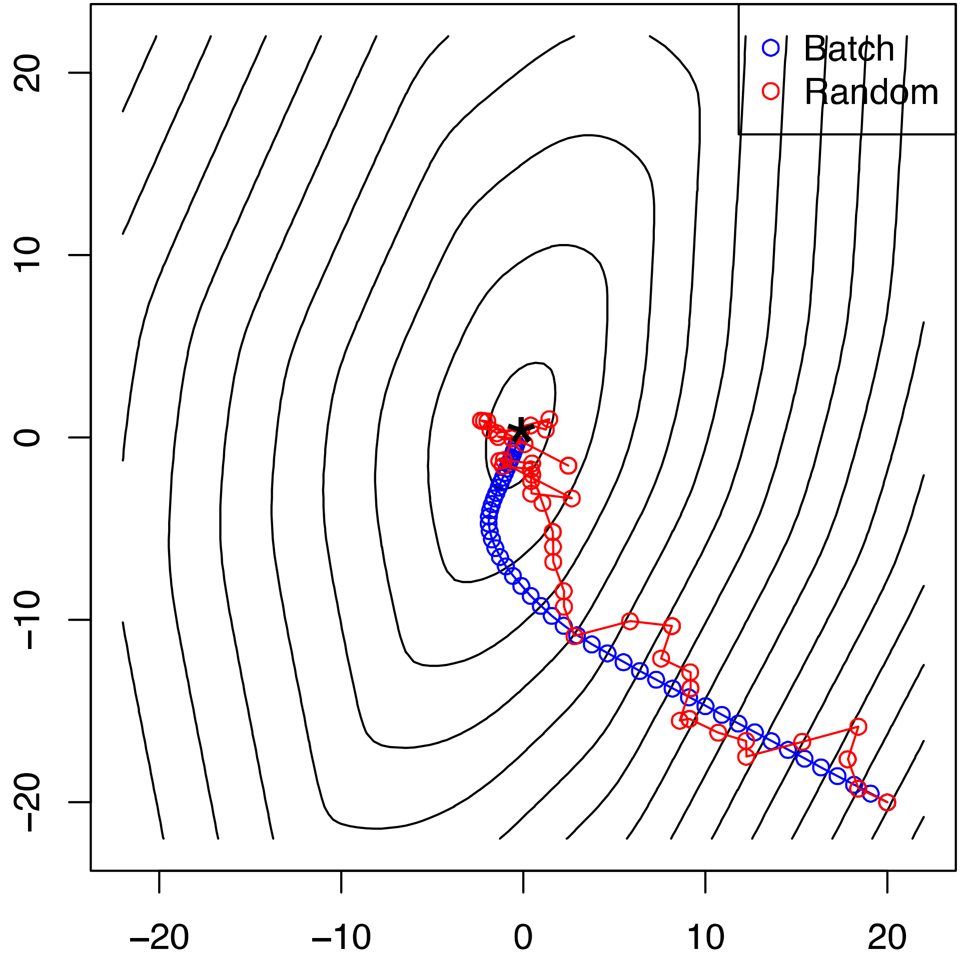
A natural choice of SGD in machine learning

- Recall that

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, (x_i, y_i))$$

- SGD samples a data point i uniformly at random while GD uses all data!
  - Use $\nabla_\theta \ell(\theta, (x_i, y_i))$

Illustration of GD vs SGD



Time complexity:

GD: $O(nd * \text{n\_iterations})$
SGD: $O(d * \text{n\_iterations})$

The power of SGD

- Extremely simple:
  - A few lines of code

- Extremely scalable
  - Just a few pass of the data, no need to store the data

- Extremely general:
  - In addition to linear regression, in practice it can solve most optimization problems of differentiable functions
    - E.g., Training neural networks, Transformer, Generative Pretrained Transformer
  - Foundational algorithm of the AI revolution as we see today!

Time complexity of direct solver and GD/SGD for solving linear regression

- Direct solver
  - $O(nd^2 + d^3)$
- GD:
  - $O(ndT)$
- SGD:
  - $O(dT)$

- $T = \text{n\_iterations}$

What's Linear Programming (LP)?

- An optimization problem of linear objective functions with linear constraints.
  - Objective function can be minimized or maximized
  - Constraints can be in equalities or inequalities
  - All functions must be linear functions

- 2 examples:

$$\begin{aligned} \min\ & x_1 + 2x_2 \\ \text{s.t. } & x_1 + x_2 \leq 3 \\ & x_1 \geq 1 \end{aligned} \qquad \begin{aligned} \max\ & x_1 + 2x_2 \\ \text{s.t. } & x_1 + x_2 = 3 \end{aligned}$$

- Discussion: Could you propose more linear programming problems?

