Applied Numerical Methods for Civil Engineering

CGN 3405 - 0002

Week 10: Linear Algebraic Equations

Xinyu Chen

Assistant Professor

University of Central Florida

Quizzes Now!

- **Today's participation** (ungraded survey): Please check out

  "`Class Participation Quiz 22`"

  Time slot: **2:30PM – 3:00PM**

  on Canvas.

Solving Linear Systems: Comparison

Solving the linear system with different methods:

$$3x_1 + 2x_2 = 18$$
$$-x_1 + 2x_2 = 2$$

1. Graphical method

$$x_2 = -\frac{3}{2}x_1 + 9$$
$$x_2 = \frac{1}{2}x_1 + 1$$

2. Cramer's rule:

$$x_1 = \frac{\det(\boldsymbol{A}_1)}{\det(\boldsymbol{A})}, \qquad x_2 = \frac{\det(\boldsymbol{A}_2)}{\det(\boldsymbol{A})}$$

3. Inverse of matrix: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \quad \Rightarrow \quad \boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \quad \Rightarrow \quad \boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$

$$\boldsymbol{A}^{-1} = \frac{\mathrm{adj}(\boldsymbol{A})}{\det(\boldsymbol{A})} \qquad \text{(adjoint matrix over determinant)}$$

4. Gauss elimination (including back substitution)
5. LU decomposition (including both forward and back substitutions)

Gauss-Seidel Method

An **iterative approach** to solving n simultaneous linear equations.

- **Motivation** for iterative methods: Traditional methods like Gauss elimination can become problematic for very large systems.
- **Efficiency**: Iterative methods are often faster for large, sparse systems where many coefficients are zero.

Gauss-Seidel Method

Learning objectives:

- Understand the iterative formulation for solving linear systems
- Derive the update equations for each unknown variable
- Apply the Gauss-Seidel iteration scheme
- Compute absolute relative approximate errors
- Determine convergence criteria

Linear Systems

- A system of linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

- A system of n linear algebraic equations:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i$$

for $i = 1, 2, \ldots, n$. We want to solve for $x_1, x_2, \ldots, x_n$.

- Matrix form $\boldsymbol{Ax = b}$

Linear Systems

If the diagonal entries $a_{11}, a_{22}, \ldots, a_{nn}$ are nonzero, then

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n}{a_{11}}$$
$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}}$$
$$\vdots$$
$$x_n = \frac{b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}}{a_{nn}}$$

Thus,

$$x_1 = \frac{b_1 - \sum_{j\neq 1} a_{1j}x_j}{a_{11}} \quad x_2 = \frac{b_2 - \sum_{j\neq 2} a_{2j}x_j}{a_{22}} \quad \cdots \quad x_n = \frac{b_n - \sum_{j\neq n} a_{nj}x_j}{a_{nn}}$$

For any $x_i, i = 1, 2, \ldots, n$, we have

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j\right) \qquad \text{(Solve each equation for one variable)}$$

Gauss-Seidel Iteration

- For a general system with right-hand side b_i:

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}x_j\right)$$

$$= \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j\right), \quad i = 1, 2, \ldots, n$$

Hint:

$$x_1 \quad x_2 \quad \cdots \quad x_{i-1} \quad \boxed{x_i} \qquad \text{(for all } j < i\text{)}$$

$$x_{i+1} \quad x_{i+2} \quad \cdots \quad x_n \qquad \text{(for all } j > i\text{)}$$

- **Key idea**: Start with an initial point, then iteratively update each x_i using the most recent values of the other variables.
- In Gauss-Seidel iteration, we use the most recent estimates:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$$

with iterate k.

Convergence Criteria

- At the end of each iteration, compute the absolute relative approximate error for each variable:

$$\varepsilon_i = \left| \frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k+1)}} \right|$$

- Stopping condition:

$$\varepsilon_i \leq \epsilon \qquad \text{for all } i = 1, 2, \ldots, n$$

where ϵ is the pre-specified tolerance (e.g., 0.01%).

Simple 2×2 Matrices

A simple linear system:

$$4x_1 + x_2 = 9$$
$$x_1 + 3x_2 = 8$$

- Gauss-Seidel iteration:

$$x_1 = \frac{1}{4}(9 - x_2) \qquad x_2 = \frac{1}{3}(8 - x_1)$$

with initial points: $x_1^{(0)} = 0$ and $x_2^{(0)} = 0$.

Simple 2×2 Matrices

A simple linear system:

$$4x_1 + x_2 = 9$$
$$x_1 + 3x_2 = 8$$

- Gauss-Seidel iteration:

$$x_1 = \frac{1}{4}(9 - x_2) \qquad x_2 = \frac{1}{3}(8 - x_1)$$

  with initial points: $x_1^{(0)} = 0$ and $x_2^{(0)} = 0$.

- Iteratively update each variable:
  - First iteration:

$$x_1^{(1)} = \frac{1}{4}(9 - x_1^{(0)}) = \frac{9}{4} \qquad x_2^{(1)} = \frac{1}{3}(8 - x_1^{(1)}) = \frac{1}{3}(8 - \frac{9}{4}) = \frac{23}{12}$$

  - Second iteration:

$$x_1^{(2)} = \frac{1}{4} \times (9 - \frac{23}{12}) = \frac{85}{48} \approx 1.771 \qquad x_2^{(2)} = \frac{1}{3}(8 - \frac{85}{48}) = \frac{299}{144} \approx 2.076$$

- Exact solution: $x_1 = \frac{19}{11} \approx 1.727$ and $x_2 = \frac{23}{11} \approx 2.091$

Algorithm

How to solve $\boldsymbol{Ax} = \boldsymbol{b}$?

