

# **Numerisk Lineær Algebra F2021**

## **Første halvdel**

Andrew Swann

Institut for Matematik  
Aarhus Universitet

Forår 2021

Sidst ændret: 9. april 2021.  
Versionskode: c48fb62.

# Indhold

<b>Indhold</b>	<b>ii</b>
<b>Figurer</b>	<b>v</b>
<b>Tabeller</b>	<b>vii</b>
<b>1 Indledning og første betragtninger</b>	<b>1</b>
1.1 Tal og fejl . . . . .	2
1.2 Flydende-komma repræsentation . . . . .	5
1.3 Fejl i flydende-komma tal . . . . .	9
<b>2 Plangeometri: vektorer og matricer</b>	<b>13</b>
2.1 Vektorer, norm og vinkelbestemmelse . . . . .	13
2.2 Indre produkt og matrixtransformationer . . . . .	15
2.3 Første tegninger . . . . .	21
<b>3 Matricer og vektorer</b>	<b>25</b>
3.1 Matricer . . . . .	25
3.2 Matricer i python . . . . .	26
3.3 Heatmap plot . . . . .	28
3.4 Multiplikation med en skalar og matrixsum . . . . .	31
3.5 Transponering . . . . .	33
3.6 Vektorer . . . . .	34
<b>4 Matrix multiplikation</b>	<b>37</b>
4.1 Række-søjleprodukt . . . . .	37
4.2 Matrixprodukt . . . . .	39
4.3 Regneregler for matrixprodukt . . . . .	45
4.4 Identitetsmatricer . . . . .	47

4.5	Ydre produkt . . . . .	48
<b>5</b>	<b>Numerisk håndtering af matricer</b>	<b>51</b>
5.1	Omkostninger ved matrixberegning . . . . .	51
5.2	Pythons for-løkke . . . . .	53
<b>6</b>	<b>Lineære ligningssystemer</b>	<b>57</b>
6.1	Elementære rækkeoperationer . . . . .	57
6.2	Rækkeoperationer i python . . . . .	59
6.3	Echelonform . . . . .	62
6.4	Løsning via rækkeoperationer . . . . .	63
<b>7</b>	<b>Matrixinvers</b>	<b>69</b>
7.1	Egenskaber af den inverse matrix . . . . .	69
7.2	Eksistens af den inverse . . . . .	73
7.3	Beregning af den inverse . . . . .	75
<b>8</b>	<b>Ortogonalitet og projektioner</b>	<b>77</b>
8.1	Standard indre produkt . . . . .	77
8.2	Vinkel mellem vektorer . . . . .	80
8.3	Projektion på en linje . . . . .	84
8.4	Ortogonalitet . . . . .	86
<b>9</b>	<b>Ortogonale matricer</b>	<b>95</b>
9.1	Definition og første eksempler . . . . .	95
9.2	Egenskaber . . . . .	96
9.3	Ortonormale vektorer og ortogonale matricer . . . . .	98
9.4	Householdermatrix . . . . .	100
9.5	Udvidelse til ortonormal basis . . . . .	105
<b>10</b>	<b>Singulærværdidekomponering</b>	<b>107</b>
10.1	Singulærværdier i dimension 2 . . . . .	108
10.2	SVD generelt . . . . .	111
10.3	Eksempler på SVD . . . . .	112
10.3.1	Punkter i planen . . . . .	112
10.3.2	Billede med støj . . . . .	114
10.3.3	Komprimering af et billede . . . . .	116
10.4	Eksistens af SVD . . . . .	120

## INDHOLD

<b>11</b>	<b>Konditionstal</b>	<b>123</b>
11.1	Konditionstal for differentiable funktioner . . . . .	123
11.2	Matrixnorm . . . . .	126
11.3	Konditionstal og lineære ligningssystemer . . . . .	129
<b>12</b>	<b>Generelle vektorrum</b>	<b>133</b>
12.1	Vektorrum . . . . .	133
12.2	Andre skalarer . . . . .	135
12.3	Underrum . . . . .	139
<b>13</b>	<b>Indre produkter</b>	<b>143</b>
13.1	Definitioner og eksempler . . . . .	143
13.2	Ortogonale samlinger og tilnærmelse . . . . .	148
13.3	Numerisk integration . . . . .	151
<b>14</b>	<b>Ortogonale samlinger: klassisk Gram-Schmidt</b>	<b>157</b>
14.1	Den klassiske Gram-Schmidt proces . . . . .	157
14.2	Klassisk Gram-Schmidt i $\mathbb{R}^n$ . . . . .	163
<b>A</b>	<b>Det græske alfabet</b>	<b>167</b>
<b>B</b>	<b>Python</b>	<b>169</b>
	<b>Bibliografi</b>	<b>171</b>
	<b>Python indeks</b>	<b>173</b>
	<b>Indeks</b>	<b>177</b>