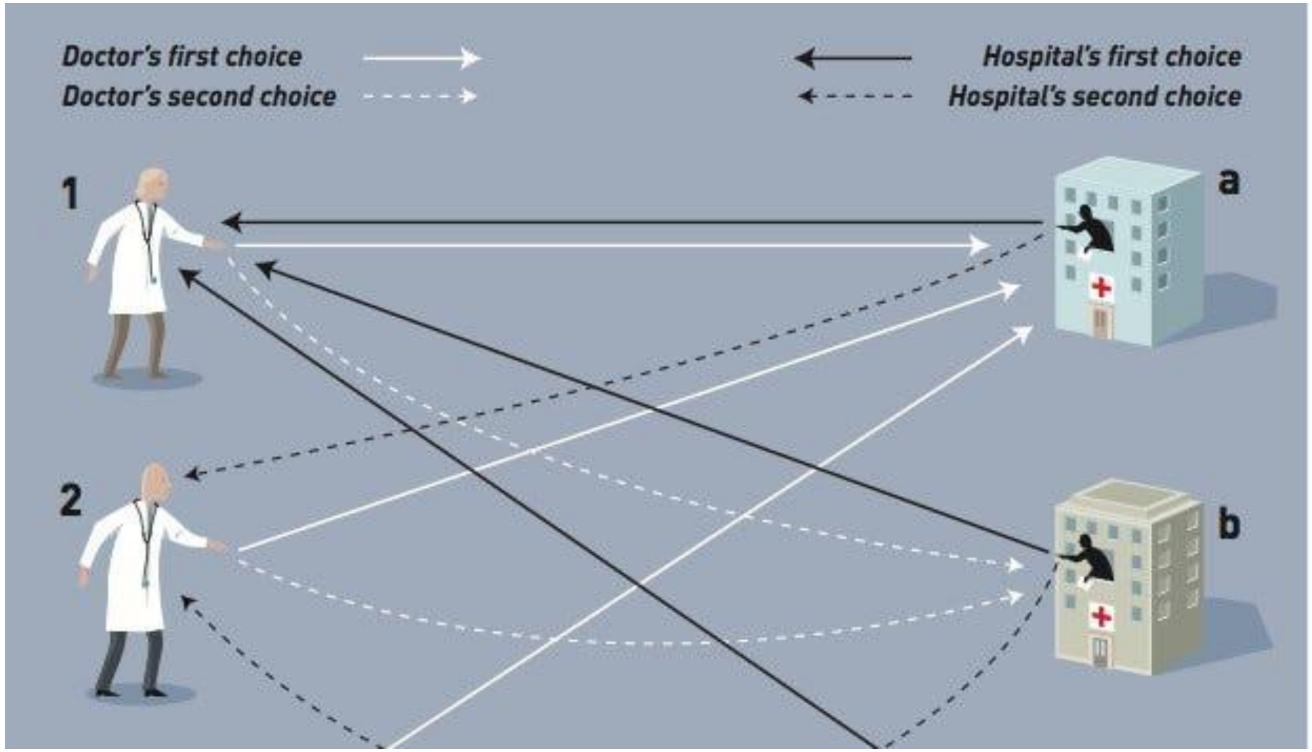
Application of LP: Matching problem

- Company (hospital) - Candidate (doctor) matching problem

- Each doctor:
  - Fits one position

- Doctors/hospitals:
  - Have their preferences

- Goal:
  - Put doctors to positions
  - Such that overall best match



Application of LP: Optimal transport

- Suppose you run a company, which has 4 factories and 3 big markets, each in a different city.

Boston demands 25

Buffalo supplies 15

Syracuse supplies 15

Rochester supplies 20

Albany supplies 35

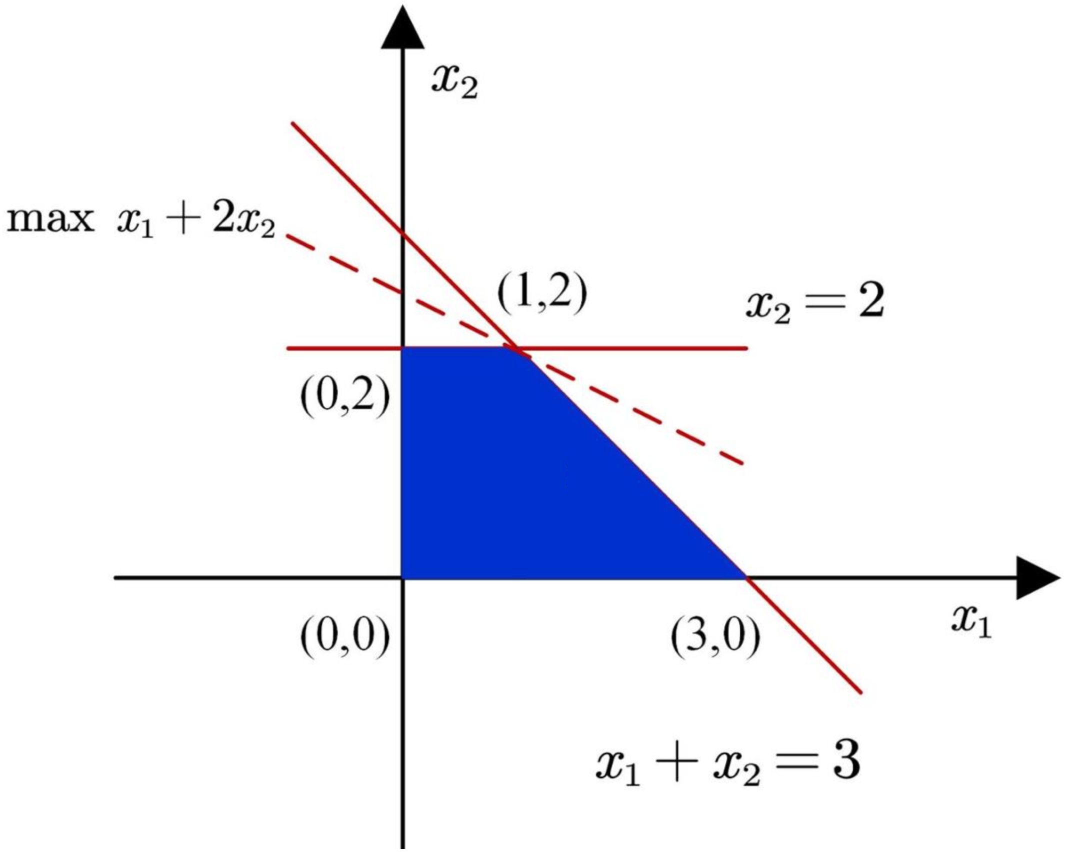
New York City demands 30

Philadelphia demands 30

- You job is to design the optimal transportation route that has minimum transportation cost of your products
  - Each route (supply to demand) costs differently
  - Each factory has its supply capacity
  - Each market must be well supplied to maximize your profit

How to solve the LP problem?

$$
\begin{aligned}
\text{maximize } & x_1 + 2x_2 \\
\text{subject to } & x_1 + x_2 \leq 3 \\
& x_2 \leq 2 \\
& x_1 \geq 0 \\
& x_2 \geq 0
\end{aligned}
$$



- For most 2-d LP problems,
  1. We can draw it's feasible region
  2. And move it's objective function

- In-class exercise: Draw the feasible region defined by constraints.

From primal to dual LP problems

- In-class exercise:
  - Work on the following two LP problems by drawing graphs

$$\max Z = x_1 + x_2$$

$$\text{s.t.} \begin{cases} x_1 + 2x_2 \leq 8, \\ 3x_1 + 2x_2 \leq 18, \\ x_1, x_2 \geq 0. \end{cases}$$

$$\min Z = 8y_1 + 18y_2$$

$$\text{s.t.} \begin{cases} y_1 + 3y_2 \geq 1, \\ 2y_1 + 2y_2 \geq 1, \\ y_1, y_2 \geq 0. \end{cases}$$

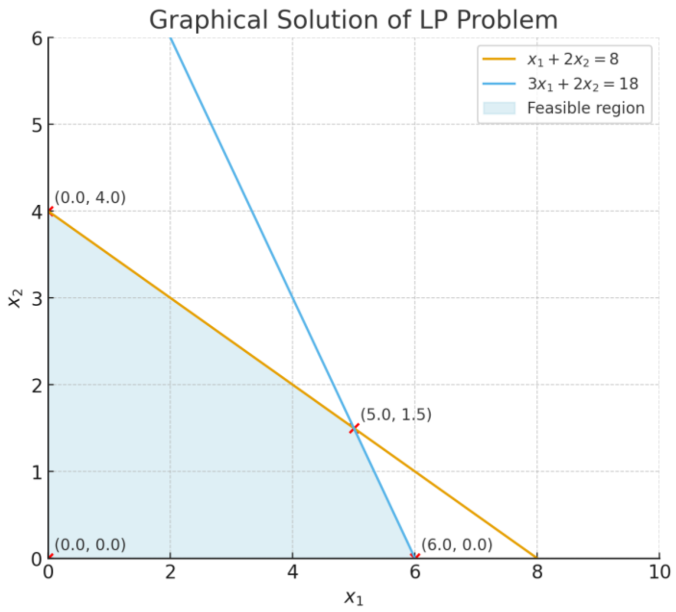
  - What can you see from their optimal Z?

