Lecture 7

The Eigen- & Singular Value Decompositions

Nonlinear matrix algorithms

CS328 - Numerical Methods for Visual Computing and Machine Learning

Prof. Wenzel Jakob

EPFL RGL Realistic Graphics Lab

MATH-111 recap

eigen, *noun (ger.)*: self

Eigenvalue

Eigenvector

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

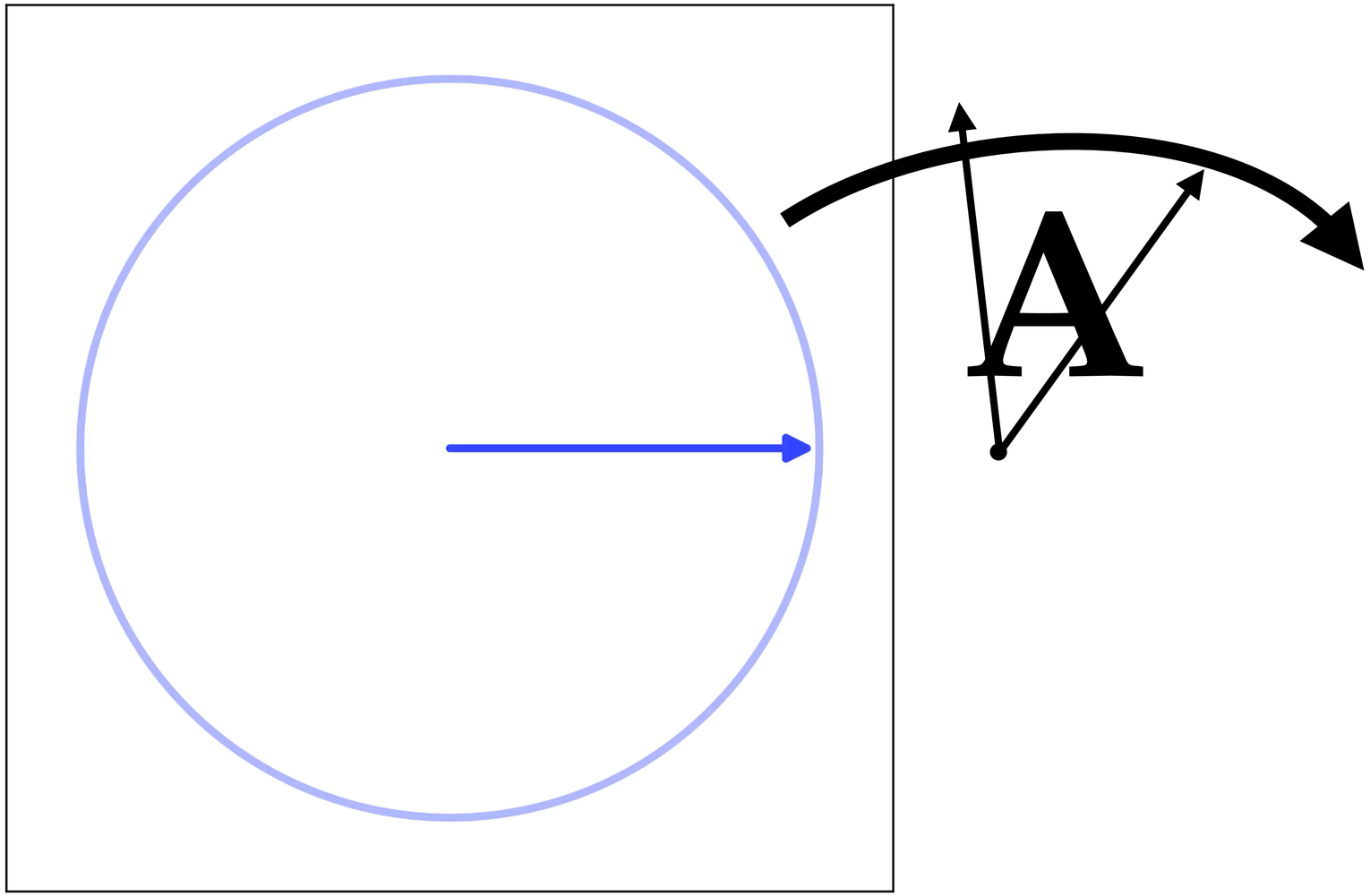
where

$$\mathbf{A} \in \mathbb{R}^{n \times n}, \quad \lambda_i \in \mathbb{R}$$

$$\mathbf{v}_i \in \mathbb{R}^n, \|\mathbf{v}_i\| = 1$$

$$(i = 1, \ldots, n)$$

Eigendecomposition: the geometry



$$\mathbf{A}\mathbf{v}_1 = \lambda_1 \mathbf{v}_1$$
$$\mathbf{A}\mathbf{v}_2 = \lambda_2 \mathbf{v}_2$$

```
# Randomly generated matrix

A = array([[ 1.16043581,  0.0566787 ],
           [ 0.56722123,  0.85777919]])

λ, V  = scipy.linalg.eig(A)
```

Eigenvectors (.. of non-singular matrices)

- For non-singular matrices $\mathbf{A}$:
  - there are n eigenvalue/eigenvector pairs
  - the eigenvectors $\mathbf{v}_i$ form a basis for $\mathbb{R}^n$
    - .. which are not generally orthogonal (guaranteed only if $\mathbf{A}$ is symmetric).
- Therefore, a matrix with those $\mathbf{v}_i$ as its columns

$$\mathbf{V} = \begin{pmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{pmatrix}$$

has rank n (i.e., *full rank*) and so is non-singular

What happens when we multiply A and V?

$$\mathbf{A} \cdot \begin{pmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ \lambda_1 \mathbf{v}_1 & \cdots & \lambda_n \mathbf{v}_n \\ | & & | \end{pmatrix} = \mathbf{V} \underbrace{\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}}_{=:\mathbf{\Lambda}}$$

In other words:

The Eigendecomposition

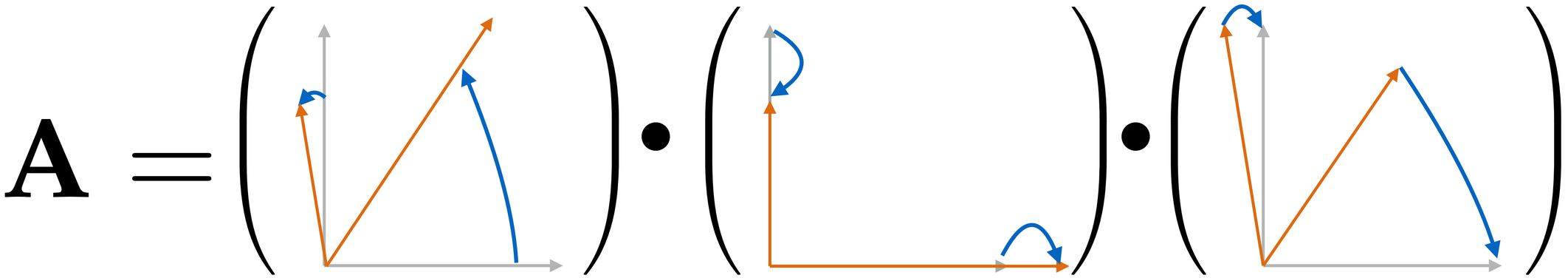
$$\mathbf{AV} = \mathbf{V\Lambda}$$

1

Geometric summary

The eigendecomp. expresses a linear transformation A as product of 3 transforms

$$\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}$$



(inverse)
change of basis

non-uniform scale

change of basis (*not necessarily orthogonal!*)

Eigendecomposition: what is it good for?

- **Taking integer & fractional powers of matrices.**
- **Solving linear ordinary differential equations (ODEs).**
  - Physics models the world using differential equations.
- **Estimating covariance matrices.**
  - Normal distribution: *the* central statistical distribution.
    Eigendecomposition "fits" normal distribution to data points.
- **Dimensionality reduction.**
  - Important processing step to remove irrelevant data.

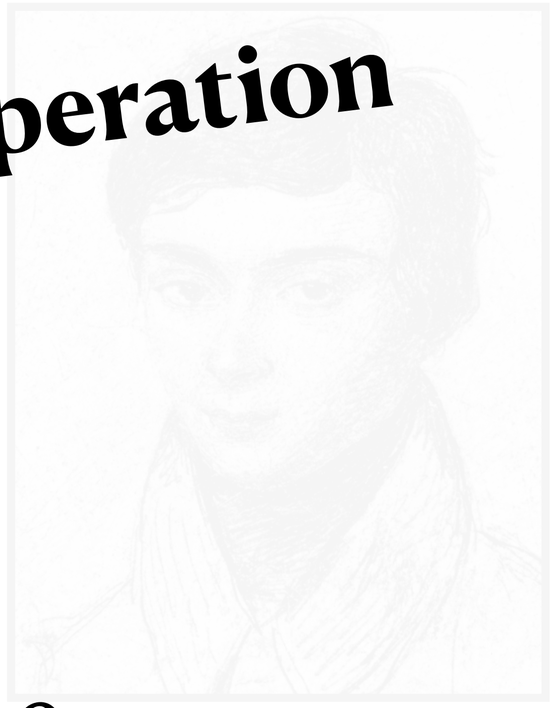
Will see examples of all of these.
(Some in the context of SVD)

How to find the eigendecomposition?

MATH-111: Use the Characteristic Polynomial!

$$\det(\lambda \mathbf{I} - \mathbf{A}) = 0$$

- As the name indicates, this is a polynomial.
- Degree = size of the matrix
- *Galois theory*: most polynomials of degree 5 or higher cannot be solved using *algebraic methods*.
- Oops.



Évariste Galois (1811-1832)

Eigendecomposition is a **nonlinear operation**

(our very first one!)

- Approximate result
- No guarantees on computation time

Matrix powers

The following works for both integer and fractional powers

- Let's compute

$$\mathbf{A}^2 = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}\mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1} = \mathbf{V}\mathbf{\Lambda}^2\mathbf{V}^{-1}$$

- General rule:

$$\mathbf{A}^k = \mathbf{V}\mathbf{\Lambda}^k\mathbf{V}^{-1}$$

- That includes the inverse as well:

$$\mathbf{A}^{-1} = \mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{V}^{-1}$$

Observation when taking high powers of a matrix

(Building on the power identity from the previous slide)

$$\mathbf{A}^k = \mathbf{V}\mathbf{\Lambda}^k\mathbf{V}^{-1}$$

$$\mathbf{\Lambda}^k \approx \begin{pmatrix} \lambda_1^k & & \\ & 0 & \\ & & \ddots \\ & & & 0 \end{pmatrix}$$

As $k \to \infty$, first entry becomes **huge** compared to others.

