Summer Class: Machine Learning and Data Science

Shuaifeng Li

Department of Physics, University of Michigan

June & July 2025

Table of Contents

Matrix as a Vector Transformer

Any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ into a new vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Matrix as a Vector Transformer

Any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ into a new vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Example:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{y} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

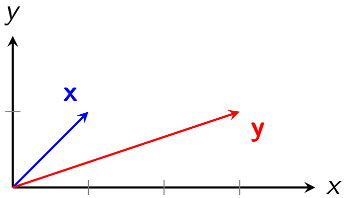
Matrix as a Vector Transformer

Any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ into a new vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Example:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{y} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$



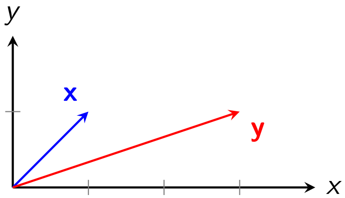
Matrix as a Vector Transformer

Any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ into a new vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Example:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{y} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$



What does this mean geometrically?

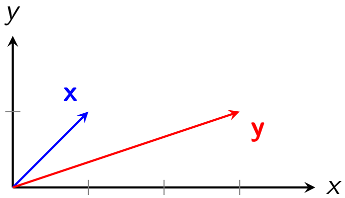
Matrix as a Vector Transformer

Any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ transforms a vector $\mathbf{x} \in \mathbb{R}^n$ into a new vector $\mathbf{y} \in \mathbb{R}^m$:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

Example:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{y} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

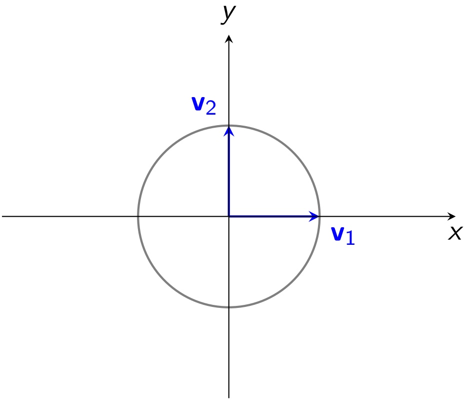


What does this mean geometrically?

- Changes length (scaling).
- Changes direction (rotation/skewing).
- May collapse dimensions.

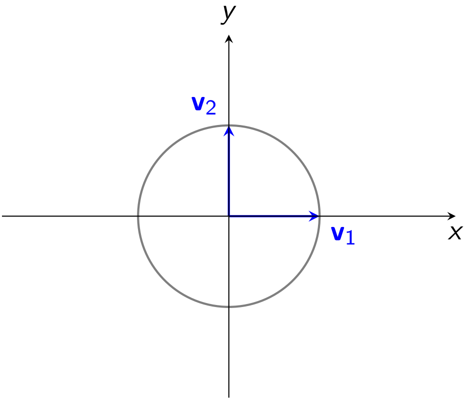
Transforming a Set of Vectors

Input space: Unit Circle

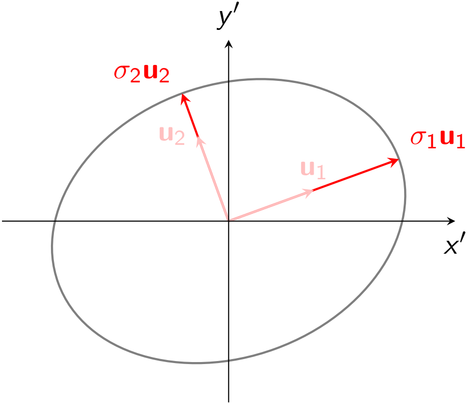


Transforming a Set of Vectors

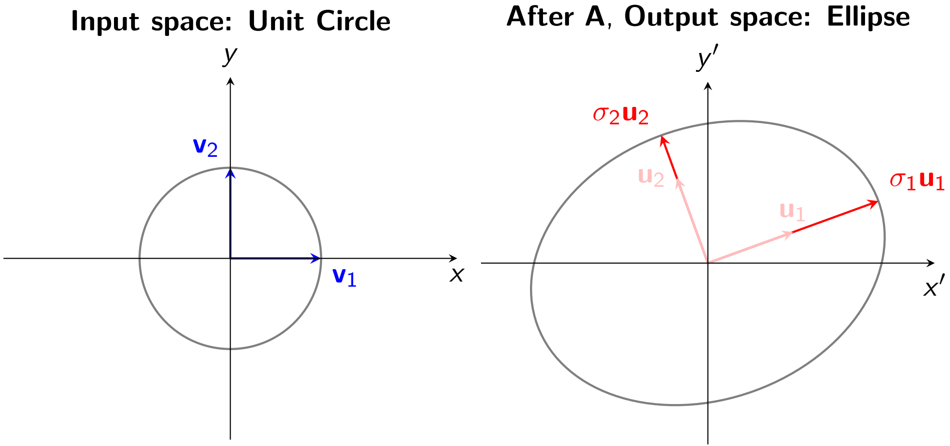
Input space: Unit Circle



After A, Output space: Ellipse



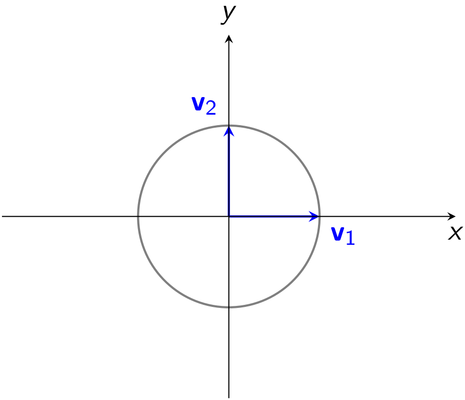
Transforming a Set of Vectors



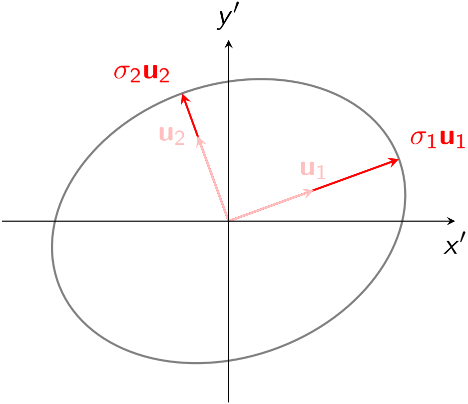
A matrix maps unit vectors along $\mathbf{v}_1, \mathbf{v}_2$ to scaled orthonormal vectors along $\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2$:

