Numerisk Lineær Algebra F2021
Notesæt 27

Andrew Swann

10. maj 2021

Sidst ændret: 10. maj 2021.
Versionskode: f68094e.

Indhold

Figurer

Tabeller

27 Singulærværdi beregning og principalkomponent analyse

Vi har brugt nogle kræfter på at vise hvordan egenværdier og egenvektorer for symmetriske matricer kan beregnes. Dette er et interessant problem i sig selv, men den er også relevant for beregning af singulærværdier, som vil vise i dette kapitel. Vil vi også skitsere hvordan singulærværdidekomponering giver en god måde at lave principalkomponent analyse, en ofte brugt værktøj i dataanalyse.

27.1 Singulærværdidekomponering og symmetriske matricer

Det er (mindst) tre måder man kan danne en ny symmetrisk matrix fra en vilkårlig matrix $A \in \mathbb{R}^{m\times n}$:

(a) $A^TA \in \mathbb{R}^{n\times n}$,

(b) $AA^T \in \mathbb{R}^{m\times m}$,

(c) $\begin{bmatrix} 0_{n\times n} & A^T \\ A & 0_{m\times m} \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$.

Hvis man bruger QR-metoden til at beregne egenværdier for disse tre symmetriske matricer, vil det koster hhv. (a) $\sim 4n^3/3$ flops, (b) $\sim 4m^3/3$ flops og (c) $\sim 4(n+m)^3/3$ flops, Da $4(n+m)^3/3 > (4n^3/3) + (4m^3/3)$, virker den sidste metode udmiddelbart til at være ret bekostelig.

Eksempel 27.1. Lad os kikke på matricen

$$A = \begin{bmatrix} 1 & -1 \\ 2 & 1 \\ 4 & 1 \\ 3 & -1 \end{bmatrix}.$$

Vi har

(a) $A^TA = \begin{bmatrix} 30 & 2 \\ 2 & 4 \end{bmatrix}$,

(b) $AA^T = \begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & 5 & 9 & 5 \\ 3 & 9 & 17 & 11 \\ 4 & 5 & 11 & 10 \end{bmatrix}$ og

(c) $\begin{bmatrix} 0_{n\times n} & A^T \\ A & 0_{m\times m} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 2 & 4 & 3 \\ 0 & 0 & -1 & 1 & 1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 & 0 \\ 3 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$.

△

Hvad siger SVD om disse symmetriske matricer? Lad os skrive

$$A = U\Sigma V^T \in \mathbb{R}^{m\times n}$$

med $U \in \mathbb{R}^{m\times m}$ og $V \in \mathbb{R}^{n\times n}$ ortogonale, og $\Sigma = \text{diag}_{m\times n}(\sigma_0, \sigma_1, \ldots, \sigma_{k-1})$, hvor $k = \min\{m, n\}$.

For (a) har vi

$$A^TA = V\Sigma^TU^TU\Sigma V^T = V\Sigma^T\Sigma V^T = VD_nV^T$$

hvor

$$D_n = \text{diag}(\sigma_0^2, \sigma_1^2, \ldots, \sigma_{k-1}^2, 0, \ldots, 0) \in \mathbb{R}^{n\times n}.$$

Tilsvarende for (b) er

$$AA^T = U\Sigma V^TV\Sigma^TU^T = U\Sigma\Sigma^TU^T = UD_mU^T$$

med $D_m \in \mathbb{R}^{m\times m}$, som ovenfor. Vi ser at i begge tilfælder giver egenværdierne af den symmetriske matrix singulærværdierne for A i anden potens. Da singulærværdier er positiver, er de entydigt bestemt af disse egenværdier. Det betyder at vi kunne beregne singulærværdier af A ved at forme enten A^TA eller AA^T og bruge metoden fra det forrige kapitel til at beregne egenværdier af den symmetriske matrix.

Der er dog flere problemer med at beregne singulærværdier af via egenværdierne af A^TA (eller AA^T). De stammer fra tre kilder:

(i) QR-metoden beregner egenværdier af A^TA indenfor $\|A^TA\|_2\epsilon_{\text{machine}} = \sigma_0^2\epsilon_{\text{machine}}$,
(ii) for $m \geqslant n$ har A^TA har konditionstal $\kappa(A^TA) = \sigma_0^2/\sigma_{n-1}^2 = \kappa(A)^2$,
(iii) selve float beregning af A^TA kan have betydningsfulde fejl.