From primal to dual LP problems

- Solutions to in-class exercise:

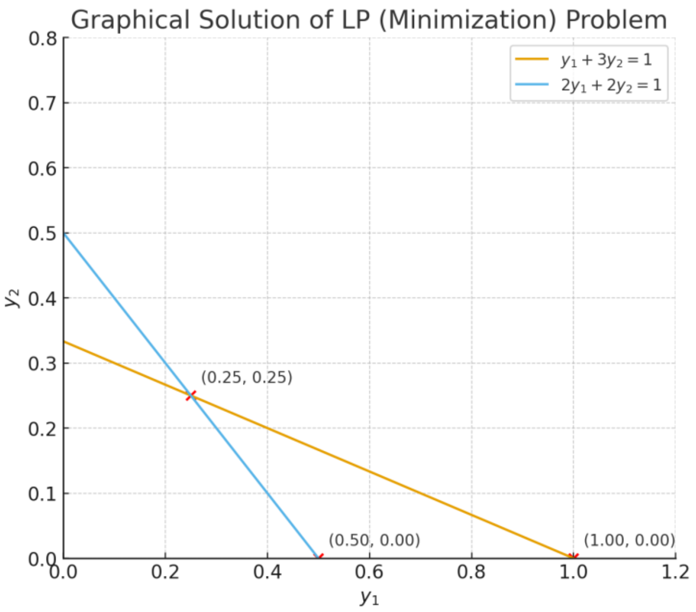
$$\max Z = x_1 + x_2$$

$$\text{s.t.} \begin{cases} x_1 + 2x_2 \leq 8, \\ 3x_1 + 2x_2 \leq 18, \\ x_1, x_2 \geq 0. \end{cases}$$



$$\min Z = 8y_1 + 18y_2$$

$$\text{s.t.} \begin{cases} y_1 + 3y_2 \geq 1, \\ 2y_1 + 2y_2 \geq 1, \\ y_1, y_2 \geq 0. \end{cases}$$



- They are primal and dual LP problems!

From primal to dual LP problems

- Primal problem:

$$\max\ z = 4x_1 + x_2 + 5x_3 + 3x_4$$

$$\begin{aligned} x_1 - x_2 - x_3 + 3x_4 &\leq 1 \\ 5x_1 + x_2 + 3x_3 + 8x_4 &\leq 55 \\ -x_1 + 2x_2 + 3x_3 - 5x_4 &\leq 3 \\ x_i &\geq 0 \end{aligned}$$

- Key idea:
  - Multiply each constraint with a non-negative multiplier and form linear combinations of constraints.

$$y_1(x_1 - x_2 - x_3 + 3x_4) \ + \ y_2(5x_1 + x_2 + 3x_3 + 8x_4) \ + \ y_3(-x_1 + 2x_2 + 3x_3 - 5x_4) \ \leq \\ y_1 + 55y_2 + 3y_3.$$

$$(y_1 + 5y_2 - y_3)x_1 \ + \ (-y_1 + y_2 + 2y_3)x_2 \ + \ (-y_1 + 3y_2 + 3y_3)x_3 \ + \ (3y_1 + 8y_2 - 5y_3)x_4 \ \leq \\ y_1 + 55y_2 + 3y_3.$$

- Finally, dual problem:

$$\min\ u = y_1 + 55y_2 + 3y_3$$

$$\begin{aligned} y_1 + 5y_2 - y_3 &\geq 4 \\ -y_1 + y_2 + 2y_3 &\geq 1 \\ -y_1 + 3y_2 + 3y_3 &\geq 5 \\ 3y_1 + 8y_2 - 5y_3 &\geq 3 \\ y_i &\geq 0 \end{aligned}$$

Dual problem of linear programming

- Economic Interpretation
  - The dual variables y represent **shadow prices** — the value of relaxing each constraint by one unit.
  - In a resource allocation problem, each y_i tells how much the objective (profit) would improve if resource i were increased slightly.

- Weak Duality:
  For any feasible x (primal) and y (dual), $c^T x \leq b^T y$.
  - The dual provides an upper bound (for maximization problems).

- Strong Duality:
  At the optimal solutions x^*, y^*, $c^T x^* = b^T y^*$.
  - Solving one problem solves the other — they share the same optimal value.

Dual problem of linear programming

- Why we study dual problems?

- Duality helps:
  - **Check optimality:** If primal and dual feasible solutions give the same objective, both are optimal.
  - **Perform sensitivity analysis:** Dual variables show how changes in constraints affect the outcome.
  - **Simplify computation:** Some LPs are easier to solve in dual form (e.g., when constraints >> variables).