Transforming a Set of Vectors

Input space: Unit Circle



After A, Output space: Ellipse



A matrix maps unit vectors along $\mathbf{v}_1, \mathbf{v}_2$ to scaled orthonormal vectors along $\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2$:

$$Av_i = \sigma_i u_i \Rightarrow AV = U\Sigma \Rightarrow A = U\Sigma V^*.$$

What is Singular Value Decomposition?

- The Singular Value Decomposition (SVD) is a fundamental matrix factorization in linear algebra. It asserts that any real or complex matrix $A \in \mathbb{C}^{m \times n}$ can be decomposed into the product of three specific matrices.

$$A = U\Sigma V^*$$

What is Singular Value Decomposition?

- The Singular Value Decomposition (SVD) is a fundamental matrix factorization in linear algebra. It asserts that any real or complex matrix $A \in \mathbb{C}^{m \times n}$ can be decomposed into the product of three specific matrices.

$$A = U \Sigma V^*$$

- $U \in \mathbb{C}^{m \times m}$: An orthogonal (or unitary for complex) matrix whose columns are the left singular vectors.
- $\Sigma \in \mathbb{C}^{m \times n}$: A diagonal matrix (with non-negative real numbers in decreasing order on the diagonal) containing the singular values σ.
- $V \in \mathbb{C}^{n \times n}$: An orthogonal (or unitary for complex) matrix whose columns are the right singular vectors.

Properties of SVD Components

- Orthogonality of U and V:

Properties of SVD Components

- Orthogonality of U and V:
    - $U^T U = I_m$ and $V^T V = I_n$.

Properties of SVD Components

- Orthogonality of U and V:
  - $U^T U = I_m$ and $V^T V = I_n$.
  - The columns of U/V are mutually orthogonal unit vectors.
  - This orthogonality is crucial as it implies that U and V represent rotations and/or reflections, which preserve lengths in a geometric sense.

Properties of SVD Components

- Orthogonality of U and V:
  - $U^T U = I_m$ and $V^T V = I_n$.
  - The columns of U/V are mutually orthogonal unit vectors.
  - This orthogonality is crucial as it implies that U and V represent rotations and/or reflections, which preserve lengths in a geometric sense.
- Ordering of Singular Values:

Properties of SVD Components

- Orthogonality of U and V:
  - $U^T U = I_m$ and $V^T V = I_n$.
  - The columns of U/V are mutually orthogonal unit vectors.
  - This orthogonality is crucial as it implies that U and V represent rotations and/or reflections, which preserve lengths in a geometric sense.
- Ordering of Singular Values:
  - $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where $r = \mathrm{rank}(A)$. Any remaining diagonal entries are zero.

Properties of SVD Components

- Orthogonality of U and V:
  - $U^T U = I_m$ and $V^T V = I_n$.
  - The columns of U/V are mutually orthogonal unit vectors.
  - This orthogonality is crucial as it implies that U and V represent rotations and/or reflections, which preserve lengths in a geometric sense.
- Ordering of Singular Values:
  - $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where $r = \mathrm{rank}(A)$. Any remaining diagonal entries are zero.
  - This ordering is fundamental for low-rank approximation and identifying dominant components. The largest singular values correspond to the directions of greatest variance or importance in the data.

Properties of SVD Components

- Orthogonality of U and V:
  - $U^T U = I_m$ and $V^T V = I_n$.
  - The columns of U/V are mutually orthogonal unit vectors.
  - This orthogonality is crucial as it implies that U and V represent rotations and/or reflections, which preserve lengths in a geometric sense.
- Ordering of Singular Values:
  - $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where $r = \mathrm{rank}(A)$. Any remaining diagonal entries are zero.
  - This ordering is fundamental for low-rank approximation and identifying dominant components. The largest singular values correspond to the directions of greatest variance or importance in the data.
  - If $m > n$, $\Sigma = \begin{bmatrix} \hat{\Sigma} \in \mathbb{C}^{n\times n} \\ 0 \end{bmatrix}$. If $m < n$, $\Sigma = \left[\hat{\Sigma} \in \mathbb{C}^{m\times m}, 0\right]$. If $m = n$, $\Sigma = \hat{\Sigma} \in \mathbb{C}^{m\times n}$.

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

1. Rotation by V^T:

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

1. Rotation by V^T:
   - V^T is an orthogonal matrix. It rotates the input vectors in the domain (input space $\mathbb{R}^n$).
   - It aligns the basis vectors of the domain along the principal axes (directions of v_i).

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

1. Rotation by V^T:
   - V^T is an orthogonal matrix. It rotates the input vectors in the domain (input space $\mathbb{R}^n$).
   - It aligns the basis vectors of the domain along the principal axes (directions of v_i).
2. Scaling by Σ:

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

1. Rotation by V^T:
   - V^T is an orthogonal matrix. It rotates the input vectors in the domain (input space $\mathbb{R}^n$).
   - It aligns the basis vectors of the domain along the principal axes (directions of v_i).
2. Scaling by Σ:
   - Σ is a diagonal matrix. It scales the rotated vectors along these new axes.
   - The scaling factor along each axis i is the singular value σ_i. Some axes might be stretched ($\sigma_i > 1$), some shrunk ($\sigma_i < 1$), and some collapsed ($\sigma_i = 0$).