# Figurer

2.1	Afstand og vinkel i planen . . . . .	13
2.2	Vinklen mellem to vektorer . . . . .	16
2.3	Skalering . . . . .	18
2.4	Rotation . . . . .	18
2.5	Spejling . . . . .	20
2.6	Projektion . . . . .	20
2.7	Skævvridning . . . . .	21
3.1	En første heatmap plot . . . . .	29
3.2	Heatmap med colorbar . . . . .	30
3.3	Heatmap med justeret farveskala . . . . .	30
3.4	Vektorsum af $u = (2, 1)$ og $v = (1, 3)$ . . . . .	35
4.1	Linjen $x + 3y = 2$ . . . . .	39
4.2	Elektrisk kredsløb . . . . .	44
4.3	Eksempler på ydre produkter . . . . .	50
5.1	Oprindelig figur og dens rotationer . . . . .	56
6.1	Rundkørsel . . . . .	64
6.2	To linjer i planen, som skærer . . . . .	66
6.3	Tre linjer i planen, som har intet fællespunkt . . . . .	67
6.4	To parallelle linjer i planen har intet fællespunkt . . . . .	67
8.1	Projektion på en linje . . . . .	84
8.2	Ortogonal samling af polynomier . . . . .	93
8.3	Numerisk approksimation af eksponentialfunktionen . . . . .	93
8.4	Eksponentialfunktionen og dens Taylor udvikling . . . . .	94
9.1	Effekten af en Householdermatrix . . . . .	101

## FIGURER

9.2	Spejling til $\varepsilon e_0$	102
10.1	Cirkel efter matrixmultiplikation	108
10.2	SVD af punktsamling	114
10.3	Cifret 0, plus støj	115
10.4	SVD af billede med støj	117
10.5	Uppsala domkirke	118
10.6	Singulærværdier for billedet af domkirken	119
10.7	SVD komprimering	120
11.1	Trekantsuligheden	126
13.1	Fourier cosinus tilnærmelse for $1 - x$	150
13.2	Integral	152
13.3	Rektanglet fra den venstre funktionsværdi	152
13.4	Venstre- og højre-tilnærmelse til integral	153
13.5	Trapez-tilnærmelse til integral	154
14.1	Legendre og Chebychev polynomier	163

# Tabeller

1.1	Python regneoperationer på tal . . . . .	3
1.2	Fejl i beregninger til 3 decimaler . . . . .	4
5.1	Omkostninger ved vektor- og matrixberegninger . . . . .	53
6.1	Rækkeoperationer i python . . . . .	60





# Kapitel 1

## Indledning og første betragtninger

**N**UMERISK LINEÆR ALGEBRA er studiet af metode for behandling af beregningsproblemer indenfor matricer og vektorer. Det er et hjørnesteen i computerberegninger baseret på matematiske modeller for problemstillinger fra alle videnskaber. Det er et fag der har udviklet sig hurtigt og er under stadig forandring. Grundlaget er teorien for vektorrum og lineære transformationer. Konkret arbejde med tal betyder at disse teorier skal vinkles og forfines for at behandle problemer baseret på eksperimentel eller observationsbaseret data og for at udføre beregninger på en computer. Moderne anvendelser ofte kræver store mængder data, og det kan ofte medføre at en direkte tilgang til implementering er ikke tilstrækkelig effektiv. Overraskende mange problemstillinger kan udtrykkes via matrixligninger, og ofte den bedste metode numerisk er at finde og anvende gode faktoriseringer af disse matricer. Af disse problemstillinger falder de fleste indfor variationer over en af de følgende to:

- (a) Givet en matrix  $A$  og en vektor  $b$  bestem alle vektorer  $x$ , som opfylder

$$Ax = b.$$

- (b) Givet en matrix  $A$  bestem alle vektorer  $v$  og alle tal  $\lambda$  således at

$$Av = \lambda v.$$

I dette første kapitel, tager vi et første kik på nogle af de grundlæggende betragtninger for emnet.