Nonlinear equation solver: Bisection method

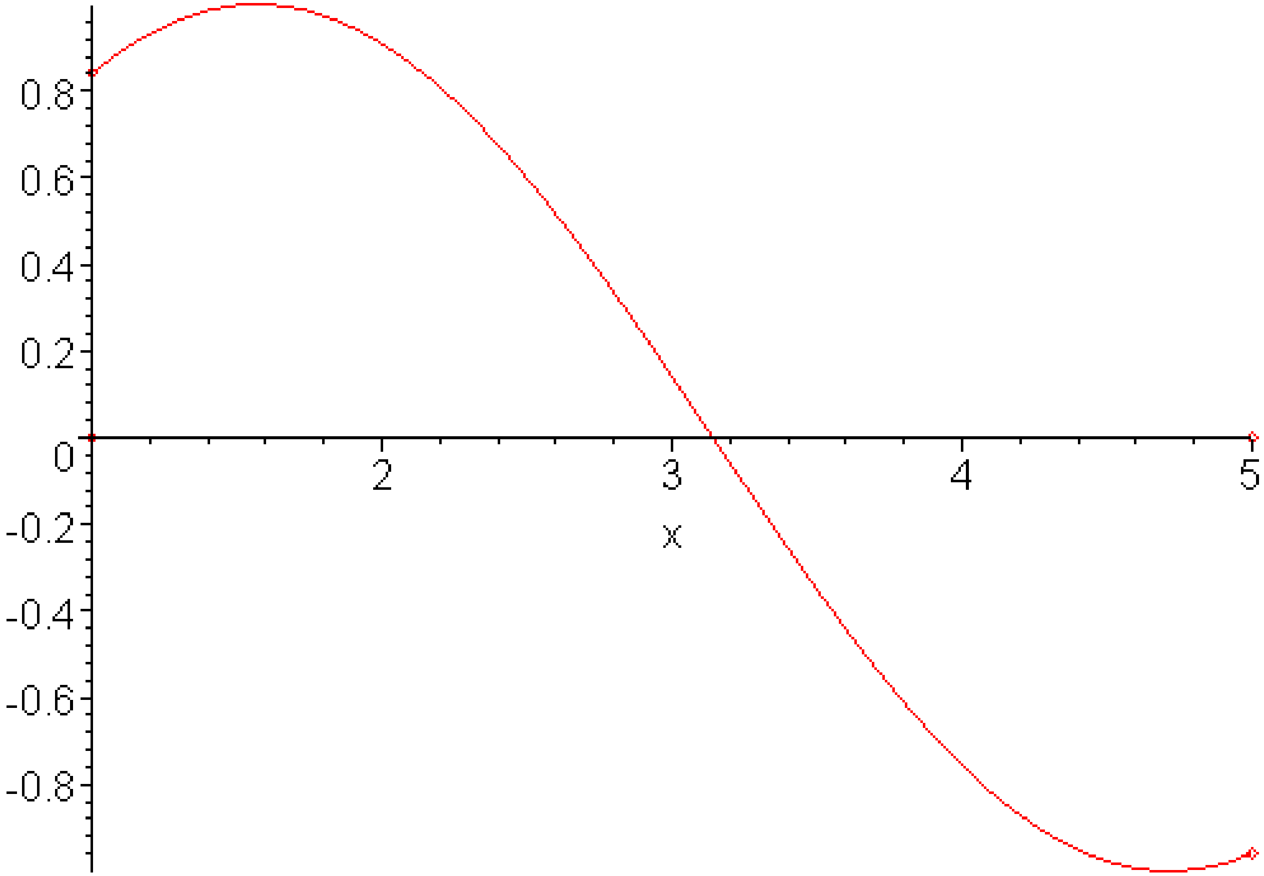
- Key idea:
  - In every iteration, we cut the interval in half while still maintaining the property that the endpoints have opposite signs. This allows us to conclude that we're getting closer and closer to a root.
- Algorithm:

1. Preprocessing: If $F(a) = 0$ or $F(b) = 0$, output whichever one was 0 and terminate. If $F(a) < 0 < F(b)$, then set $inc = 1$. Otherwise, set $inc = 0$.
2. Compute $z = \frac{a+b}{2}$, the midpoint of the interval $[a, b]$.
3. If $F(z) = 0$, return z and terminate.
4. If $inc = 0$ (so $F(a) > 0 > F(b)$):
   (a) If $F(z) < 0$, then set $b = z$.
   (b) If $F(z) > 0$, then set $a = z$.
5. If $inc = 1$ (so $F(a) < 0 < F(b)$):
   (a) If $F(z) < 0$, then set $b = z$.
   (b) If $F(z) > 0$, then set $a = z$.

After k iterations, we output the midpoint of the resulting interval.

Illustration of the bisection method

- Initial interval: [1, 5]
- 3 steps in each iteration:
  - Given a, b, find midpoint
  - Check midpoint value
  - Update a or b



Nonlinear equation solver: Newton's method

- Key idea:
  - Take F, find its local linear approximation at a starting point x_0, solve for x to get x_1, and use that as our new initial point.
  - Iterate until (hopefully) convergence.
- So, how to find the local linear approximation of F at x_0?
  - First-order Taylor expansion at x_0

$$P_1(x) = F(x_0) + F'(x_0)(x - x_0).$$

$$0 = F(x_0) + F'(x_0)(x - x_0) \implies -\frac{F(x_0)}{F'(x_0)} = x - x_0 \implies x = x_0 - \frac{F(x_0)}{F'(x_0)}.$$

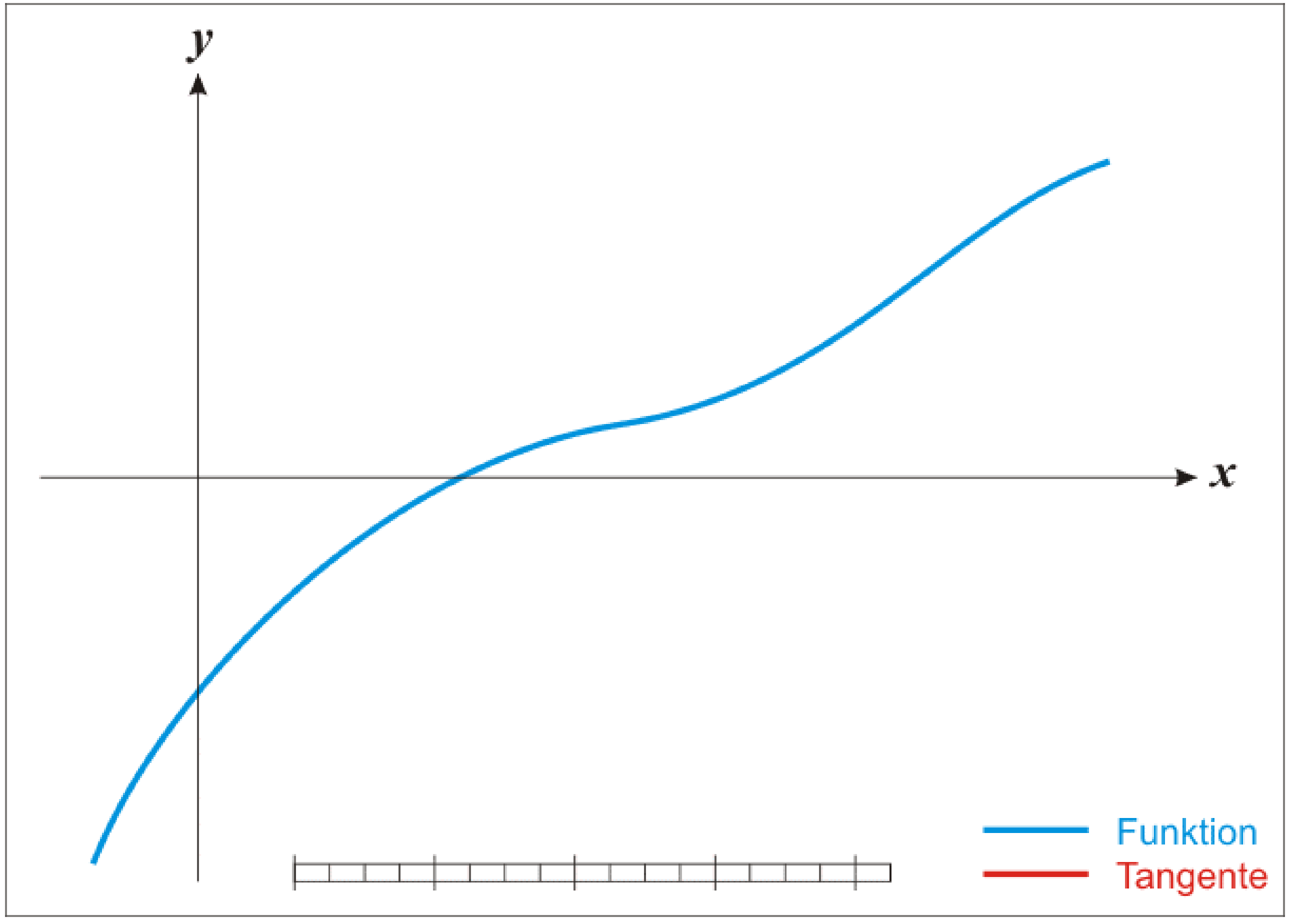
- Algorithm: (Newton update equation)