Power iteration

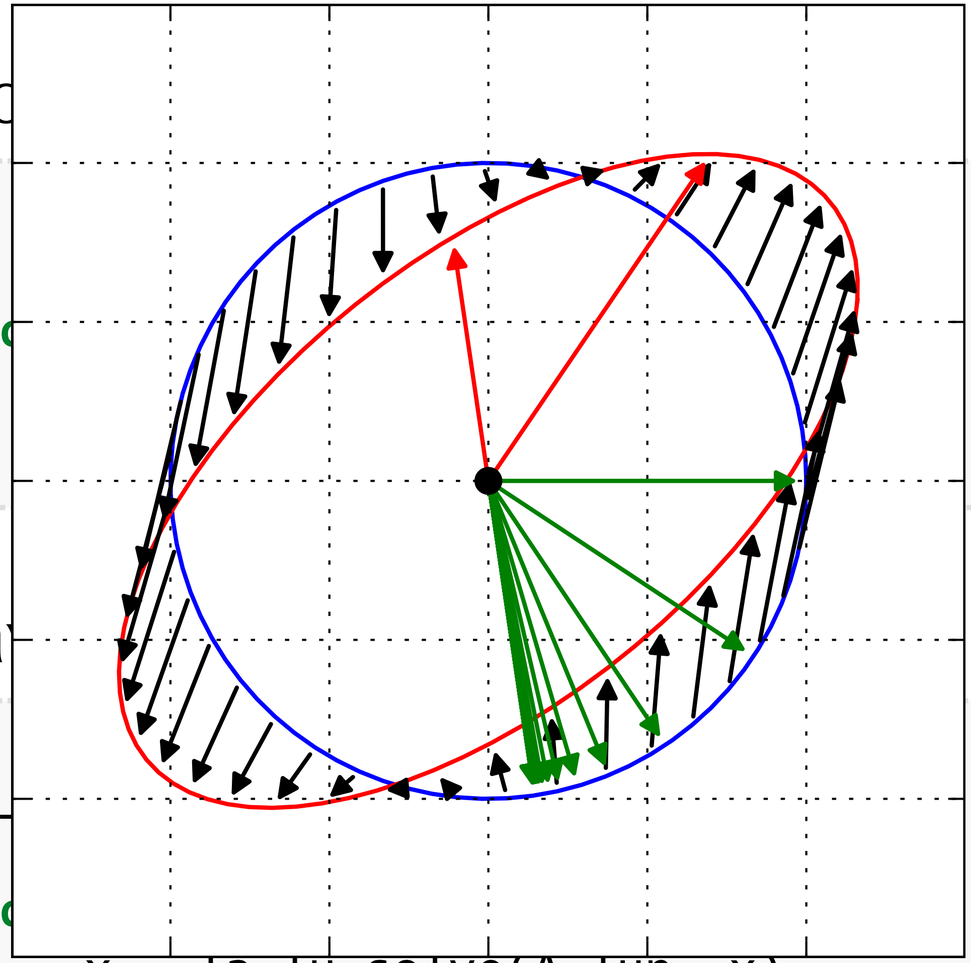
A simple method to compute the dominant eigenvector

Po

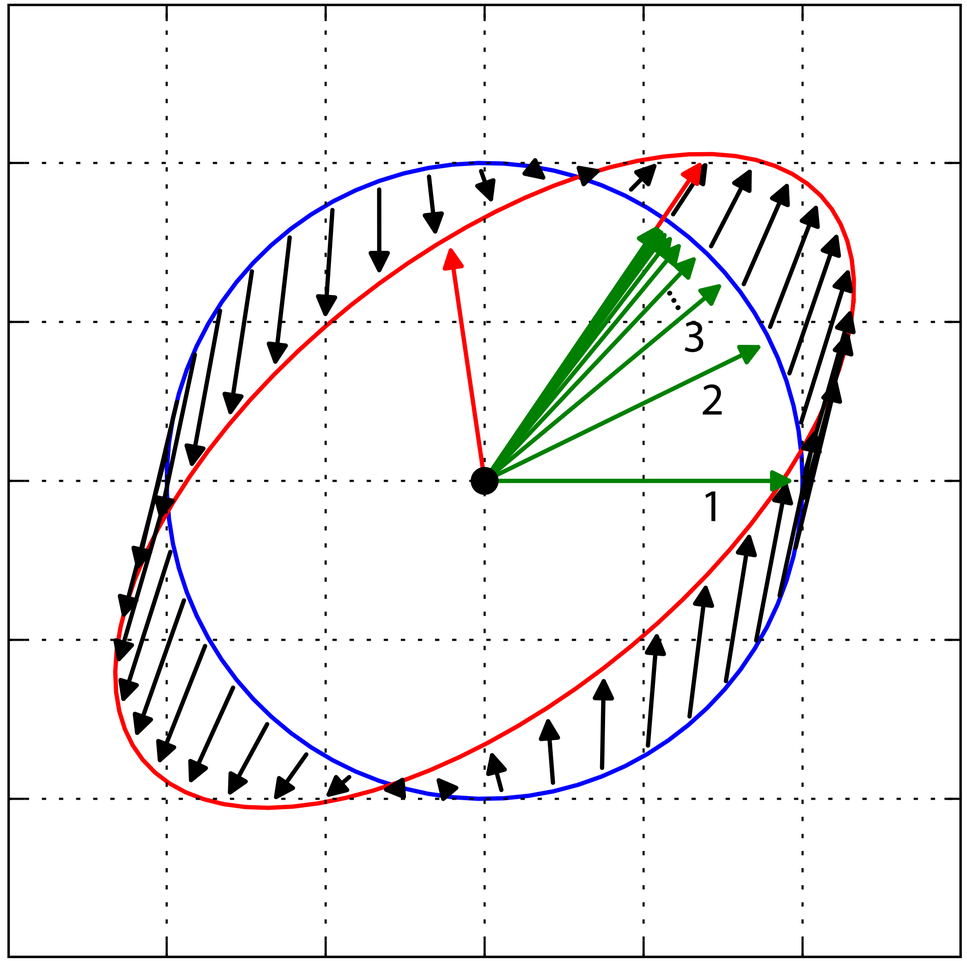
```
x
fo
```

In

```
x
A_
fo
    x = la.lu_solve(A_lup, x)
    x /= np.linalg.norm(x)
```



Inverse iteration



Power iteration

QR algorithm

```
def eig(A, n_it=10):
    for i in range(n_it):
        Q, R = la.qr(A)
        A = R @ Q
    return A
```

Similarity transformation: Same eigenvalues

$$\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k = \underline{\mathbf{Q}_k^{-1}\mathbf{Q}_k\mathbf{R}_k}\mathbf{Q}_k = \mathbf{Q}_k^{-1}\mathbf{A}_k\mathbf{Q}_k$$

- **Intuition**: Run n instances of the power iteration at the same time.
  - QR factorization renormalizes vectors and keeps the eigenvector estimates from all collapsing onto the same solution.

[Link: Detailed explanation]

Demo time

QR algorithm

```
def eig(A, n_it=10):
    for i in range(n_it):
        Q, R = la.qr(A)
        A = R @ Q
    return A
```

Similarity transformation: Same eigenvalues

$$\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k = \underline{\mathbf{Q}_k^{-1}\mathbf{Q}_k\mathbf{R}_k}\mathbf{Q}_k = \mathbf{Q}_k^{-1}\mathbf{A}_k\mathbf{Q}_k$$

- Intuition: Run n instances of the power iteration at the same time.
  - QR factorization renormalizes vectors and keeps the eigenvector estimates from all collapsing onto the same solution.
- **In practice**: many subtleties, use libraries and don't build your own version.

[Link: Detailed explanation]

Application: Solving linear ODEs

Simple homogeneous ODE: $x'(t) = ax(t)$

Solutions have the form: $x(t) = e^{at}x(0)$

let's double-check that: $x'(t) = ae^{at}x(0)$

Same approach also works for systems of ODEs!

System of homogeneous ODEs: $\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t)$

Solutions have the form: $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0)$

The Matrix Exponential

A way to evaluate $\exp(\mathbf{A})$ when $\mathbf{A}$ is a matrix.

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) \quad \overset{?!?!}{}$$

Standard series definition: $e^{a} = \sum_{k=0}^{\infty} \frac{1}{k!} a^{k}$ $\qquad$ $e^{\mathbf{A}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{A}^{k}$

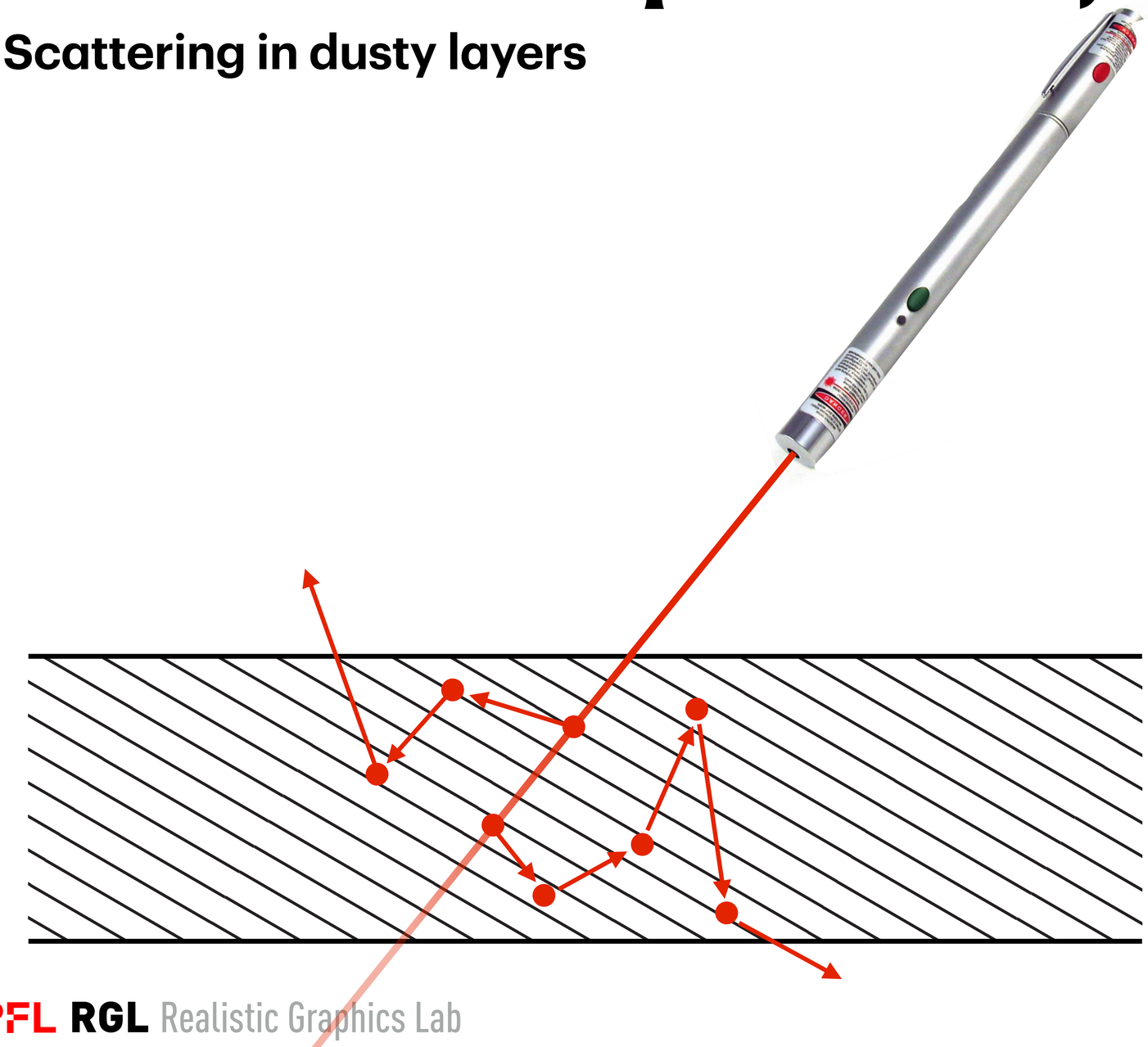
$$e^{\mathbf{A}} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{V}\mathbf{\Lambda}^{k}\mathbf{V}^{-1}$$

$$= \mathbf{V}\left[\sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{\Lambda}^{k}\right]\mathbf{V}^{-1} = \mathbf{V}e^{\mathbf{\Lambda}}\mathbf{V}^{-1}$$

$$e^{\mathbf{\Lambda}} = \begin{pmatrix} e^{\lambda_1} & & \\ & \ddots & \\ & & e^{\lambda_n} \end{pmatrix}$$