- Choose initial points $\boldsymbol{x}$ and tolerance ϵ
- Set iteration $k = 0$
- Repeat until convergence
  - For $i = 1$ to n:
    - Compute the most recent estimate:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right)$$

    - Compute absolute relative approximate error:

$$\varepsilon_i = \left| \frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k+1)}} \right|$$

  - Check convergence: If $\varepsilon_i < \epsilon$, stop
  - Set $k = k + 1$

Solving Linear Systems

Problem statement:

$$\begin{aligned} 12x_1 + 3x_2 - 5x_3 &= 1 \\ x_1 + 5x_2 + 3x_3 &= 28 \\ 3x_1 + 7x_2 + 13x_3 &= 76 \end{aligned}$$

- Initial points $x_1 = 1, x_2 = 0, x_3 = 1$
- **Check diagonal dominance**:
  - ✓ First row: $|a_{11}| = 12 \geq |a_{12}| + |a_{13}| = 8$
  - ✓ Second row: $|a_{22}| = 5 \geq |a_{21}| + |a_{23}| = 4$
  - ✓ Third row: $|a_{33}| = 13 \geq |a_{31}| + |a_{32}| = 10$

  The inequality is strictly greater for **at least one row** $\rightarrow$ **Convergence is guaranteed**.
- Does the following linear system ensure the convergence?

$$\begin{aligned} 3x_1 + 7x_2 + 13x_3 &= 76 \\ 12x_1 + 3x_2 - 5x_3 &= 1 \\ x_1 + 5x_2 + 3x_3 &= 28 \end{aligned}$$

Solving Linear Systems

- Gauss-Seidel iteration:

$$x_1 = \frac{1}{12}(1-3x_2+5x_3) \quad x_2 = \frac{1}{5}(28-x_1-3x_3) \quad x_3 = \frac{1}{13}(76-3x_1-7x_2)$$

- Initial points $x_1^{(0)} = 1, x_2^{(0)} = 0, x_3^{(0)} = 1$
- **First iteration**:

Solving Linear Systems

- Gauss-Seidel iteration:

$$x_1 = \frac{1}{12}(1-3x_2+5x_3) \quad x_2 = \frac{1}{5}(28-x_1-3x_3) \quad x_3 = \frac{1}{13}(76-3x_1-7x_2)$$

- Initial points $x_1^{(0)} = 1, x_2^{(0)} = 0, x_3^{(0)} = 1$
- **First iteration**:
  - Update x_1:

$$x_1^{(1)} = \frac{1}{12}(1 - 3x_2^{(0)} + 5x_3^{(0)}) = \frac{1}{12}(1 - 0 + 5) = 0.5 \qquad \varepsilon_1 = 100\%$$

  - Update x_2:

$$x_2^{(1)} = \frac{1}{5}(28 - x_1^{(1)} - 3x_3^{(0)}) = \frac{1}{5}(28 - 0.5 - 3) = 4.9 \qquad \varepsilon_2 = 100\%$$

  - Update x_3:

$$x_3^{(1)} = \frac{1}{13}(76-3x_1^{(1)}-7x_2^{(1)}) = \frac{40.2}{13} = 3.0923077 \qquad \varepsilon_3 = 67.66\%$$

Solving Linear Systems

Iteration (k)	x_1	ε_1	x_2	ε_2	x_3	ε_3
1	0.5	100%	4.9	100%	3.0923	67.662%
2	0.14679	240.61%	3.7153	31.889%	3.8118	18.874%
3	0.74275	80.236%	3.1644	17.408%	3.9708	4.0064%
4	0.94675	21.546%	3.0281	4.4996%	3.9971	0.65772%
5	0.99177	4.5391%	3.0034	0.82499%	4.0001	0.07438%

- Exact solution: $x_1 = 1$, $x_2 = 3$, and $x_3 = 4$

Python Code

```python
import numpy as np

def gauss_seidel(A, b, x_init, tolerance, max_iterations=100):
    n = len(b)
    x = x_init.copy().astype(float)
    for k in range(max_iterations):
        x_old = x.copy()
        for i in range(n):
            # Compute sum for j < i (using updated x values)
            sum_j_less = np.inner(A[i, :i], x[:i])
            # Compute sum for j > i (using old x values)
            sum_j_greater = np.inner(A[i, i+1:], x_old[i+1:])
            # Update x[i]
            x[i] = (b[i] - sum_j_less - sum_j_greater) / A[i, i]
        # Calculate absolute relative approximate error
        # Avoid division by zero if x[i] is 0
        error = np.max(np.abs((x - x_old) / np.where(x != 0, x, 1)))
        if error < tolerance:
            print(f"Converged in {k+1} iterations.")
            return x
    print("Max iterations reached without convergence.")
    return x

# Example
A = np.array([[8, 2, -2], [10, 2, 4], [12, 2, 2]])
b = np.array([-2, 4, 6])
x_start = np.zeros(3) # Initial points
solution = gauss_seidel(A, b, x_start, tolerance=0.001)
print(f"Gauss-Seidel Solution: {solution}")
```

Advantage & Limitation

Advantages:

- Simple to implement
- Uses less memory than direct methods
- Good for large sparse systems
- Self-correcting (round-off not amplified)

Limitations:

- May not converge for all systems
- Requires diagonal dominance or symmetry for convergence
- Slower than direct methods for dense systems
- Convergence can be slow for ill-conditioned systems

Quick Summary

Monday's Class:

- How to solve linear systems iteratively?
- Basic idea of Gauss-Seidel method
- Algorithm & convergence criteria
- Diagonal dominance for guaranteed convergence

Quizzes Now!