## 1.1 Tal og fejl

**T**AL I VORES DATA er yderst sjælden præcis. Måler man et lufttryk til at være 1005 hPa er der usikkerhed omkring det sidste tal; har man et computer-billed er den røde farve i et given punkt typisk gemt som et heltal mellem 0 og 255, og så kan ikke afspejle virkeligheden eksakt. Det er kun når vi tæller objekter 1, 2, 3, ... at vi får et eksakt tal at arbejde med; men også her hvis der er mange objekter eller objekterne er svære at tælle, som f.eks. ulve i Jylland, er tallet ikke nødvendigvis korrekt.

Selvom man har et eksakt tal opstår der også praktiske problemer når en computer skal opbevare og behandle den. Medmindre vi arbejder symbolsk, er tal typisk opbevaret i form der svarer til visse rationelle tal  $a/b$ . Det betyder at tal som  $\sqrt{2} \approx 1,414\,21$ ,  $e \approx 2,718\,28$ ,  $\pi \approx 3,141\,59$  har ikke eksakte repræsentationer. Desuden er nævneren  $b$  typisk en potens af 2 (binærbrøk). Som konsekvens kan hverken  $1/3 \approx 0,333\,33$  eller  $1/7 \approx 0,142\,86$  behandles præcist.

Dette forværres af at computeren også giver os rige mulighed for at lave beregninger med mange tal. En computer vil sagtens kunne beregne en sum for et datasæt bestående af en million tal, men så risikerer man også at fejleffekterne bidrager en million gange.

Selv for heltal er der problemer. Computerhukommelse bliver billigere og billigere, men det er alligevel dyrt at arbejde med større heltal. Et heltal med mange cifre beslaglægger en del hukommelsesplads, og beregninger bliver langsommere.

I dette kursus har vi programmeringssproget python som eksempelsprog. Hvad et python program kan behandle er ikke fuldstændigt fastlagt og kan variere, afhængig af hvilke version og implementering der er tale om, samt egenskaber af den underliggende computer den køres på.

I python skal man skelne mellem heltal og decimalbrøk: skriver man 143 har man et heltal, ønskes derimod en decimalbrøk skrives 143.0 i stedet. Når vi bruger decimalbrøk, så viser nogle af de ovennævnte problemer sig ret hurtigt:

```
>>> 1.1 + 1.2
2.3
>>> 1.9 + 1.2
3.0999999999999996
>>> 1.1 - 1.2
-0.09999999999999987
```

python	matematik	navn
$x + y$	$x + y$	sum
$x - y$	$x - y$	differens
$x * y$	$xy$	produkt
$x / y$	$x/y$	kvotient
$x ** y$	$x^y$	potens

Tabel 1.1: Python regneoperationer på tal.

```
>>> 2.1 * 2.7
5.6700000000000001
>>> 2.1 / 2.0
1.05
>>> 2.1 / 2.5
0.8400000000000001
```

hvor vi har brugt pythons regneoperationer som i tabel 1.1. For reelle tal ville alle svar ovenfor har haft højst 2 decimaler. Desuden har vi

```
>>> 1.1 + (1.3 + 1.5)
3.9
>>> (1.1 + 1.3) + 1.5
3.9000000000000004
```

selvom begge sum er det samme for reelle tal. Når vi betragter regneoperationer, har manglende præcision nogle konsekvenser. Som eksempel lad os arbejder med decimalbrøk med kun 3 cifre efter kommaet. Så vil  $a' = 10,153$  blive nødt til at repræsentere et hvilken som helst tal  $a$  mellem  $a_{\min} = 10,1525$  og  $a_{\max} = 10,1535$ .

