Dimensionality Reduction & POD

AERO 689: Machine Learning for Aerospace Engineers

Raktim Bhattacharya

Table of contents

1 Overview

1.1 Topics Covered

1. PCA Recap (from Week 5)
2. SVD & Proper Orthogonal Decomposition
3. Application: Cylinder Wake Flow
4. Application: Structural Vibrations
5. SVD for Aerospace Engineers
6. Limitations of Linear Methods & Nonlinear Alternatives

2 PCA Recap

2.1 Quick Review: PCA Intuition

Key idea from Week 5: Variance = Information

- Find directions of **maximum variance** in the data
- Project data onto those directions (principal components)
- Discard low-variance directions → reduce dimensions

Recall: PCA works by eigendecomposition of the covariance matrix

$$C = \frac{1}{n-1} X_c^T X_c, \quad Cv_i = \lambda_i v_i$$

where X_c is the centered data matrix

2.2 PCA Algorithm Summary

Four steps:

1. **Center**: Subtract the mean from each feature

$$X_c = X - \bar{X}$$

2. **Covariance**: Compute $C = \frac{1}{n-1} X_c^T X_c$
3. **Eigendecompose**: Solve $Cv_i = \lambda_i v_i$
4. **Project**: $Z = X_c V_k$ where $V_k = [v_1 | \cdots | v_k]$

Result: $n \times d$ data $\rightarrow$ $n \times k$ data ($k \ll d$)

2.3 PCA Limitations

What PCA does well:

- Fast and well-understood
- Optimal linear dimensionality reduction
- Preserves global variance structure

What PCA cannot do:

- X Capture nonlinear relationships
- X Handle manifold structure (e.g., Swiss roll)
- X Scale to very high-dimensional data efficiently

This week: Tools that address these limitations

2.4 From PCA to SVD

Key insight: PCA is SVD in disguise

The eigendecomposition of $C = \frac{1}{n-1} X_c^T X_c$ is equivalent to the **SVD** of X_c:

$$X_c = U \Sigma V^T$$

- Principal components: $Z = X_c V = U\Sigma$

- Eigenvalues: $\lambda_i = \frac{\sigma_i^2}{n-1}$

Why learn SVD separately?

- Works on **any** matrix (not just square covariance)
- More numerically stable
- Enables **truncated** computation (don't need all components)
- Foundation for **POD** in engineering

3 SVD & Proper Orthogonal Decomposition

3.1 SVD: Definition

Any $m \times n$ matrix A can be decomposed as:

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

- $U \in \mathbb{R}^{m\times m}$: Left singular vectors (orthonormal columns)
- $\Sigma \in \mathbb{R}^{m\times n}$: Diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$
- $V \in \mathbb{R}^{n\times n}$: Right singular vectors (orthonormal columns)

Properties:

- Always exists for any real matrix
- Singular values σ_i are unique
- $\sigma_i = \sqrt{\lambda_i(A^TA)}$

3.2 Truncated SVD: Low-Rank Approximation

Eckart–Young Theorem: The best rank-k approximation to A is:

$$A_k = U_k\Sigma_k V_k^T = \sum_{i=1}^{k} \sigma_i\, u_i\, v_i^T$$

- U_k: first k columns of U
- Σ_k: top-left $k \times k$ block of Σ
- V_k: first k columns of V

Optimality: $A_k = \arg\min_{\text{rank}(B)=k} \|A - B\|_F$ where the **Frobenius norm** is

$$\|M\|_F = \sqrt{\sum_{i,j} M_{ij}^2} = \sqrt{\text{tr}(M^TM)}$$

It measures the **least-squares error across all matrix entries** — minimizing $\|A - B\|_F^2$ is equivalent to minimizing the sum of squared element-wise errors $\sum_{i,j}(A_{ij}-B_{ij})^2$. We use it because it equals $\sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$, which directly connects the approximation error to discarded singular values.

3.3 Truncated SVD: Error and Energy

Approximation error:

$$\|A - A_k\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_r^2}$$

Energy captured by rank-k approximation:

$$E(k) = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2}$$

Rule of thumb: Keep enough modes to capture 95–99% of energy

Practical impact: If singular values decay fast, very few modes suffice

---

3.4 Image Compression via SVD

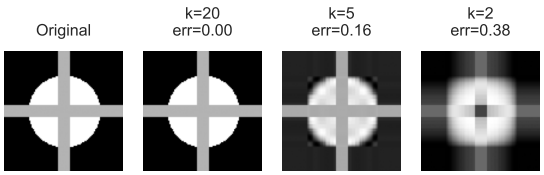


Image = matrix → apply SVD

- Original: rank 100 (full)
- $k = 20$: Near-perfect (error ≈ 0)
- $k = 5$: Main shapes preserved
- $k = 2$: Only gross structure

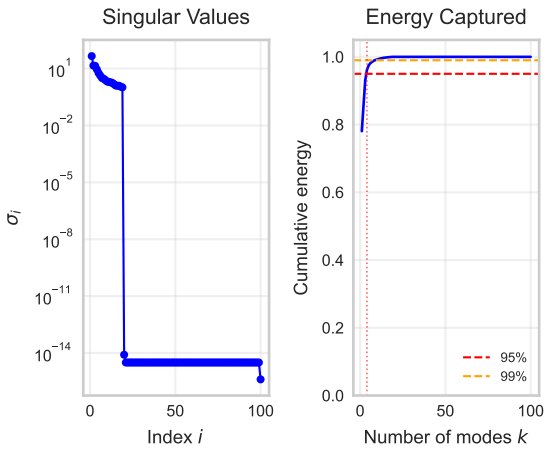
Compression ratio: $k = 5$ stores $5 \times (100 + 1 + 100) = 1005$ values vs 10,000 original

---

3.5 Image Compression: Code

```python
import numpy as np

# Load or create image as matrix A (m x n)
U, s, Vt = np.linalg.svd(A, full_matrices=False)

# Reconstruct with k modes
k = 10
A_k = U[:, :k] @ np.diag(s[:k]) @ Vt[:k, :]

# Relative error
error = np.linalg.norm(A - A_k, 'fro') / np.linalg.norm(A, 'fro')

# Energy captured
energy = np.sum(s[:k]**2) / np.sum(s**2)
print(f"k={k}: error={error:.4f}, energy={energy:.4f}")
```

3.6 Singular Value Spectrum



Left: Singular values decay rapidly — most energy in first few modes. **Right**: 95% energy captured with only $k \approx$ np.int64(4) modes out of 100.