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

1. Rotation by V^T:
   - V^T is an orthogonal matrix. It rotates the input vectors in the domain (input space $\mathbb{R}^n$).
   - It aligns the basis vectors of the domain along the principal axes (directions of v_i).
2. Scaling by Σ:
   - Σ is a diagonal matrix. It scales the rotated vectors along these new axes.
   - The scaling factor along each axis i is the singular value σ_i. Some axes might be stretched ($\sigma_i > 1$), some shrunk ($\sigma_i < 1$), and some collapsed ($\sigma_i = 0$).
3. Rotation by U:

Geometric Interpretation of SVD

Any linear transformation $X_{new} = AX_0$ represented by a matrix A can be decomposed into a sequence of three fundamental geometric operations:

1. Rotation by V^T:
   - V^T is an orthogonal matrix. It rotates the input vectors in the domain (input space $\mathbb{R}^n$).
   - It aligns the basis vectors of the domain along the principal axes (directions of v_i).
2. Scaling by Σ:
   - Σ is a diagonal matrix. It scales the rotated vectors along these new axes.
   - The scaling factor along each axis i is the singular value σ_i. Some axes might be stretched ($\sigma_i > 1$), some shrunk ($\sigma_i < 1$), and some collapsed ($\sigma_i = 0$).
3. Rotation by U:
   - U is an orthogonal matrix. It rotates the scaled vectors into their final positions in the codomain (output space $\mathbb{R}^m$).
   - It aligns the scaled vectors with the basis vectors of the range (u_i).

Visualizing the Transformation

Let's start with an ellipse and apply the transformation matrix:

$$A = \begin{pmatrix} 0.4330 & -0.7500 \\ 1.2500 & -0.4330 \end{pmatrix}$$

.

- **Rotation (V^T):** Rotation by 30°.

$$V^T = \begin{pmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{pmatrix} \approx R(30^\circ)$$

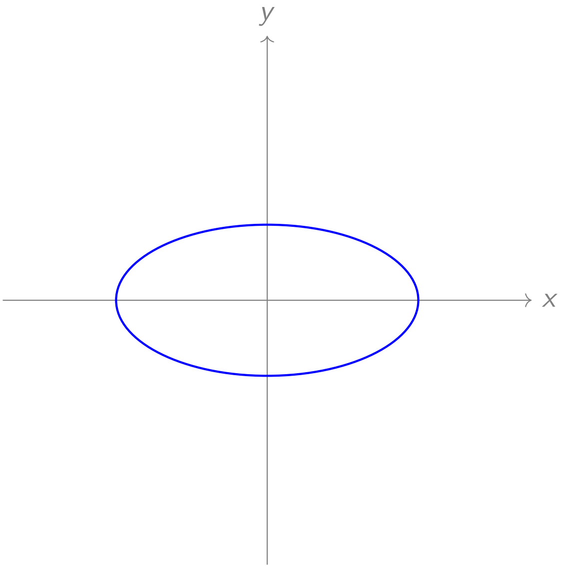
- **Scaling (Σ):**

$$\Sigma = \begin{pmatrix} 1.5 & 0 \\ 0 & 0.5 \end{pmatrix}$$

- **Rotation (U):** Rotation by 60°.

$$U = \begin{pmatrix} 0.5 & -0.866 \\ 0.866 & 0.5 \end{pmatrix} \approx R(60^\circ)$$

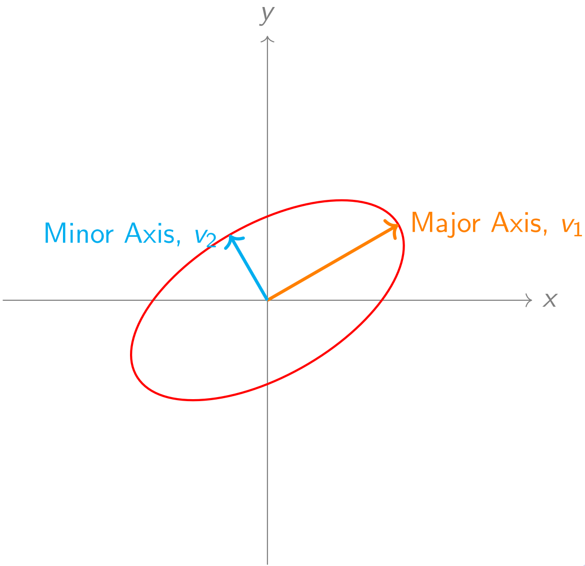
Visualizing the Transformation



Start: Ellipse

$x_0 \in \mathbb{R}^2$

Visualizing the Transformation

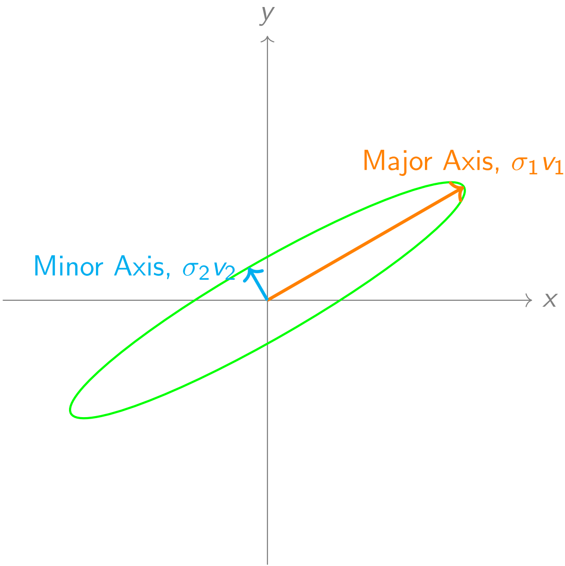


Step 1: Apply $V^\top$

$$x_1 = V^\top x_0$$

$V^\top$ rotates the entire coordinate system so that the directions we want to stretch and shrink are perfectly aligned with the main x and y axes (major and minor axes are rotated to lie along the 30° and 120° lines).

