Big Data Analysis Application and Practice (XAI605)

PCA & MDS

2023 Spring

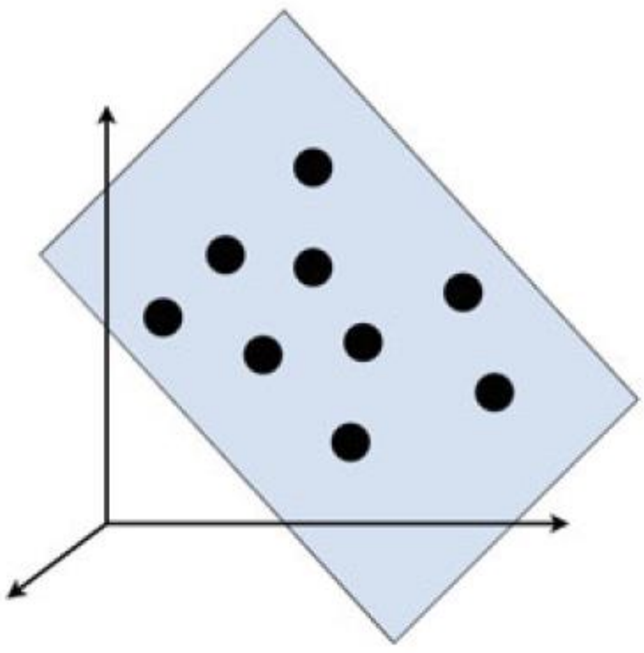
Instructor: Sejun Park

Geometric embedding

- **Goal:** embed given data to Euclidean space
  - PCA (Principal component analysis)
    - High-dimensional data to Low-dimensional affine space
  - MDS (Multidimensional scaling)
    - Distance between data to Euclidean space

Principal component analysis

- **Goal:** find a best affine approximation of data $x_1, \ldots, x_n \in \mathbb{R}^p$
  - What is the best approximation?
  - i.e., measure of quality of approximation?



Principal component analysis

- **Mathematical objective**

$$\min_{\beta,\mu,U} L := \sum_{i=1}^{n} \|x_i - (\mu + U\beta_i)\|^2$$

- Data: $X = [x_1, \ldots, x_n] \in \mathbb{R}^{p \times n}$
- Low-dim. representation: .$\beta = [\beta_1, \ldots, \beta_n] \in \mathbb{R}^{n \times k}$, $\sum_{i=1}^{n} \beta_i = 0$
- Bias: $\mu \in \mathbb{R}^p$
- Low-dim. basis: $U = [u_1, \ldots, u_k] \in \mathbb{R}^{p \times k}$
- Our basis is orthonormal: $\|u_i\| = 1$, $u_i^\top u_j = 0$ for all $i \neq j$ (or $U^\top U = I$)

Principal component analysis

- **Finding optimal solution**
  - First-order optimality condition under fixed basis (U)

$$\min_{\beta,\mu,U} L := \sum_{i=1}^{n} \|x_i - (\mu + U\beta_i)\|^2$$

$$\frac{\partial L}{\partial \mu} = -2\sum_{i=1}^{n}(x_i - \mu - U\beta_i) = 0 \iff \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\frac{\partial L}{\partial \beta_i} = (x_i - \mu - U\beta_i)^\top U = 0 \iff \beta_i = U^\top (x_i - \mu)$$

Principal component analysis

- **Finding optimal solution**
  - Reformulate the objective

$$
\begin{aligned}
L &= \sum_{i=1}^{n} \|x_i - (\hat{\mu} + U\beta_i)\|^2 \\
&= \sum_{i=1}^{n} \|x_i - \hat{\mu} - UU^\top (x_i - \mu)\|^2 \\
&= \sum_{i=1}^{n} \|y_i - UU^\top y_i\|^2 \quad (y_i := x_i - \hat{\mu})
\end{aligned}
$$

Principal component analysis

- **Finding optimal solution**
  - $Y = [y_1, \dots, y_n]$

$$
\begin{aligned}
&\arg\min_U \sum_{i=1}^n \|y_i - UU^\top y_i\|^2 \\
&= \arg\min_U \operatorname{Tr}\big((Y - UU^\top Y)^\top (Y - UU^\top Y)\big) \\
&= \arg\min \operatorname{Tr}\big(Y^\top (I - UU^\top)^\top (I - UU^\top) Y\big) \\
&= \arg\min \operatorname{Tr}\big(Y^\top (I - UU^\top)^2 Y\big) \\
&= \arg\min \operatorname{Tr}\big(Y^\top (I - UU^\top) Y\big) \\
&= \arg\min \operatorname{Tr}\big(YY^\top (I - UU^\top)\big) \\
&= \arg\max \operatorname{Tr}\big(YY^\top UU^\top\big) \\
&= \arg\max \operatorname{Tr}\big(U^\top YY^\top U\big)
\end{aligned}
$$

Principal component analysis

- **Finding optimal solution**

$$\arg\min_U \sum_{i=1}^{n} \|y_i - UU^\top y_i\|^2 = \arg\max \operatorname{Tr}\big(U^\top Y Y^\top U\big)$$

$$= \arg\max \operatorname{Tr}\big(U^\top \hat{\Sigma} U\big)$$

$$= \arg\max \sum_{i=1}^{k} u_i^\top \hat{\Sigma} u_i$$

$$\hat{\Sigma} := \frac{1}{n} Y Y^\top = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})(x_i - \hat{\mu})^\top$$

or $\displaystyle \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \hat{\mu})(x_i - \hat{\mu})^\top$

Principal component analysis

- **Finding optimal solution**
  - Top-k eigenvectors of the covariance matrix are the solution

$$\arg\min_{U} \sum_{i=1}^{n} \|y_i - UU^\top y_i\|^2 = \arg\max_{u_1,\ldots,u_k} \sum_{i=1}^{k} u_i^\top \hat{\Sigma} u_i$$

Principal component analysis

- **Finding optimal solution**
  - Top-k eigenvectors of the covariance matrix are the solution

$$\arg\min_{U} \sum_{i=1}^{n} \|y_i - UU^\top y_i\|^2 = \arg\max_{u_1,\ldots,u_k} \sum_{i=1}^{k} u_i^\top \hat{\Sigma} u_i$$