3.7 SVD $\leftrightarrow$ PCA Connection

Aspect	PCA	SVD
Input	Covariance matrix $C = \frac{1}{n-1} X_c^T X_c$	Data matrix X_c directly
Decomposition	$C = V \Lambda V^T$ (eigendecomposition)	$X_c = U \Sigma V^T$
Principal components	$Z = X_c V$	$Z = U\Sigma$
Variance	λ_i	$\sigma_i^2/(n-1)$
Works on	Square covariance matrix	Any $m \times n$ matrix

Bottom line: Use SVD when you want PCA without forming $X^T X$ explicitly

3.8 When SVD Beats Eigendecomposition

Numerical stability:

- Eigendecomposing $X^T X$ **squares** the condition number
- SVD works on X directly — more stable for ill-conditioned data

Rectangular matrices:

- PCA requires square covariance matrix
- SVD works on any $m \times n$ matrix (e.g., images, snapshot matrices)

Truncated computation:

- Only compute the top k singular values/vectors
- Much faster when $k \ll \min(m, n)$

Sparse/iterative methods:

- `scipy.sparse.linalg.svds` for large sparse matrices

3.9 Computing the SVD: Algorithms and Cost

Full SVD of $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$:

- Classical approach: bidiagonalization + QR iteration

- Cost: $O(mn^2)$ flops — e.g., 1000×500 matrix $\approx 2.5 \times 10^8$ flops
- `numpy.linalg.svd`: LAPACK `dgesdd` (divide-and-conquer, very fast in practice)

Truncated SVD (only top k triplets):

- Krylov/Lanczos iteration: cost $O(mnk)$ — huge savings when $k \ll n$
- `scipy.sparse.linalg.svds`: uses ARPACK (Arnoldi), works on sparse matrices
- `sklearn.decomposition.TruncatedSVD`: calls the same under the hood

3.10 Scalability: Randomized SVD

Problem: Even $O(mnk)$ is too slow when $m, n \sim 10^6$ (CFD grids, image datasets)

Randomized SVD (Halko, Martinsson & Tropp, 2011):

1. Draw a random matrix $\Omega \in \mathbb{R}^{n \times (k+p)}$ ($p \sim 5$–10 oversampling)
2. Form $Y = A\Omega$ — a single matrix multiply, $O(mn(k+p))$
3. Compute QR: $Y = QR$, then $B = Q^T A$ (small matrix)
4. SVD of B (cheap: $(k+p) \times n$)

Result: Approximate rank-k SVD in **one or two passes** over the data

Variant	Cost	Memory	Best for
Full SVD (`np.linalg.svd`)	$O(mn^2)$	$O(mn)$	Small–medium dense matrices
Truncated (`scipy.sparse.linalg.svds`)	$O(mnk)$	$O(m+n)k$	Sparse matrices, moderate k
Randomized (`sklearn.utils.extmath.randomized_svd`)	$O(mn(k+p))$	$O(m+n)(k+p)$	Very large dense/sparse, small k
Method of Snapshots (Sirovich)	$O(m^2 n)$	$O(m^2)$	POD when $m \ll n$ (few snapshots)

Rule of thumb: If you only need $k \leq 100$ modes from a million-row matrix, randomized SVD is orders of magnitude faster than full SVD.

3.11 SVD in the Real World

SVD is not just a textbook tool — it is embedded in products used by billions of people:

Application	How SVD is used	Scale
Google Search	PageRank relies on low-rank approximations of the web link matrix to rank pages	Billions of pages
Netflix / Spotify	Collaborative filtering: user×item rating matrix $\approx U_k \Sigma_k V_k^T$ predicts what you'll like	10^8 users × 10^6 items
iPhone Face ID	Eigenfaces (PCA/SVD of face images) underpins facial recognition pipelines	Real-time on device
GPS / Surveying	Least-squares position estimation solved via SVD (numerically stable pseudoinverse $A^\dagger = V\Sigma^{-1}U^T$)	Every GPS fix
MRI / CT imaging	Image reconstruction from incomplete Fourier samples uses truncated SVD regularization	Millions of voxels
Natural Language (LLMs)	Latent Semantic Analysis: SVD of term×document matrix discovers topic structure; weight matrices in large language models are compressed via low-rank SVD	GPT-scale models
Finance	PCA/SVD of asset return matrices identifies market factors (e.g., "the market", sector rotations)	Thousands of assets × years of daily data
Genomics	SVD of gene expression matrices (samples × genes) reveals cell-type clusters	10^4 genes × 10^6 cells

The common thread: Whenever you have a large matrix and need to find its dominant structure — whether it is user preferences, pixel intensities, gene expressions, or web links — SVD is the workhorse.

3.12 What is POD?

Proper Orthogonal Decomposition = SVD applied to physical data

- Originated in turbulence research (Lumley, 1967)
- **Method of Snapshots** (Sirovich, 1987) made it practical

Key idea: Collect time-varying field measurements as columns of a **snapshot matrix**, then apply SVD

$$X = [x(t_1) \,|\, x(t_2) \,|\, \cdots \,|\, x(t_m)] \in \mathbb{R}^{n \times m}$$

- n = number of spatial points (e.g., grid cells in CFD)
- m = number of time snapshots

3.13 POD Framework

Step 1: Assemble snapshot matrix and subtract temporal mean

$$\bar{x} = \frac{1}{m} \sum_{j=1}^{m} x(t_j), \qquad \tilde{X} = X - \bar{x}\mathbf{1}^T$$

Step 2: Compute SVD of the centered snapshot matrix

$$\tilde{X} = U \Sigma V^T$$

Step 3: Interpret the decomposition

- Columns of U: **Spatial modes** (POD modes / coherent structures)
- Diagonal of Σ: **Mode amplitudes** (importance ranking)
- Rows of V^T: **Temporal coefficients** (how modes evolve in time)