Hvis et reelt tal  $x$  er repræsenteret af tallet  $x'$  vil vi kalde

$$x' - x$$

for *fejlen*. Vi vil også kalde

$$\delta = \frac{x' - x}{x}$$

## 1 INDLEDNING OG FØRSTE BETRAGTNINGER

	beregnet	min værdi	max værdi	fejl	relativ fejl %
+	20,139	20,138 000	20,140 000	0,0010	0,004 97
−	0,167	0,166 000	0,168 000	0,0010	0,602 41
×	101,388	101,377 789	101,397 928	0,0102	0,010 07
/	1,017	1,016 622	1,016 824	0,0002	0,017 27

Tabel 1.2: Fejl i beregninger til 3 decimaler ved brug af repræsentanter  $a' = 10,153$  og  $b' = 9,986$ .

for den *relative fejl*, så længe  $x$  er ikke 0. Den opfylder

$$x' = x(1 + \delta). \quad (1.1)$$

Ofte udtrykker vi den relative fejl i procent, ved at gange  $\delta$  med 100.

For  $a$  mellem  $a_{\min}$  og  $a_{\max}$  ovenfor har vi så, at fejlen  $a' - a$  ligger mellem  $a' - a_{\max} = -0,0005$  og  $a' - a_{\min} = 0,0005$ , samt at den relative fejl ligger mellem

$$\frac{a' - a_{\max}}{a_{\max}} = -0,000\,049\,244\,1 \approx -0,004\,92\%$$

og

$$\frac{a' - a_{\min}}{a_{\min}} = 0,000\,049\,249\,0 \approx 0,004\,92\%.$$

Betragt nu  $b' = 9,986$ , repræsenterende et reelt tal  $b$  mellem  $b_{\min} = 9,9855$  og  $b_{\max} = 9,9865$ . Fejlen for  $b'$  er den samme som for  $a'$ , mens den relative fejl er  $\pm 0,005\,01\%$ .

Summen forventes repræsenteret af  $a' + b' = 20,139$ , men  $a + b$  ligger mellem  $a_{\min} + b_{\min} = 20,138$  og  $a_{\max} + b_{\max} = 20,140$ . Fejlen er  $\pm 0,001$ , mens den relative fejl er mellem  $\pm 0,004\,97\%$ . For den plus + operation, ser vi at fejlen er dobbelt så stort som for  $a$ , mens i dette eksempel er den relative fejl er næsten uændret. For de andre regneoperationer difference −, produkt  $\times$  og kvotient /, kan fejlen regnes på tilsvarende måde. Resultaterne gives i tabel 1.2. Her ser vi en del udsving i fejl og relativ fejl. Mest bekymrende er den store relativ fejl for difference operationen, som afspejler færre betydende cifre i resultatet.

Generelle regneregler for disse type uligheder er givet ved:

**Proposition 1.1.** For reelle tal  $a$  og  $b$  med  $a_{\min} \leq a \leq a_{\max}$  og  $b_{\min} \leq b \leq b_{\max}$  gælder

## 1.2 FLYDENDE-KOMMA REPRÆSENTATION

$$(a) \ a_{\min} + b_{\min} \leq a + b \leq a_{\max} + b_{\max},$$

$$(b) \ a_{\min} - b_{\max} \leq a - b \leq a_{\max} - b_{\min}.$$

Hvis yderligere  $a_{\min}, b_{\min} > 0$ , så har vi

$$(c) \ a_{\min} b_{\min} \leq ab \leq a_{\max} b_{\max},$$

$$(d) \ a_{\min}/b_{\max} \leq a/b \leq a_{\max}/b_{\min}.$$

□

## 1.2 Flydende-komma repræsentation

Et par eksempler på tal i flydende-komma form, eller tal i videnskabelig notation, er

$$1,639\ 01 \cdot 10^2$$

$$-2,704\ 62 \cdot 10^{-3}$$

$$5,672\ 91 \cdot 10^{100}$$

Disse skrives i python som

```
>>> 1.63901e2
163.901
>>> -2.70462e-3
-0.00270462
>>> 5.67291e100
5.67291e+100
```

og som set vil tal, der er ikke for store, bliver skrevet ud som en almindelig decimalbrøk. Her benyttes grundtallet 10.