A random example from my own research

Scattering in dusty layers



Problem has the form

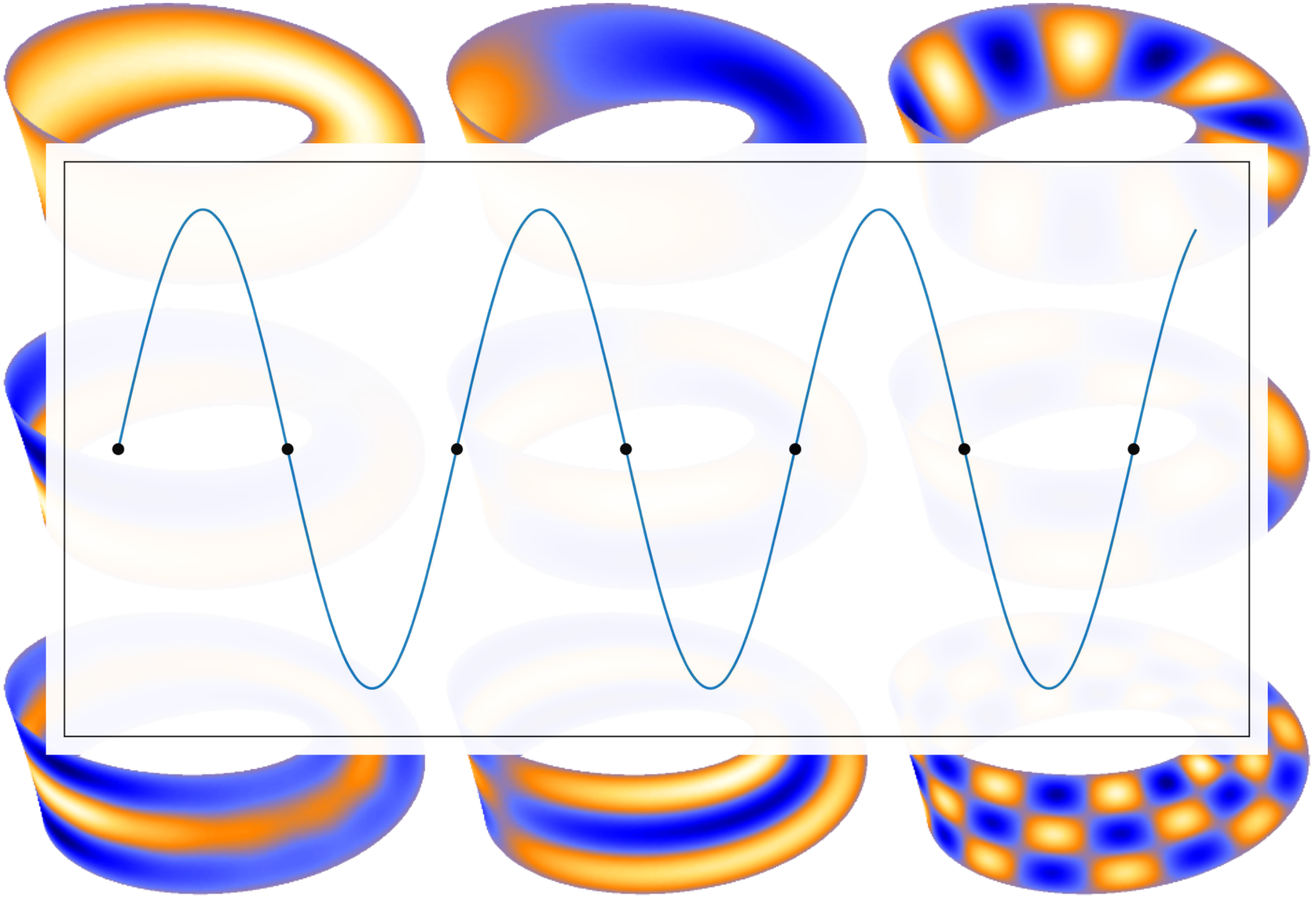
$$\mathbf{x}'(t) = \mathbf{A}\mathbf{x}(t)$$

$\mathbf{x}(t)$ is the light traveling in different directions.

t is the depth within the layer.


α=0.02
τ=0.000, g=0.900

Oscillations on surfaces



[Langlois, An, Jin, and James, SIGGRAPH 2014]

Diameter: 19 cm
Vertices: 51434
Compressed: 26 kB
Uncompressed: 28 MB

Complex eigenvalues / eigenvectors

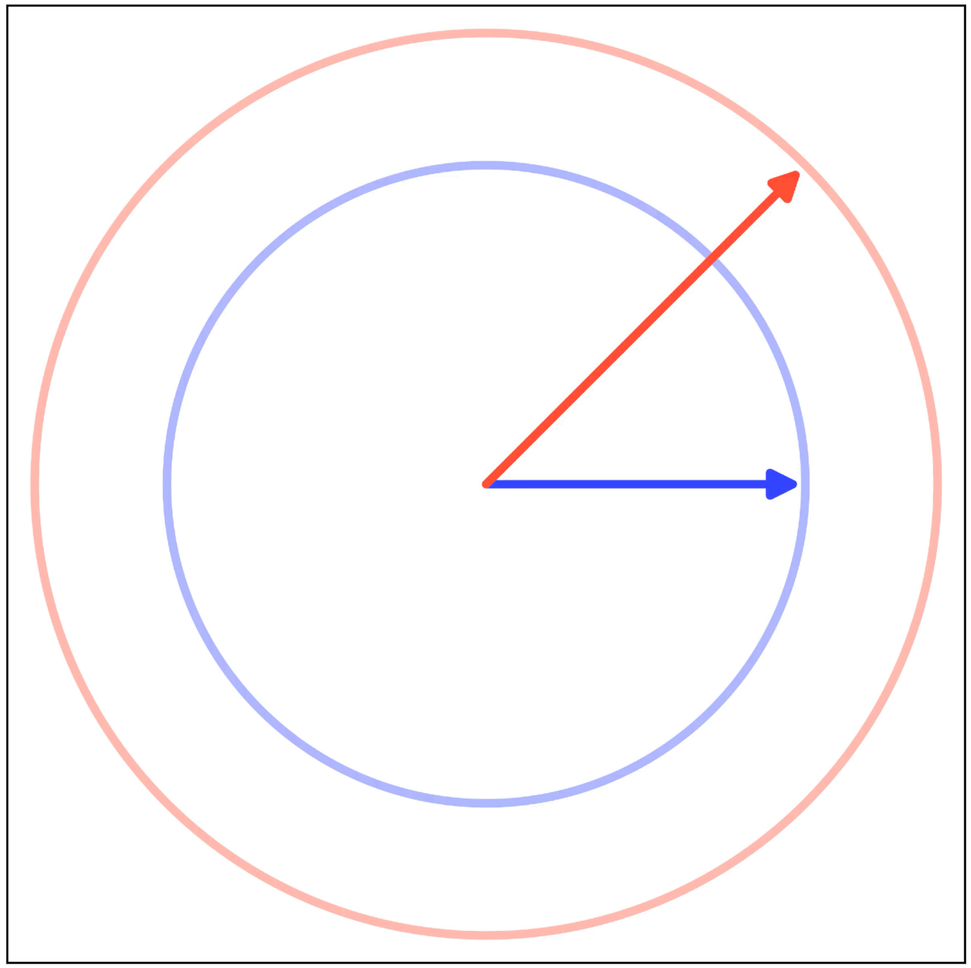
An awkward special case

$$\mathbf{A} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$\det\left(\lambda\mathbf{I} - \mathbf{A}\right) = 0.$$

$$\Leftrightarrow \quad \lambda^2 - 2\lambda + 2 = 0.$$

$$\Leftrightarrow \quad \lambda = \frac{2 \pm \sqrt{4 - 8}}{2} = 1 \pm i.$$



Eigenanalysis in practice: it's complicated..

non-symmetric case

1. an eigenvalue/eigenvector "pair" is actually a *triple* $(\mathbf{x}, \mathbf{y}, \lambda)$

where $\mathbf{y}^T\mathbf{A} = \lambda\mathbf{y}^T$ *"left eigenvector"* $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ *"right eigenvector"*

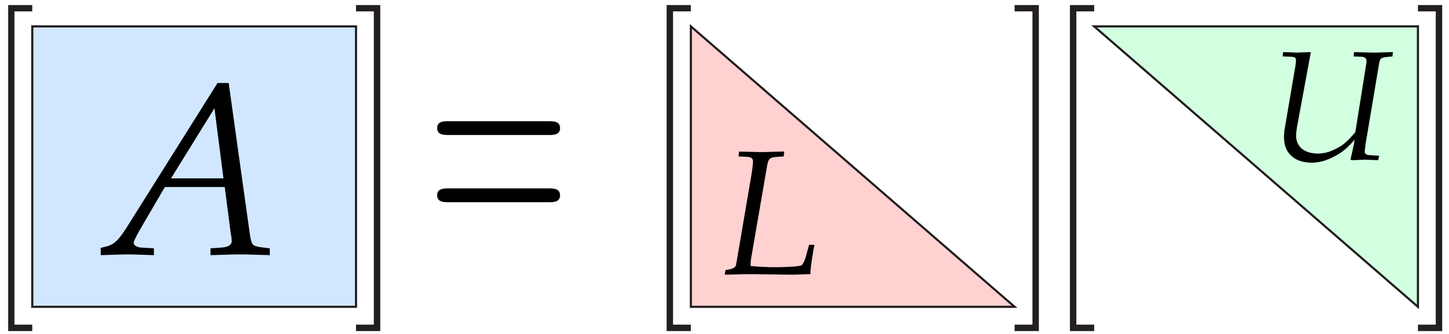
2. complex arithmetic generally required.

3. methods can be very unstable if eigenvalues not well-separated.

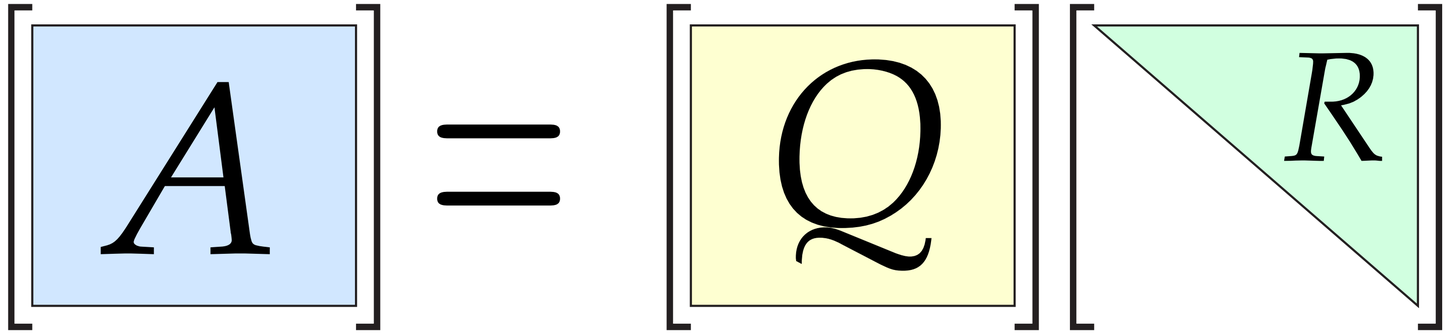
4. Doesn't make sense for non-square $\mathbf{A}$!