3.14 POD Energy and Mode Selection

Energy in mode i:

$$E_i = \frac{\sigma_i^2}{\sum_{j=1}^{r} \sigma_j^2}$$

Truncated reconstruction with k modes:

$$x(t) \approx \bar{x} + \sum_{i=1}^{k} a_i(t)\, \phi_i$$

where $\phi_i = u_i$ (spatial modes) and $a_i(t) = \sigma_i v_i(t)$ (temporal coefficients)

Mode selection rule: Choose k such that $\sum_{i=1}^{k} E_i \geq 0.95$

3.15 Why POD Matters for Engineers

PCA (statistics): finds variance directions in feature space

POD (engineering): finds **coherent structures** in physical systems

Domain	Snapshots	POD modes reveal…
Fluid dynamics	Velocity fields	Vortex structures
Structures	Vibration data	Natural mode shapes
Aero loads	Pressure fields	Dominant load patterns
Thermal	Temperature maps	Heat transfer modes

Benefit: Data-driven modes — no physical model required!

4 Application: Cylinder Wake Flow

4.1 Why Apply SVD to Flow Data?

The engineering problem: CFD simulations are expensive

- Navier–Stokes on a fine grid: $\sim 10^5$–10^8 unknowns per time step
- Design optimization, control, or uncertainty quantification require **many** solves
- Running full CFD each time is often impractical

What SVD/POD gives us:

1. **Compression**: Represent the flow with $k \ll n$ modes instead of n grid values
2. **Physical insight**: Identify the dominant flow structures (vortex shedding, shear layers) directly from data — no modeling assumptions
3. **Reduced-order models (ROMs)**: Project the governing equations onto k modes $\rightarrow$ solve a k-dimensional ODE instead of the full PDE
4. **Sensor placement**: Know *where* to measure to capture the most information

POD answers: What is the *smallest* basis that captures most of the dynamics?

4.2 Cylinder Wake: A Classic POD Benchmark

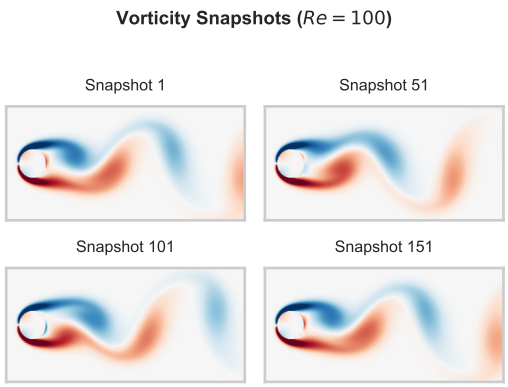
Physical setup: Uniform flow past a circular cylinder

- At $Re > 47$: periodic **vortex shedding** (von Kármán street)
- Alternating vortices create oscillating velocity field
- Ideal POD test case: low-rank, periodic, well-understood

Dataset: Brunton & Kutz, *Data-Driven Science and Engineering* (2019)

- Direct numerical simulation at $Re = 100$ (Taira & Colonius)
- Vorticity field on a 449×199 grid (89,351 spatial points)
- 151 time snapshots capturing periodic vortex shedding

4.3 Cylinder Wake: Vorticity Snapshots



DNS vorticity field (Taira & Colonius):

- Von Kármán vortex street at $Re = 100$
- Positive (red) and negative (blue) vortices
- Alternating pattern convecting downstream
- Periodic shedding creates low-rank structure

Snapshot matrix: $89{,}351 \times 151$

4.4 Cylinder Wake: Loading Data & Computing POD

```
import scipy.io

# Load DNS vorticity data (Re = 100)
data = scipy.io.loadmat('VORTALL.mat')
snapshots = data['VORTALL']  # (89351, 151)
nx, ny = 449, 199

# Compute POD via SVD
snap_mean = snapshots.mean(axis=1, keepdims=True)
X_centered = snapshots - snap_mean
U, s, Vt = np.linalg.svd(X_centered,
                         full_matrices=False)
```

4.5 POD Modes of the Wake

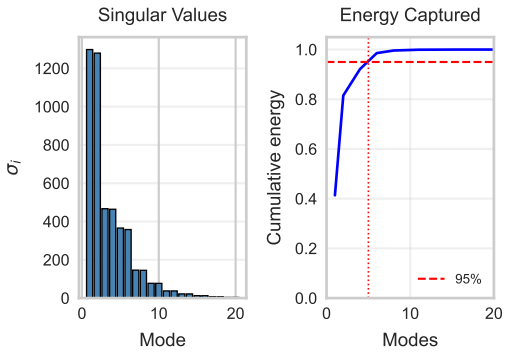
Mode 1–2: Vortex shedding pair (sin/cos phase shift) — together form the **traveling wave** ($\approx 82\%$ energy)

Mode 3–4: First harmonic pair ($\approx 10\%$ energy)

Mode 5–6: Second harmonic pair ($\approx 6\%$)

Modes appear in **pairs** — characteristic of periodic flows with traveling-wave structure

4.6 Wake: Singular Value Spectrum



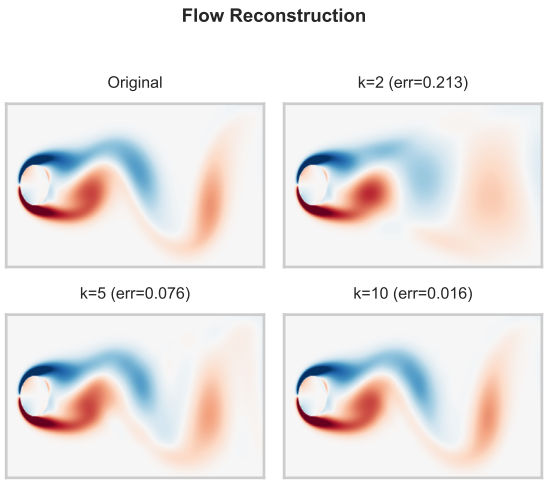
Key observations:

- Modes 1–2 dominate (vortex shedding pair)
- Modes appear in pairs (sin/cos at each harmonic)
- Rapid decay → **low-rank structure**