Generelt er en *flydende-komma repræsentation* med grundtal 10 er et reelt tal  $x$  af formen

$$x = \pm m \cdot 10^k$$

hvor

$$m = "d_1, d_2 \dots d_r" = d_1 + \frac{d_2}{10} + \dots + \frac{d_r}{10^{r-1}}$$

er en decimalbrøk med  $r$  cifre,  $d_i \in \{0, 1, 2, \dots, 9\}$ ,  $d_1 \neq 0$ , og  $k$  er et heltal. Decimaltet  $m$  kaldes *mantissen* og  $k$  kaldes *eksponenten*. Så tallet  $-2,704\ 62 \cdot 10^{-3}$  har fortegn  $-1$ , mantisse  $2,704\ 62$  og eksponent  $-3$ .

I python repræsenteres alle decimaltal internt i en tilsvarende flydende-komma form, men med grundtallet 2 i stedet for 10, dvs. en binærbrøk ganget med en potens som  $2^{17}$ . Denne type kaldes for *float*:

## 1 INDLEDNING OG FØRSTE BETRAGTNINGER

```
>>> type(143)
<class 'int'>
>>> type(143.0)
<class 'float'>
>>> type(-2.70462e-3)
<class 'float'>
```

Der afsættes en fast mængde hukommelse til enhver float. Vi vil arbejde med NumPy pakken. På min maskine fortæller

```
>>> import numpy as np
>>> np.finfo(float).bits
64
```

at der afsættes 64 bit = 8 byte for hver `float`. Din computer råder sikkert over hukommelse med adskillelige gigabyte =  $2^{30}$  byte = 1 073 741 824 byte. I hver gigabyte er der så plads til at opbevare  $134\,217\,728 \approx 1,3 \cdot 10^8$  tal i `float` form.

Da der afsættes 64 bit til en `float`, er der højst

$$2^{64} = 18\,446\,744\,073\,709\,551\,616 \approx 1,8 \cdot 10^{19}$$

forskellige reelle tal der kan bruges som repræsentanter i `float` form. Dette står i kontrast til at der er uendelige mange reelle tal, og har nogle væsentlige konsekvenser

(a) Der er et største tal  $\text{float}_{\max}$  og et mindste tal  $\text{float}_{\min}$  af typen `float`:

```
>>> np.finfo(float).max
1.7976931348623157e+308
>>> np.finfo(float).min
-1.7976931348623157e+308
```

(b) Der er et mindste tal  $\epsilon_{\text{machine}} > 0$ , *machine epsilon* eller *nøjagtigheden*, af type `float` således at  $1 + \epsilon_{\text{machine}}$  er igen en `float` og  $1 + \epsilon_{\text{machine}} > 1$ :

```
>>> np.finfo(float).eps
2.220446049250313e-16
```

(c) Der er et mindste tal  $\text{float}_{\text{tiny}}$  af type `float` som er skarpt større end 0:

## 1.2 FLYDENDE-KOMMA REPRESENTATION

```
>>> np.finfo(float).tiny
2.2250738585072014e-308
```

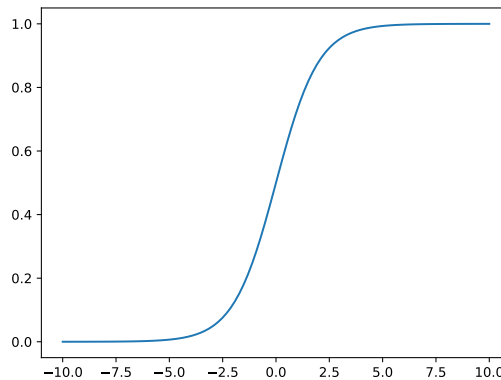
(d) Resultater af regneoperationer med tal af type `float` skal generelt afrundes for at give et svar der igen er af type `float`.

(e) Resultater der er for store kan ikke repræsenteres som `float`; dette kaldes *overflow*.

(f) Resultater der er for tæt på 0 riskærer at bliver afrundet til `0.0`; dette kaldes *underflow*.