Punkterne (i) og (ii) peger begge i retning af en halvering af det forventede antal korrekte cifre i hver beregnede singulærværdi. Punkt del (iii) kan også være ret alvorligt.

Eksempel 27.2. For (iii) betragt det følgende eksempel.

```
>>> import numpy as np
>>> s = 1e-9
>>> a = np.array([[1.0, 1.0],
...               [  s, 0.0],
...               [0.0,   s]])
```

Vi har

```
>>> a.T @ a
array([[1., 1.],
       [1., 1.]])
```

og alle indgange i matricen `a.T @ a` er præcis lige med `1.0`, som float:

```
>>> np.all(a.T @ a == np.ones((a.T @ a).shape))
True
```

Dette betyder at beregnede egenværdier af `a.T @ a` er nødvendigvis `2.0` og `0.0` fra egenvektorerne hhv. `[[1.0], [1.0]]` og `[[1.0], [-1.0]]`, som ville give singulærværdier `np.sqrt(2.0)` og `0.0`. Men de korrekte singulærværdier for `a` er $\sqrt{2+s^2}$ og s; ingen af disse er 0.

```
>>> np.linalg.svd(a, compute_uv=False)
array([1.41421356e+00, 1.00000000e-09])

>>> np.allclose(np.linalg.svd(a, compute_uv=False),
...             [np.sqrt(2.0 + s**2), s],
...             atol = np.finfo(float).eps)
True
```

Problemet stammer fra dannelsen af `a.T @ a`, hvor f.eks. den øverste venstre indgang er

```
>>> (1.0)**2 + (s)**2 + (0.0)**2
1.0
```

men `s**2` er mindre end machine epsilon, så dens sum med `1.0` giver blot `1.0`.

For denne matrix kunne problemet delvis forbigås ved at betragte i stedet den større matrix `a @ a.T`

```
>>> a @ a.T
array([[2.e+00, 1.e-09, 1.e-09],
       [1.e-09, 1.e-18, 0.e+00],
       [1.e-09, 0.e+00, 1.e-18]])
>>> np.sort(np.sqrt(np.abs(np.linalg.eigvals(a @ a.T))))
array([1.38777878e-17, 1.00000000e-09, 1.41421356e+00])
>>> np.sort(np.linalg.svd(a, compute_uv=False))
array([1.00000000e-09, 1.41421356e+00])
```

Men generelt kan AA^T være væsentlige større end A^TA og der kan være de samme underliggende problemer i forhold til float beregning. △

Indtil videre har vi kun kikket på de symmetriske matrice af type (a) og (b). For den tredje symmetrisk matrix (c), som er af størrelse $(n+m)\times(n+m)$, har vi

$$\begin{bmatrix} 0_{n\times n} & A^T \\ A & 0_{m\times m} \end{bmatrix} = \begin{bmatrix} 0 & V\Sigma^T U^T \\ U\Sigma V^T & 0 \end{bmatrix} = \begin{bmatrix} V & 0 \\ 0 & U \end{bmatrix} \begin{bmatrix} 0 & \Sigma^T U^T \\ \Sigma V^T & 0 \end{bmatrix}$$
$$= \begin{bmatrix} V & 0 \\ 0 & U \end{bmatrix} \begin{bmatrix} 0 & \Sigma^T \\ \Sigma & 0 \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & U^T \end{bmatrix}.$$

Her er de ydre matricer ortogonale med

$$\begin{bmatrix} V & 0 \\ 0 & U \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & U^T \end{bmatrix} = \begin{bmatrix} VV^T & 0 \\ 0 & UU^T \end{bmatrix} \begin{bmatrix} I_n & 0 \\ 0 & I_m \end{bmatrix} = I_{n+m}.$$

