Numerisk Lineær Algebra F2021
Notesæt 16

Andrew Swann

24. marts 2021

Sidst ændret: 24. marts 2021.
Versionskode: 2621ea8.

Indhold

Figurer

16 Mindste kvadraters metode

Det er ikke usædvanligt at vi ønsker at estimere nogle parametre i en model. Nogle gange kan dette gøres ved at udføre konkrete eksperimeter og målinger. For at få en god estimering af parametrene tager man gerne så mange målinger, som muligt. Men typisk er disse målinger forbundet med fejl, så de opnåede datapunkter ligger ikke præcis på den kurve der forventes, og de resulterende ligninger er inkonsistent. Den mindste kvadraters metode forsøge at give et bedste bud på disse parametre.

16.1 Problemformulering

Givet et lineært ligningssystem

$$Ax = b \tag{16.1}$$

med flere ligninger end variabler, er der for typisk b ingen løsning. Lad os tage $A \in \mathbb{R}^{m\times n}$, med $m > n$, så $x \in \mathbb{R}^n$ og $b \in \mathbb{R}^m$. Systemet (16.1) kan skrives

$$x_0 a_0 + x_1 a_1 + \cdots + x_{n-1} a_{n-1} = b,$$

hvor $a_0, a_1, \ldots, a_{n-1}$ er søjlerne af A. Det følger at systemet har en løsning kun hvis b ligger i søjlerummet $S(A)$ af A.

For generel $b \in \mathbb{R}^m$, betragter vi *restvektoren*

$$r = b - Ax \in \mathbb{R}^m.$$

I stedet for at løse (16.1), arbejder vi på at finde $x \in \mathbb{R}^n$ således at restvektoren er så lille, som muligt. Lad os måle vektorstørrelse via det standard indre produkt. Så er vores problem: givet $A \in \mathbb{R}^{m\times n}$ og $b \in \mathbb{R}^m$, bestem $x \in \mathbb{R}^n$, som minimerer

$$\|r\|_2 = \|b - Ax\|_2.$$

Da x varierer, giver Ax alle vektorer i søjlerummet $S(A)$. Så $\|r\|_2$ minimeres når Ax er projektionen Pb af b på søjlerummet af A. Så den mindste kvadraters metode er ækvivalent med at løse

$$Ax = Pb. \tag{16.2}$$

Lad os give to forskellig løsningsmetoder.

16.2 Løsning via QR-dekomponering

Givet en tynd QR-dekomponering af $A = QR$, har vi

$$Ax = QRx = Qy = y_0 q_0 + y_1 q_1 + \cdots + y_{n-1} q_{n-1}$$

hvor $q_0, q_1, \ldots, q_{n-1}$ er søjlerne af Q. Lad os antage at $S(Q) = S(A)$. Så er projektion på søjlerummet af A givet ved

$$\begin{aligned} P &= q_0 q_0^T + q_1 q_1^T + \cdots + q_{n-1} q_{n-1}^T \\ &= QQ^T, \end{aligned}$$

da q_i er ortogonal af længde 1, sammenlign (13.3). Så vi ønsker at løse

$$QRx = QQ^T b. \tag{16.3}$$

Men Q har ortonormale søjler $q_0, q_1, \ldots, q_{n-1}$. Så dens Grammatrix er $Q^T Q = I_n$. Fra (16.3) har vi $Rx = I_n Rx = Q^T Rx = Q^T QQ^T b = I_n Q^T b = Q^T b$. Omvendt $Rx = Q^T b$ medfører $QRx = QQ^T b$. Så ser vi at (16.3) er det samme som

$$Rx = Q^T b. \tag{16.4}$$

Bemærk at R er øvre triangulær, så vi kan forvente at løse (16.4) via back substitution.

Vi får

MINDSTE KVADRATER VIA QR(A, b)
Krav: søjlerne af A er lineært uafhængig
1 Beregn en tynd QR-dekomponering $A = QR$
2 Beregn $Q^T b$
3 Løs $Rx = Q^T b$ via back substitution

Antallet af flops for denne metode er cirka $2mn^2$ hvis vi beregner QR-dekomponering via den forbedrede Gram-Schmidt proces.

16.3 Løsning via singulærværdidekomponering

En alternative metode er at udnytte singulærværdidekomponering. Husk at SVD dekomponering af $A = U\Sigma V^T \in \mathbb{R}^{m \times n}$ er ækvivalent med

$$A = \sigma_0 u_0 v_0^T + \sigma_1 u_1 v_1^T + \cdots + \sigma_{k-1} u_{k-1} v_{k-1}^T,$$

hvor $\sigma_0 \geqslant \sigma_1 \geqslant \cdots \geqslant \sigma_{k-1} \geqslant 0$, $k = \min\{m, n\}$.