- **Today's participation** (ungraded survey): Please check out

  "`Class Participation Quiz 23`"

  Time slot: **2:30PM – 3:00PM**

  on Canvas.

Linear Systems

- A system of linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

- Rewrite the problem:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}_{\boldsymbol{A}\in\mathbb{R}^{n\times n}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{\boldsymbol{x}\in\mathbb{R}^{n}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{\boldsymbol{b}\in\mathbb{R}^{n}}$$

- Matrix form $\boldsymbol{Ax = b}$

Update Equations

How to solve $\boldsymbol{Ax} = \boldsymbol{b}$?

- Essential idea: Conjugate gradient solves linear system iteratively.
- $\boldsymbol{x}$ is updated by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \underbrace{\alpha_k}_{\text{step size (unknown)}} \boldsymbol{d}_k$$

  with initial points $\boldsymbol{x}_0$

- $\boldsymbol{d}_k$ is the search direction (initial search direction $\boldsymbol{d}_0 = \boldsymbol{r}_0$):

$$\boldsymbol{d}_{k+1} = \underbrace{\boldsymbol{r}_{k+1}}_{\text{residual}} + \underbrace{\beta_k}_{\text{step size (unknown)}} \boldsymbol{d}_k$$

- How to write down the update of residual $\boldsymbol{r}_{k+1}$?

Update Equations

How to solve $\boldsymbol{Ax} = \boldsymbol{b}$?

- Essential idea: Conjugate gradient solves linear system iteratively.
- $\boldsymbol{x}$ is updated by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \underbrace{\alpha_k}_{\text{step size (unknown)}} \boldsymbol{d}_k$$

  with initial points $\boldsymbol{x}_0$

- $\boldsymbol{d}_k$ is the search direction (initial search direction $\boldsymbol{d}_0 = \boldsymbol{r}_0$):

$$\boldsymbol{d}_{k+1} = \underbrace{\boldsymbol{r}_{k+1}}_{\text{residual}} + \underbrace{\beta_k}_{\text{step size (unknown)}} \boldsymbol{d}_k$$

- How to write down the update of residual $\boldsymbol{r}_{k+1}$?

$$\begin{aligned} & \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k \\ \Rightarrow \quad & \boldsymbol{A}\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \alpha_k \boldsymbol{A}\boldsymbol{d}_k \\ \Rightarrow \quad & \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k \\ \Rightarrow \quad & \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k \end{aligned}$$

Update Equations

To summarize, we have the following update equations to solve $\boldsymbol{Ax} = \boldsymbol{b}$

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{(variables)} \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k && \text{(residuals)} \\ \boldsymbol{d}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k && \text{(search direction)} \end{aligned}$$

In an n-dimensional space, conjugate gradient is guaranteed to converge to the exact solution in at most n steps.

Update Equations

To summarize, we have the following update equations to solve $\boldsymbol{Ax} = \boldsymbol{b}$

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{(variables)} \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k && \text{(residuals)} \\ \boldsymbol{d}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k && \text{(search direction)} \end{aligned}$$

In an n-dimensional space, conjugate gradient is guaranteed to converge to the exact solution in at most n steps.

△ But how to write down the update equations of α_k and β_k?

Residual orthogonality + Search direction conjugacy ($\boldsymbol{A}$-orthogonality)

$$\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_k = 0 \qquad \text{and} \qquad \boldsymbol{d}_{k+1}^\top \boldsymbol{A}\boldsymbol{d}_k = 0$$

for any iterations k and $k+1$.

Update Equations of α_k, β_k

Recall that

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{(variables)} \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k && \text{(residuals)} \\ \boldsymbol{d}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k && \text{(search direction)} \end{aligned}$$

- Imposing $\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_k = \boldsymbol{r}_k^\top \boldsymbol{r}_{k+1} = 0$ (orthogonality), we have

$$\boldsymbol{r}_k^\top (\underbrace{\boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k}_{\boldsymbol{r}_{k+1}}) = 0 \quad \Rightarrow \quad \alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{r}_k^\top \boldsymbol{A}\boldsymbol{d}_k}$$

Update Equations of α_k, β_k

Recall that

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{(variables)} \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k && \text{(residuals)} \\ \boldsymbol{d}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k && \text{(search direction)} \end{aligned}$$

- Imposing $\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_k = \boldsymbol{r}_k^\top \boldsymbol{r}_{k+1} = 0$ (orthogonality), we have

$$\boldsymbol{r}_k^\top (\underbrace{\boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k}_{\boldsymbol{r}_{k+1}}) = 0 \quad \Rightarrow \quad \alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{r}_k^\top \boldsymbol{A}\boldsymbol{d}_k}$$

  Denominator: $\boldsymbol{r}_k^\top \boldsymbol{A}\boldsymbol{d}_k = (\boldsymbol{d}_k - \beta_{k-1}\boldsymbol{d}_{k-1})^\top \boldsymbol{A}\boldsymbol{d}_k = \boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k$

- Update equation:

$$\alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k}$$

Update Equations of α_k, β_k

Recall that

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{(variables)} \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k && \text{(residuals)} \\ \boldsymbol{d}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k && \text{(search direction)} \end{aligned}$$

- Imposing $\boldsymbol{d}_{k+1}^\top \boldsymbol{A}\boldsymbol{d}_k = 0$ ($\boldsymbol{A}$-orthogonality), we have

$$\underbrace{(\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k)}_{\boldsymbol{d}_{k+1}}{}^\top \boldsymbol{A}\boldsymbol{d}_k = 0 \quad \Rightarrow \quad \beta_k = -\frac{\boldsymbol{r}_{k+1}^\top \boldsymbol{A}\boldsymbol{d}_k}{\boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k}$$