Desuden er matricen i midten symmetrisk, og vi kan beregne dens egenværdier. For hver singulærværdi σ har vi en delmatrix af formen

$$\begin{bmatrix} 0 & \sigma \\ \sigma & 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \sigma & 0 \\ 0 & -\sigma \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Vi ser så at matricen

$$\begin{bmatrix} 0 & \Sigma^T \\ \Sigma & 0 \end{bmatrix}$$

er diagonaliserbar med egenværdier

$$\sigma_0, -\sigma_0, \ldots, \sigma_{k-1}, -\sigma_{k-1}, 0, \ldots, 0.$$

Så den symmetriske matrice af type (c) har samme 2-norm og konditionstal, som A. Det betyder at vi kan forvente en præcise beregning af dens egenværdier, og dermed singulærværdierne for A, via QR-metoden. Selvfølgelig er ulempen at matricen er en $((n+m) \times (n+m))$-matrix, som kan være meget stort.

I praksis beregnes SVD via QR-metoden for A^TA. Men man undlader at danne matricen A^TA og i stedet for kikker på tilsvarende operationerne på den oprindelige matrix A. For eksempel, er det første trin at reducere A til en bidiagonal form

$$A = U_0 B V_0^T, \qquad B = \begin{bmatrix} * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Dette svarer netop til at B^TB er tridiagonal. Derefter findes oplysningen til »$Q^{(1)}$« fra QR-metoden for B^TB med shift $\mu^{(1)}$ givet ved en egenværdi af den sidste (2×2)-blok af B^TB. Dette anvendes på B, men ødelægger dens bidiagonal form. Så man gentager bidiagonaliseringstrinnet og forsætter. Til sidst vil metoden konvergere mod en SVD for A.

27.2 Principalkomponent analyse

Relationerne mellem $(m \times n)$-matricer og symmetriske matricer bruges i en del forskellige sammenhæng. Et eksempel er principalkomponent analyse.

Betragt datasættet givet i tabel 27.1. For en given række

$$x = \begin{bmatrix} x_0 & x_1 & \ldots & x_{n-1} \end{bmatrix} \in \mathbb{R}^{1 \times n}$$

Person	0	1	2	3	4
Vægt (kg)	54	56	56	61	66
Højde (m)	1,54	1,52	1,63	1,72	1,82

Tabel 27.1: Vægt og højde for nogle personer.

kan vi beregne dens *middelværdien*

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i \in \mathbb{R}$$

og dens *varians*

$$\operatorname{var}(x) = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2 \in \mathbb{R}.$$

Middelværdi er den gennemsnitlige værdi af $x_0, x_1, \ldots, x_{n-1}$. Variansen måler hvor tæt disse værdier ligger til middelværdien. Er variansen 0, så er alle x_i lige med middelværdien; er variansen stort, så er der stor spredning i x_i værdierne.

Ved at sætte $1_n = \begin{bmatrix}1 & 1 & \ldots & 1\end{bmatrix} \in \mathbb{R}^{1\times n}$ kan variansen skrives

$$\operatorname{var}(x) = \frac{1}{n}(x - \bar{x}1_n)(x - \bar{x}1_n)^T \in \mathbb{R}.$$

For to rækker kan vi også beregne deres indbyrdes *kovarians*

$$\operatorname{cov}(x, y) = \frac{1}{n-1}(x - \bar{x}1_n)(y - \bar{y}1_n)^T.$$

Kovarians kan betragtes, som et mål for hvor meget y_i-værdier følger de tilsvarende x_i-værdier. Er $\operatorname{cov}(x, y) = 0$, betragtes variablerne x og y, som værende uafhængig af hinanden.

Har man flere datarækker kan man udføre disse operationer i matrix form. Lad os antage at tabellen har m rækker og n søjler. F.eks. har vi n personer hvor vi har taget m forskellige mål. Så kan vi betragte vores data givet ved n søjlevektorer $v_0, \ldots, v_{n-1} \in \mathbb{R}^m$. Hver vektor består af de forskellige målinger for en enkelt person.

Vektoren af middelværdier er så

$$\bar{v} = \frac{1}{n}(v_0 + \cdots + v_{n-1}) \in \mathbb{R}^m.$$

For tabel 27.1 er $\bar{v} \in \mathbb{R}^2$, med den 0'te indgang middelvægten for gruppen og 1'te indgang middelhøjden.