Lad r være det største tal så at $\sigma_{r-1} > 0$. Så er

$$A = \sigma_0 u_0 v_0^T + \sigma_1 u_1 v_1^T + \cdots + \sigma_{r-1} u_{r-1} v_{r-1}^T$$

og

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

med $U \in \mathbb{R}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$, $V \in \mathbb{R}^{n\times r}$. Dette kaldes den *reducerede* singulærværdidekomponering.

Bemærk at $u_0, u_1, \ldots, u_{r-1}$ er nu en ortonormal basis for $S(A)$. Vi kalder r *rangen* af A, og observerer at den er lige med antallet af pivotsøjler i echelonform for A.

Givet den reducerede SVD $A = U\Sigma V^T$, er $u_0, u_1, \ldots, u_{r-1}$ er ortonormal basis for $S(A)$ og projektionen på $S(A)$ er

$$P = UU^T.$$

For de mindste kvadraters metode ønsker vi at løse

$$U\Sigma V^T x = UU^T b$$

Da søjlerne af U er ortonormal er dette ækvivalent med

$$\Sigma V^T x = U^T b$$

Husk at $\Sigma = \operatorname{diag}(\sigma_0, \sigma_1, \ldots, \sigma_{r-1})$, med alle $\sigma_i > 0$. Det giver at den inverse er

$$\Sigma^{-1} = \operatorname{diag}(1/\sigma_0, 1/\sigma_1, \ldots, 1/\sigma_{r-1}),$$

og vi kan nemt løse $\Sigma y = U^T b$ for et y, som $y = \Sigma^{-1} U^T b$.

Desuden er VV^T projektion på $S(V)$. Så ligningen $y = V^T x$ har en løsning givet som $x = Vy$. Alt i alt har vi

$$x = V\Sigma^{-1} U^T b. \tag{16.5}$$

Det kan bekræftes ved udregningen at

$$\begin{aligned} Ax &= U\Sigma V^T V\Sigma^{-1} U^T b = U\Sigma I_r \Sigma^{-1} U^T b \\ &= UI_r U^T b = UU^T b = Pb, \end{aligned}$$

som ønsket.

Matricen $A^+ = V\Sigma^{-1}U^T$ i (16.5) kaldes *pseudoinversen* til A, og (16.5) er løsningen til $Ax = Pb$, som har $\|x\|_2$ mindst muligt.

MINDSTE KVADRATER VIA SVD(A, b)
1 Beregn en reduceret SVD $A = U\Sigma V^T$
2 Beregn $U^T b$
3 Løs $\Sigma y = U^T b$
4 Sæt $x = Vy$

Antallet af flops for denne metode er cirka $2mn^2+11n^3$. Så generelt er denne metode langsommere end via QR-dekomponering. Men beregnes sådan en QR-dekomponering via den forbedrede Gram-Schmidt metode er resultatet ikke så præcis, som den fra SVD metoden; vi vil se et eksempel om lidt. Desuden er SVD metoden gyldig når søjlerne af A er ikke nødvendigvis lineært uafhængig, i modsætning til QR-metoden.

16.4 Polynomier

Lad os kikke på et par eksempler hvor vi tilpasser polynomier til given data.

Vi begynder med et simpelt eksempel hvor der er kun 3 datapunkter.

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

x = np.array([0.0, 1.0, 2.0])
y = np.array([0.2, 0.5, 1.2])

fig, ax = plt.subplots()
ax.plot(x, y, 'o')
```

Her har vi indlæst punkterne og plottet dem som den første plot i figur 16.1. Da der er kun 3 punkter, findes der et entydig andengradspolynomium igennem disse punkter. Dette polynomium kan findes på følgende måde. Vi danner Vandermondematricen hvis søjler er x-koordinaterne opløftet i potenserne 2, 1 og 0:

```
cols = len(x)
print(cols)
```

```
a = np.vander(x, cols)
print(a)
```

```
3
[[0. 0. 1.]
 [1. 1. 1.]
 [4. 2. 1.]]
```

Bestemmelsen af polynomiet $ax^2 + bx + c$, som har de 3 givne y-værdier i de givne punkter, er det samme som at løse det lineære ligningssystem

$$\begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}.$$

Dette kan gøres i python på følgende vis