The Singular Value Decomposition

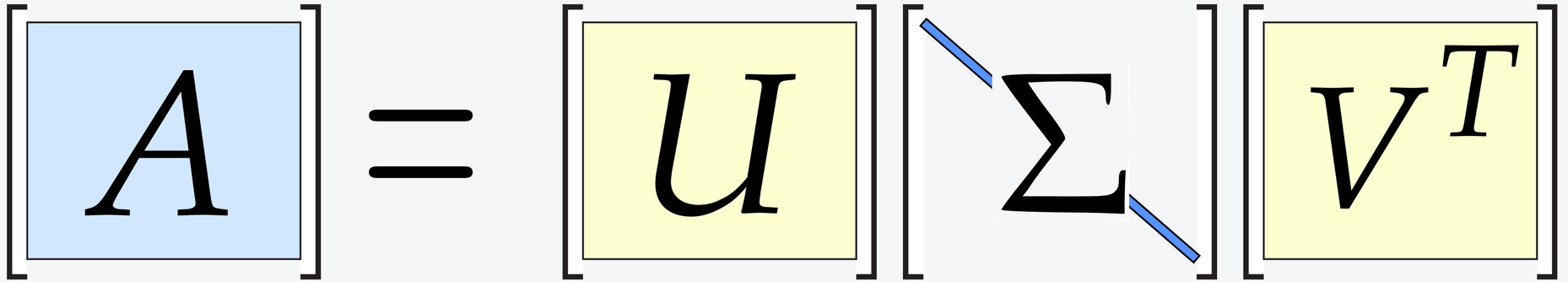
LU



QR



SVD



The Singular Value Decomposition

The Singular Value Decomposition

- Can be computed for **any** matrix (non-square, non-symmetric, singular, ..)
- Involves only orthogonal and diagonal matrices.
- **Impeccable** numerical properties.
- No complex arithmetic necessary.
- Expensive to compute (~5-10 times the cost of LU)
- SVD and Eigendecomposition are *identical** when **A** is symmetric.

* Except for tiny differences: ordering of entries, etc.

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

Terminology

Let's look at the m=n case first.

$$\mathbf{V} = \begin{pmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{pmatrix}$$

"*Right singular vectors*"
Orthogonal.

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_n \\ | & & | \end{pmatrix}$$

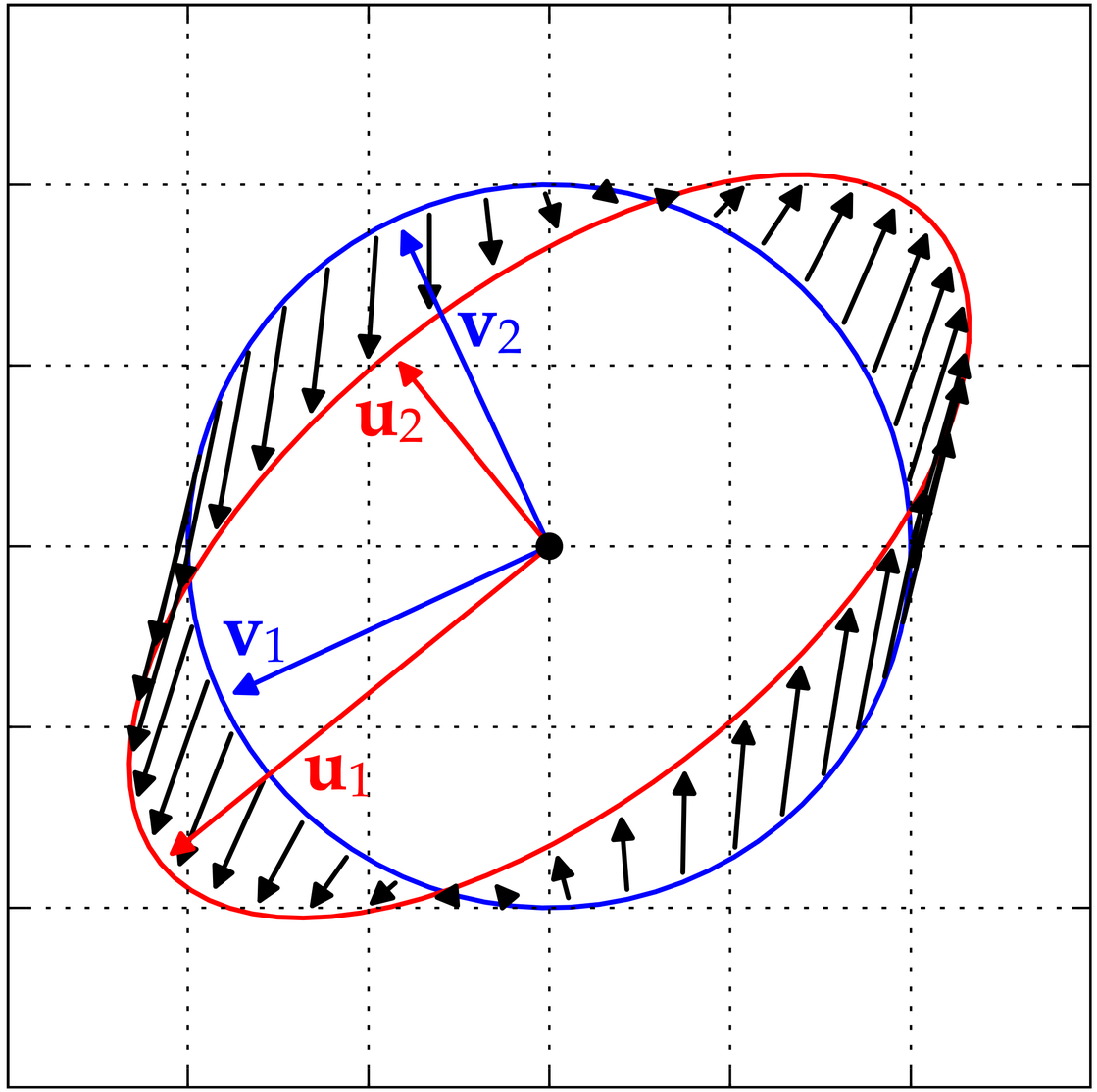
"*Left singular vectors*"
Orthogonal.

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}$$

"*Singular values*"
Positive, sorted in decreasing order

SVD

```
U, Σ, V = scipy.linalg.svd(A)
```



Another view of the SVD

$$\mathbf{A} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T$$

where

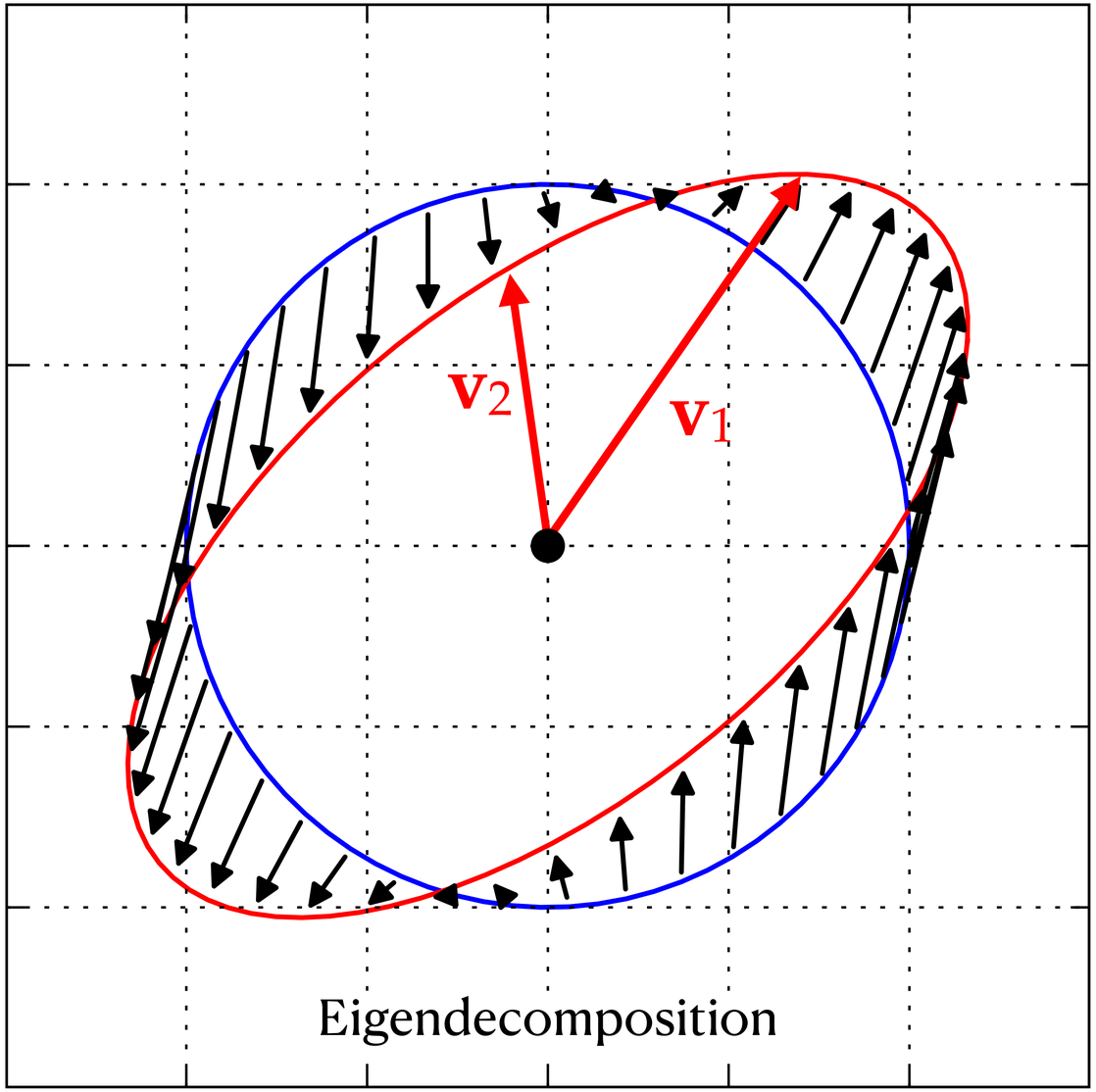
$$\mathbf{V} = \begin{pmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{pmatrix} \quad \mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_n \\ | & & | \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}$$

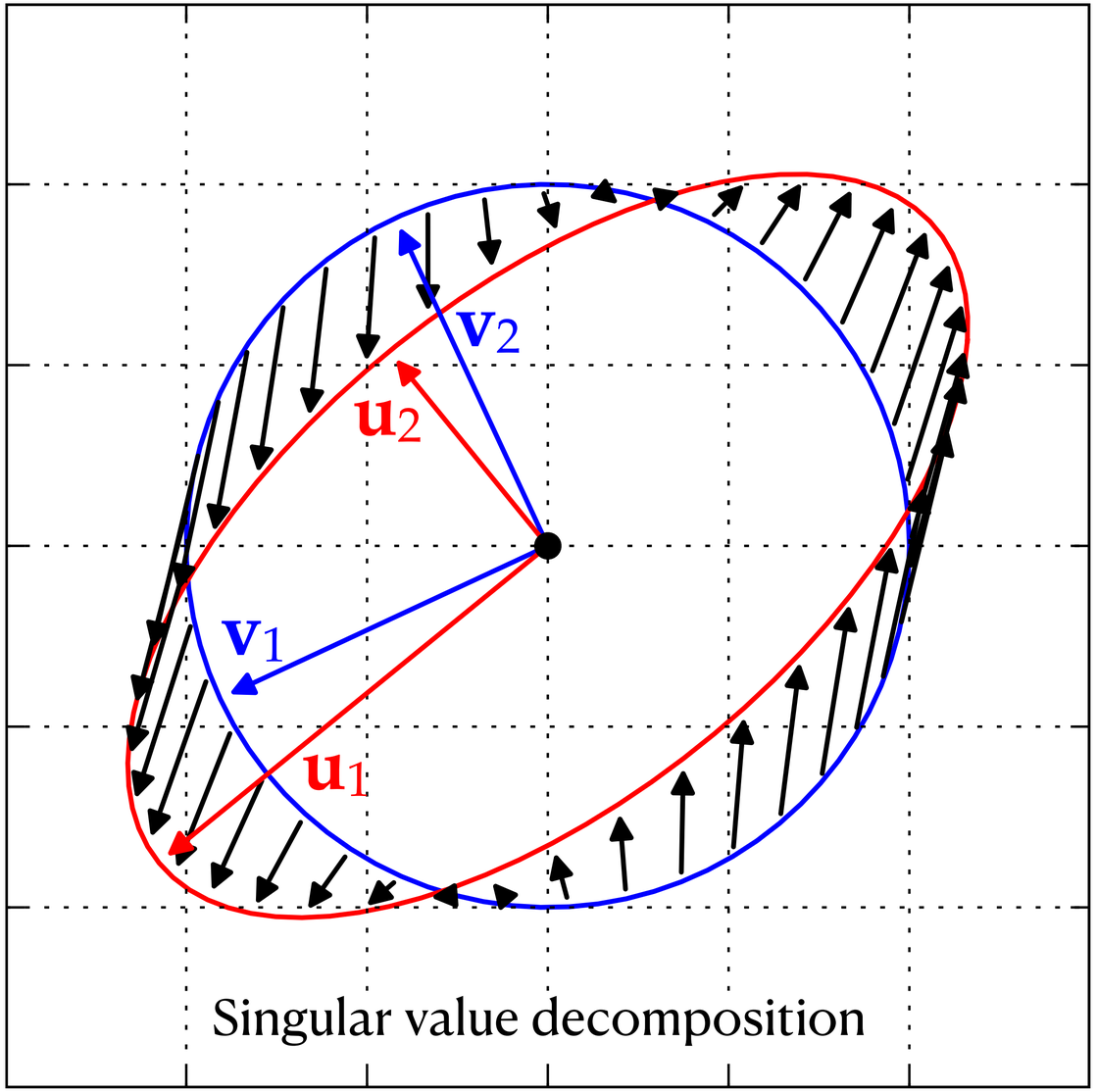
$$\text{Then } \mathbf{A} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