Energy capture:

- 2 modes: $\approx 82\%$ (fundamental)
- 4 modes: $\approx 92\%$ (+ first harmonic)
- 6 modes: $\approx 99\%$ (+ second harmonic)

4.7 Wake: Flow Reconstruction



Reconstruction quality:

- $k = 2$: Captures fundamental shedding pattern
- $k = 5$: Close approximation ($\approx 95\%$ energy)
- $k = 10$: Nearly exact ($< 2\%$ error)

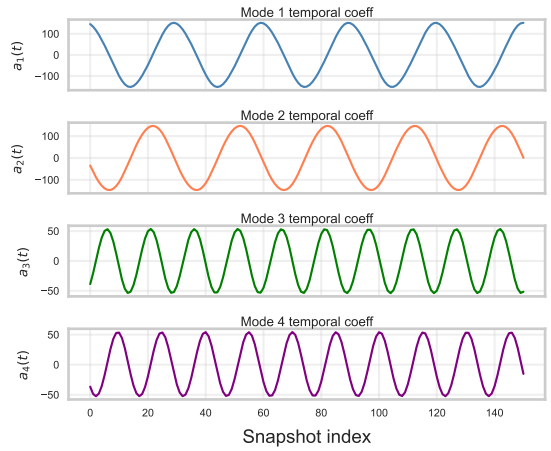
Compression: From $89{,}351 \times 151$ to storing only k modes + coefficients

Physical insight: 6 modes capture 99% of the dynamics — wake is effectively **low-rank**

4.8 Wake Reconstruction: Code

```
# POD of snapshot matrix
snap_mean = snapshots.mean(axis=1, keepdims=True)
X_centered = snapshots - snap_mean
U, s, Vt = np.linalg.svd(X_centered, full_matrices=False)

# Reconstruct snapshot j with k modes
k = 5
j = 50  # snapshot index
x_recon = snap_mean.flatten() + (
    U[:, :k] @ np.diag(s[:k]) @ Vt[:k, j])

# Relative error
err = np.linalg.norm(snapshots[:, j] - x_recon)
err /= np.linalg.norm(snapshots[:, j])
```

4.9 Wake: Temporal Coefficients



Temporal coefficients $a_i(t) = \sigma_i v_i(t)$:

- Modes 1–2: Fundamental shedding frequency (sin/cos pair)
- Modes 3–4: First harmonic (2× shedding frequency)

POD recovered the physics: Periodic coefficients at the shedding frequency and its harmonics

5 Application: Structural Vibrations

5.1 Cantilever Beam: Modal Analysis via POD

Physical setup: Instrumented cantilever beam under vibration

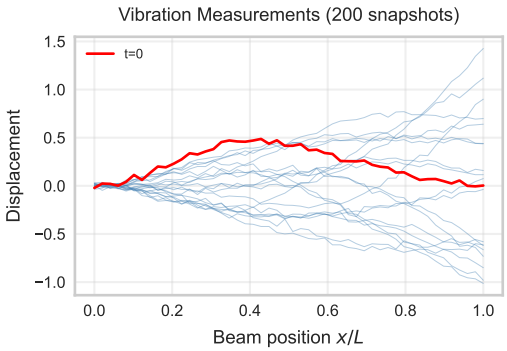
- Sensors measure displacement at n locations along the beam
- Response is a superposition of natural mode shapes
- Classical approach: FEM eigenvalue problem
- **POD approach**: Extract modes directly from measurement data

Synthetic model:

- 50 sensor locations along beam length
- 3 analytical cantilever mode shapes (clamped-free eigenfunctions)
- 200 time snapshots with decaying oscillation + noise

Same SVD, different physics: Identical workflow to the cylinder wake — only the interpretation changes

5.2 Beam: Generating Vibration Data



200 time snapshots of beam displacement

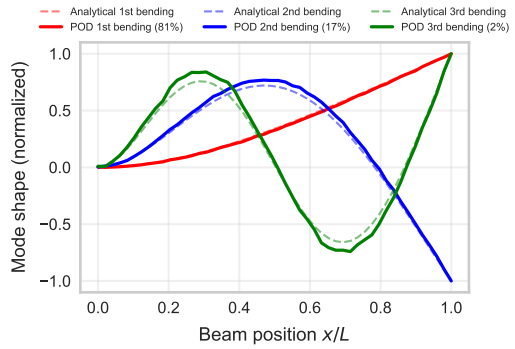
- Red: initial condition (t=0)
- Blue traces: subsequent measurements
- Response decays over time (damping)
- Multiple frequencies visible (superposition of modes)

Goal: Extract the underlying mode shapes from this data alone

5.3 Beam: Generating Vibration Data Code

```
# Exact cantilever (clamped-free) mode shapes
def cantilever_mode(x, beta_L, L):
    beta = beta_L / L
    s = (np.sinh(beta_L) - np.sin(beta_L)) / \
        (np.cosh(beta_L) + np.cos(beta_L))
    return (np.cosh(beta*x) - np.cos(beta*x)
            - s*(np.sinh(beta*x) - np.sin(beta*x)))

# beta_k*L: roots of cosh(bL)cos(bL) + 1 = 0
beta_L = [1.87510407, 4.69409113, 7.85475744]
phi1 = cantilever_mode(x, beta_L[0], L)

# Decaying oscillation at each natural frequency
a1 = A1 * np.sin(w1*t) * np.exp(-zeta1*t)
snapshots[:, j] = a1[j]*phi1 + a2[j]*phi2 + noise
```

5.4 POD Extracts Mode Shapes



Solid lines: POD modes (from data)

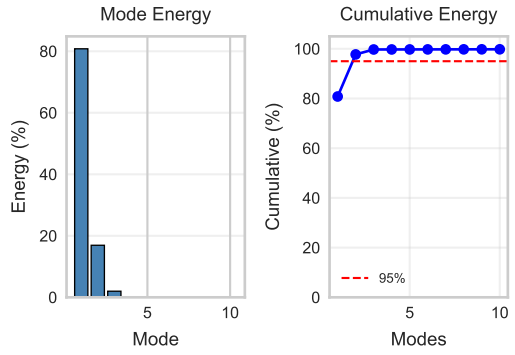
Dashed lines: Analytical mode shapes

Observations:

- POD modes match analytical modes closely
- Mode 1: dominant (highest energy)
- Modes ranked by energy, not frequency

Key point: POD extracted the physics without any beam model!