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}.$$

Illustration of the Newton's method

- In each iteration:

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}.$$



Summary of nonlinear equation solvers

- Things to know about:
    - Problem statement
    - Assumptions behind each method
    - Benefits/drawbacks of each method
    - Key theorems from calculus that feature in their analysis
    - How does each method look, visually?
    - How do we code each method up in Matlab/Python?
- Technical summary table:

Methods	Bisection method	Newton’s method
Assumptions	Continuity, opposite sign condition	Continuous, differentiable, initial point close to root
Associated theorem	Intermediate value theorem	Taylor’s remainder theorem
Guarantee	Linear convergence	Quadratic convergence

Problem setup of Interpolation

- For given data
  - $(t_1, y_1), (t_2, y_2), \ldots, (t_m, y_m)$ with $t_1 < t_2 < \cdots < t_m$
- determine function $f: R \to R$ such that
  - $f(t_i) = y_i, \forall i = 1, \ldots, m$
  - Exactly crossing all data points!

- f is **interpolating function**, or **interpolant**, for given data.
  - f could be function of more than one variable, but let's focus on the 1-dimensional case first.

Interpolation vs. Regression

- By definition, interpolating function fits given data points exactly
- Interpolation is inappropriate if data points subject to significant errors
  - Regression is a better choice in this case
- It is usually preferable to smooth noisy data
- Regression is more appropriate for special function libraries
  - Linear regression

Basis Functions

- Family of functions for interpolating:
  - Set of basis functions $\phi_1(t), \ldots, \phi_n(t)$
- Interpolating function f is chosen as linear combination of them

$$f(t) = \sum_{j=1}^{n} x_j \phi_j(t)$$

- Requiring f to interpolate data (t_i, y_i) means

$$f(t_i) = \sum_{j=1}^{n} x_j \phi_j(t_i) = y_i, \quad i = 1, \ldots, m$$

  - Discussion: What is this system?
    - A system of linear equations $Ax = y$ for n-vector x of parameters x_j, where entries of $m \times n$ matrix A are given by $a_{ij} = \phi_j(t_i)$.

Basic polynomial interpolation

- Simplest and most common type of interpolation using polynomials

- Unique polynomial of degree at most $n-1$ passes through n data points $(t_i, y_i), i = 1, \ldots, n$, where t_i are distinct