SVD vs Eigendecompositions

- A linear transformation maps a circle/sphere onto an ellipse/ellipsoid
- **Eigendecomposition**
  - Question: *which directions stay aligned when transforming by A?*
- **SVD**
  - Question: *what are the principal axes of the ellipse/ellipsoid, and which directions on the circle map to them?*
- For symmetric matrices, this is the same question.

Eigendecomposition vs. SVD





MATH-232 Review: The Normal Distribution

Univariate (1D) case. Also known as "*Gaussian distribution*"

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

Variance (square of std. deviation)

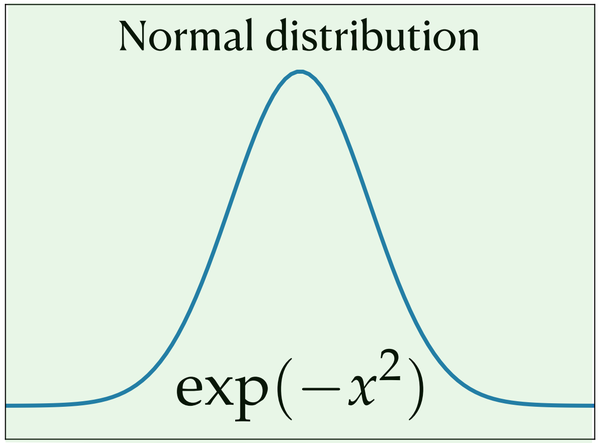
Density function:

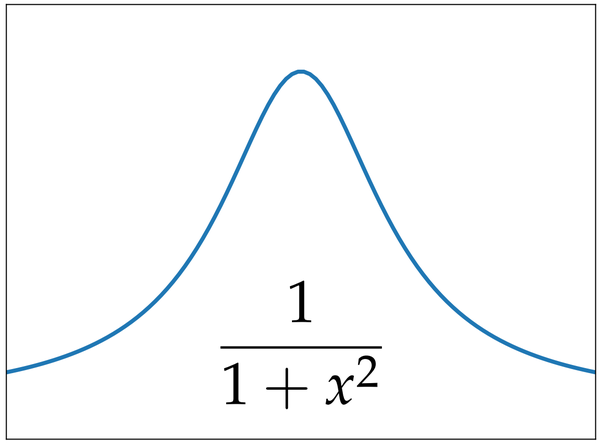
$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

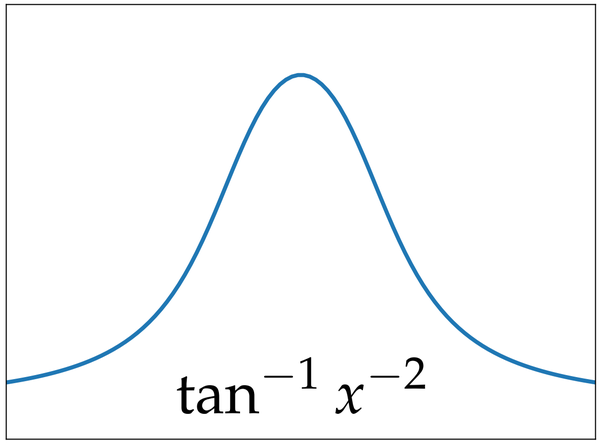
normalization constant

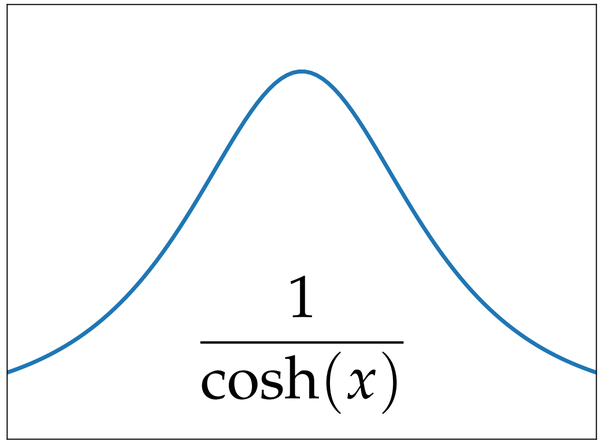
exponent

Standard deviation









The Normal Distribution

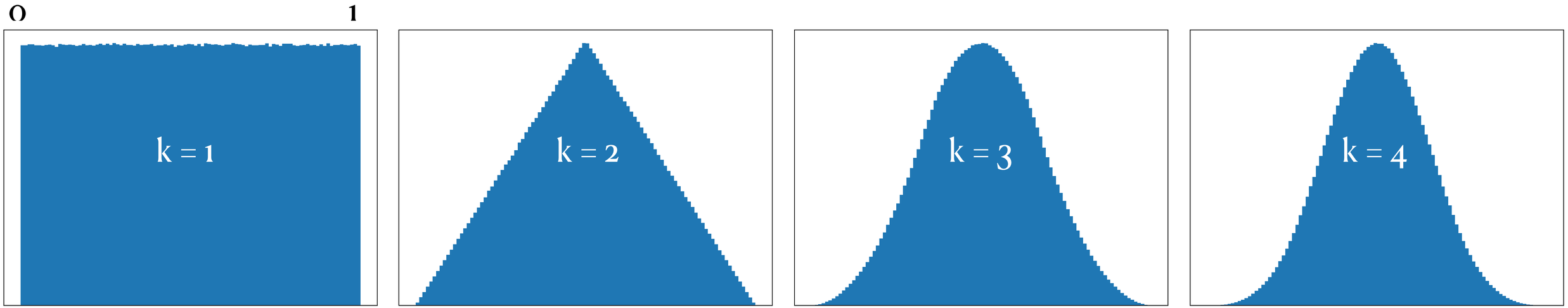
Why is *this particular* function interesting?

$$X_1, X_2, \ldots \quad \text{iid. random variables with } E[X_i] = \mu \text{ and } \mathrm{Var}[X_i] = \sigma^2$$

$$\bar{X}_k = \frac{X_1 + \ldots + X_k}{k}$$

$$\frac{\sqrt{k}}{\sigma}\left(\bar{X}_k - \mu\right) \xrightarrow{\text{i.d.}} \mathcal{N}(0,1) \quad (k \to \infty)$$

Central Limit Theorem



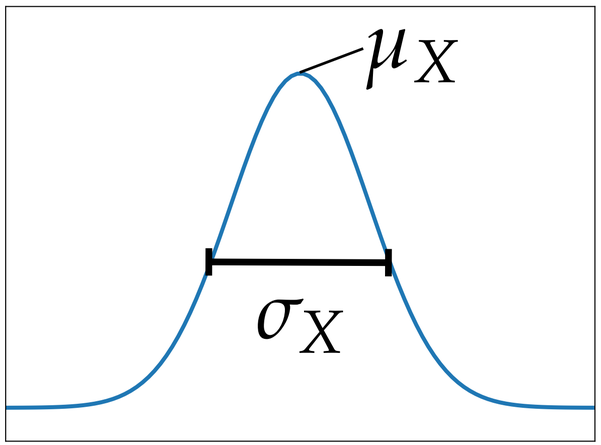
Analysing data in higher dimensions (2D)

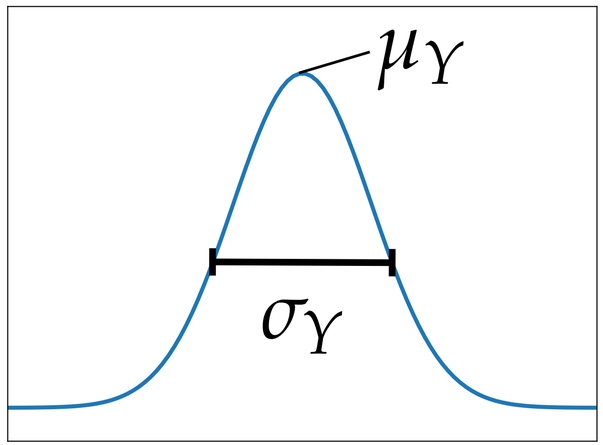
$$X_1, X_2, \ldots \qquad\qquad Y_1, Y_2, \ldots$$

$$\bar{X}_k = \frac{X_1 + \ldots + X_k}{k} \qquad\qquad \bar{Y}_k = \frac{Y_1 + \ldots + Y_k}{k}$$

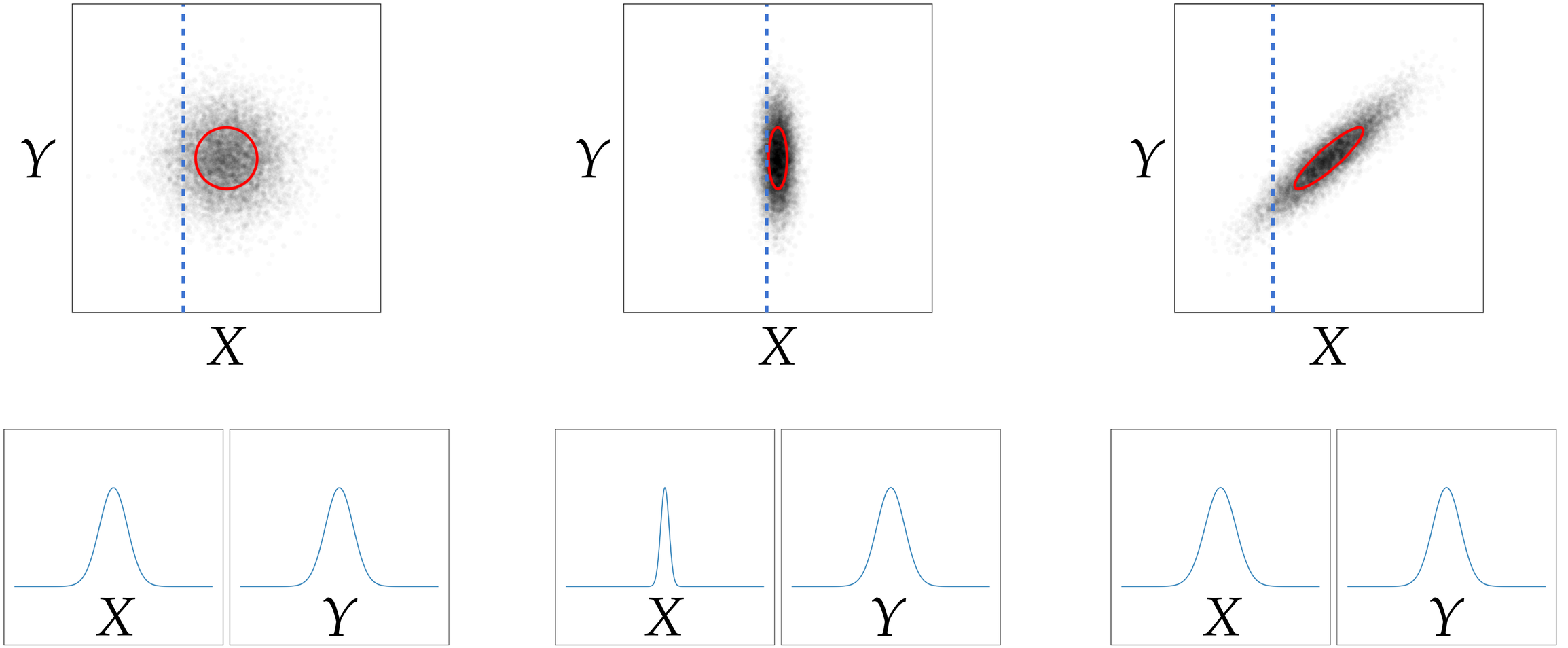
$$\bar{X}_k \to \mu_X \qquad \text{(averages converge to mean)} \qquad \bar{Y}_k \to \mu_Y$$