Update Equations of α_k, β_k

Recall that

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{(variables)} \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k && \text{(residuals)} \\ \boldsymbol{d}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k && \text{(search direction)} \end{aligned}$$

- Imposing $\boldsymbol{d}_{k+1}^\top \boldsymbol{A}\boldsymbol{d}_k = 0$ ($\boldsymbol{A}$-orthogonality), we have

$$\underbrace{(\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k)}_{\boldsymbol{d}_{k+1}}{}^\top \boldsymbol{A}\boldsymbol{d}_k = 0 \quad \Rightarrow \quad \beta_k = -\frac{\boldsymbol{r}_{k+1}^\top \boldsymbol{A}\boldsymbol{d}_k}{\boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k}$$

- Considering

$$\boldsymbol{A}\boldsymbol{d}_k = \frac{\boldsymbol{r}_k - \boldsymbol{r}_{k+1}}{\alpha_k} \qquad \alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k}$$

- Update equation (with residuals):

$$\beta_k = \frac{\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^\top \boldsymbol{r}_k}$$

Conjugate Gradient

To summarize, we have the following update equations to solve $\boldsymbol{Ax} = \boldsymbol{b}$

$$\alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k} \qquad \text{(step size)}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k \qquad \text{(variables)}$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k \qquad \text{(residuals)}$$

$$\beta_k = \frac{\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^\top \boldsymbol{r}_k} \qquad \text{(step size)}$$

$$\boldsymbol{d}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k \qquad \text{(search direction)}$$

A Simple Linear System

Problem statement:

$$\begin{aligned} 4x_1 + x_2 &= 1 \\ x_1 + 3x_2 &= 2 \end{aligned} \quad \text{with} \quad \boldsymbol{A} = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Suppose we have initial points $x_1 = 0, x_2 = 0$ (i.e., $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ for the initialization)

- Initial residual:

$$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

- Initial search direction:

$$\boldsymbol{d}_0 = \boldsymbol{r}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

A Simple Linear System

First iteration:

Given $\boldsymbol{A} = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}$, $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\boldsymbol{r}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, and $\boldsymbol{d}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, we have

$$\alpha_0 = \frac{\boldsymbol{r}_0^\top \boldsymbol{r}_0}{\boldsymbol{d}_0^\top \boldsymbol{A} \boldsymbol{d}_0} = \frac{1}{4} \qquad \text{(step size)}$$

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \alpha_0 \boldsymbol{d}_0 = \begin{bmatrix} 1/4 \\ 1/2 \end{bmatrix} \qquad \text{(variables)}$$

$$\boldsymbol{r}_1 = \boldsymbol{r}_0 - \alpha_0 \boldsymbol{A} \boldsymbol{d}_0 = \begin{bmatrix} -1/2 \\ 1/4 \end{bmatrix} \qquad \text{(residuals)}$$

$$\beta_0 = \frac{\boldsymbol{r}_1^\top \boldsymbol{r}_1}{\boldsymbol{r}_0^\top \boldsymbol{r}_0} = 1/16 \qquad \text{(step size)}$$

$$\boldsymbol{d}_1 = \boldsymbol{r}_1 + \beta_0 \boldsymbol{d}_0 = \begin{bmatrix} -7/16 \\ 3/8 \end{bmatrix} \qquad \text{(search direction)}$$

A Simple Linear System

Second iteration:

Given $\boldsymbol{A} = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}$, $\boldsymbol{x}_1 = \begin{bmatrix} 1/4 \\ 1/2 \end{bmatrix}$, $\boldsymbol{r}_1 = \begin{bmatrix} -1/2 \\ 1/4 \end{bmatrix}$, and $\boldsymbol{d}_1 = \begin{bmatrix} -7/16 \\ 3/8 \end{bmatrix}$, we have

$$\alpha_1 = \frac{\boldsymbol{r}_1^\top \boldsymbol{r}_1}{\boldsymbol{d}_1^\top \boldsymbol{A}\boldsymbol{d}_1} = \frac{4}{11} \qquad \text{(step size)}$$

$$\boldsymbol{x}_2 = \boldsymbol{x}_1 + \alpha_1 \boldsymbol{d}_1 = \begin{bmatrix} 1/11 \\ 7/11 \end{bmatrix} \qquad \text{(variables)}$$

$$\boldsymbol{r}_2 = \boldsymbol{r}_1 - \alpha_1 \boldsymbol{A}\boldsymbol{d}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \text{(residuals)}$$

$$\beta_1 = \frac{\boldsymbol{r}_2^\top \boldsymbol{r}_2}{\boldsymbol{r}_1^\top \boldsymbol{r}_1} = 0 \qquad \text{(step size)}$$

$$\boldsymbol{d}_2 = \boldsymbol{r}_2 + \beta_1 \boldsymbol{d}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \text{(search direction)}$$