Visualizing the Transformation

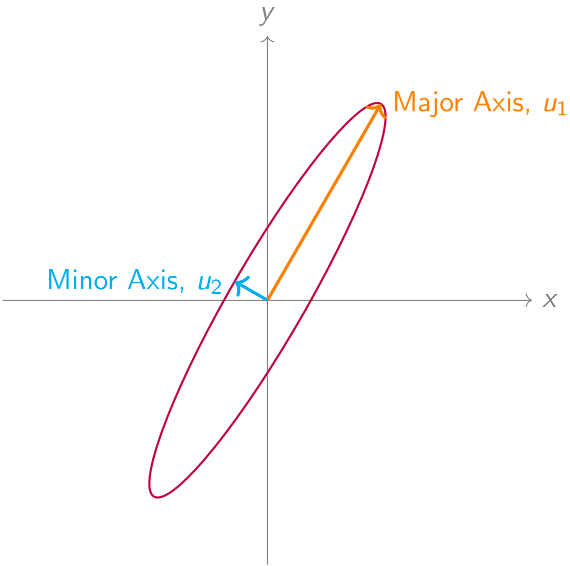


Step 2: Apply Σ

$$x_2 = \Sigma x_1 = \Sigma V^\top x_0$$

It scales everything by 1.5 along the new 30° direction and by 0.5 along the new 120° direction.

Visualizing the Transformation



Step 3: Apply U

$$x_{\text{new}} = Ux_2 = U\Sigma V^\top x_0$$

The columns of the U define the directions of the final ellipse's axes.

Final rotation gives:

$$\boxed{x_{\text{new}} = Ax_0}$$

Full vs Economy (Reduced) SVD

- Full SVD: $U \in \mathbb{C}^{m \times m}$, $\Sigma \in \mathbb{C}^{m \times n}$, $V \in \mathbb{C}^{n \times n}$.

Full vs Economy (Reduced) SVD

- Full SVD: $U \in \mathbb{C}^{m\times m}$, $\Sigma \in \mathbb{C}^{m\times n}$, $V \in \mathbb{C}^{n\times n}$.
- Economy SVD:
  - **Case 1: Tall-and-Skinny Matrix** $(m > n)$

$$A_{m\times n} = \underbrace{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_n \\ | & & | \end{bmatrix}}_{U\in\mathbb{C}^{m\times n}} \underbrace{\begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{bmatrix}}_{\Sigma\in\mathbb{C}^{n\times n}} \underbrace{\begin{bmatrix} - & v_1^\top & - \\ & \vdots & \\ - & v_n^\top & - \end{bmatrix}}_{V^\top\in\mathbb{C}^{n\times n}}$$

Full vs Economy (Reduced) SVD

- Full SVD: $U \in \mathbb{C}^{m\times m}$, $\Sigma \in \mathbb{C}^{m\times n}$, $V \in \mathbb{C}^{n\times n}$.
- Economy SVD:
  - **Case 1: Tall-and-Skinny Matrix** $(m > n)$

$$A_{m\times n} = \underbrace{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_n \\ | & & | \end{bmatrix}}_{U\in\mathbb{C}^{m\times n}} \underbrace{\begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{bmatrix}}_{\Sigma\in\mathbb{C}^{n\times n}} \underbrace{\begin{bmatrix} - & v_1^\top & - \\ & \vdots & \\ - & v_n^\top & - \end{bmatrix}}_{V^\top\in\mathbb{C}^{n\times n}}$$

  - **Case 2: Short-and-Fat Matrix** $(m < n)$

$$A_{m\times n} = \underbrace{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{bmatrix}}_{U\in\mathbb{C}^{m\times m}} \underbrace{\begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_m \end{bmatrix}}_{\Sigma\in\mathbb{C}^{m\times m}} \underbrace{\begin{bmatrix} - & v_1^\top & - \\ & \cdots & \\ - & v_m^\top & - \end{bmatrix}}_{V^\top\in\mathbb{C}^{m\times n}}$$

How to Compute SVD

- Let $A = U\Sigma V^T$.

How to Compute SVD

- Let $A = U\Sigma V^T$.
- Then:
  - Row-space correlation matrix:

$$AA^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma^2 U^T \Rightarrow U = \text{eigenvectors of } AA^T$$

  - Column-space correlation matrix:

$$A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T \Rightarrow V = \text{eigenvectors of } A^T A$$

How to Compute SVD

- Let $A = U\Sigma V^T$.
- Then:
  - Row-space correlation matrix:

$$AA^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma^2 U^T \Rightarrow U = \text{eigenvectors of } AA^T$$

  - Column-space correlation matrix:

$$A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T \Rightarrow V = \text{eigenvectors of } A^T A$$

- Eigenvalues: Σ^2

Randomized SVD (rSVD)

Challenge:

- Standard SVD algorithms can be computationally expensive for very large matrices.

Randomized SVD (rSVD)

Challenge:

- Standard SVD algorithms can be computationally expensive for very large matrices.
- For "Big Data" scenarios where m and n are in the tens of thousands, millions, or more, computing the full SVD becomes impractical in terms of time and memory.

Randomized SVD (rSVD)

Challenge:

- Standard SVD algorithms can be computationally expensive for very large matrices.
- For "Big Data" scenarios where m and n are in the tens of thousands, millions, or more, computing the full SVD becomes impractical in terms of time and memory.
- However, many large real-world matrices are approximately low-rank, or we are only interested in a low-rank approximation (e.g., top k singular values/vectors).