$$\overline{(X_k - \bar{X}_k)^2} \to \sigma_X^2 \qquad \text{(squared deviations converge to variance)} \qquad \overline{(Y_k - \bar{Y}_k)^2} \to \sigma_Y^2$$





Seeing the whole picture.



Correlations in multidimensional data

1. Age
2. Hardness
3. Smell intensity
4. Taste intensity
5. Amount of mold
6. Average color (RGB)
7. Average number of sun spots during maturation
8. Temperature on the back side of the moon

Intuitively, collecting data will reveal that some fields have an approximately linear relationship (they are **correlated**), e.g.:

- **Very old** cheeses have a **strong taste** and are **smelly**.
- Blue cheeses are generally blue because of mold.

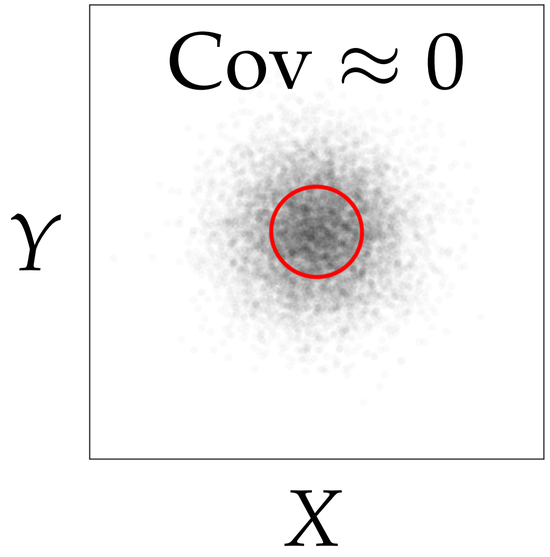
Other axes will show **no correlation** in plots.

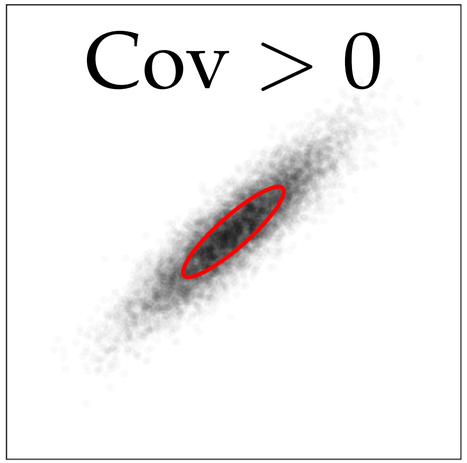
MidJourney: *Board with cheese selection, white background.*

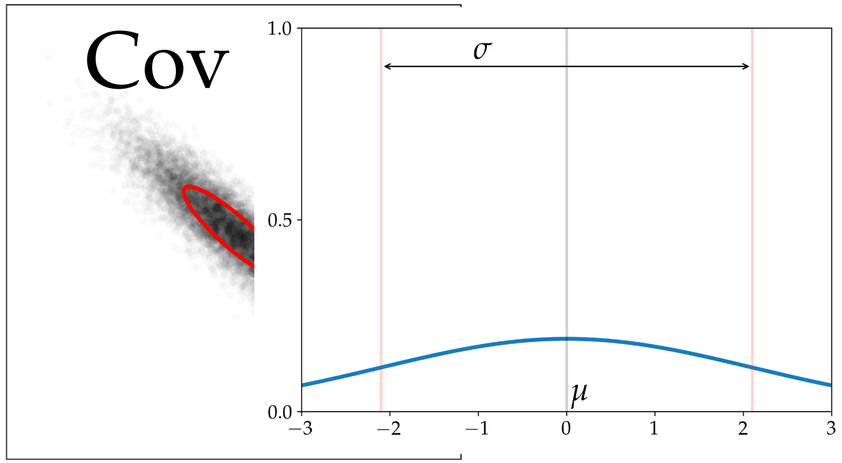
Covariance: in 2 dimensions

Definition: $$\mathrm{Cov}(X,Y) = E\left[(X - E[X])(Y - E[Y])\right]$$

Estimator: $$\overline{(X_k - \bar{X})(Y_k - \bar{Y})} \to \mathrm{Cov}(X,Y) \quad (k \to \infty)$$







Covariance: in *m* dimensions

$$\mathbf{X} = \begin{bmatrix} X_0 - \bar{X} \\ \\ \\ \\ \end{bmatrix}$$

n observations

m dimensions

$$\mathbf{\Sigma} = \frac{1}{N}\mathbf{X}^T\mathbf{X}$$

Sometimes *N-1* is used here instead. It depends on the setup (beyond the scope of CS328).

Multivariate Normal Distribution

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Mean $\in \mathbb{R}^n$ Covariance matrix $\in \mathbb{R}^{n \times n}$

Density function:

$$f_{\mathbf{x}}(\mathbf{x}) = \underbrace{(2\pi)^{-\frac{n}{2}} \det(\boldsymbol{\Sigma})^{-\frac{1}{2}}}_{\text{normalization constant}} \exp\left(\underbrace{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}_{\text{exponent}} \right)$$

Compare to 1D case:

$$f_x(x) = (2\pi)^{-\frac{1}{2}} \sigma^{-1} \exp\left(-\frac{1}{2}(x-\mu)\sigma^{-2}(x-\mu) \right)$$

Creating Normal Variates

Creating general normal variates (a.k.a. "simulating" or "sampling" them)

1-D case: $X \sim \mathcal{N}(0,1) \quad \Rightarrow \quad \sigma X + \mu \sim \mathcal{N}(\mu, \sigma^2)$

NumPy recipe:

```
sigma * np.random.randn() + mu
```

N-D case: $\mathbf{X} \sim \mathcal{N}(0,\mathbf{I}) \quad \Rightarrow \quad \mathbf{A}\mathbf{X} + \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{A}\mathbf{A}^T)$

$$\Downarrow$$

$$\boldsymbol{\Sigma}^{\frac{1}{2}}\mathbf{X} + \boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

NumPy recipe:

```
sqrt_sigma @ np.random.randn(n) + mu
```

Statistical Analysis of Multivariate Data

Principal component analysis (PCA): using eigenvectors and eigenvalues to describe data.

- **Step 1**: compute the mean of all data points. Subtract this value from all data points.

- *Version 1* (via Eigenanalysis):
  - **Step 2**: estimate the covariance matrix $\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X} \,/\, n$
  - **Step 3:** compute $\mathbf{V}, \mathbf{\Lambda} = \text{eig}(\mathbf{\Sigma})$
  - **Profit**: principal directions given by $\mathbf{V}$, radii given by $\sqrt{\mathbf{\Lambda}}$

- *Version 2* **(via SVD):**
  - **Step 2**: compute $\mathbf{U}, \mathbf{S}, \mathbf{V} = \text{svd}(\mathbf{X} \,/\, \sqrt{n})$
  - **Profit**: principal directions given by $\mathbf{V}$, radii given by $\mathbf{S}$

Why does this work?

- Covariance matrix is **symmetric**
  - MATH-111: It is therefore a linear transformation that expresses **scaling** along a set of axes. That's it. (It, e.g., cannot rotate or shear its input)
  - Eigendecomposition: explains symmetric matrices as scaling operation in an orthogonal coordinate system

Demo time

The Outer Product

Also often called a rank-1 matrix.

$$\mathbf{u}\mathbf{v}^T = \begin{bmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{bmatrix}$$

Another view of the SVD

$$\mathbf{A} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T$$

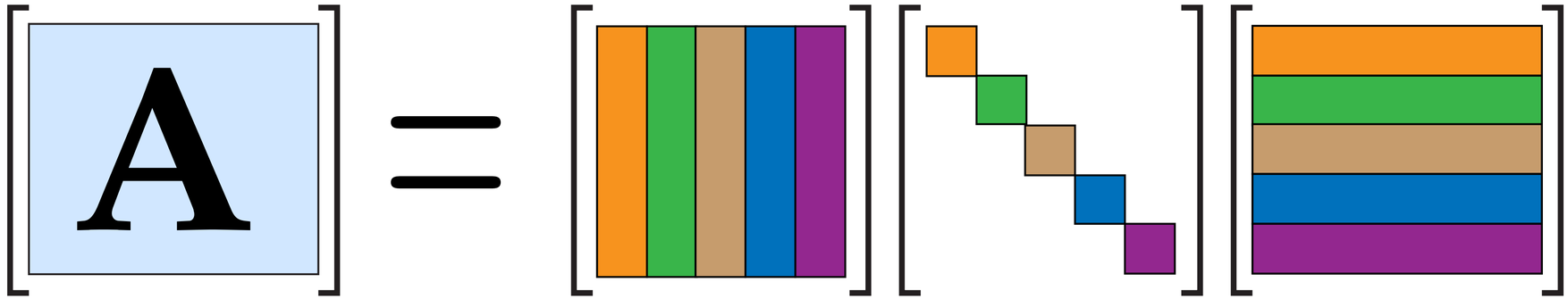
where

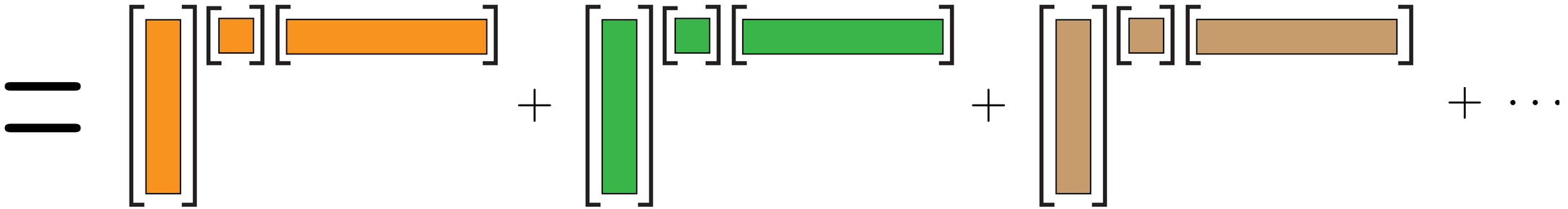
$$\mathbf{V} = \begin{pmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{pmatrix} \quad \mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdots & \mathbf{u}_n \\ | & & | \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}$$

$$\text{Then } \mathbf{A} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

Another view of the SVD

Coloring each combination of left/right singular vector & value reveals outer product structure