  - Or equivalently, we can choose top-k left singular vector of Y

$$\hat{\Sigma} := \frac{1}{n} YY^\top = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})(x_i - \hat{\mu})^\top$$

$$Y = \hat{U}\hat{S}\hat{V}^\top$$

Principal component analysis

- **Finding optimal solution**
  - Optimal low-dimensional representation

$$\beta_i = \hat{U}_k^\top y_i$$

$$\beta = \hat{U}_k^\top Y = \hat{S}_k \hat{V}_k^\top$$

Principal component analysis

- **SVD-based algorithm**
  - Compute $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and compute $Y = [x_1 - \hat{\mu}, \ldots, x_n - \hat{\mu}]$
  - For $q = \min\{p, n\}$, compute $\hat{U} \in \mathbb{R}^{p \times q}, \hat{S} \in \mathbb{R}^{q \times q}, \hat{V} \in \mathbb{R}^{n \times q}$ using SVD so that $Y = \hat{U}\hat{S}\hat{V}^\top$
  - Choose entries corresponding to top-k singular values: $\hat{U}_k \in \mathbb{R}^{p \times k}$, $\hat{S}_k \in \mathbb{R}^{k \times k}, \hat{V}_k \in \mathbb{R}^{n \times k}$
  - Return $\hat{\mu}, U \leftarrow \hat{U}_k, \beta \leftarrow U^\top Y$

Principal component analysis

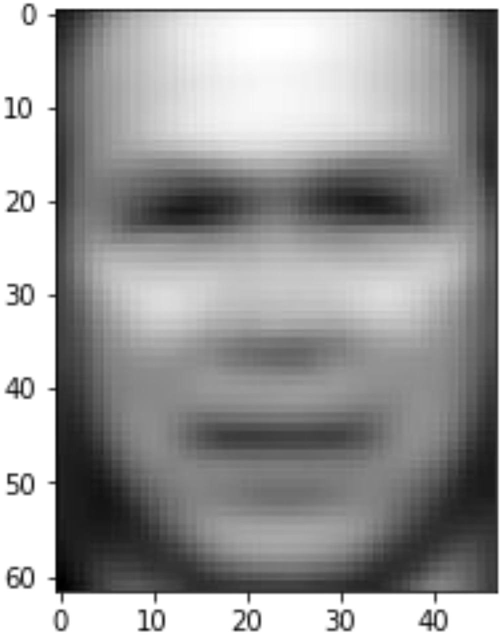
- **SVD-based algorithm**
  - Compute $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and compute $Y = [x_1 - \hat{\mu}, \ldots, x_n - \hat{\mu}]$
  - For $q = \min\{p, n\}$, compute $\hat{U} \in \mathbb{R}^{p\times q}, \hat{S} \in \mathbb{R}^{q\times q}, \hat{V} \in \mathbb{R}^{n\times q}$ using SVD so that $Y = \hat{U}\hat{S}\hat{V}^\top$
  - Choose entries corresponding to top-k singular values: $\hat{U}_k \in \mathbb{R}^{p\times k}$, $\hat{S}_k \in \mathbb{R}^{k\times k}, \hat{V}_k \in \mathbb{R}^{n\times k}$
  - Return $\hat{\mu}, U \leftarrow \hat{U}_k, \beta \leftarrow U^\top Y$

- We will later discuss improving the running time of PCA by approximating SVD using the power iteration

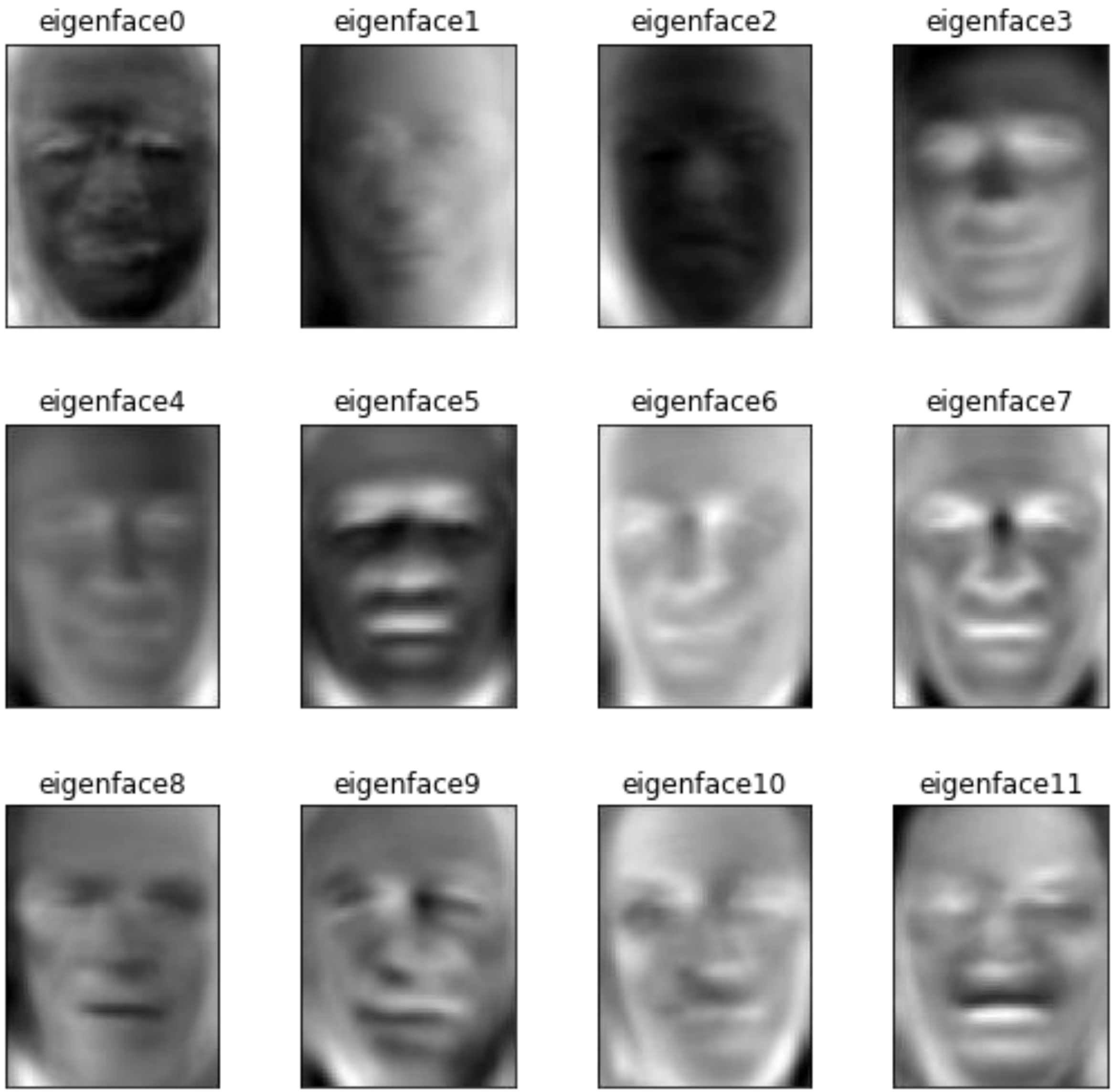
Example: eigenfaces (p = 2914)