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

1. Finding an Approximate Basis for the Span of the Columns of A.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

1. Finding an Approximate Basis for the Span of the Columns of A.
   - Goal: Construct a smaller matrix $Q \in \mathbb{R}^{m \times l}$ (where l is slightly larger than the target rank k) whose columns are orthonormal and span a subspace that captures most of column space of A.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

1. Finding an Approximate Basis for the Span of the Columns of A.
   - Goal: Construct a smaller matrix $Q \in \mathbb{R}^{m \times l}$ (where l is slightly larger than the target rank k) whose columns are orthonormal and span a subspace that captures most of column space of A.
   - How:
     1. Generate a random matrix $\Omega \in \mathbb{R}^{n \times l}$.
     2. Form the "sketch" or "sample" matrix $Y = A\Omega \in \mathbb{R}^{m \times l}$. Multiplying A by random vectors effectively samples its range.
     3. Compute an orthonormal basis for the columns of Y, typically via QR decomposition: $Y = QR$. The matrix $Q \in \mathbb{R}^{m \times l}$ is our approximate basis.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

1. Finding an Approximate Basis for the Span of the Columns of A.
   - Goal: Construct a smaller matrix $Q \in \mathbb{R}^{m\times l}$ (where l is slightly larger than the target rank k) whose columns are orthonormal and span a subspace that captures most of column space of A.
   - How:
     1. Generate a random matrix $\Omega \in \mathbb{R}^{n\times l}$.
     2. Form the "sketch" or "sample" matrix $Y = A\Omega \in \mathbb{R}^{m\times l}$. Multiplying A by random vectors effectively samples its range.
     3. Compute an orthonormal basis for the columns of Y, typically via QR decomposition: $Y = QR$. The matrix $Q \in \mathbb{R}^{m\times l}$ is our approximate basis.

   The insight here is that random projections can preserve the essential geometric structure of the original matrix A, particularly its dominant components, allowing us to work with a much smaller sketch Y.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

2. SVD on Smaller Matrix

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

2. SVD on Smaller Matrix
   - Once we have $Q \in \mathbb{R}^{m \times l}$ with orthonormal columns, where $A \approx QQ^T A$:

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

2. SVD on Smaller Matrix
   - Once we have $Q \in \mathbb{R}^{m \times l}$ with orthonormal columns, where $A \approx QQ^T A$:
     1. Form a much smaller matrix $B = Q^T A$. Dimensions of B are $l \times n$.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

2. SVD on Smaller Matrix
   - Once we have $Q \in \mathbb{R}^{m \times l}$ with orthonormal columns, where $A \approx QQ^T A$:
     1. Form a much smaller matrix $B = Q^T A$. Dimensions of B are $l \times n$.

The key is that computing SVD of B is much faster than SVD of A because $l \ll m, n$.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

2. SVD on Smaller Matrix
   - Once we have $Q \in \mathbb{R}^{m \times l}$ with orthonormal columns, where $A \approx QQ^T A$:
     1. Form a much smaller matrix $B = Q^T A$. Dimensions of B are $l \times n$.
     2. Compute the SVD of this smaller matrix $B = \tilde{U} \Sigma V^T$.

The key is that computing SVD of B is much faster than SVD of A because $l \ll m, n$.

Basic Idea of rSVD

Randomized SVD (rSVD) algorithms aim to compute a near-optimal low-rank approximation much faster than traditional SVD, especially for large matrices. The core idea involves two stages:

2. SVD on Smaller Matrix
   - Once we have $Q \in \mathbb{R}^{m \times l}$ with orthonormal columns, where $A \approx QQ^T A$:
     1. Form a much smaller matrix $B = Q^T A$. Dimensions of B are $l \times n$.
     2. Compute the SVD of this smaller matrix $B = \tilde{U} \Sigma V^T$.
     3. The final SVD approximation for A is then $A \approx (Q\tilde{U})\Sigma V^T$. Let $U_{rSVD} = Q\tilde{U}$ and then $A \approx U_{rSVD} \Sigma V^T$.

   The key is that computing SVD of B is much faster than SVD of A because $l \ll m, n$.

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m \times n}$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m \times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n \times (k+p)} \quad (l = k + p)$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m \times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n \times (k+p)} \quad (l = k + p)$
2. **Sketch the range of A:** $Y = A\Omega \in \mathbb{R}^{m \times (k+p)}$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m \times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n \times (k+p)} \quad (l = k + p)$
2. **Sketch the range of A:** $Y = A\Omega \in \mathbb{R}^{m \times (k+p)}$
3. **Orthonormalize by QR Decomposition:**
   $Y = QR \quad \Rightarrow \quad Q \in \mathbb{R}^{m \times (k+p)}$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m\times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n\times(k+p)} \quad (l = k + p)$
2. **Sketch the range of A:** $Y = A\Omega \in \mathbb{R}^{m\times(k+p)}$
3. **Orthonormalize by QR Decomposition:**
   $Y = QR \quad \Rightarrow \quad Q \in \mathbb{R}^{m\times(k+p)}$
4. **Project A into lower dimension:** $B = Q^T A \in \mathbb{R}^{(k+p)\times n}$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m\times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n\times(k+p)} \quad (l = k+p)$
2. **Sketch the range of A:** $Y = A\Omega \in \mathbb{R}^{m\times(k+p)}$
3. **Orthonormalize by QR Decomposition:**
   $Y = QR \quad \Rightarrow \quad Q \in \mathbb{R}^{m\times(k+p)}$
4. **Project A into lower dimension:** $B = Q^T A \in \mathbb{R}^{(k+p)\times n}$
5. **Compute SVD on small matrix:** $B = \tilde{U}\Sigma V^T$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m\times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n\times(k+p)}$ $\quad (l = k + p)$
2. **Sketch the range of A:** $Y = A\Omega \in \mathbb{R}^{m\times(k+p)}$
3. **Orthonormalize by QR Decomposition:**
   $Y = QR \quad \Rightarrow \quad Q \in \mathbb{R}^{m\times(k+p)}$
4. **Project A into lower dimension:** $B = Q^T A \in \mathbb{R}^{(k+p)\times n}$
5. **Compute SVD on small matrix:** $B = \tilde{U}\Sigma V^T$
6. **Form approximate SVD of A:**
   $A \approx (Q\tilde{U})\Sigma V^T \quad \Rightarrow \quad A_{rSVD} = A_k = U_k \Sigma_k V_k^T$

rSVD: Step-by-Step

Approximate the top k singular values/vectors of a large matrix $A \in \mathbb{R}^{m\times n}$