5.5 Beam: POD Energy Distribution



Energy distribution:

- Mode 1: $\approx 81\%$ (1st bending)
- Mode 2: $\approx 17\%$ (2nd bending)
- Mode 3: $\approx 2\%$ (3rd bending)
- Rest: noise ($< 0.1\%$)

3 modes capture $> 99\%$ of the vibrational energy

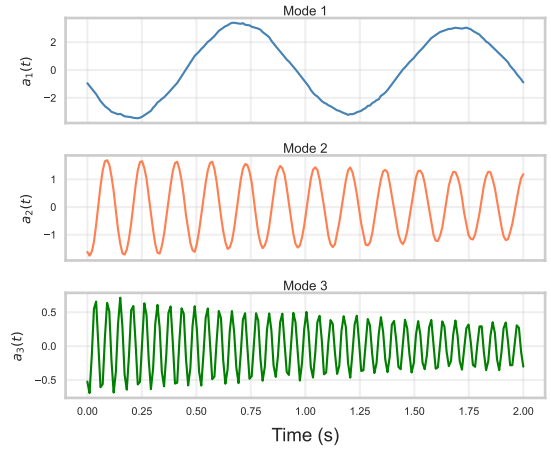
Pattern: Higher modes have less energy — smaller excitation amplitudes and faster damping decay

5.6 POD vs FEM Modal Analysis

Aspect	FEM Modal Analysis	POD from Data
Requires	Physical model (mass, stiffness matrices)	Only measurement data
Computes	Eigenmodes of $K\phi = \omega^2 M\phi$	SVD of snapshot matrix
Assumptions	Known geometry, material, BCs	None (data-driven)
Modes	Ordered by frequency	Ordered by energy
Updates with damage	Need updated model	Automatic from new data
Cost	High (modeling effort)	Low (sensors + SVD)

Structural Health Monitoring: Compare POD modes over time — mode shape changes indicate damage!

5.7 Beam: Temporal Coefficients and Damping



Compare with cylinder wake:

- Wake: periodic (steady shedding) → constant-amplitude oscillations
- Beam: **decaying** oscillations → transient response

Damping estimation: Fit exponential envelope to $a_i(t)$ to extract damping ratios ζ_i — useful for structural health monitoring

Different frequencies visible: a_1 slow, a_2 faster, a_3 fastest

5.8 Reduced-Order Modeling: The Big Picture

POD gives us a low-dimensional basis $\{\phi_1, \ldots, \phi_k\}$

Idea: Approximate the full state using this basis

$$x(t) \approx \bar{x} + \sum_{i=1}^{k} a_i(t)\,\phi_i = \bar{x} + \Phi_k\, a(t)$$

Galerkin projection: Substitute into governing equations, project onto the POD basis

$$\dot{a}(t) = \Phi_k^T f(\bar{x} + \Phi_k a(t))$$

Cost reduction: Full model with $n = 10^6$ DoFs → POD-ROM with $k = 10$ DoFs

5.9 What Do People Do with ROMs?

Design optimization: Evaluate 1000s of design variants in seconds

- Wing shape optimization: full CFD takes hours per run; POD-ROM takes seconds
- Turbine blade cooling channel design at GE, Rolls-Royce

Real-time control: ROM is fast enough for feedback loops

- Active flow control (delay separation, suppress vortex shedding)
- Flutter suppression in aeroelastic systems

Uncertainty quantification: Monte Carlo with 10^4–10^6 samples

- Propagate manufacturing tolerances through aerodynamic models
- Quantify thermal margins in hypersonic vehicle design

Digital twins: Continuously updated model running alongside physical system

- Structural health monitoring of aircraft wings
- Engine performance tracking with sensor fusion

Common thread: All require *many* model evaluations — too expensive with full-order PDE solvers

6 SVD for Aerospace Engineers

6.1 Trajectory Database for Real-Time Guidance

Problem: A Mars entry vehicle needs real-time guidance, but the optimal trajectory depends on entry conditions (velocity, flight path angle, atmosphere). Precomputing trajectories for all conditions fills 100+ GB.

SVD solution: Compute trajectories on a grid of initial conditions → stack as columns of a snapshot matrix → SVD

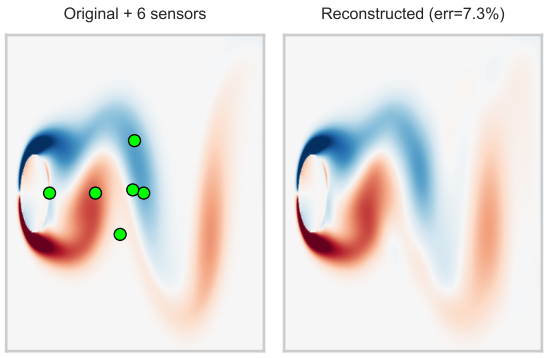
$$\text{Trajectory}(\text{conditions}) \approx \bar{x} + \sum_{i=1}^{k} a_i(\text{conditions})\, \phi_i$$

Result: Store only $k \sim 10$ basis trajectories + a small interpolation table for the coefficients a_i. Memory: KB instead of GB.

Onboard use: Given current measured entry conditions, interpolate a_i and reconstruct the reference trajectory in microseconds.

Impact: NASA's Mars Science Laboratory (Curiosity) and Mars 2020 (Perseverance) entry guidance used trajectory databases built with techniques closely related to POD compression.

6.2 Demo: Optimal Sensor Placement



Key idea: QR pivoting on $\mathbf{U}_k^T$ selects sensors that maximize information

$$\mathbf{U}_k^T \mathbf{P} = \mathbf{Q}\mathbf{R}$$

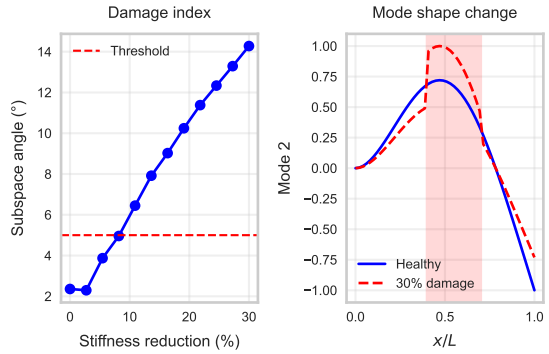
The first k pivot columns identify the **most informative** spatial locations.