Example: eigenfaces

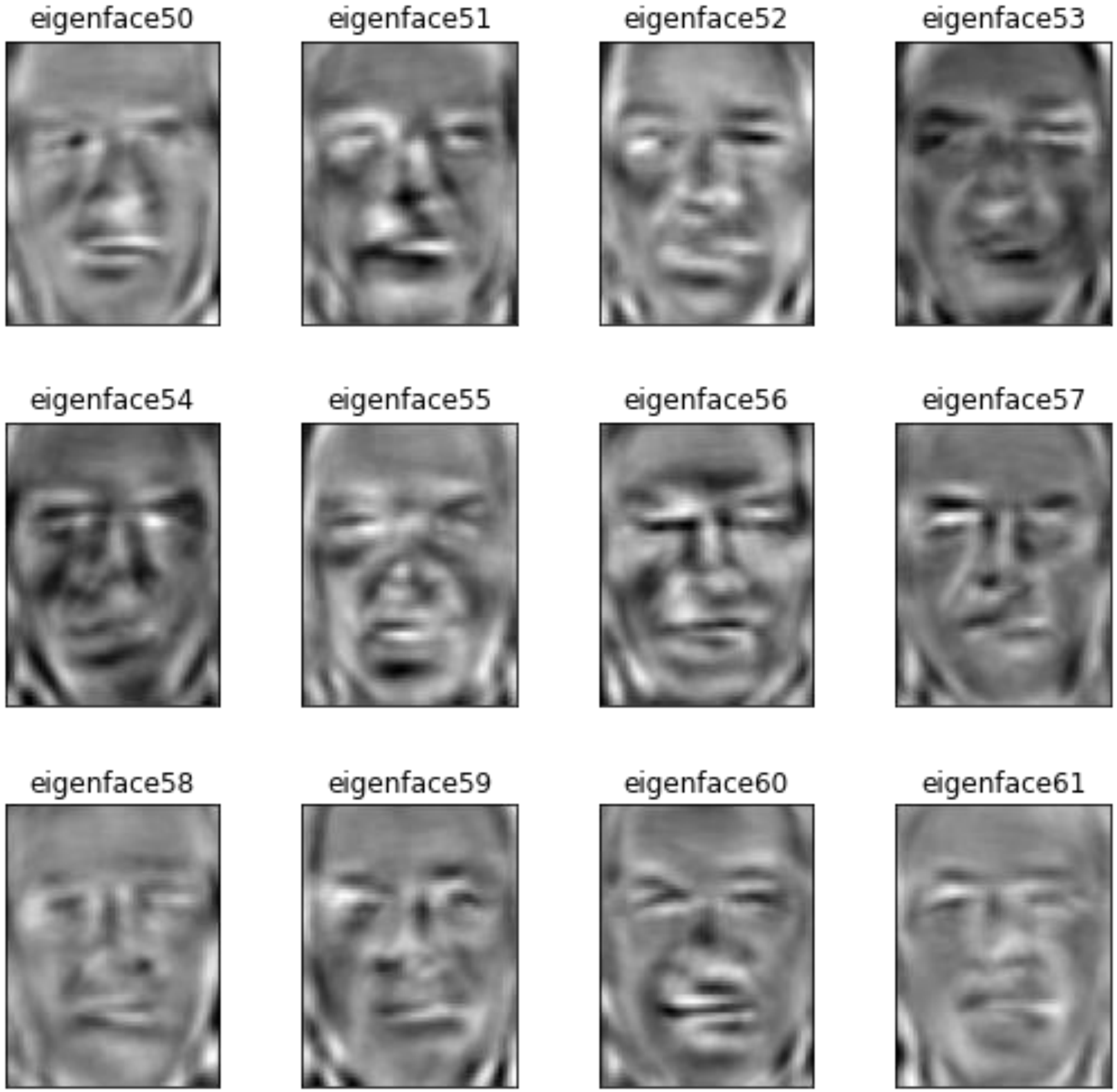
- **Average face**
  - Mu in PCA



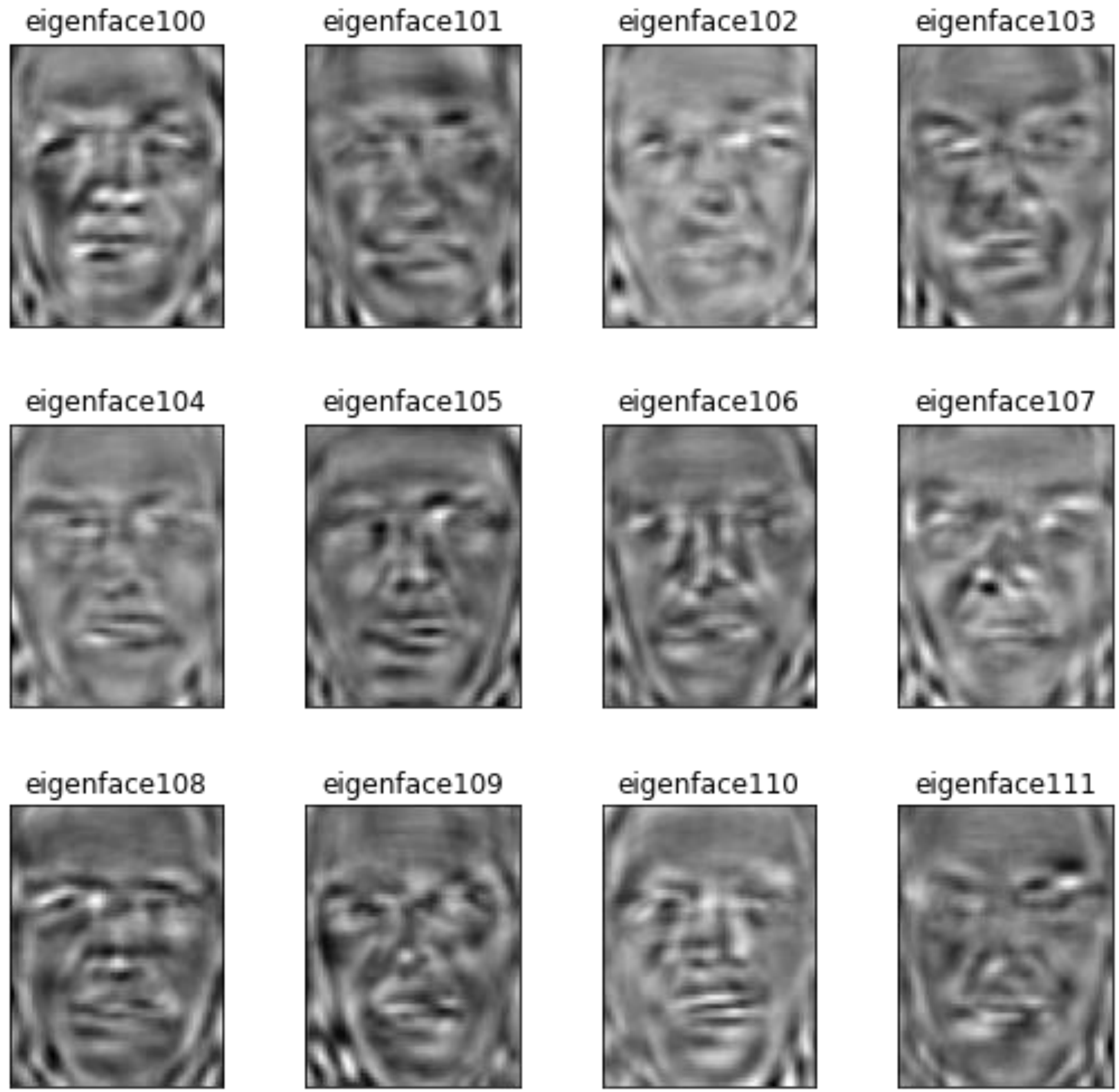
Example: eigenfaces



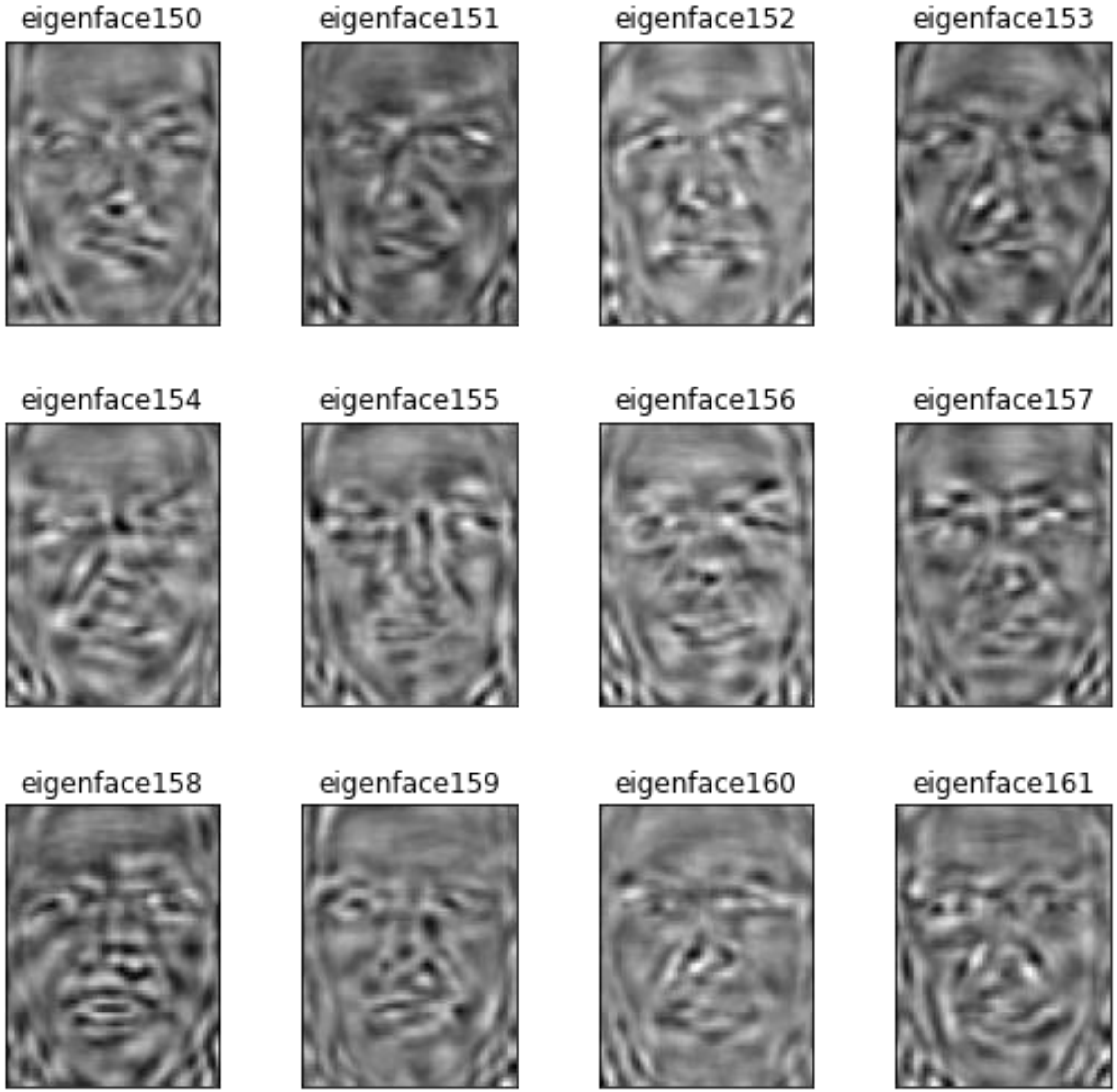
Example: eigenfaces



Example: eigenfaces



Example: eigenfaces



Example: reconstruction (k = 50)

Example: reconstruction (k = 100)

Example: reconstruction (k = 150)

Example: reconstruction (k = 200)

Example: original images

Principal component analysis

- **Approximation loss incurred by PCA**

$$\begin{aligned}
\sum_{i=1}^{n} \|y_i - UU^\top y_i\|^2 &= \mathrm{Tr}\big(YY^\top (I - UU^\top)\big) \\
&= \mathrm{Tr}\big(YY^\top\big) - (U^\top YY^\top U)\big) \\
&= n\sum_{i=1}^{p} \lambda_i - n\sum_{i=1}^{k} \lambda_i \\
&= n\sum_{i=k+1}^{p} \lambda_i
\end{aligned}$$