Basic polynomial interpolation

- Basis functions

$$\phi_j(t) = t^{j-1}, \quad j = 1, \ldots, n$$

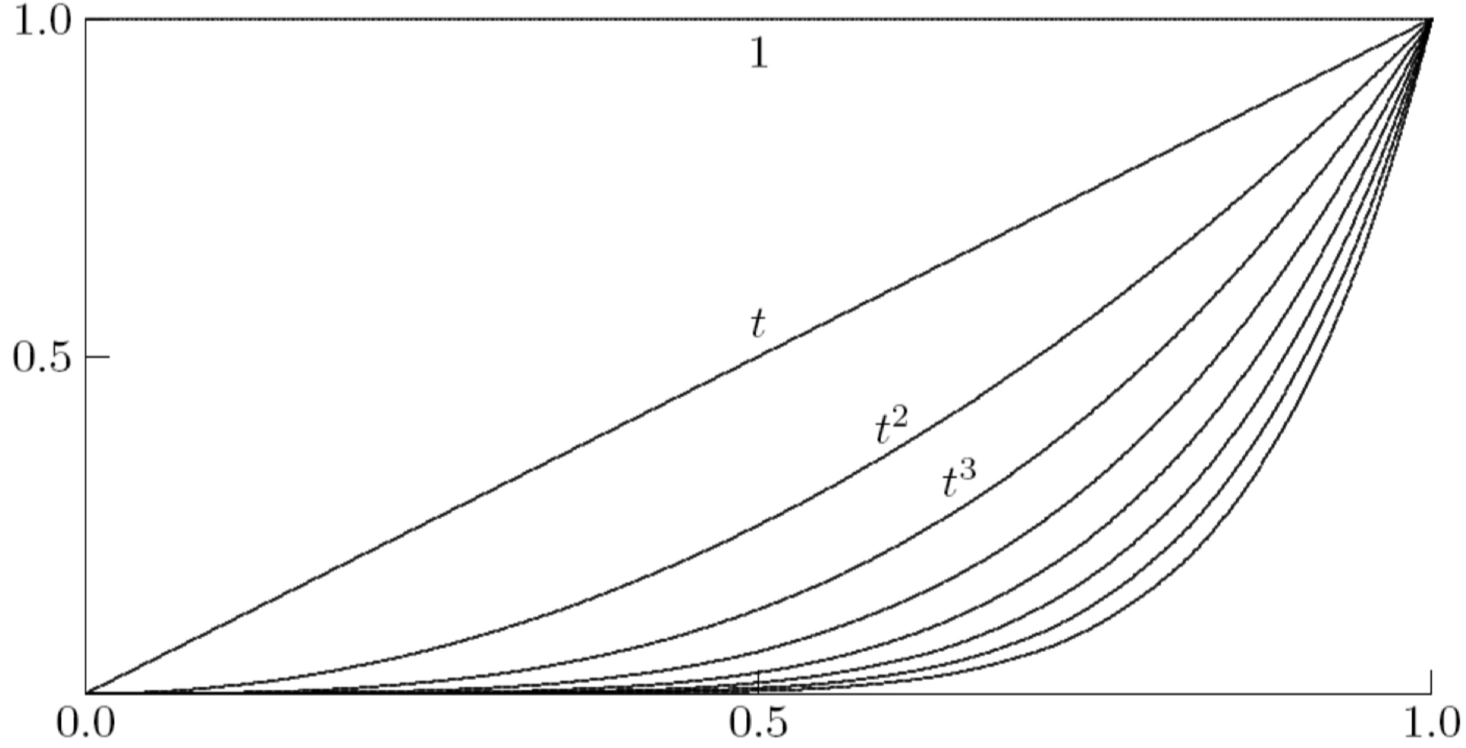
- give interpolating polynomial of form

$$p_{n-1}(t) = x_1 + x_2 t + \cdots + x_n t^{n-1}$$

- with coefficients x given by $n \times n$ linear system

$$\boldsymbol{Ax} = \begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & \cdots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \boldsymbol{y}$$

Basis functions



Lagrange interpolation

- For given set of data points $(t_i, y_i), i = 1, \dots, n$, let

$$\ell(t) = \prod_{k=1}^{n} (t - t_k) = (t - t_1)(t - t_2) \cdots (t - t_n)$$

- Define weights

$$w_j = \frac{1}{\ell'(t_j)} = \frac{1}{\prod_{k=1,\, k \neq j}^{n} (t_j - t_k)}, \quad j = 1, \dots, n$$

- Lagrange basis functions are then given by

$$\ell_j(t) = \ell(t) \frac{w_j}{t - t_j}, \quad j = 1, \dots, n$$

  - From definition, $\ell_j(t)$ is polynomial of degree $n - 1$

Lagrange interpolation

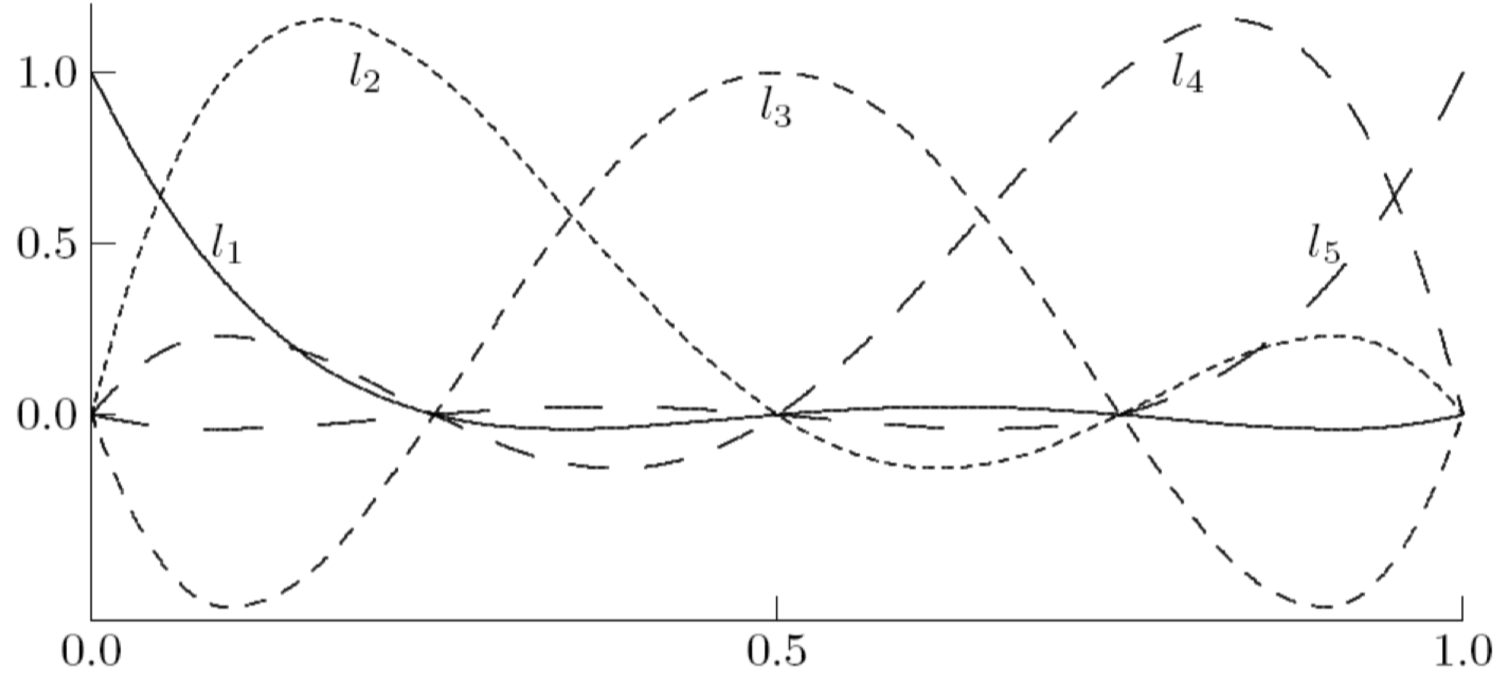
- Assuming common factor $(t_i - t_j)$ in $\ell(t_j)/(t_i - t_j)$ is canceled to avoid division by zero when evaluating $\ell_j(t_i)$, then

$$\ell_j(t_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \quad i, j = 1, \ldots, n$$

  - Matrix of linear system $Ax = y$ is identity matrix I
  - Coefficients x for Lagrange basis functions are just data values y
- Polynomial of degree $n - 1$ interpolating data points $(t_i, y_i), i = 1, \ldots, n$ is given by

$$p_{n-1}(t) = \sum_{j=1}^{n} y_j \ell_j(t) = \sum_{j=1}^{n} y_j \ell(t) \frac{w_j}{t - t_j} = \ell(t) \sum_{j=1}^{n} y_j \frac{w_j}{t - t_j}$$

Lagrange Basis Functions



Newton interpolation

- For given set of data points $(t_i, y_i), i = 1, \ldots, n$, Newton basis functions are defined by