From 6 sensors we reconstruct the full 89,351-point wake field — this is the power of POD-aware sensing.

Applications: wind tunnel instrumentation, flight test sensor arrays, structural monitoring networks

---

6.3 Demo: Structural Health Monitoring



Damage detection via subspace tracking:

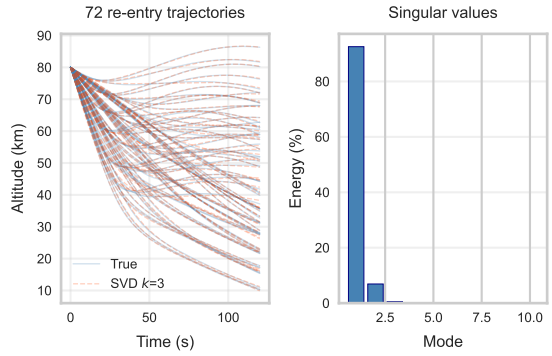
1. Compute POD basis from **healthy** data
2. Periodically recompute POD from new measurements
3. Track the **subspace angle** between healthy and current bases

$$\theta = \arccos(\sigma_{\min}(\mathbf{U}_h^T \mathbf{U}_d))$$

The subspace angle increases monotonically with stiffness reduction, crossing the detection threshold at $ $8% damage.

Right panel: Mode 2 shape shifts locally near the damage site — the SVD "sees" the structural change.

6.4 Demo: Trajectory Database Compression



Scenario: 72 lifting re-entry trajectories (altitude vs. time) from 80 km, varying flight-path angle, entry speed, ballistic coefficient $\beta = m/(C_D A)$, and **lift-to-drag ratio** L/D.

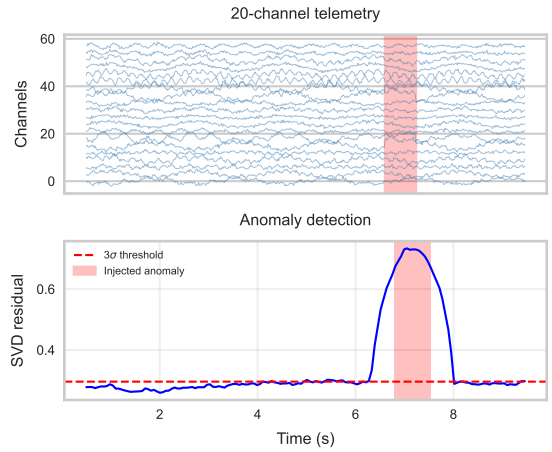
SVD compression: Mode 1 captures $ $93% (mean descent), mode 2 adds $ $7% (lift-induced curvature). Three modes reach **99.96%** energy.

Store 3 basis functions + 3 coefficients per trajectory instead of 300 time points — a **50**× compression.

Applications:

- Mars EDL guidance: real-time trajectory interpolation
- Missile defense: rapid threat assessment from partial track
- Reusable launch vehicle corridor design

6.5 Demo: Flight-Test Anomaly Detection



Method: Fixed-baseline SVD anomaly detection

1. Learn a rank-k subspace $\mathbf{P} = \mathbf{U}_k \mathbf{U}_k^T$ from **healthy** data
2. For each sliding window, project onto $\mathbf{P}$ and track the **residual** $\|\mathbf{X}_w - \mathbf{P}\mathbf{X}_w\|_F / \|\mathbf{X}_w\|_F$
3. Flag when residual exceeds 3σ threshold

At $t \approx 7$ **s** an anomalous event (simulated sensor fault) appears in 2 of 20 channels. The residual **spikes immediately** — the anomaly lies outside the healthy rank-3 subspace.

Advantage: No labeled anomaly data needed — purely **unsupervised**. Works for any multi-sensor system: flight test, engine monitoring, structural fatigue.

6.6 Optimal Sensor Placement

Problem: You can only afford 10 pressure sensors on a wing, but CFD gives you 10^5 grid points. Where do you put them?

SVD solution: Compute the POD modes U_k, then use **QR pivoting** on U_k^T to find the k rows (spatial locations) that are maximally independent

$$U_k^T \Pi = QR \quad \Longrightarrow \quad \text{pivot columns} = \text{sensor locations}$$

Result: With $k = 10$ sensors at SVD-optimal locations, you can reconstruct the full pressure field via $\hat{x} = U_k (U_k^T|_{\text{sensors}})^{-1} x_{\text{sensors}}$

Impact: DLR and NASA use this for wind tunnel instrumentation — 10 sensors can recover 95% of the aerodynamic load distribution. This is the foundation of **sparse reconstruction** in fluid mechanics.

6.7 Aeroacoustic Source Separation

Problem: A jet engine produces noise from the fan, core, turbine, and jet exhaust simultaneously. How do you isolate each source?

Setup: Phased microphone arrays (100+ microphones) record pressure time histories during engine tests or aircraft flyovers

SVD solution: Form the cross-spectral matrix $S(\omega) \in \mathbb{C}^{m \times m}$ at each frequency, then decompose it: $S = U \Sigma U^H$. Each singular vector is a **coherent noise source pattern**

Why it works:

- Independent sources contribute rank-1 components
- Dominant singular values → loudest sources
- Spatial patterns in U reveal source locations (fan vs jet vs airframe)

Impact: Boeing and Airbus use array-SVD processing for noise certification — required by FAA/EASA to measure community noise at airports.

6.8 Structural Health Monitoring

Problem: An aircraft wing develops a fatigue crack. Can we detect it from vibration data alone, without visual inspection?

Approach: Instrument the structure with accelerometers. Collect vibration snapshots periodically over months/years.