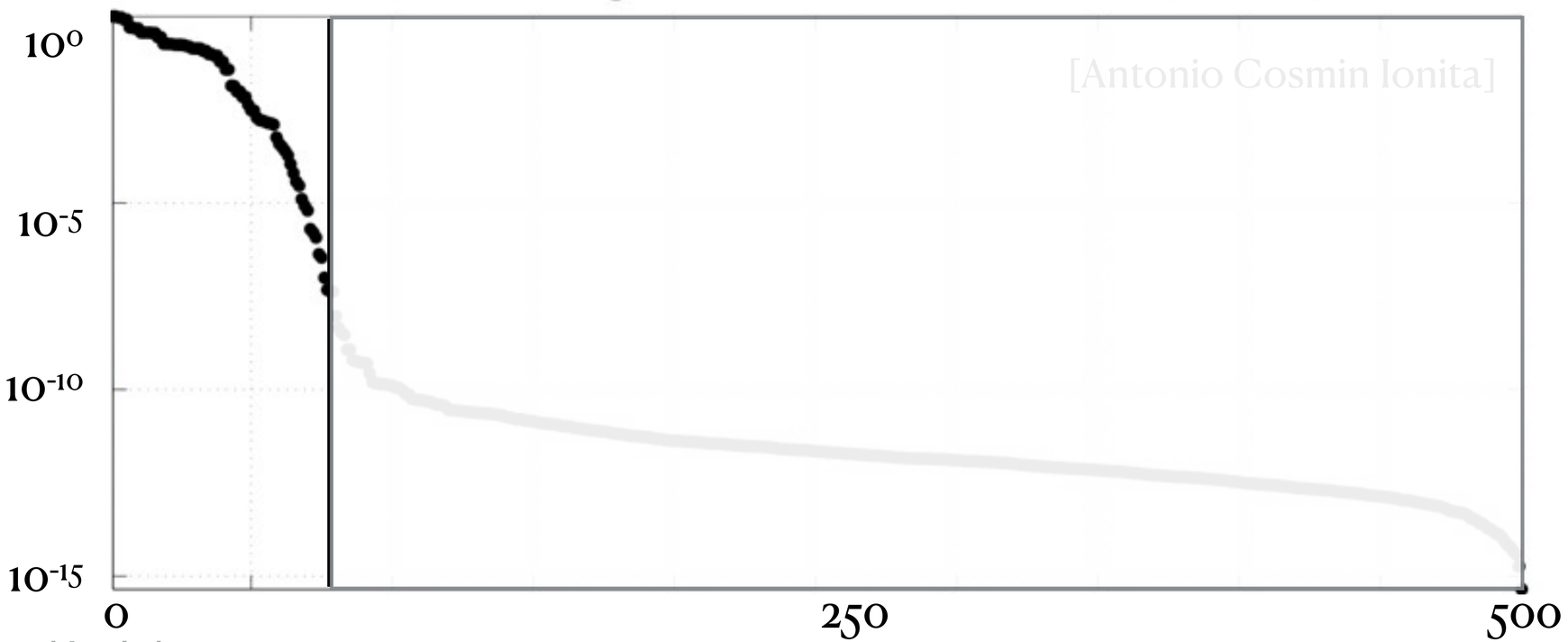




Matrix approximation

$$\mathbf{A} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \longrightarrow \mathbf{A} \approx \sum_{i=1}^{n_{\max}} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

Plot of singular values (in decreasing magnitude)



Eckart-Young Theorem

Stated without proof.

Suppose that **A'** is obtained from **A** by truncating all but the largest k singular values from its singular value decomposition.

Then **A'** minimizes both

$$(i) \quad \|\mathbf{A} - \mathbf{A}'\|_F \quad \textit{and}$$

$$(ii) \quad \|\mathbf{A} - \mathbf{A}'\|_2$$

among all matrices **A'** with rank k.

$$\|\mathbf{A}\|_F^2 := \sum_{ij} A_{ij}^2$$

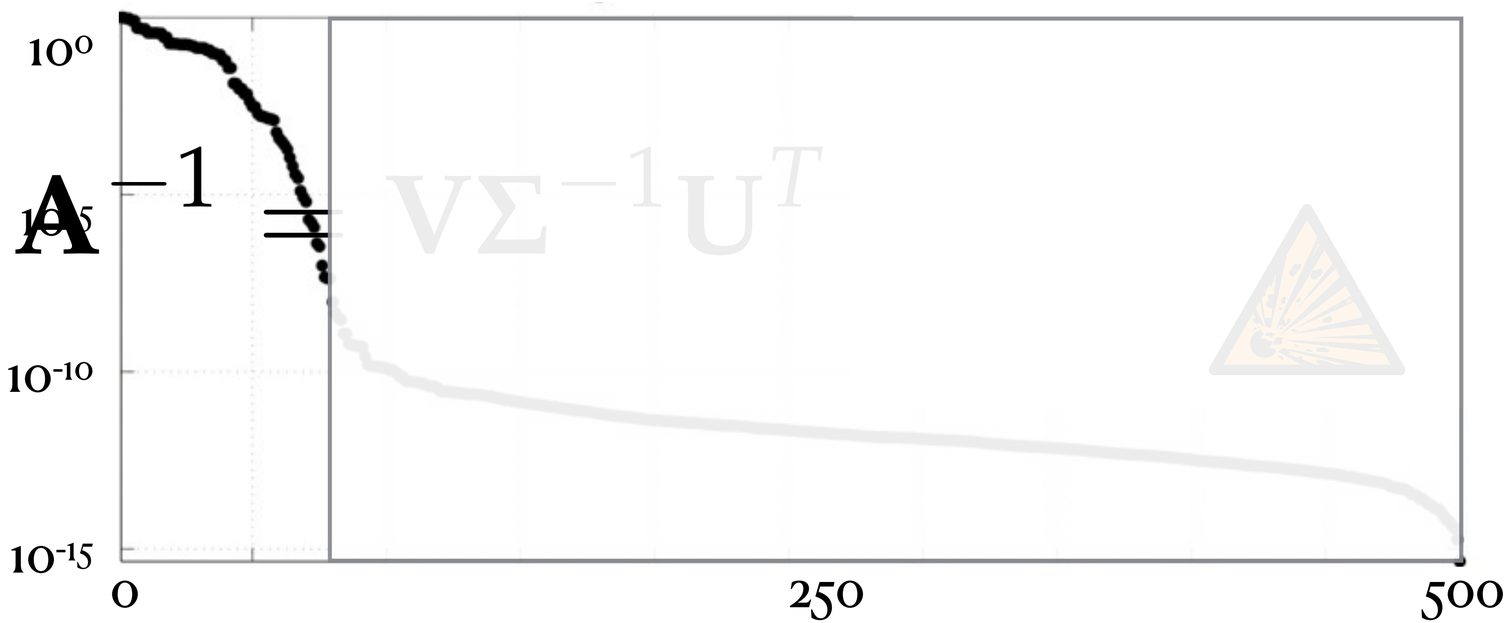
("Frobenius norm")

Demo time

Computing inverses via the SVD

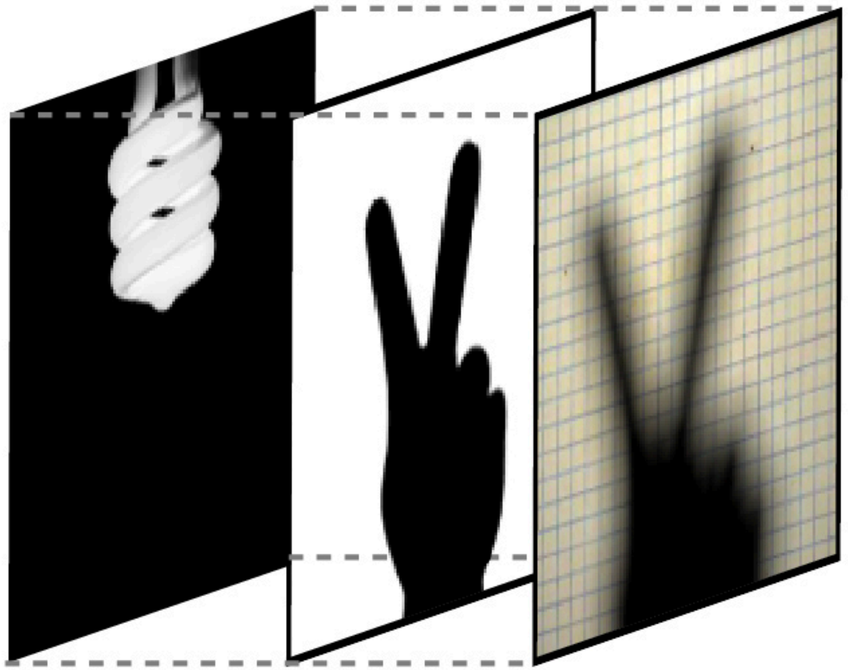
$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \approx \sum_{i=1}^{n_{\max}} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T$$

Plot of singular values (in decreasing magnitude)



$\mathbf{A}^{-1} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^T$

SVD as a regularization strategy



The *SVD-powered X-ray glasses*
(originally by Doug James @ Stanford U.)