*Eksempel 1.2.* I machine learning er der ofte brug for sigmoidfunktionen

$$\sigma(x) = \frac{e^x}{1 + e^x},$$



som er en differentiabel funktion med værdier skarpt mellem 0 og 1. Beregning af  $\sigma(x)$  indebærer udregning af eksponentialfunktionen  $e^x$ , som hurtige antage meget store værdier for  $x$  positivt, og værdier tæt på 0 for  $x$  negativt. Eksponentialfunktionen er tilgængelig i `numpy` pakken som `np.exp`

```
>>> import numpy as np
>>> np.exp(1.75)
5.754602676005731
>>> np.exp(700.0)
1.0142320547350045e+304
>>> np.exp(-700.0)
9.85967654375977e-305
```

Prøver man en lidt større værdi, får man et svar der er ikke et `float` tal

## 1 INDLEDNING OG FØRSTE BETRAGTNINGER

```
>>> np.exp(750.0)
inf
```

som standard ledsaget af en advarsel

```
<stdin>:1: RuntimeWarning: overflow encountered in exp
```

Hvis vi forsøger at beregne  $\sigma(750)$ , får vi

```
>>> np.exp(750.0)/(1 + np.exp(750.0))
nan
```

```
<stdin>:1: RuntimeWarning: invalid value encountered in
↪ double_scalars
```

Men vi kan omskrive udtrykket for  $\sigma$  til

$$\sigma(x) = \frac{e^x}{1 + e^x} = \frac{e^x}{1 + e^x} \cdot \frac{e^{-x}}{e^{-x}} = \frac{1}{1 + e^{-x}}$$

Nu vil beregning af  $\sigma(750)$  bruge `np.exp(-750.0)`, som afrundes til 0.0, uden nogen advarsel, og beregning af sigmoidfunktionen giver værdien

```
>>> 1/(1 + np.exp(-750.0))
1.0
```

som kan være mere brugbart. Der er en anden omskrivning

$$\sigma(x) = \frac{e^x}{1 + e^x} = \frac{e^x}{1 + e^x} \cdot \frac{e^{-x/2}}{e^{-x/2}} = \frac{e^{x/2}}{e^{x/2} + e^{-x/2}}$$

som undgår både overflow og underflow ved denne værdi

```
>>> np.exp(750.0/2)/(np.exp(750.0/2) + np.exp(-750.0/2))
1.0
```

men det hjælper ikke med udregning af  $\sigma(1500)$ .

△



*Bemærkning 1.3.* Det er ikke nødvendigvis godt, at overflow eller underflow opstår ubemærket. De er tegn på at noget er gået galt. Python kan tvinges til at stoppe ved sådan en fejl i `float`-beregninger via

```
>>> import numpy as np
>>> save_err = np.seterr(all = 'raise')
```

◇

## 1.3 Fejl i flydende-komma tal

Moderne computer plejer at følge en standard fastlagt af IEEE (Institute of Electrical and Electronics Engineers) for tal af type `float`. Ignorerer vi problemer med overflow og underflow, kan man generelt forvente en ideal implementering af float at

(F1) for ethvert reel tal  $x \in \mathbb{R}$  findes der et tal  $\text{float}(x)$  af type `float`, der repræsenterer  $x$  inden for en relativ fejl af højst  $\epsilon_{\text{machine}}/2$ , og

(F2) operationerne  $+$ ,  $-$ ,  $\times$ ,  $/$  og  $\sqrt{\cdot}$  er implementeret på tal af type `float` således at svar er korrekt inden for en relative fejl af højst  $\epsilon_{\text{machine}}/2$ .

Punkt (F1) siger, at for alle reelle tal  $x$  har vi

$$\text{float}(x) = x(1 + \delta_x)$$

hvor den relative fejl  $\delta_x$  opfylder  $|\delta_x| \leq \epsilon_{\text{machine}}/2$ . Dette er det samme, som at siger

$$|\text{float}(x) - x| \leq |x|\epsilon_{\text{machine}}/2.$$

For python `float`, vi kan forvente at mantissen afrundet til de første

```
>>> np.finfo(float).precision
15
```

cifre er korrekt.

Punkt (F2) siger tilsvarende, at for  $a$  og  $b$  af type `float` gælder

$$\begin{aligned} a +_{\text{float}} b &= (a + b)(1 + \delta_1), & a -_{\text{float}} b &= (a - b)(1 + \delta_2), \\ a *_{\text{float}} b &= ab(1 + \delta_3), & a /_{\text{float}} b &= (a/b)(1 + \delta_4), \\ \text{sqrt}(a) &= \sqrt{a}(1 + \delta_5), \end{aligned}$$

## 1 INDLEDNING OG FØRSTE BETRAGTNINGER

med  $|\delta_i| \leq \epsilon_{\text{machine}}/2$  for alle relative fejl  $\delta_i$ . Disse fejl  $\delta_i$  varierer med  $a$  og  $b$ , men er altid kontrolleret af det samme tal,  $\epsilon_{\text{machine}}/2$ .