```
koeffs = np.linalg.solve(a, y[:, np.newaxis])
print(koeffs)
```

```
[[0.2]
 [0.1]
 [0.2]]
```

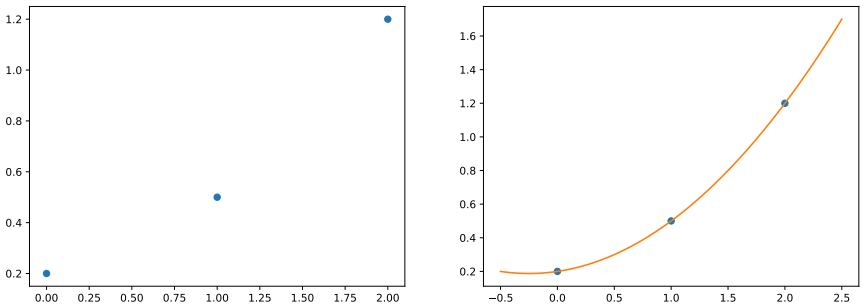
Nu kan vi plotte kurven fra andengradspolynomiet

```
t = np.linspace(-0.5, 2.5, 100)

fig, ax = plt.subplots()
ax.plot(x, y, 'o')
ax.plot(t, np.vander(t, cols) @ koeffs)
```

Resultatet vises til højre i figur 16.1.

Som alternativt kan vi finde en ret linje, dss. polynomium af grad 1, som ligger tæt på den givne punkter, ved hjælp af den mindste kvadraters metode. Dette kaldes også lineær regression. Lad os bruge QR-metoden via den forbedrede Gram-Schmidt.



Figur 16.1: Andengradspolynomium igennem 3 datapunkter.

```
def forbedret_gram_schmidt(a):
    k = a.shape[1]
    q = np.copy(a)
    r = np.zeros((k, k))
    for i in range(k):
        r[i, i] = np.linalg.norm(q[:, i])
        q[:, i] /= r[i, i]
        r[[i], i+1:] = q[:, [i]].T @ q[:, i+1:]
        q[:, i+1:] -= q[:, [i]] @ r[[i], i+1:]
    return q, r
```

Vi sætter graden til 1, men det andet argument til `np.vander` er antallet af søjle i Vandermondematricen, som er én mere end graden:

```
cols = 2
a = np.vander(x, cols)
print(a)
```

```
[[0. 1.]
 [1. 1.]
 [2. 1.]]
```

QR-dekomponeringen af a beregnes

```
q, r = forbedret_gram_schmidt(a)
print(q)
```

```
print()
print(r)
```

```
[[ 0.          0.91287093]
 [ 0.4472136   0.36514837]
 [ 0.89442719 -0.18257419]]

[[2.23606798 1.34164079]
 [0.         1.09544512]]
```

Vektoren $c = Q^T b$ er så

```
c = q.T @ y[:, np.newaxis]
print(c)
```

```
[[1.29691943]
 [0.14605935]]
```

og koefficienterne til førsteordenspolynomiet er nu

```
koeffs = np.linalg.solve(r, c)
print(koeffs)
```

```
[[0.5       ]
 [0.13333333]]
```

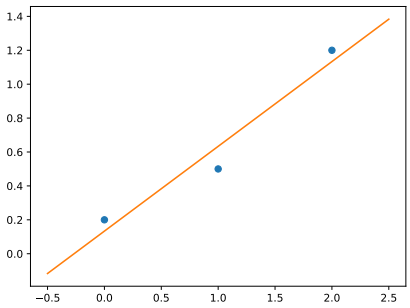
En plot af resultatet

```
t = np.linspace(-0.5, 2.5, 100)

fig, ax = plt.subplots()
ax.plot(x, y, 'o')
ax.plot(t, np.vander(t, cols) @ koeffs)
```

gives i figur 16.2.

Denne samme fremgangsmåde kan anvendes på flere datapunkter. Her vil man ofte gerne bruger polynomier af højere grad. Lad os tage et eksempel og først finde et polynomium, der går igennem alle datapunkter.



Figur 16.2: Lineær tilnærmelse af data.