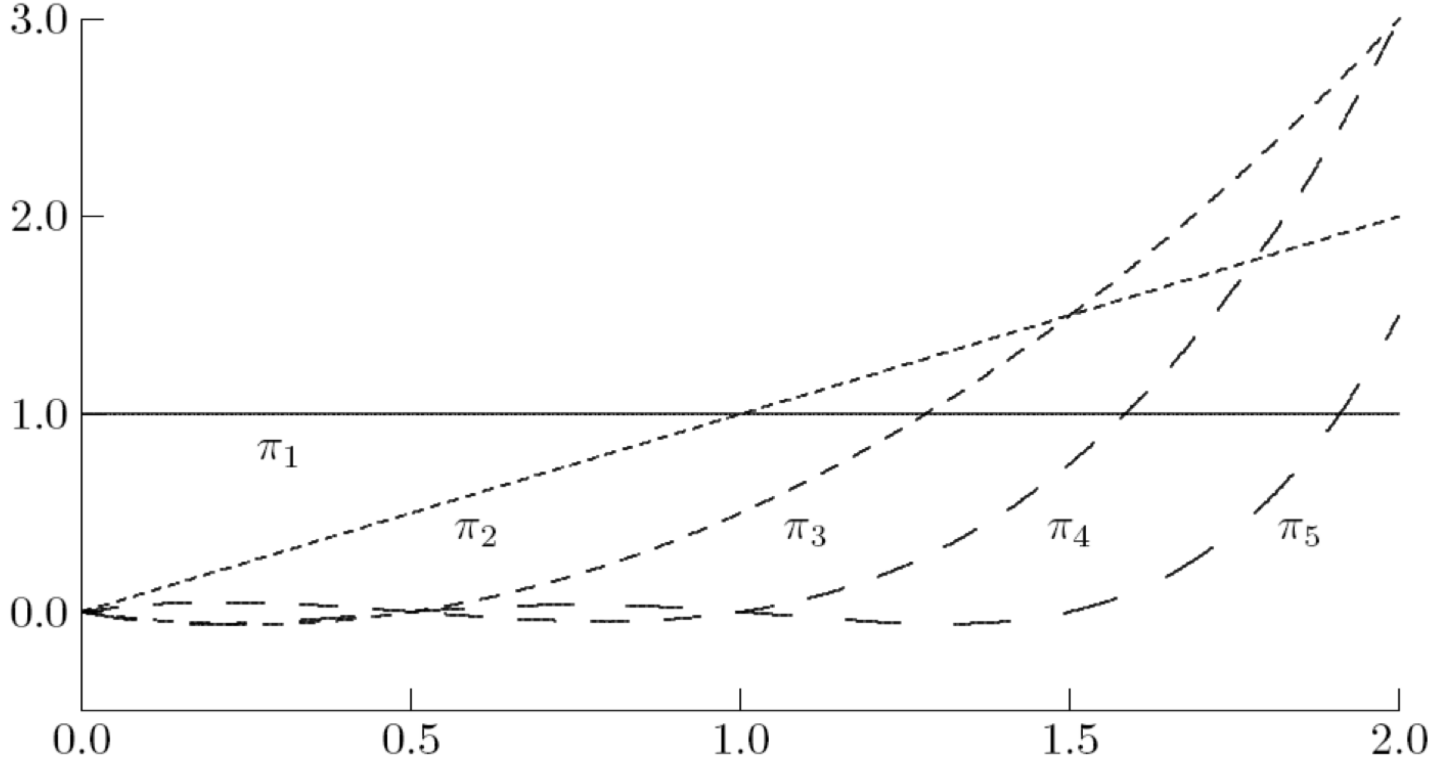
$$\pi_j(t) = \prod_{k=1}^{j-1}(t - t_k), \quad j = 1, \ldots, n$$

- Newton interpolating polynomial has form

$$\begin{aligned} p_{n-1}(t) \quad &= \quad x_1 + x_2(t - t_1) + x_3(t - t_1)(t - t_2) + \\ &\quad \cdots + x_n(t - t_1)(t - t_2) \cdots (t - t_{n-1}) \end{aligned}$$

  - For $i < j, \pi_j(t_i) = 0$, so basis matrix A is lower triangular, where $a_{ij} = \pi_j(t_i)$.

Newton basis functions



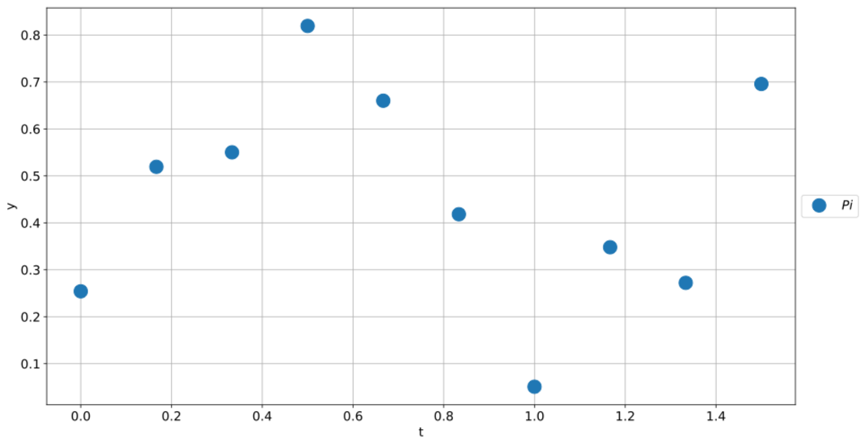
Piecewise polynomial interpolation

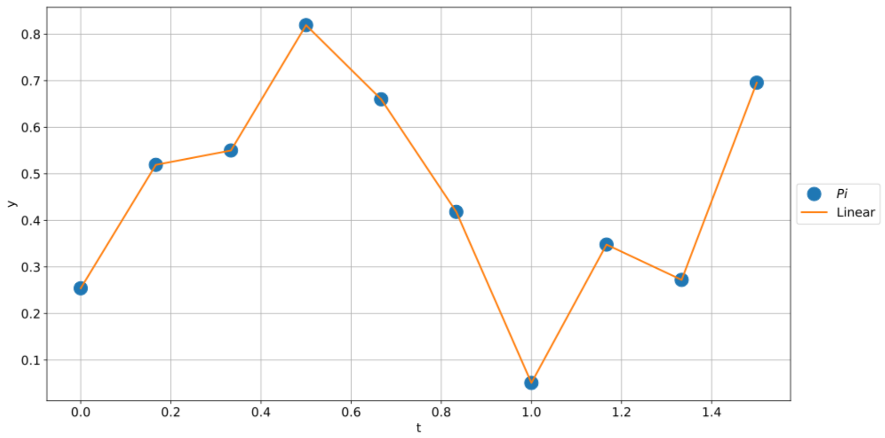
- Motivation:
  - Fitting single polynomial to large number of data points is likely to yield unsatisfactory behavior in interpolant
- Main advantage:
  - Large number of data points can be fit with low-degree polynomials
- How:
  - Given data points (t_i, y_i), different function is used in each subinterval $[t_i, t_{i+1}]$
    - t_i is called knot or breakpoint, at which interpolant changes from one function to another

Piecewise polynomial interpolation

- Discussion: Could you provide an example of a piecewise polynomial interpolation?