På trods af  $\epsilon_{\text{machine}}$  er ret lille er vi stadigvæk plaget af problemerne i afsnit 1.1, især med hensyn til differenceoperationen og tab af betydende cifre.

*Eksempel 1.4.* Betragt andengradsligningen

$$2x^2 + 98x + \frac{1}{4} = 0.$$

Vi husker at  $ax^2 + bx + c = 0$  har rødder

$$q = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{og} \quad r = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

I vores tilfælde er  $b = 98$  og kvadratroden af diskriminanten er  $\sqrt{b^2 - 4ac} = \sqrt{9602.0} \approx 97,990$ . Dette betyder at når vi regner  $q$ , hvor skal vi tage differencen af  $b$  og  $\sqrt{b^2 - 4ac}$ , kan vi miste flere betydende cifre. Selvom vi bruger `float` kan det give fejl i den beregnede værdi for  $q$ . Vi kan delvis se dette ved at sætter  $q$  tilbage i polynomiet:

```
>>> import numpy as np
>>> a = 2.0 #kommentar: sæt 'a' til værdien 2.0
>>> b = 98.0
>>> c = 1.0 / 4.0
>>> sqrt_discriminant = np.sqrt(b**2 - 4*a*c)
>>> q = (-b + sqrt_discriminant)/(2*a)
>>> q
-0.0025511532323037045
>>> a*q**2 + b*q + c
-1.3367085216486885e-13
```

Tilgængæld er udregning af  $r$  uden dette problem

```
>>> r = (-b - sqrt_discriminant)/(2*a)
>>> r
-48.997448846767696
>>> a*r**2 + b*r + c
0.0
```

Husker vi at  $q$  og  $r$  opfylder

$$\begin{aligned} ax^2 + bx + c &= a(x - q)(x - r) \\ &= ax^2 - a(q + r)x + aqr \end{aligned}$$

kan vi sammenligne koefficienter af de forskellige potenser af  $x$ , som giver

$$b = -a(q + r) \quad \text{og} \quad c = aqr.$$

Den sidste kan løses for  $q$ :

$$q = \frac{c}{ar} = \frac{2c}{-b - \sqrt{b^2 - 4ac}}.$$

Bruges denne formel for  $q$ , får vi et bedre resultat:

```
>>> q_new = (2*c)/(-b - sqrt_discriminant)
>>> q_new
-0.0025511532323023406
>>> a*q_new**2 + b*q_new + c
0.0
```

Den relative fejl i  $q$  er så

```
>>> (q - q_new)/q_new
5.346312858502275e-13
```

som er væsentlig større end machine epsilon. △

Generelt afhænger valget af beregningsmetoden af fortegnet  $\text{sgn}(b)$  af  $b$ .

**Proposition 1.5.** *Givet*

$$\text{sgn}(b) = \begin{cases} 1, & \text{for } b \geq 0, \\ -1, & \text{for } b < 0, \end{cases}$$

er rødderne  $r_1, r_2$  af

$$ax^2 + bx + c = 0 \quad (a \neq 0, (b, c) \neq (0, 0))$$

givet ved

$$r_1 = \frac{-b - \text{sgn}(b)\sqrt{b^2 - 4ac}}{2a}, \quad r_2 = \frac{2c}{-b - \text{sgn}(b)\sqrt{b^2 - 4ac}}. \quad \square$$

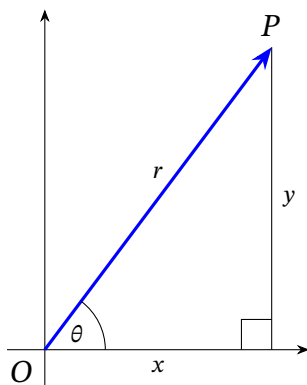


## Kapitel 2

# Plangeometri: vektorer og matricer

### 2.1 Vektorer, norm og vinkelbestemmelse

Vi er vant til at repræsentere punkter  $P$  i planen via deres koordinater:  $P = (x, y)$ . F.eks.  $P = (3, 4)$  er punktet med  $x$ -koordinat 3 og  $y$ -koordinat 4. Sådan et koordinatpar kan også betragtes som endepunkt for en pil  $\overrightarrow{OP}$  fra origo  $O = (0, 0)$  til punktet  $P = (x, y)$ . Denne pil kan ses som den længste side i en retvinklet trekant med kateter af længde hhv.  $x$  og  $y$ , se figur 2.1. Den



Figur 2.1: Afstand og vinkel i planen.