Principal component analysis

- **Variance explained by PCA**
  - Total variation is an informal measure of “spread” of data

$$\text{Total variation: } \mathrm{Tr}(\hat{\Sigma}) = \sum_{i=1}^{p} \hat{\lambda}_i$$

$$\text{Variance explained by PCA: } \frac{\sum_{i=1}^{k} \hat{\lambda}_i}{\mathrm{Tr}(\hat{\Sigma})}$$

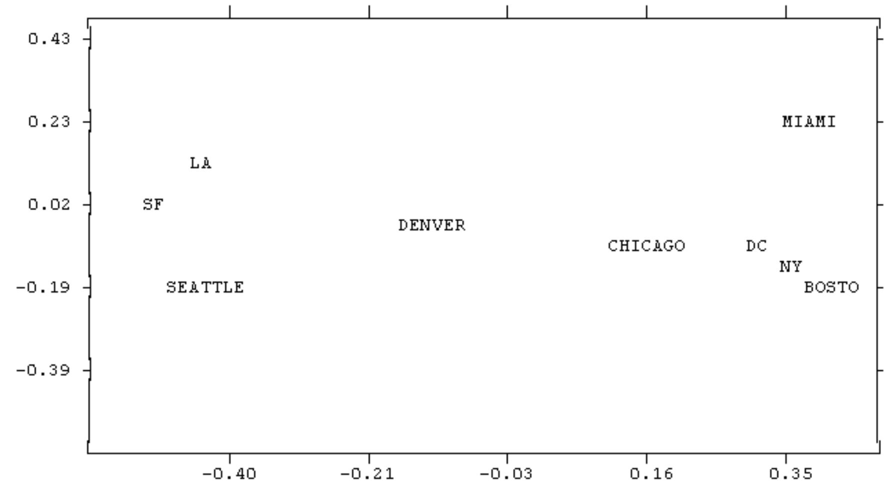
  - Given some threshold, we can choose #principal components satisfying

$$\frac{\sum_{i=1}^{k} \hat{\lambda}_i}{\mathrm{Tr}(\hat{\Sigma})} \geq \text{threshold}$$

Multidimensional scaling

- **Goal:** given distance between data, represent data in Euclidean space preserving the distance

	BOST	NY	DC	MIAM	CHIC	SEAT	SF	LA	DENV
BOSTON	0	206	429	1504	963	2976	3095	2979	1949
NY	206	0	233	1308	802	2815	2934	2786	1771
DC	429	233	0	1075	671	2684	2799	2631	1616
MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
CHICAGO	963	802	671	1329	0	2013	2142	2054	996
SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
SF	3095	2934	2799	3053	2142	808	0	379	1235
LA	2979	2786	2631	2687	2054	1131	379	0	1059
DENVER	1949	1771	1616	2037	996	1307	1235	1059	0



Multidimensional scaling

- **Goal:** given a squared distance matrix $D = [d_{ij}^2]$, find $X = [x_1, \ldots, x_n]$ satisfying

$$d_{ij}^2 = \|x_i - x_j\|^2 = x_i^\top x_i + x_j^\top x_j - 2x_i^\top x_j$$

- We first assume that a given matrix D is computed from **data in Euclidean space**

Multidimensional scaling

- **Key observation:** Suppose that X satisfies the below

$$H := I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$$
$$-\frac{1}{2}HDH = (XH)^\top (XH)$$

- Then, the squared distance between data represented by X matches with D

$$d_{ij}^2 = \|x_i - x_j\|^2 = x_i^\top x_i + x_j^\top x_j - 2x_i^\top x_j$$

Multidimensional scaling

- **Proof of key observation**
  - If $X = [x_1, \ldots, x_n]$ and $Z = [z_1, \ldots, z_n]$ satisfies $X^\top X = Z^\top Z$, then $\|x_i - x_j\|^2 = \|z_i - z_j\|^2$ for all i, j

$$X^\top X = [x_i^\top x_j]$$

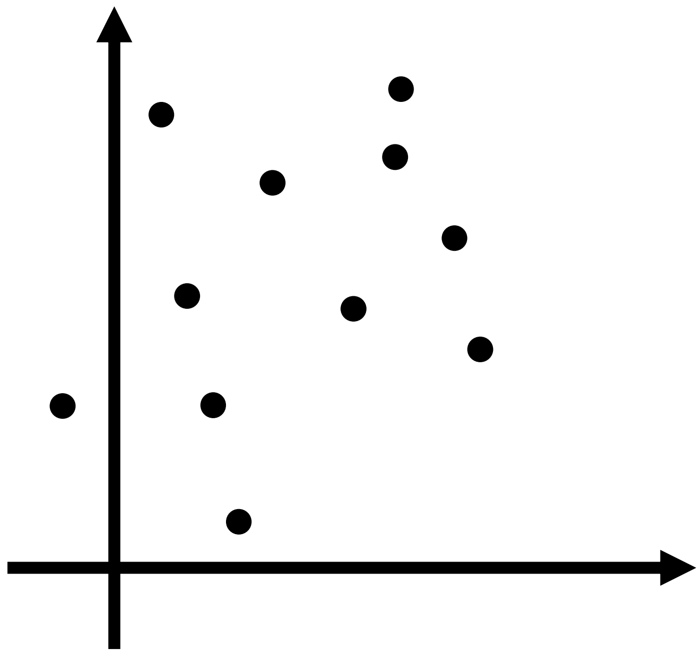
$$\|x_i - x_j\|^2 = x_i^\top x_i + x_j^\top x_j - 2x_i^\top x_j$$

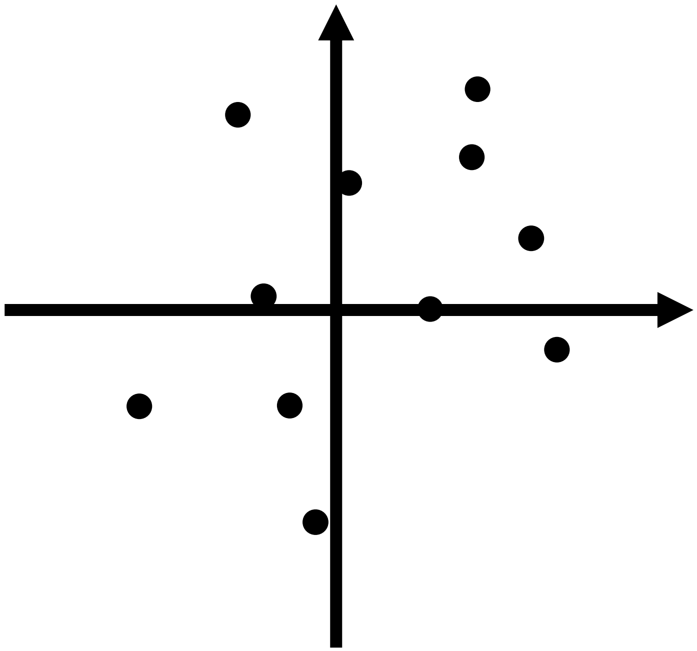
$$\begin{aligned} D_X :=& [\|x_i - x_j\|^2] \\ =& \mathrm{diag}(X^\top X)\mathbf{1}^\top + \mathbf{1}\mathrm{diag}(X^\top X)^\top - 2X^\top X \end{aligned}$$