1. **Generate random test matrix:** $\Omega \in \mathbb{R}^{n\times(k+p)} \quad (l = k + p)$
2. **Sketch the range of A:** $Y = A\Omega \in \mathbb{R}^{m\times(k+p)}$
3. **Orthonormalize by QR Decomposition:**
   $Y = QR \quad \Rightarrow \quad Q \in \mathbb{R}^{m\times(k+p)}$
4. **Project A into lower dimension:** $B = Q^T A \in \mathbb{R}^{(k+p)\times n}$
5. **Compute SVD on small matrix:** $B = \tilde{U}\Sigma V^T$
6. **Form approximate SVD of A:**
   $A \approx (Q\tilde{U})\Sigma V^T \quad \Rightarrow \quad A_{rSVD} = A_k = U_k \Sigma_k V_k^T$

MATLAB Demonstration.

Matrix Approximation

Matrix Approximation

SVD as a Sum of Rank-One Matrices:

Matrix Approximation

SVD as a Sum of Rank-One Matrices:

The SVD $A = U\Sigma V^T$ can be written as an outer product expansion:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

Matrix Approximation

SVD as a Sum of Rank-One Matrices:

The SVD $A = U\Sigma V^T$ can be written as an outer product expansion:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where:

- $r = \text{rank}(A)$.
- σ_i is the i-th singular value.
- u_i is the i-th left singular vector (column of U).
- v_i is the i-th right singular vector (column of V).
- Each item $\sigma_i u_i v_i^T$ is an $m \times n$ matrix of rank 1.

Matrix Approximation

SVD as a Sum of Rank-One Matrices:

The SVD $A = U\Sigma V^T$ can be written as an outer product expansion:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where:

- $r = \text{rank}(A)$.
- σ_i is the i-th singular value.
- u_i is the i-th left singular vector (column of U).
- v_i is the i-th right singular vector (column of V).
- Each item $\sigma_i u_i v_i^T$ is an $m \times n$ matrix of rank 1.

This means any matrix A can be expressed as a weighted sum of r rank-one matrices. The weights are the singular values σ_i. Since $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$, the first few terms in this sum capture the most "significant" parts of the matrix A.

Matrix Approximation

Low-Rank Approximation:

Matrix Approximation

Low-Rank Approximation:
If we truncate the sum of rank-one matrices after k terms ($k < r$):

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

This matrix A_k is a rank-k approximation of A.

Matrix Approximation

Low-Rank Approximation:
If we truncate the sum of rank-one matrices after k terms ($k < r$):

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

This matrix A_k is a rank-k approximation of A.

- A_k is constructed using the k largest singular values and their corresponding singular vectors.
- This process effectively filters out information associated with smaller singular values, which often corresponds to noise or less significant details.

Optimal Truncation – Eckart-Young-Mirsky Theorem

The Eckart-Young-Mirsky theorem formalizes the optimality of the SVD-based low-rank approximation.

Optimal Truncation – Eckart-Young-Mirsky Theorem

The Eckart-Young-Mirsky theorem formalizes the optimality of the SVD-based low-rank approximation.

- Frobenius norm

$$\|A\|_F = \left(\sum_{i=1}^{n}\sum_{j=1}^{p}|a_{ij}|^2\right)^{\frac{1}{2}} = (\mathrm{tr}A^TA)^{\frac{1}{2}}, \|A\|_F = \left(\sum_{i=1}^{r}\sigma_i^2\right)^{\frac{1}{2}}$$

Optimal Truncation – Eckart-Young-Mirsky Theorem

The Eckart-Young-Mirsky theorem formalizes the optimality of the SVD-based low-rank approximation.

- Frobenius norm

$$\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{p} |a_{ij}|^2 \right)^{\frac{1}{2}} = (\mathrm{tr} A^T A)^{\frac{1}{2}}, \|A\|_F = \left(\sum_{i=1}^{r} \sigma_i^2 \right)^{\frac{1}{2}}$$

- p-norms

$$\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \sup_{x : \|x\|_p = 1} \|Ax\|_p, \|A\|_2 = \sigma_1$$

Optimal Truncation – Eckart-Young-Mirsky Theorem

Theorem (Eckart-Young-Mirsky)

For either $\|\cdot\|_2$ or $\|\cdot\|_F$,

$$\|A - A_k\| \leq \|A - B\| \text{ for all rank-k matrices } B.$$

Moreover,

$$\|A - A_k\| = \begin{cases} \sigma_{k+1}, \|\cdot\|_2 \text{ norm} \\ \left(\sum_{i=k+1}^{r} \sigma_i^2\right)^{\frac{1}{2}}, \|\cdot\|_F \text{ norm.} \end{cases}$$

Image Compression with SVD

- Every image can be seen as a large matrix of pixel intensity values.

Image Compression with SVD

- Every image can be seen as a large matrix of pixel intensity values.
- Goal: To reduce the amount of data needed to store the image without a significant loss in visual quality.

Image Compression with SVD

- Every image can be seen as a large matrix of pixel intensity values.
- Goal: To reduce the amount of data needed to store the image without a significant loss in visual quality.
- The most important features of an image are captured by the largest singular values in the Σ matrix. By keeping only the top k singular values (and their corresponding vectors in U and V), we can create a lower-rank approximation of the original image that requires much less storage.