SVD as a diagnostic:

- Compute SVD of each measurement campaign: $X_t = U_t \Sigma_t V_t^T$
- Track the **singular value spectrum** $\{\sigma_i(t)\}$ over time
- Damage redistributes energy: a new crack changes mode shapes → the subspace U_t rotates

Detection metric: Subspace angle between healthy and current modes

$$\theta = \arccos(\sigma_{\min}(U_{\text{healthy}}^T U_{\text{current}}))$$

If θ exceeds a threshold → flag for inspection.

Impact: Sandia National Labs and the US Air Force use POD-based damage indices for aging aircraft fleet management.

6.9 Balanced Truncation for Flight Control

Problem: An aeroservoelastic model of a flexible aircraft has 10,000+ states. The flight control computer needs a model with ~ 50 states that runs in real time.

SVD solution — Balanced truncation:

1. Solve for controllability and observability Gramians: P, Q
2. Compute $PQ = U\Sigma^2 U^T$ (via SVD of the **Hankel matrix**)
3. The **Hankel singular values** σ_i measure how much each state contributes to input-output behavior
4. Discard states with small σ_i → reduced model with provable error bounds

Key property: Unlike POD (which captures energy), balanced truncation captures **controllability** — the states that matter for the controller.

Impact: Standard method at Boeing, Lockheed Martin, and Airbus for reducing aeroservoelastic models before control law design. Implemented in MATLAB's `balred()`.

6.10 Radar Cross-Section Database Compression

Problem: A stealth aircraft's radar cross-section (RCS) is a function of azimuth angle, elevation angle, frequency, and polarization — the full database can be 10+ GB.

SVD solution: Reshape the RCS data into a matrix (e.g., angles × frequencies) and compute truncated SVD

$$\text{RCS}(\theta, \phi, f) \approx \sum_{i=1}^{k} \sigma_i \, u_i(\theta, \phi) \, v_i(f)$$

Compression: Typical military RCS databases compress 100× to 1000× because the physics is smooth — a few scattering mechanisms dominate.

Onboard use: Fighter aircraft carry compressed RCS libraries for threat identification — SVD enables real-time lookup that would be impossible with the full database.

6.11 Combustion Instability Analysis

Problem: Rocket engine combustion chambers experience thermoacoustic instabilities — pressure oscillations couple with heat release and can destroy hardware in milliseconds.

Data: High-speed pressure transducers and chemiluminescence cameras capture 10^4 frames/sec inside the combustor.

SVD solution: Stack frames as columns of a snapshot matrix. SVD separates:

- **Mode 1–2**: Dominant acoustic mode (e.g., first transverse mode at ~ 3 kHz)
- **Mode 3–4**: Secondary acoustic modes
- **Higher modes**: Turbulent background

Why it matters: The temporal coefficients reveal whether modes are **growing** (unstable) or **decaying** (stable) — an early warning system.

Impact: NASA's Rotating Detonation Engine program and SpaceX Raptor development use POD of high-speed combustion imagery to identify instability modes and validate injector designs.

6.12 Flight Test Telemetry Analysis

Problem: A flight test campaign records 1,000+ sensor channels (strain gauges, accelerometers, pitot probes, control surface positions, engine parameters) at high rates. How do you find the important patterns?

SVD solution: Form the sensor × time matrix and compute SVD

- **First few modes**: Dominant flight dynamics (phugoid, short period, Dutch roll)
- **Mode amplitudes**: Which maneuvers excite which modes
- **Anomaly detection**: Residual $\|X - X_k\|$ suddenly spikes → something unusual happened

Key advantage: No need to label or model anything — SVD finds structure automatically across *all* channels simultaneously.

Impact: Flight test engineers at Edwards AFB and Patuxent River use SVD/PCA-based tools to screen thousands of flight hours for anomalies that manual inspection would miss.

6.13 Manufacturing Quality Control

Problem: You manufacture 500 composite wing panels. Each panel is measured at 10,000 points by a coordinate measurement machine (CMM). Are deviations random or systematic?

SVD solution: Stack all 500 shape deviation profiles into a matrix ($10{,}000 \times 500$) and decompose with SVD

- **Mode 1**: Systematic thermal warping (same pattern in every panel)
- **Mode 2**: Tooling wear trend (grows linearly with panel number)
- **Mode 3+**: Random manufacturing variation

Actionable insight: If Mode 1 captures 60% of the deviation energy, fixing the thermal management in the autoclave eliminates 60% of all quality problems at once.

Impact: Spirit AeroSystems and GKN Aerospace use POD of CMM data to separate tooling drift from random variation in composite part production.

6.14 Hyperspectral Satellite Image Denoising

Problem: Earth observation satellites capture images in 200+ spectral bands. Each band is noisy, but the underlying terrain signal is smooth across bands.

SVD solution: Reshape the hyperspectral cube as a (pixels × bands) matrix and compute SVD

- **First k modes**: True spectral signatures (vegetation, water, soil, urban)
- **Remaining modes**: Sensor noise, atmospheric artifacts

Reconstruction: $\hat{X} = U_k \Sigma_k V_k^T$ gives a denoised image cube with dramatically improved signal-to-noise ratio.

Why aerospace cares: UAVs performing precision agriculture, environmental monitoring, or intelligence reconnaissance all carry hyperspectral sensors — SVD denoising runs onboard or in ground processing.

6.15 Aeroelastic Flutter Prediction

Problem: Flutter is the most dangerous aeroelastic instability — it can destroy a wing in seconds. Flight testing approaches the flutter boundary incrementally, but how do you predict the boundary from **subcritical** (safe) data?