Multidimensional scaling

- **Proof of key observation**
  - $D_X = D_{XH}$

$$XH = [x_1 - \mu, \ldots, x_n - \mu]$$





Multidimensional scaling

- **Proof of key observation**
  - Suppose that $-\frac{1}{2}HD_ZH := (XH)^\top (XH)$
  - Then $(ZH)^\top (ZH) = (XH)^\top (XH)$

$$\left(\mathrm{diag}(Z^\top Z)\mathbf{1}^\top\right) H = 0$$
$$H\left(\mathbf{1}\mathrm{diag}(Z^\top Z)^\top\right) = 0$$

Multidimensional scaling

- **Proof of key observation**
  - Suppose that $-\frac{1}{2}HD_ZH := (XH)^\top(XH)$
  - Then $(ZH)^\top(ZH) = (XH)^\top(XH)$

$$
\begin{aligned}
-\frac{1}{2}HD_ZH &= -\frac{1}{2}H\left(\mathrm{diag}(Z^\top Z)\mathbf{1}^\top + \mathbf{1}\mathrm{diag}(Z^\top Z)^\top - 2Z^\top Z\right)H \\
&= HZ^\top ZH \\
&= (ZH)^\top(ZH)
\end{aligned}
$$

Multidimensional scaling

- **Key observation:** Suppose that X satisfies the below

$$H := I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$$
$$-\frac{1}{2}HDH = (XH)^\top(XH)$$

- Then, the squared distance between data represented by X matches with D

Multidimensional scaling

- **Remark**
  - Suppose that D is generated from data $Z = [z_1, \ldots, z_n]$
  - Then for MDS, we only require inner product, instead of the distance

$$-\frac{1}{2} HDH = HZ^\top ZH$$

  - In kernel MDS, we use inner product; we will see this later

- **Remark2**
  - For the exact representation, we need n dimension for an n x n matrix D

Multidimensional scaling

- **Eigendecomposition-based algorithm**
  - Compute $\hat{K} = -\frac{1}{2} HDH$
  - Compute the Eigendecomposition $\hat{K} = \hat{V}\hat{\Lambda}\hat{V}^\top$
  - Compute $\hat{\Lambda}_k \in \mathbb{R}^{k \times k}$, $\hat{V}_k \in \mathbb{R}^{n \times k}$ corresponding to the top-k eigenvalues
  - Return $\hat{\Lambda}_k^{1/2} \hat{V}_k^\top$

Duality of PCA and MDS

- **PCA solution = MDS solution**
  - If D is from data (say Z) from Euclidean space

- Low-dimensional representation of PCA satisfies
  - $Y = XH = \hat{U}\hat{S}\hat{V}^\top$
  - $\hat{U}_k$: chosen principal components (left singular vectors)

$$\beta = \hat{U}_k^\top Y = \hat{S}_k \hat{V}_k^\top$$

- MDS solution is given by right singular vectors

$$X = \hat{S}\hat{V}^\top$$

Metric

- d is a metric if it satisfies the following conditions
  - $d(x, x) = 0 \quad \forall x$
  - $d(x, y) > 0 \quad \forall x \neq y$
  - $d(x, y) = d(y, x)$
  - $d(x, z) = d(y, x) + d(y, z)$

Non-metric MDS

- We can also consider the setup: D is not exactly the squared distance matrix
  - e.g., some measurement noise can be added
  - We may use other notion of distance that is not a metric

$$\arg\min_X \sum_{i,j} \left(\|x_i - x_j\|^2 - d_{ij}^2 \right)$$

- We will study non-metric MDS later

Condition for Euclidean embedding

- What if D is from data in Euclidean space
- Can we check whether D can be represented by Euclidean data or not?

	BOST	NY	DC	MIAM	CHIC	SEAT	SF	LA	DENV
BOSTON	0	206	429	1504	963	2976	3095	2979	1949
NY	206	0	233	1308	802	2815	2934	2786	1771
DC	429	233	0	1075	671	2684	2799	2631	1616
MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
CHICAGO	963	802	671	1329	0	2013	2142	2054	996
SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
SF	3095	2934	2799	3053	2142	808	0	379	1235
LA	2979	2786	2631	2687	2054	1131	379	0	1059
DENVER	1949	1771	1616	2037	996	1307	1235	1059	0

Condition for Euclidean embedding

- What if D is from data in Euclidean space
- Can we check whether D can be represented by Euclidean data or not?

Theorem D can be represented by Euclidean data if and only if $-HDH$ is positive semi-definite

- $M \in \mathbb{R}^{n \times n}$ is positive semi-definite if $x^\top M x \geq 0 \; \forall x \in \mathbb{R}^n$

Singular value decomposition

- A Typical singular value decomposition algorithm of an m x n matrix requires $O(mn^2)$ flops
  - This is based on variants of QR decomposition and linear algebra techniques

- This computes all left/right singular vectors and all singular values

Power iteration

- An iterative algorithm for finding the largest eigenvalue & corresponding eigenvector

- We can use this for efficiently computing top-k singular values/vectors

$$M = USV^\top \implies M^\top M = V(S^\top S)V^\top, MM^\top = U(SS^\top)U^\top$$

Power iteration

- Power iteration for finding the largest eigenvalue & corresponding eigenvector of $A \in \mathbb{R}^{n \times n}$
  - Initialize $b^{(0)} \in \mathbb{R}^n$ (you can choose any)
  - Repeat $b^{(t+1)} \leftarrow Ab^{(t)} / \|Ab^{(t)}\|$, $t \leftarrow t + 1$ until $\|b^{(t)} - b^{(t-1)}\| < \varepsilon$
    - ε is typically choosen as a very small constant (e.g., $\varepsilon = 10^{-6}$)
  - Largest eigenvalue: $Ab^{(t)} / \|Ab^{(t)}\|$, corresponding eigenvector: $b^{(t)}$