- Simplest example is piecewise linear interpolation, in which successive pairs of data points are connected by straight lines
  - Discussion: what are the drawbacks of linear interpolation?



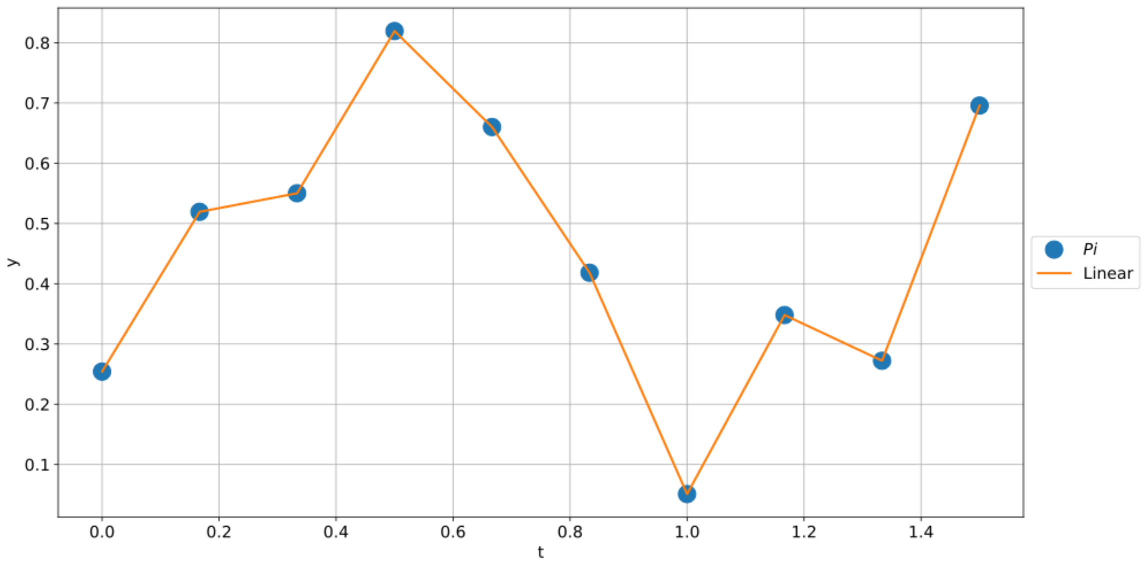


Spline interpolation

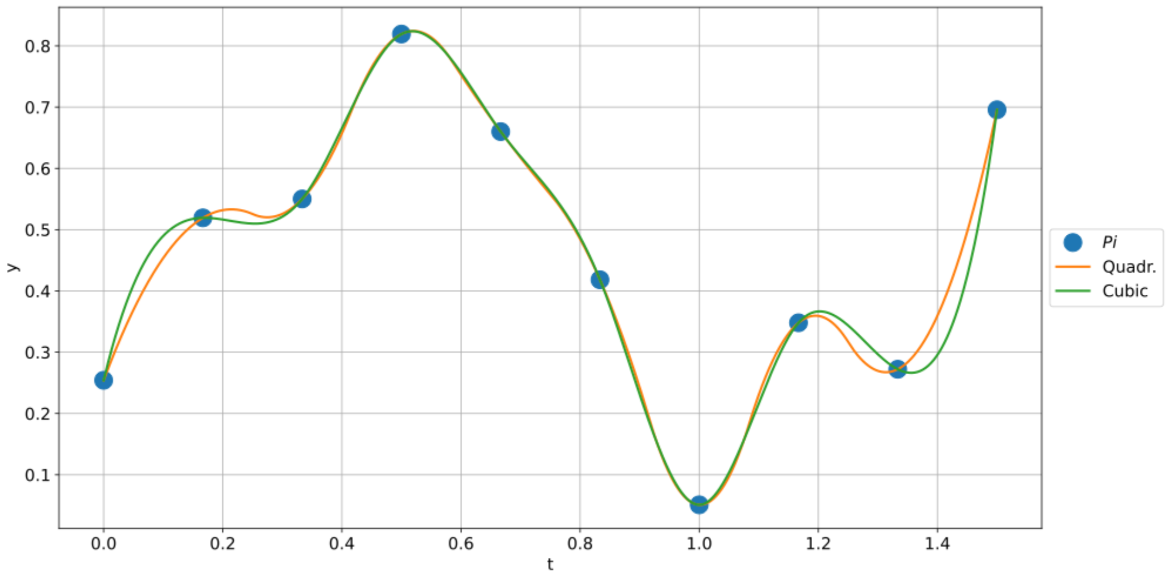
- A spline is a smooth piecewise polynomial function.
  - Two popular model:
    - Quadratic spline, Cubic spline
- Quadratic spline interpolation
  - each segment is a **second-degree polynomial** function.
  - Formally, we have data points $(t_i, y_i), i = 1, \dots, n$
  - For each interval $[t_i, t_{i+1}]$, we define a quadratic polynomial
    - $f_i(t) = a_i + b_i(\mathrm{t} - t_i) + c_i(t - t_i)^2$.
    - There are $n - 1$ such polynomials (one per interval).
  - Discussion: how many coefficients need to be determined? How many equations do we need?
    - $3(n - 1)$

Illustration of piecewise polynomial interpolation (scipy.interpolate)

- Piecewise linear



- Spline – quadratic
- Spline - cubit



Summary

- Interpolating function fits given data points exactly, which is not appropriate if data are noisy
- Interpolating function given by linear combination of basis functions, whose coefficients are to be determined
- Existence and uniqueness of interpolant depend on whether number of parameters to be determined matches number of data points to be fit
- Piecewise polynomial (e.g., spline) interpolation can fit large number of data points with low-degree polynomials
- Cubic spline interpolation is excellent choice when smoothness is important

Finally, …

- HW4 due tonight
- Final practice exam and solution reviews next week
- Looking forward to your final presentations on Mon Dec 8!
- I'll teach CSI 436 Machine Learning next Spring.
  - If you enjoy my teaching or want to learn more about AI/ML, feel free to register
  - Let's see how numerical methods are applied in real-world exciting algorithms and applications!