Sæt

$$w_i = v_i - \bar{v}$$

og saml disse vektorer i en matrix

$$W = \left[w_0 \mid w_1 \mid \ldots \mid w_{n-1} \right] \in \mathbb{R}^{m \times n}.$$

Hver række i W har middelværdi 0, så er centreret omkring sin middelværdi. Nu kan vi danne *kovariansmatricen*

$$C = \frac{1}{n-1} W W^T \in \mathbb{R}^{m \times m}.$$

For $i = j$ er c_{ii} variansen af række i; for $i \neq j$ er c_{ij} kovariansen mellem række i og række j.

Kovariansmatricen er en symmetrisk matrix. Dens egenvektorer giver forskellige lineære kombinationer af datarække. Den tilhørende egenværdier er variansen for denne kombination. Oftest er man interesseret i de kombinationer der giver den største varians, da det beskriver bedst hvordan datapunkter adskiller sig mest. Dette svarer til den (numerisk) største egenværdi.

Som i vores diskussion ovenfor, er det mest præcist at beregne egenværdier af C fra singulærværdierne af W. Hvis $W = U\Sigma V^T$ er en SVD for W, så har vi

$$C = \frac{1}{n-1} U \Sigma \Sigma^T U^T = U \left(\frac{1}{n-1} \Sigma \Sigma^T \right) U^T.$$

Egenværdierne af C er dermed $\frac{1}{n-1}\sigma_i^2$, hvor σ_i er singulærværdierne for W.

Lad os nu betragte $U^T W$. Dette har kovariansmatrix

$$\begin{aligned} \frac{1}{n-1}(U^T W)(U^T W)^T &= \frac{1}{n-1}(\Sigma V)(\Sigma V)^T = \frac{1}{n-1}\Sigma V V^T \Sigma^T \\ &= \frac{1}{n-1}\Sigma\Sigma^T, \end{aligned}$$

som er diagonal. Dette er udtryk for at rækkerne i $U^T W$ er uafhængig af hinanden. Det vil sige disse lineær kombinationer af målinger er uafhængige af hinanden.

Eksempel 27.3. For datasættet i tabel 27.1 har vi

```
import matplotlib.pyplot as plt
import numpy as np

v = np.array([[54, 56, 56, 61, 66],
              [1.54, 1.52, 1.63, 1.72, 1.82]])

fig, ax = plt.subplots()
ax.plot(*v, 'o')
```

Middelværdien af en række kan beregnes via `np.mean`

```
print(np.mean(v[0]))
print(np.mean(v[1]))
```

```
58.6
1.6459999999999997
```

Vi kan beregne middelværdierne for alle rækker på en gang ved at bruge `np.mean` på `v` og angive `axis=1`

```
print(np.mean(v, axis=1))
```

```
[58.6    1.646]
```

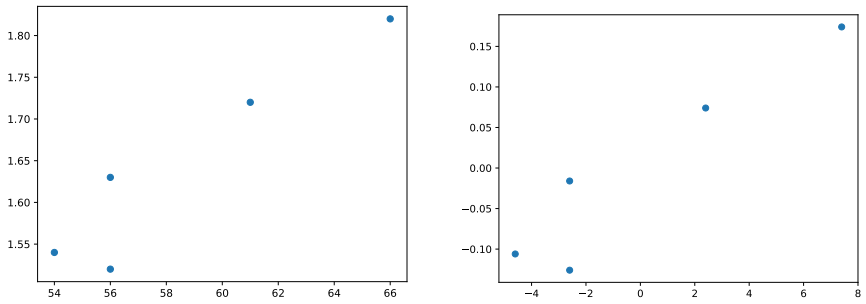
Resultatet kan fås som søjlevektor ved hjælp af `keepdims=True`

```
v_bar = np.mean(v, axis=1, keepdims=True)
print(v_bar)
```

```
[[58.6  ]
 [ 1.646]]
```

så den centrede matrix `w` er blot

```
w = v - v_bar
print(w)
```



Figur 27.1: Plot af data og den centrede version.

```
[[-4.6   -2.6   -2.6    2.4    7.4  ]
 [-0.106 -0.126 -0.016  0.074  0.174]]
```

```
fig, ax = plt.subplots()
ax.plot(*w, 'o')
```

Se figur 27.1 for plots af den oprindelige datamatrix `v` og den centrede matrix `w`.