Power iteration

- Power iteration for finding the second largest eigenvalue & corresponding eigenvector of $A \in \mathbb{R}^{n \times n}$
  - Find the largest eigenvalue/vector λ_1, b_1 of A using the power iteration
  - $A_2 \leftarrow A - \lambda_1 b_1 b_1^\top$
  - Find the largest eigenvalue/vector λ_2, b_2 of A_2 using the power iteration
  - λ_2, b_2 are the second largest eigenvalue/vector of A

- We can generalize this to find top-k eigenvalues/vectors

Power iteration

- **Complexity for finding the largest eigenvalue/vector**
  - Required #iterations$= O(\frac{\log(n/\varepsilon)}{\log(\lambda_1/\lambda_2)})$
  - Each iteration requires $O(n^2)$ FLOPS (matrix-vector product)
  - Total complexity$= O(\frac{n^2\log(n/\varepsilon)}{\log(\lambda_1/\lambda_2)})$

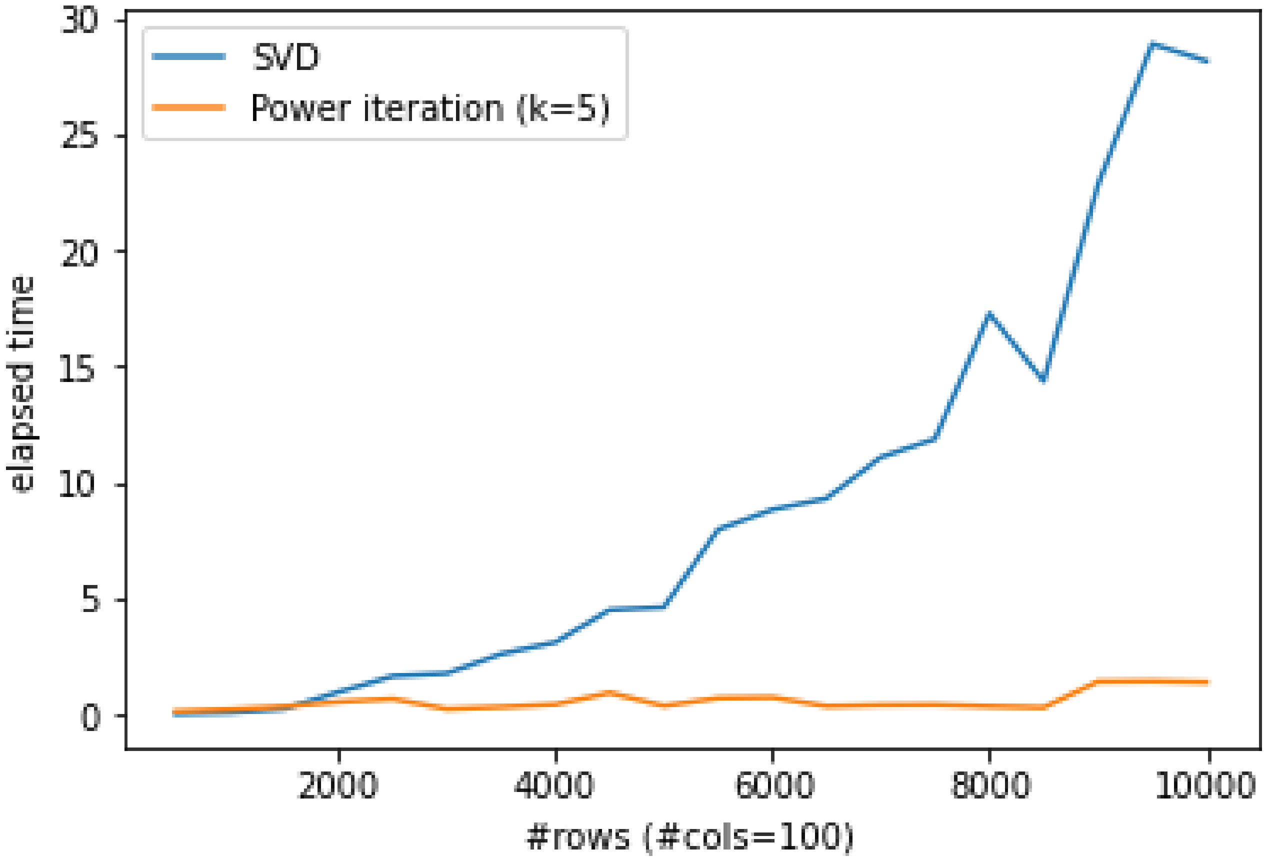
Power iteration

- **Complexity for finding the largest eigenvalue/vector**
  - Required #iterations$= O(\frac{\log(n/\varepsilon)}{\log(\lambda_1/\lambda_2)})$
  - Each iteration requires $O(n^2)$ FLOPS (matrix-vector product)
  - Total complexity$= O(\frac{n^2 \log(n/\varepsilon)}{\log(\lambda_1/\lambda_2)})$

- For SVD, we additionally require a matrix-matrix product

$$M = USV^\top \implies M^\top M = V(S^\top S)V^\top, MM^\top = U(SS^\top)U^\top$$