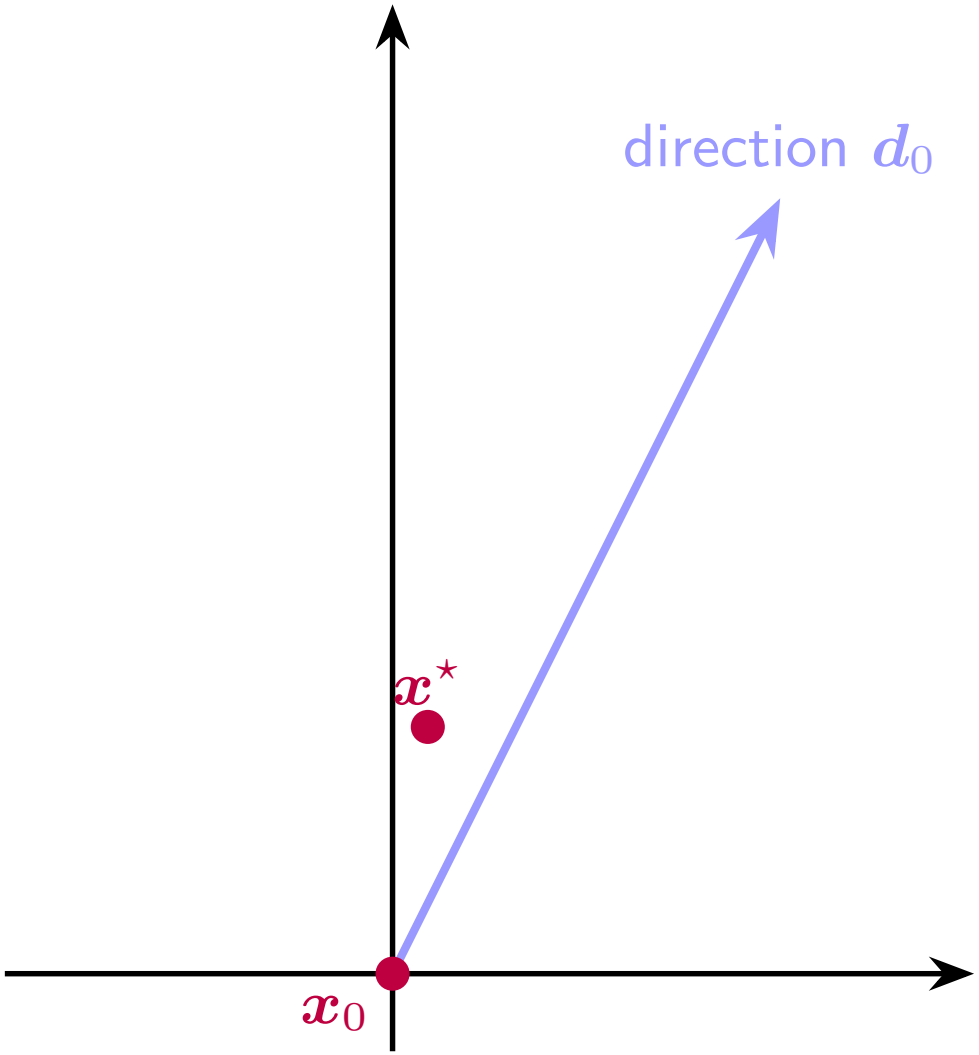
Zero residuals $\rightarrow$ exact solution.

Quick Summary

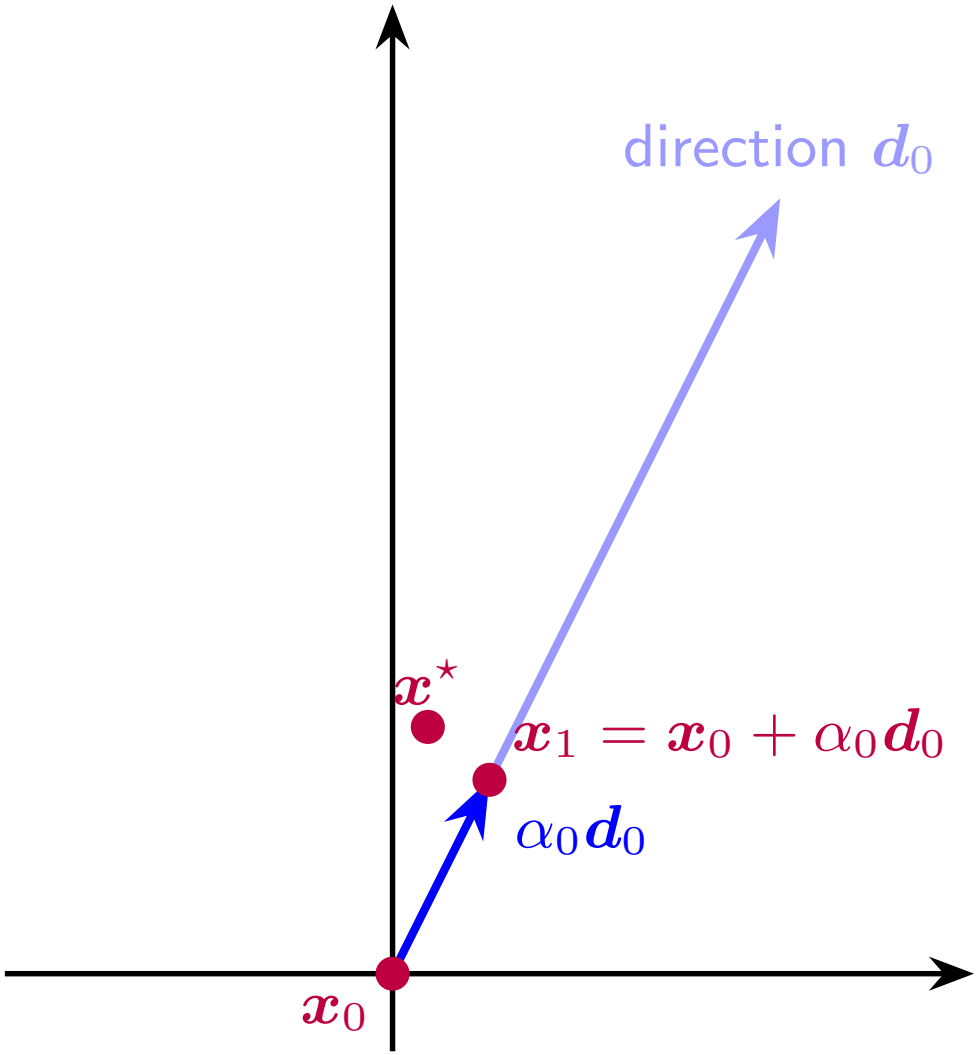
Wednesday's Class:

- Conjugate gradient: iterative method for solving linear systems
- How to update $\boldsymbol{x}$, $\boldsymbol{r}$, and search direction $\boldsymbol{d}$?
- How to write down step sizes α_k and β_k?
- A simple linear system for demonstrating conjugate gradient

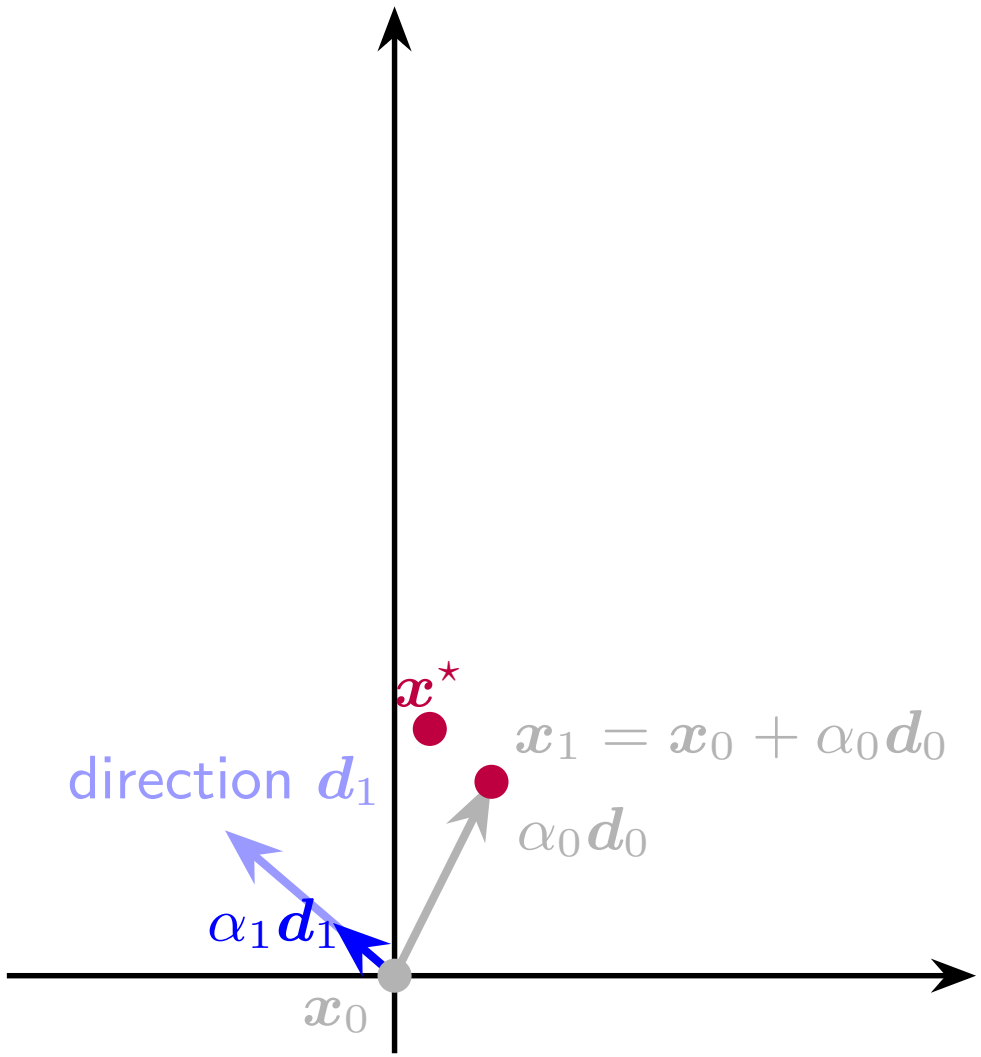
Geometric Interpretation: $x_{k+1} = x_k + \alpha_k d_k$



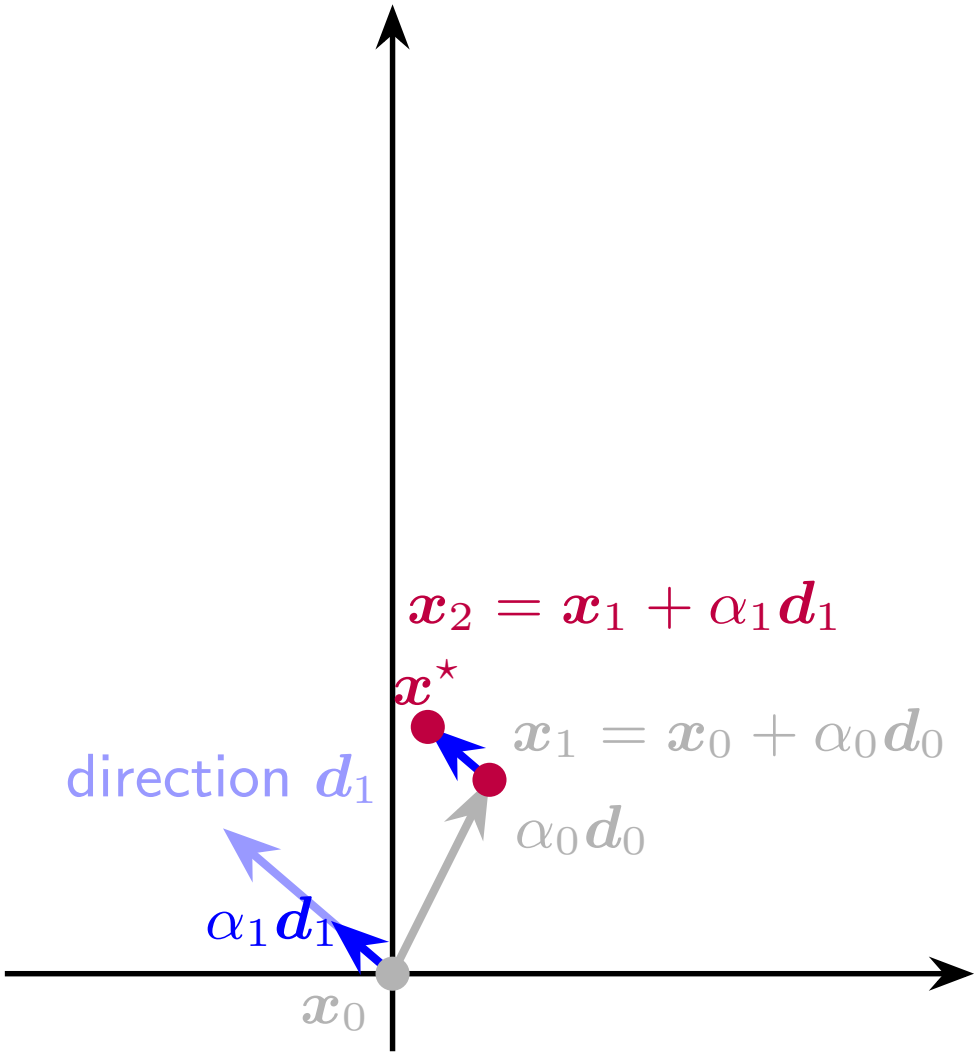
Geometric Interpretation: $x_{k+1} = x_k + \alpha_k d_k$