Kovariansmatricen er nu

```
c = (1 / (v.shape[1] - 1)) * w @ w.T
print(c)
```

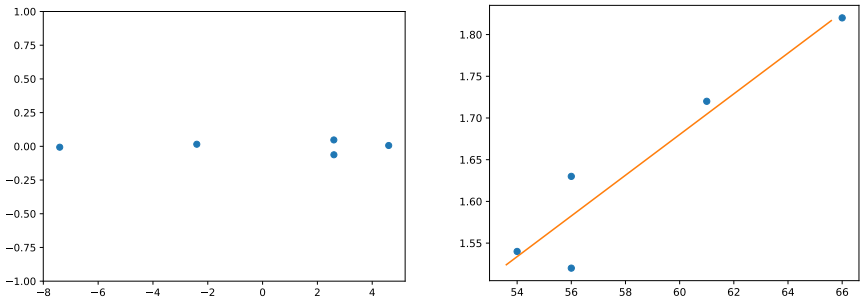
```
[[2.380e+01 5.805e-01]
 [5.805e-01 1.578e-02]]
```

som viser større variation i vægterne end i højderne. Beregner vi SVD for `w`, får vi følgende singulærværdier

```
u, s, _ = np.linalg.svd(w, full_matrices=False)
print(s)
```

```
[9.75995078 0.08050347]
```

Så den første retning har en relativt stort variation. En plot af den datasættet i via de nye måle kombinationer er givet i figur 27.2: til venstre plotte via de



Figur 27.2: Principale komponenter, uden normalisering.

to nye målekombinationer mod hinanden; til højre tegner vi den første principalretning, svarende til den største singulærværdi og dens målekombination, ovenpå den oprindelige datasæt.

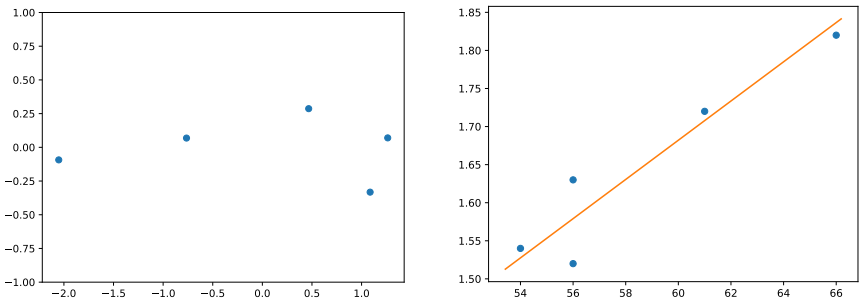
```
fig, ax = plt.subplots()
ax.set_ylim(-1, 1)
ax.plot(*(u.T @ w), 'o')
```

```
fig, ax = plt.subplots()
ax.plot(*v, 'o')
ax.plot(*(u[:, [0]] * np.linspace(-7, 5, 2) + v_bar))
```

En del af forklaringen på variationsstørrelsen er de forskellige enheder, og så numeriske størrelser, brugt for de forskellige målinger (vægt og højde). En standard måde at kompensere for dette, er at sørge for at dataet justeres så hver række har middelværdi 0 og varians 1. Disse skalæringsfaktorer et givet via kvadratroden af de diagonale indgange i kovariansmatricen `c`:

```
skalæring = np.sqrt(np.diag(c))[:, np.newaxis]
w_ny = w / skalæring
c_ny = (1 / (v.shape[1] - 1)) * w_ny @ w_ny.T
print(c_ny)
```

Vi kan beregne principale komponenter for det normaliserede data vi singulærværdidekomponeringen



Figur 27.3: Principalkomponenter, normaliserede.

```
u_ny, s_ny, _ = np.linalg.svd(w_ny)
print(s_ny)
```

```
[2.79087117 0.45938884]
```

De nye plots disse giver er

```
fig, ax = plt.subplots()
ax.set_ylim(-1, 1)
ax.plot(*(u_ny.T @ w_ny), 'o')
```

```
fig, ax = plt.subplots()
ax.plot(*v, 'o')
ax.plot(*((u_ny * skalæring)[:, [0]]
          * np.linspace(-2.2, 1.5, 2) + v_bar))
```

Se figur 27.3. △

Python indeks

Indeks