Power iteration vs. SVD



Complexity of matrix multiplication

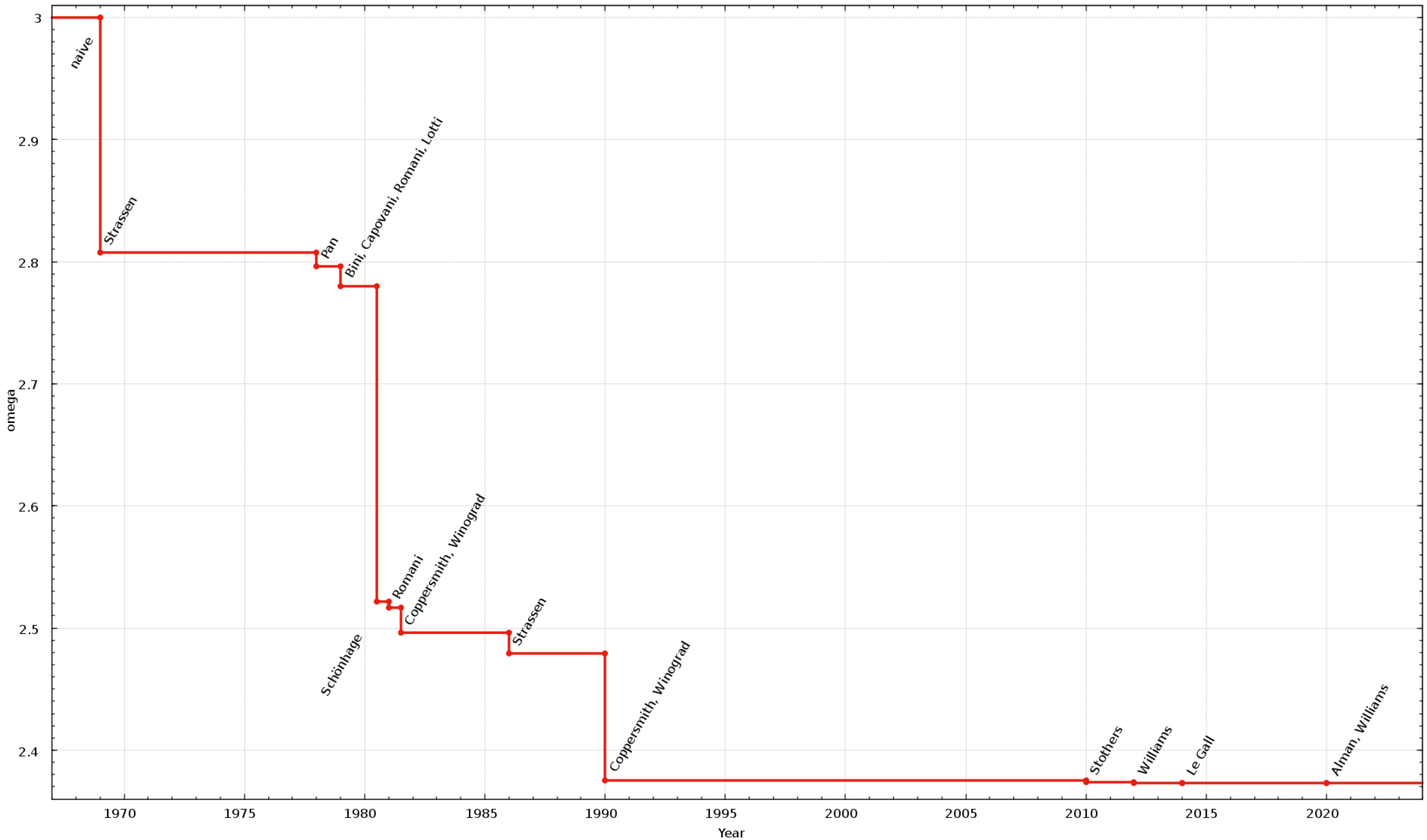
- Naïve multiplication of an n x m matrix and an m x n matrix requires $O(mn^2)$ FLOPS
  - $O(n^3)$ if $m = n$

Complexity of matrix multiplication

- Naïve multiplication of an n x m matrix and an m x n matrix requires $O(mn^2)$ FLOPS
  - $O(n^3)$ if $m = n$
- What happens in theory?

Rather surprisingly, this complexity is not optimal, as shown in 1969 by Volker Strassen, who provided an algorithm, now called Strassen's algorithm, with a complexity of $O(n^{\log_2 7}) \approx O(n^{2.8074})$.[14] Strassen's algorithm can be parallelized to further improve the performance.[*citation needed*] As of December 2020, the best matrix multiplication algorithm is by Josh Alman and Virginia Vassilevska Williams and has complexity $O(n^{2.3728596})$.[15] It is not known whether matrix multiplication can be performed in $n^{2 + o(1)}$ time. This would be optimal, since one must read the n^2 elements of a matrix in order to multiply it with another matrix.

Complexity of matrix multiplication



Complexity of matrix multiplication

k	upper bound on $\omega(k)$
0.30298	2
0.31	2.000063
0.32	2.000371
0.33	2.000939
0.34	2.001771
0.35	2.002870
0.40	2.012175
0.45	2.027102
0.50	2.046681
0.5302	2.060396
0.55	2.070063
0.60	2.096571
0.65	2.125676
0.70	2.156959
0.75	2.190087
0.80	2.224790

k	upper bound on $\omega(k)$
0.85	2.260830
0.90	2.298048
0.95	2.336306
1.00	2.375477
1.10	2.456151
1.20	2.539392
1.30	2.624703
1.40	2.711707
1.50	2.800116
1.75	3.025906
2.00	3.256689
2.50	3.727808
3.00	4.207372
4.00	5.180715
5.00	6.166736

Table 2: Upper bounds from [17] on the exponent of the multiplication of an $n \times n^k$ matrix by an $n^k \times n$ matrix, obtained by analyzing the second power of the Coppersmith-Winograd tensor.

Complexity of matrix multiplication

- **Initial idea for decreasing matrix multiplication cost**
  - Consider the square matrix multiplication, i.e., m=n
  - Suppose that $n = 2^k$

The Strassen algorithm partitions A, B and C into equally sized block matrices

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix},$$

$$(A, B, C \in \mathbb{R}^{2^k \times 2^k}, A_{ij}, B_{ij}, C_{ij} \in \mathbb{R}^{2^{k-1} \times 2^{k-1}})$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Complexity of matrix multiplication

- **Initial idea for decreasing matrix multiplication cost**
  - Consider the square matrix multiplication, i.e., m=n
  - Suppose that $n = 2^k$

$$
\begin{aligned}
&\text{Complexity of } 2^k \times 2^k \text{ matrix multiplication} \\
&= O(8 \times \text{Complexity of } 2^{k-1} \times 2^{k-1} \text{ matrix multiplication}) \\
&= O(8^k) = O(2^{3k}) = O(n^3)
\end{aligned}
$$

Complexity of matrix multiplication

The Strassen algorithm defines instead new matrices:

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22});\\
M_2 &= (A_{21} + A_{22})B_{11};\\
M_3 &= A_{11}(B_{12} - B_{22});\\
M_4 &= A_{22}(B_{21} - B_{11});\\
M_5 &= (A_{11} + A_{12})B_{22};\\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12});\\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}),
\end{aligned}$$

using only 7 multiplications (one for each M_k) instead of 8.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}.$$

Complexity of matrix multiplication

Complexity of $2^k \times 2^k$ matrix multiplication

$= O(7 \times \text{Complexity of } 2^{k-1} \times 2^{k-1} \text{ matrix multiplication})$

$= O(7^k) = O(2^{k \log_2 7}) = O(n^{\log_2 7})$

$= O(n^{2.8073549\ldots})$

For practice session

- You should bring your laptop!!!
- You will use google colab in practice session
  - I will provide you a jupyter notebook file with some empty entries
  - And you will fill those entries in the next week