```
x = np.array([-2.1, -1.9, -1.5, -0.8, -0.3, 0.1,
              0.5, 1.2, 1.3, 1.7, 2.4])
y = np.array([ 0.1, 0.4, -0.1, -0.6, -0.5, 0.1,
              1.0, 1.3, 0.7, 0.1, 0.2])

cols = len(x)
print(cols)
```

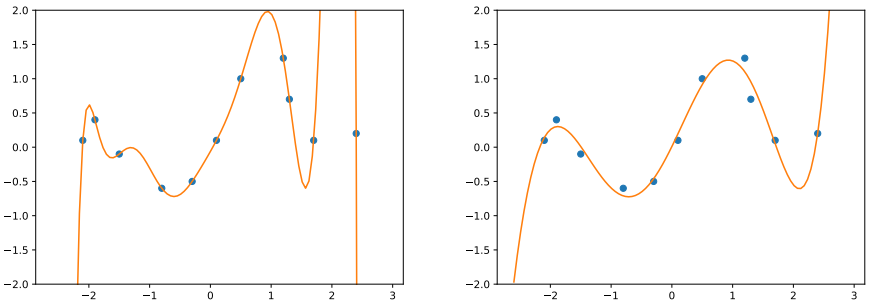
11

```
a = np.vander(x, cols)
koeffs = np.linalg.solve(a, y[:, np.newaxis])

t = np.linspace(x.min()-0.5, x.max()+0.5, 100)

fig, ax = plt.subplots()
ax.set_ylim(-2.0, 2.0)
ax.plot(x, y, 'o')
ax.plot(t, np.vander(t, cols) @ koeffs)
```

Kikker vi på plottet til venstre i figur 16.3, ses at dette polynomium er en meget dårlig repræsentation af formen af datapunkterne, den stræber for meget efter at gå igennem hver eneste punkt. Dette kaldes *over fitting*. Reduceres graden af polynomiet, og bruges de mindste kvadraters metode



Figur 16.3: Højere grad.

```
cols = 7
a = np.vander(x, cols)
q, r = forbedret_gram_schmidt(a)
c = q.T @ y[:, np.newaxis]
koeffs = np.linalg.solve(r, c)

fig, ax = plt.subplots()
ax.set_ylim(-2.0, 2.0)
ax.plot(x, y, 'o')
ax.plot(t, np.vander(t, cols) @ koeffs)
```

fås en bedre gengivelse, se plottet til højre i figur 16.3. Afvigelserne af dette polynomium fra de givne datapunkter ligger i restvektoren

```
rest = y[:, np.newaxis] - np.vander(x, cols) @ koeffs
print(rest)
```

```
[[-0.04036569]
 [ 0.09963581]
 [-0.11044335]
 [ 0.11108815]
 [-0.03960429]
 [-0.09563144]
 [ 0.08603725]
 [ 0.19850296]
```

```
 [-0.25445747]
 [ 0.04795903]
 [-0.00272098]]
```

Et godt mål for den samlede afvigelsen er normen

```
print(np.linalg.norm(rest))
```

```
0.400837595209125
```

Beregningen via SVD i stedet for QR-dekomponering, er givet nedenfor. I dette tilfælde er resultatet meget tæt på den forrige fra QR-metoden, så vi nøjes med at give længden af restvektoren.

```
u, s, vt = np.linalg.svd(a, full_matrices=False)
koeffs_svd = vt.T @ (np.diag(1/s) @ (u.T @ y[:, np.newaxis]))
rest_svd = y[:, np.newaxis] - np.vander(x, cols) @ koeffs_svd
print(np.linalg.norm(rest_svd))
```

```
0.4008375952091247
```

Differencen mellem restvektorene for de to metoder har længde

```
print(np.linalg.norm(rest_svd - rest))
```

```
1.9638163267674327e-14
```

i dette eksempel.

16.5 Løsning via normalligninger

Der er en tredje løsningsmetode for det mindste kvadraters problem, som burde nævnes, nemlig via normalligningerne. Der er mange tekster, som giver kun denne løsningsmetode, men i det næste kapitel vil vi se at den er numerisk meget upræcist. Så i praksis er metoderne ovenfor altid at foretrække.

Vi antager at $A \in \mathbb{R}^{m \times n}$, $m \geqslant n$, har lineært uafhængige søjler. Hvis A har reduceret SVD $A = U\Sigma V^T$, så har vi at $r = \text{rang}(A) = n$ og at

$$\begin{aligned} A^T A &= (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T \\ &= V\Sigma I_r \Sigma V^T = V\Sigma^2 V^T \end{aligned}$$

er invertibel med invers $V\Sigma^{-2}V^T$. Dette giver

$$\begin{aligned} (A^T A)^{-1} A^T &= V\Sigma^{-2}V^T V\Sigma U^T = V\Sigma^{-2} I_n \Sigma U^T \\ &= V\Sigma^{-1}U^T = A^+. \end{aligned}$$

Løsning af den mindste kvadraters problem $Ax = Pb$ for A med lineært uafhængige søjler er ækvivalent med løsning af

$$A^T Ax = A^T b,$$

som kaldes *normalligningerne*. Som nævnt ovenfor er denne metode dog numerisk upræcist og anbefales ikke.

Python indeks

Indeks