SVD approach: At each test airspeed, collect acceleration time histories and compute SVD

$$X_v = U_v \Sigma_v V_v^T$$

Key observable: As airspeed $v \to v_{\text{flutter}}$:

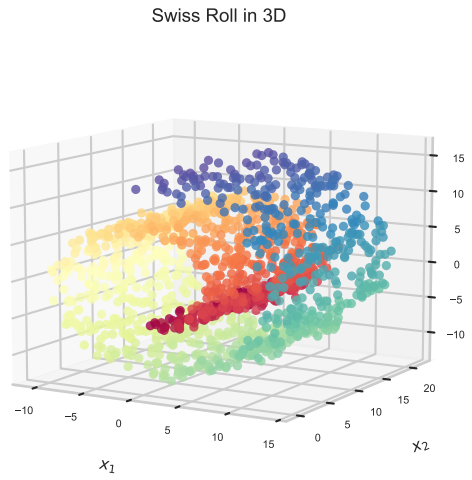
- The **dominant singular values grow** (aeroelastic mode becomes more energetic)
- The **temporal coefficients** transition from decaying to sustained oscillation
- The **damping ratio** (estimated from V_v) crosses zero → instability

Impact: AGARD and the US Air Force developed flutter margin methods based on tracking modal parameters from SVD — this allows safe prediction of the flutter speed without actually reaching it.

---

7 Nonlinear Dimensionality Reduction

7.1 Limitations of Linear Methods



The Swiss Roll Problem:

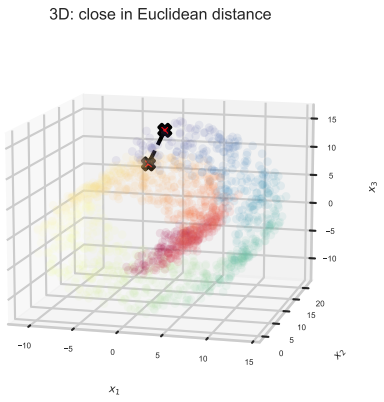
The data is intrinsically **2D** — it lies on a flat sheet that has been rolled up in 3D space

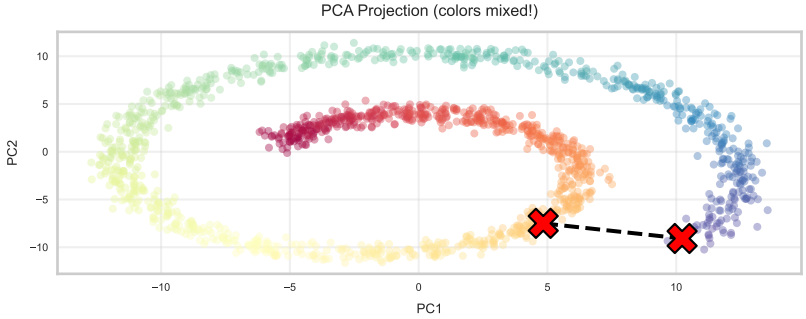
The color encodes position along the roll: nearby colors (red → yellow → green → blue) should stay close

PCA projects onto a flat plane — it crushes the roll, mixing colors that were far apart on the manifold

Need: Methods that "unroll" the manifold by preserving **neighborhood relationships**

7.2 Why Does PCA Fail on Manifolds?





Euclidean vs Geodesic Distance:

PCA preserves **Euclidean** (straight-line) distances. But on a curved manifold, two points can be:

- **Close in Euclidean distance** (through the roll)
- **Far in geodesic distance** (along the surface)

The $\star$ markers show two points that are nearby in 3D (dashed line cuts through the roll) but live on **opposite sides** of the manifold (different colors)

Key insight: A linear projection cannot distinguish Euclidean proximity from manifold proximity

7.3 Swiss Roll: Code

```
from sklearn.datasets import make_swiss_roll
from sklearn.decomposition import PCA

# Generate Swiss roll (2D manifold in 3D)
X, color = make_swiss_roll(n_samples=1500, noise=0.5)

# PCA fails: projects onto flat plane
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# Colors will be mixed -- manifold structure lost!

# Visualize in 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color)
```

7.4 Beyond Linear Methods: A Roadmap

PCA/SVD fail when data lies on a **curved manifold** — the Swiss Roll is the classic example. Several families of nonlinear methods exist:

Graph / Geodesic methods — preserve manifold distances:

- **Isomap** (Tenenbaum et al., 2000): Shortest-path distances on a k-NN graph $\rightarrow$ classical MDS
- **Locally Linear Embedding (LLE)** (Roweis & Saul, 2000): Reconstruct each point from its neighbors, preserve those weights in low-D

Spectral methods — eigendecompose a similarity matrix:

- **Laplacian Eigenmaps** (Belkin & Niyogi, 2003): Graph Laplacian of k-NN graph
- **Diffusion Maps** (Coifman & Lafon, 2006): Random walk on the data graph

Probabilistic / optimization methods — match neighborhood distributions:

- **t-SNE** (van der Maaten & Hinton, 2008): Pairwise Gaussian $\rightarrow$ Student-t affinities; minimizes KL divergence. Great for visualization, but $O(n^2)$ and non-invertible
- **UMAP** (McInnes et al., 2018): Topological approach; faster ($O(n \log n)$), preserves more global structure, supports inverse transforms

For this course: Know that these exist and when to reach for them. We will revisit nonlinear methods when we cover **autoencoders** in the deep learning unit.

8 Summary

8.1 Method Comparison

	PCA	Truncated SVD	POD
Type	Linear	Linear	Linear
Input	Features	Any matrix	Snapshots
Preserves	Variance	Frobenius norm	Energy
Invertible			
Scalability	$O(nd^2)$	$O(ndk)$	$O(nmk)$
Use case	General ML	Compression	Physics ROM

8.2 Key Takeaways

1. **PCA = SVD** on centered data — SVD is the more general and numerically stable tool
2. **Truncated SVD** gives the optimal rank-k approximation (Eckart–Young theorem)
3. **POD = SVD on physical snapshots** — extracts coherent structures from data without a model
4. **Aerospace applications**: POD reveals vortex structures in flows and vibration mode shapes in structures — both with < 5 modes capturing $> 95\%$ energy
5. **SVD is a Swiss-army knife**: sensor placement (QR-pivoted SVD), structural health monitoring (subspace angles), trajectory compression, and anomaly detection all follow from the same decomposition
6. **Nonlinear methods** (Isomap, LLE, t-SNE, UMAP, autoencoders) exist for manifold data that PCA cannot handle — we will return to these when we cover deep learning

8.3 Next Topic Preview

Neural Networks — Function Approximation

- From linear models to nonlinear: activation functions
- Universal approximation theorem
- Backpropagation and gradient descent
- First steps toward deep learning