Image Compression with SVD

- Every image can be seen as a large matrix of pixel intensity values.
- Goal: To reduce the amount of data needed to store the image without a significant loss in visual quality.
- The most important features of an image are captured by the largest singular values in the Σ matrix. By keeping only the top k singular values (and their corresponding vectors in U and V), we can create a lower-rank approximation of the original image that requires much less storage.
- Low k: high compression and low image quality.
- High k: Low compression and high image quality.

Image Compression with SVD

- Every image can be seen as a large matrix of pixel intensity values.
- Goal: To reduce the amount of data needed to store the image without a significant loss in visual quality.
- The most important features of an image are captured by the largest singular values in the Σ matrix. By keeping only the top k singular values (and their corresponding vectors in U and V), we can create a lower-rank approximation of the original image that requires much less storage.
- Low k: high compression and low image quality.
- High k: Low compression and high image quality.
- MATLAB Demonstration (SVD and rSVD).

Matrix Completion

Matrix Completion

We often encounter datasets with missing entries. The goal of matrix completion is to intelligently fill in these missing values.

Matrix Completion

We often encounter datasets with missing entries. The goal of matrix completion is to intelligently fill in these missing values.

- **Problem:** Given a partially filled matrix, how can we accurately estimate the unknown entries?

Matrix Completion

We often encounter datasets with missing entries. The goal of matrix completion is to intelligently fill in these missing values.

- **Problem:** Given a partially filled matrix, how can we accurately estimate the unknown entries?
- **Examples:**
  - **Movie Ratings:** A user has only rated a few movies out of thousands.
  - **Image Corruption:** A photo has a scratch or a block of missing pixels.
  - **Sensor Data:** A sensor in a network fails, leaving gaps in the data.

Matrix Completion

We often encounter datasets with missing entries. The goal of matrix completion is to intelligently fill in these missing values.

- **Problem:** Given a partially filled matrix, how can we accurately estimate the unknown entries?
- **Examples:**
  - **Movie Ratings:** A user has only rated a few movies out of thousands.
  - **Image Corruption:** A photo has a scratch or a block of missing pixels.
  - **Sensor Data:** A sensor in a network fails, leaving gaps in the data.

Can we predict the values of the '?' cells?

$$\begin{bmatrix} 5 & ? & 1 & ? \\ ? & 2 & ? & 3 \\ 4 & ? & ? & 5 \\ ? & 1 & 2 & ? \end{bmatrix}$$

Matrix Completion

The core assumption is that the complete matrix has a simple, underlying structure, meaning it is low-rank.

Matrix Completion

The core assumption is that the complete matrix has a simple, underlying structure, meaning it is low-rank.

- **Why is Low Rank?**
  - The columns (and rows) are not independent; they are combinations of a few "basis" vectors.
  - Think of movie ratings: they are driven by a few latent factors like genre, actors, etc.

Matrix Completion

The core assumption is that the complete matrix has a simple, underlying structure, meaning it is low-rank.

- **Why is Low Rank?**
  - The columns (and rows) are not independent; they are combinations of a few "basis" vectors.
  - Think of movie ratings: they are driven by a few latent factors like genre, actors, etc.
- **How does SVD help?**
  - SVD finds the best low-rank approximation of any matrix.
  - It decomposes a matrix A into:

$$A = USV^T$$

  - By keeping only the top k singular values in S and setting others to zero (S_k), we get the best rank-k approximation: $A_k = US_kV^T$.

Matrix Completion

We can't apply SVD directly to a matrix with holes. So, we:

Matrix Completion

We can't apply SVD directly to a matrix with holes. So, we:

1. **Initialize:** Make an initial guess for the missing entries. A simple choice is to fill them with the average of the known values, or just zero.

Matrix Completion

We can't apply SVD directly to a matrix with holes. So, we:

1. **Initialize:** Make an initial guess for the missing entries. A simple choice is to fill them with the average of the known values, or just zero.
2. **Approximate (The SVD Step):** Take this "filled" matrix and compute its SVD. Create a **low-rank approximation** by keeping only the top k singular values.

Matrix Completion

We can't apply SVD directly to a matrix with holes. So, we:

1. **Initialize:** Make an initial guess for the missing entries. A simple choice is to fill them with the average of the known values, or just zero.
2. **Approximate (The SVD Step):** Take this "filled" matrix and compute its SVD. Create a **low-rank approximation** by keeping only the top k singular values.
3. **Update:** Use the values from this new low-rank matrix to update your guess for the missing entries. Crucially, leave the original, known values untouched.

Matrix Completion

We can't apply SVD directly to a matrix with holes. So, we:

1. **Initialize:** Make an initial guess for the missing entries. A simple choice is to fill them with the average of the known values, or just zero.
2. **Approximate (The SVD Step):** Take this "filled" matrix and compute its SVD. Create a **low-rank approximation** by keeping only the top k singular values.
3. **Update:** Use the values from this new low-rank matrix to update your guess for the missing entries. Crucially, leave the original, known values untouched.
4. **Repeat:** Go back to Step 2 and repeat. The guesses for the missing values will converge.

Matrix Completion

Recommender Systems.

Matrix Completion

Recommender Systems.

- **Matrix:** Rows are users, columns are movies, and entries are ratings (1-5 stars).
- **Problem:** This matrix is extremely sparse—most users have rated only a tiny fraction of the available movies.
- **Goal:** Predict how a user would rate movies they haven't seen to provide personalized recommendations.

Matrix Completion

Recommender Systems.

- **Matrix:** Rows are users, columns are movies, and entries are ratings (1-5 stars).
- **Problem:** This matrix is extremely sparse—most users have rated only a tiny fraction of the available movies.
- **Goal:** Predict how a user would rate movies they haven't seen to provide personalized recommendations.