Geometric Interpretation: $x_{k+1} = x_k + \alpha_k d_k$



Geometric Interpretation: $x_{k+1} = x_k + \alpha_k d_k$



Quizzes Now!

- **Today's participation** (ungraded survey): Please check out

  "`Class Participation Quiz 24`"

  Time slot: **2:30PM – 3:00PM**

  on Canvas.

Conjugate Gradient

To summarize, we have the following update equations to solve $\boldsymbol{Ax} = \boldsymbol{b}$

$$\alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{d}_k^\top \boldsymbol{A}\boldsymbol{d}_k} \qquad \text{(step size)}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k \qquad \text{(variables)}$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{d}_k \qquad \text{(residuals)}$$

$$\beta_k = \frac{\boldsymbol{r}_{k+1}^\top \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^\top \boldsymbol{r}_k} \qquad \text{(step size)}$$

$$\boldsymbol{d}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{d}_k \qquad \text{(search direction)}$$

Convergence criteria: If

$$\sqrt{\boldsymbol{r}_k^\top \boldsymbol{r}_k} \leq \epsilon$$

then stop the iterative process.

Conjugate Gradient Algorithm

```python
import numpy as np

def conj_grad(A, b, x, tol = 1e-8, max_iter = 100):
    r = b - A @ x
    d = r.copy()
    r_inner = np.inner(r, r)
    for it in range(max_iter):
        if np.sqrt(r_inner) <= tol:
            print(f"Converged in {it} iterations.")
            break

        Ad = A @ d
        alpha = r_inner / np.inner(d, Ad)

        x = x + alpha * d
        r = r - alpha * Ad

        r_inner_new = np.inner(r, r)
        beta = r_inner_new / r_inner
        d = r + beta * d
        r_inner = r_inner_new

    return x
```

Ill-Conditioned Systems

- Problem:

$$x_1 + 2x_2 = 10$$
$$1.1x_1 + 2x_2 = 10.4$$

- Exact solution: Solving this system yields $x_1 = 4$ and $x_2 = 3$.
- What happens with a small change in $\boldsymbol{b}$?
  - Let's change the right-hand side constant b_2 from 10.4 to 10.8:

$$x_1 + 2x_2 = 10$$
$$1.1x_1 + 2x_2 = 10.8$$

  - Subtract the equations: $0.1x_1 = 0.8 \Rightarrow x_1 = 8$
  - Substitute back: $8 + 2x_2 = 10 \Rightarrow x_2 = 1$
- **Result**: A small change in the input vector $\boldsymbol{b}$ caused the solution to jump from $(4, 3)$ to $(8, 1)$.
- **Small change in $\boldsymbol{b} \rightarrow$ remarkable change in $\boldsymbol{x}$**

Ill-Conditioned Systems

Why is this happening?

- For matrix $\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 1.1 & 2 \end{bmatrix}$
  - Frobenius norm of $\boldsymbol{A}$:

$$\|\boldsymbol{A}\|_F = \sqrt{1^2 + 2^2 + 1.1^2 + 2^2} = \sqrt{10.21} \approx 3.195$$

  - Inverse of $\boldsymbol{A}$:

$$\boldsymbol{A}^{-1} = \frac{\text{adj}(\boldsymbol{A})}{\det(\boldsymbol{A})} = \frac{1}{-0.2} \begin{bmatrix} 2 & -2 \\ -1.1 & 1 \end{bmatrix} = \begin{bmatrix} -10 & 10 \\ 5.5 & -5 \end{bmatrix}$$

  - Frobenius norm of $\boldsymbol{A}^{-1}$:

$$\|\boldsymbol{A}^{-1}\|_F = \sqrt{(-10)^2 + 10^2 + 5.5^2 + (-5)^2} = \sqrt{255.25} \approx 15.977$$

- Condition number:

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|_F \cdot \|\boldsymbol{A}^{-1}\|_F \approx 51.05$$

Ill-Conditioned Systems

Python programming

- Input:

```
A = np.array([[1, 2], [1.1, 2]])
b = np.array([10, 10.4])
x = np.zeros(2)
x = conj_grad(A, b, x, tol = 1e-4, max_iter = 55)
print(x)
```