fluorescent

Light sources

hand

hand

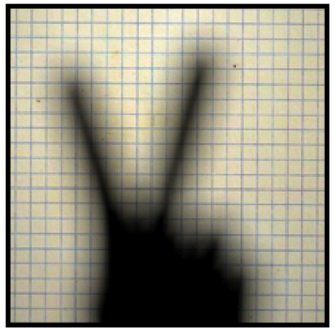
qmark

decoupage

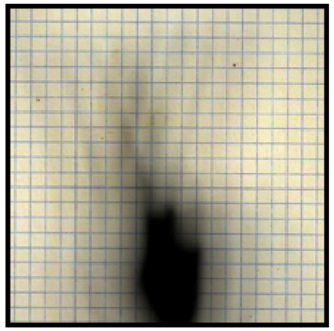
epfl

Blockers

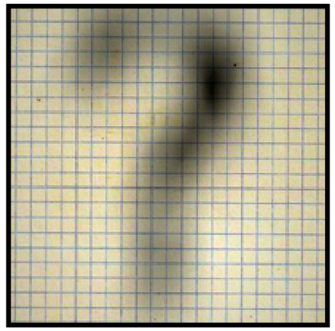
shadow1



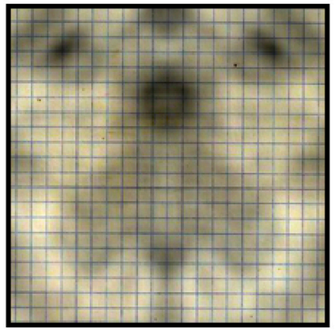
shadow2



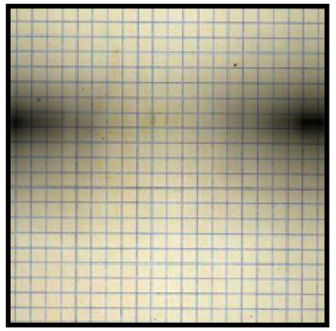
shadow3



shadow4

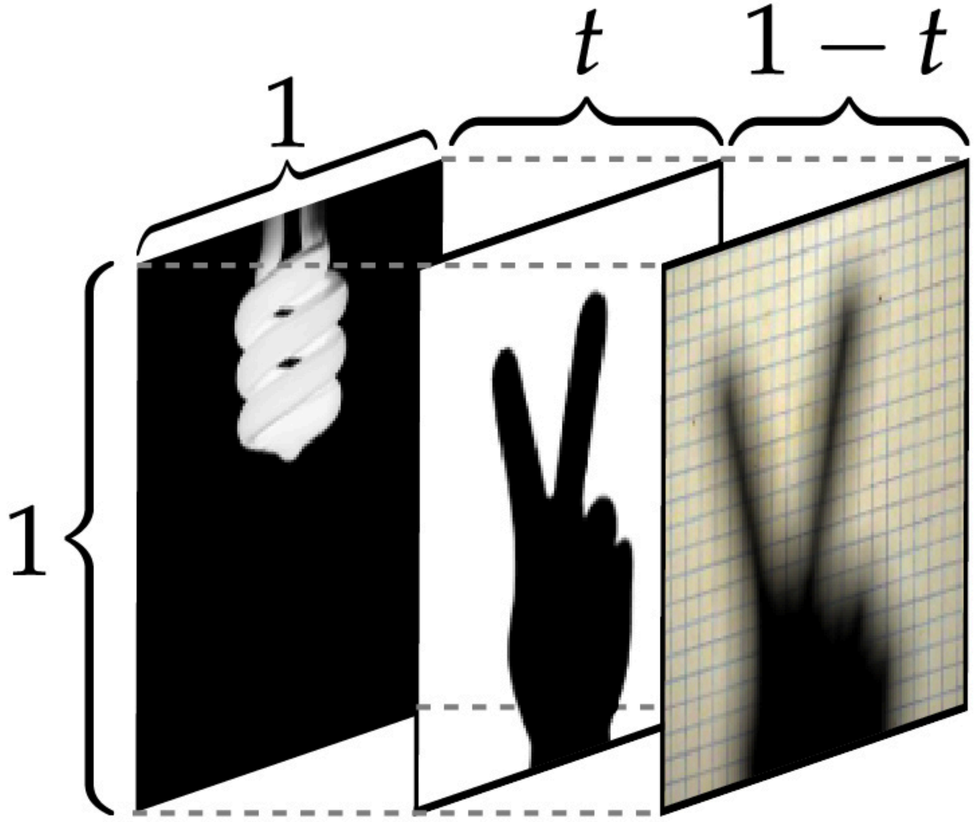


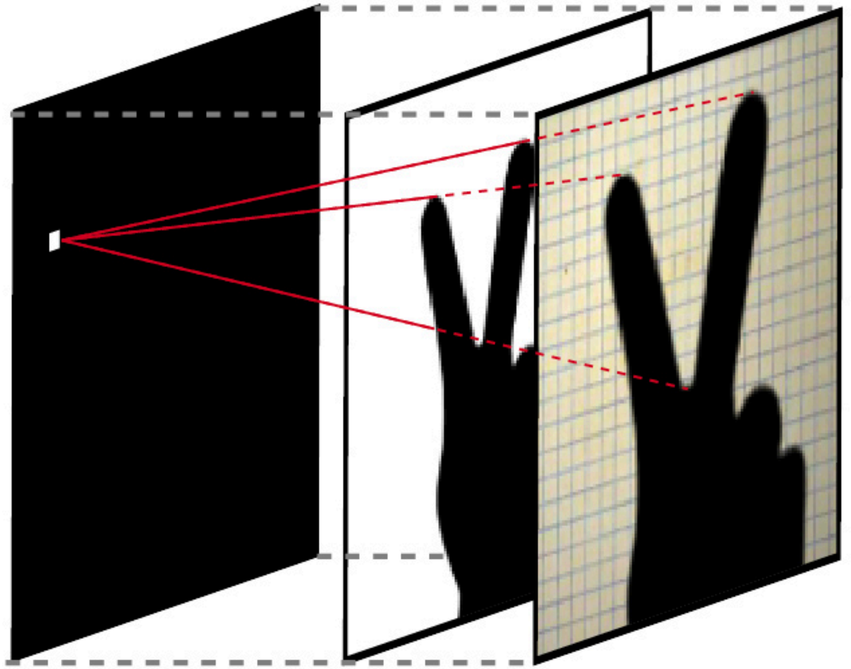
shadow5

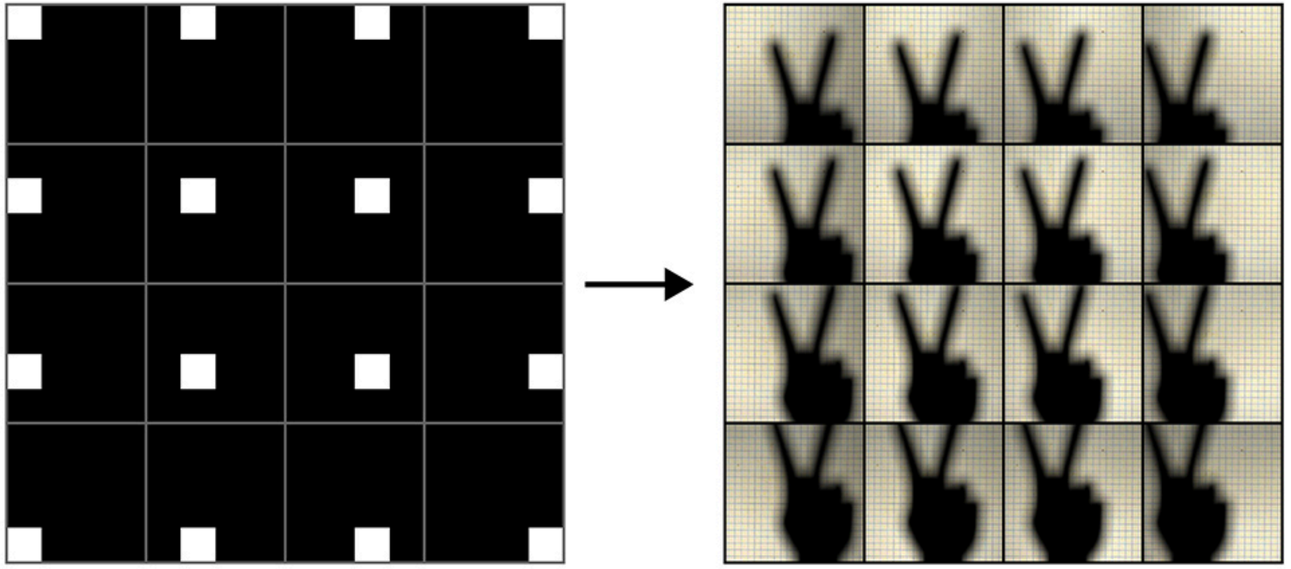


Shadow

Problem setup







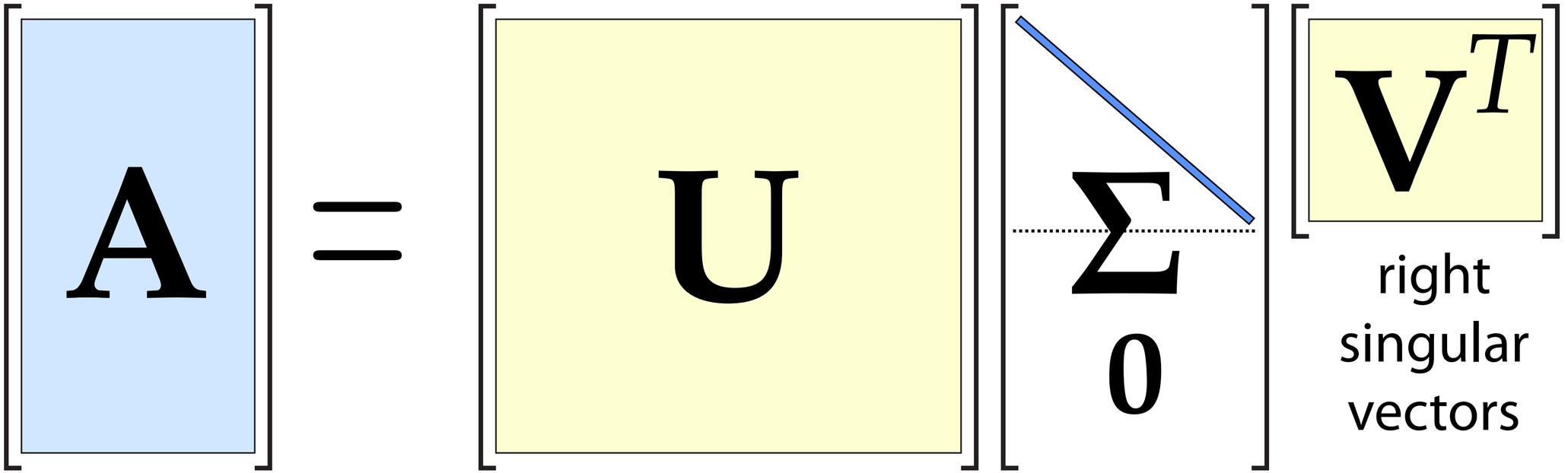
Demo time

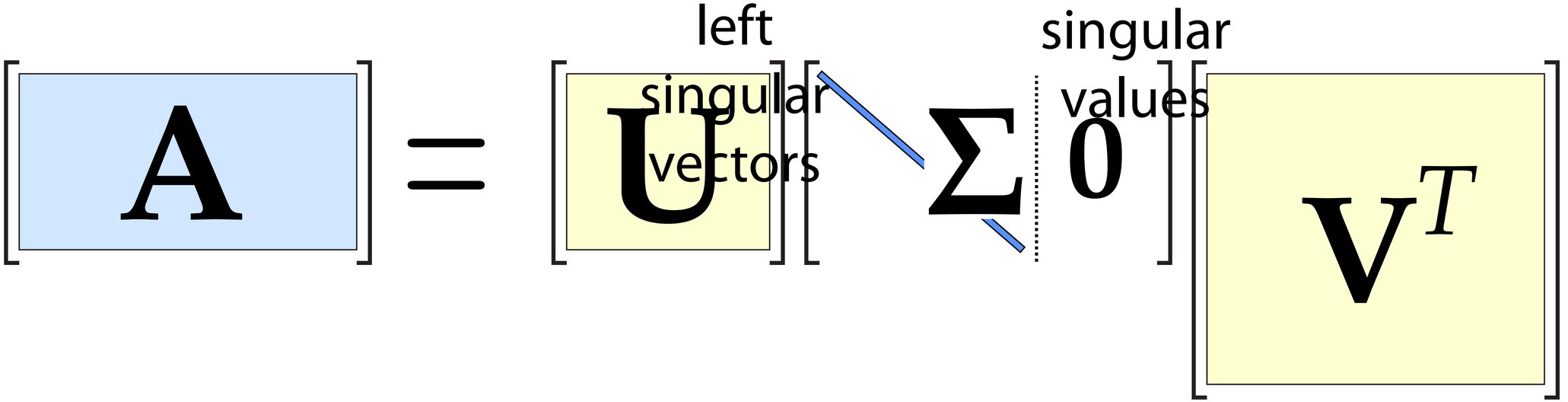
Revisiting the condition number

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$

$$= \frac{\sigma_1}{\sigma_n}.$$

SVD shape (tall & wide case)





The Pseudoinverse

SVD provides the "ultimate" form of a matrix inverse

$$\mathbf{A}^{+} = \sum_{i=1}^{\min\{n,m\}} \mathbf{v}_i \mathbf{u}_i^T \begin{cases} \frac{1}{\sigma_i}, & \sigma_i \neq 0 \\ 0, & \text{otherwise.} \end{cases}$$

1. Identical to inverse for square & full-rank matrices.
2. Gives least-squares solution for overconstrained / tall linear systems
3. Gives minimum-norm solution for underconstrained /wide linear systems

For homework: Newton's method N dimensions

(Will go in more detail in next week's lecure)

1D case:

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}$$

N-D case:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \left[\nabla \mathbf{f}(\mathbf{x}_{k-1})\right]^{-1} \mathbf{f}(\mathbf{x}_{k-1})$$