	1	2	3	4	5	6
a	5		1	1		2
b		2		4		4
c	4	5		1	1	2
d			3	5	2	
e	2		1		4	4

Matrix Completion

Recovering corrupted or missing parts of an image.

Matrix Completion

Recovering corrupted or missing parts of an image.

- **Matrix:** An image is just a matrix of pixel values (or three matrices for R, G, B channels).
- **Problem:** A block of pixels is missing due to a scratch, text overlay, or digital removal.
- **Goal:** Recover corrupted or missing parts of an image.

Matrix Completion

Recovering corrupted or missing parts of an image.

- **Matrix:** An image is just a matrix of pixel values (or three matrices for R, G, B channels).
- **Problem:** A block of pixels is missing due to a scratch, text overlay, or digital removal.
- **Goal:** Recover corrupted or missing parts of an image.
- MATLAB Demonstration.

Linear Regression with SVD

Linear Regression with SVD

Goal: Model a dependent variable y as a linear combination of independent variables (features) in a data matrix X.

$$y = X\beta + \epsilon$$

Linear Regression with SVD

Goal: Model a dependent variable y as a linear combination of independent variables (features) in a data matrix X.

$$y = X\beta + \epsilon$$

- y: vector of observed outcomes ($m \times 1$).
- X: design matrix of features ($m \times n$, where each row is an observation and each column is a feature).
- β: vector of unknown regression coefficients ($n \times 1$).
- ϵ: vector of random errors.

Linear Regression with SVD

Goal: Model a dependent variable y as a linear combination of independent variables (features) in a data matrix X.

$$y = X\beta + \epsilon$$

- y: vector of observed outcomes ($m \times 1$).
- X: design matrix of features ($m \times n$, where each row is an observation and each column is a feature).
- β: vector of unknown regression coefficients ($n \times 1$).
- ϵ: vector of random errors.

Objective: Find the coefficient vector β that minimizes the sum of squared errors:

$$\min_{\beta} \|y - X\beta\|_2^2$$

Linear Regression with SVD

Goal: Model a dependent variable y as a linear combination of independent variables (features) in a data matrix X.

$$y = X\beta + \epsilon$$

- y: vector of observed outcomes ($m \times 1$).
- X: design matrix of features ($m \times n$, where each row is an observation and each column is a feature).
- β: vector of unknown regression coefficients ($n \times 1$).
- ϵ: vector of random errors.

Objective: Find the coefficient vector β that minimizes the sum of squared errors:

$$\min_{\beta} \|y - X\beta\|_2^2$$

Standard Solution: $\beta = (X^T X)^{-1} X^T y$ ($X^T X$ is invertible).

Linear Regression with SVD

Challenges: Multicollinearity. This occurs when two or more feature columns in the data matrix X are highly correlated. This means they are nearly linearly dependent.

Linear Regression with SVD

Challenges: Multicollinearity. This occurs when two or more feature columns in the data matrix X are highly correlated. This means they are nearly linearly dependent.

- If columns of X are linearly dependent, the matrix $X^T X$ becomes singular (not invertible), and the normal equation solution $\beta = (X^T X)^{-1} X^T y$ cannot be computed.

Linear Regression with SVD

Challenges: Multicollinearity. This occurs when two or more feature columns in the data matrix X are highly correlated. This means they are nearly linearly dependent.

- If columns of X are linearly dependent, the matrix $X^T X$ becomes singular (not invertible), and the normal equation solution $\beta = (X^T X)^{-1} X^T y$ cannot be computed.
- If columns are nearly linearly dependent (high multicollinearity), $X^T X$ is ill-conditioned, leading to the numerical instability.

Linear Regression with SVD

SVD provides a robust and stable way to solve linear regression, even in the presence of multicollinearity.

Linear Regression with SVD

SVD provides a robust and stable way to solve linear regression, even in the presence of multicollinearity.

- The Moore-Penrose Pseudoinverse (X^+): any matrix; best-fit solution for $X\beta = y$.

Linear Regression with SVD

SVD provides a robust and stable way to solve linear regression, even in the presence of multicollinearity.

- The Moore-Penrose Pseudoinverse (X^+): any matrix; best-fit solution for $X\beta = y$.
- Computing the Pseudoinverse with SVD:

$$X = U\Sigma V^T \Rightarrow X^+ = V\Sigma^+ U^T$$

Linear Regression with SVD

SVD provides a robust and stable way to solve linear regression, even in the presence of multicollinearity.

- The Moore-Penrose Pseudoinverse (X^+): any matrix; best-fit solution for $X\beta = y$.
- Computing the Pseudoinverse with SVD:

$$X = U\Sigma V^T \Rightarrow X^+ = V\Sigma^+ U^T$$

- The regression coefficients are found by:

$$\beta = X^+ y = V\Sigma^+ U^T y$$

This solution minimizes the least-squares error $\|y - X\beta\|_2^2$.

Linear Regression with SVD

MATLAB Demonstration.

$$\begin{bmatrix} | & | & \cdots & | \\ 1 & x_1 & \cdots & x_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Summary of What You've Learned Today

- Properties of SVD, U, Σ, V.
- Computation of SVD: Conventional method and randomized SVD.
- Geometric Interpretation of SVD
- Applications
  - Matrix Approximation: Image Compression.
  - Matrix Completion: Image Inpainting.
  - Linear Regression.

Summary of What You've Learned Today

- Properties of SVD, U, Σ, V.
- Computation of SVD: Conventional method and randomized SVD.
- Geometric Interpretation of SVD
- Applications
    - Matrix Approximation: Image Compression.
    - Matrix Completion: Image Inpainting.
    - Linear Regression.

See you on Wednesday (7/2) for Lecture 3 on
Singular Value Decomposition (PCA, POD, LSA).