- Output:

```
Converged in 18 iterations.
[3.99917683 3.00043664]
```

$$\begin{aligned} x_1 + 2x_2 &= 10 \\ 1.1x_1 + 2x_2 &= 10.4 \end{aligned} \quad \Rightarrow \quad \boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 1.1 & 2 \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} 10 \\ 10.4 \end{bmatrix}$$

Sylvester Equation

A **Sylvester equation** is a matrix equation of the form:

$$\boldsymbol{AX} + \boldsymbol{XB} = \boldsymbol{C}$$

- Known: $\boldsymbol{A} \in \mathbb{R}^{m \times m}$, $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{C} \in \mathbb{R}^{m \times n}$
- Unknown: $\boldsymbol{X} \in \mathbb{R}^{m \times n}$

How to solve this linear system?

Vectorization

- Toy example:

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \Rightarrow \quad \text{vec}(\boldsymbol{A}) = \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix}$$

- For any matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$:

$$\boldsymbol{X} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \\ | & | & & | \end{bmatrix} \quad \Rightarrow \quad \text{vec}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{m1} \\ \vdots \\ x_{1n} \\ x_{2n} \\ \vdots \\ x_{mn} \end{bmatrix} \in \mathbb{R}^{mn}$$

Kronecker Product $\otimes$

- Mathematical expression:

$$\boldsymbol{X} \otimes \boldsymbol{Y} = \begin{bmatrix} x_{11}\boldsymbol{Y} & x_{12}\boldsymbol{Y} & \cdots & x_{1n}\boldsymbol{Y} \\ x_{21}\boldsymbol{Y} & x_{22}\boldsymbol{Y} & \cdots & x_{2n}\boldsymbol{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}\boldsymbol{Y} & x_{m2}\boldsymbol{Y} & \cdots & x_{mn}\boldsymbol{Y} \end{bmatrix}$$

Kronecker Product $\otimes$

- Verify that the Kronecker product of

$$\boldsymbol{X} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad \boldsymbol{Y} = \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$$

is

$$\boldsymbol{X} \otimes \boldsymbol{Y} = \begin{bmatrix} 1 \times \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} & 2 \times \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} \\ 3 \times \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} & 4 \times \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 6 & 7 & 10 & 12 & 14 \\ 8 & 9 & 10 & 16 & 18 & 20 \\ 15 & 18 & 21 & 20 & 24 & 28 \\ 24 & 27 & 30 & 32 & 36 & 40 \end{bmatrix}$$

- Size: 4 rows & 6 columns

Property of Kronecker Product

For any $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, it always holds that

$$\operatorname{vec}(\boldsymbol{AXB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

- Recall that the unknown $\boldsymbol{X} \in \mathbb{R}^{m \times n}$:

  $$\boldsymbol{AX} + \boldsymbol{XB} = \boldsymbol{C}$$

  for the known $\boldsymbol{A} \in \mathbb{R}^{m \times m}$, $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{C} \in \mathbb{R}^{m \times n}$.

Property of Kronecker Product

For any $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, it always holds that

$$\operatorname{vec}(\boldsymbol{AXB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

- Recall that the unknown $\boldsymbol{X} \in \mathbb{R}^{m \times n}$:

  $$\boldsymbol{AX} + \boldsymbol{XB} = \boldsymbol{C}$$

  for the known $\boldsymbol{A} \in \mathbb{R}^{m \times m}$, $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{C} \in \mathbb{R}^{m \times n}$.
- **First** term of the left-hand side:

  $$\operatorname{vec}(\boldsymbol{AX}) = \operatorname{vec}(\boldsymbol{AXI}_n) = (\boldsymbol{I}_n \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

- **Second** term of the right-hand side:

  $$\operatorname{vec}(\boldsymbol{XB}) = \operatorname{vec}(\boldsymbol{I}_m\boldsymbol{XB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{I}_m) \operatorname{vec}(\boldsymbol{X})$$

Property of Kronecker Product

For any $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, it always holds that

$$\operatorname{vec}(\boldsymbol{AXB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

- Recall that the unknown $\boldsymbol{X} \in \mathbb{R}^{m \times n}$:

  $$\boldsymbol{AX} + \boldsymbol{XB} = \boldsymbol{C}$$

  for the known $\boldsymbol{A} \in \mathbb{R}^{m \times m}$, $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{C} \in \mathbb{R}^{m \times n}$.
- **First** term of the left-hand side:

  $$\operatorname{vec}(\boldsymbol{AX}) = \operatorname{vec}(\boldsymbol{AXI}_n) = (\boldsymbol{I}_n \otimes \boldsymbol{A}) \operatorname{vec}(\boldsymbol{X})$$

- **Second** term of the right-hand side:

  $$\operatorname{vec}(\boldsymbol{XB}) = \operatorname{vec}(\boldsymbol{I}_m\boldsymbol{XB}) = (\boldsymbol{B}^\top \otimes \boldsymbol{I}_m) \operatorname{vec}(\boldsymbol{X})$$

- Thus, we have

  $$(\underbrace{\boldsymbol{I}_n \otimes \boldsymbol{A} + \boldsymbol{B}^\top \otimes \boldsymbol{I}_m}_{(mn)\times(mn)}) \underbrace{\operatorname{vec}(\boldsymbol{X})}_{(mn)\times 1} = \operatorname{vec}(\boldsymbol{C})$$

Quick Summary

Friday's Class:

- Geometric interpretation of conjugate gradient
- Convergence criteria
- Python programming
- Ill-conditioned systems
- Matrix equation: Sylvester equation