## 2 PLANGEOMETRI: VEKTORER OG MATRICER

pythagoræiske læresætning fortæller os at  $\overrightarrow{OP}$  har længde  $r = \sqrt{x^2 + y^2}$ :

$$|\overrightarrow{OP}| = r = \sqrt{x^2 + y^2}.$$

Desuden kan vinklen  $\theta$  bestemmes fra de trigonometriske relationer

$$c = \cos(\theta) = \frac{x}{\sqrt{x^2 + y^2}}, \quad s = \sin(\theta) = \frac{y}{\sqrt{x^2 + y^2}}.$$

Oftest har vi ikke brug for at kende selve værdien af  $\theta$ , men kan nøjes med parret  $(c, s)$ .

Et koordinatpar  $(x, y)$  giver anledning til en vektor

$$v = \begin{bmatrix} x \\ y \end{bmatrix}.$$

Vi indfører *normen* af denne vektor til at være længde af  $\overrightarrow{OP}$ :

$$\|v\|_2 = \left\| \begin{bmatrix} x \\ y \end{bmatrix} \right\|_2 = \sqrt{x^2 + y^2} = r.$$

Skalæres figur 2.1 med faktoren  $1/\|v\|_2$ , fås en ny vektor

$$w = \frac{1}{\|v\|_2} v = \begin{bmatrix} x/\sqrt{x^2 + y^2} \\ y/\sqrt{x^2 + y^2} \end{bmatrix} = \begin{bmatrix} c \\ s \end{bmatrix}. \quad (2.1)$$

Denne vektor  $w$  har norm 1, da længden af kateteret i den nye retvinklede trekant er 1. Dette kan bekræftes med regnestykket

$$\|w\|_2^2 = \left( \frac{x}{\sqrt{x^2 + y^2}} \right)^2 + \left( \frac{y}{\sqrt{x^2 + y^2}} \right)^2 = \frac{x^2}{x^2 + y^2} + \frac{y^2}{x^2 + y^2} = 1.$$

Vektorer af norm 1 kaldes for *enhedsvektorer*. Relationen (2.1) kan bruges til at bestemme vektoren  $v$  ud fra længde  $r$  og parret  $(c, s)$  ved

$$v = \|v\|_2 w = r \begin{bmatrix} c \\ s \end{bmatrix} = \begin{bmatrix} rc \\ rs \end{bmatrix}. \quad (2.2)$$

Givet  $r \geq 0$  og  $(c, s)$  med  $c^2 + s^2 = 1$ , fås fra (2.2) vektoren  $v = \begin{bmatrix} rc \\ rs \end{bmatrix}$  med  $\|v\|_2 = r$  og enhedsvektor  $\frac{1}{\|v\|_2} v = \begin{bmatrix} c \\ s \end{bmatrix}$ .

Bemærk at vektorerne  $e_0$  og  $e_1$  givet ved

$$e_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{og} \quad e_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

er enhedsvektorer, som peger langs koordinataksene.

## 2.2 Indre produkt og matrixtransformationer

Givet to vektorer  $u = \begin{bmatrix} a \\ b \end{bmatrix}$  og  $v = \begin{bmatrix} x \\ y \end{bmatrix}$  sætter vi

$$\langle u, v \rangle = \left\langle \begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} x \\ y \end{bmatrix} \right\rangle = ax + by.$$

Tallet  $\langle u, v \rangle$  kaldes det *indre produkt* mellem  $u$  og  $v$ . Nogle steder skrives dette som  $u \cdot v$ , og dermed får det også navnet *prikproduktet*.

Vi får straks at

$$\langle v, v \rangle = \left\langle \begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} x \\ y \end{bmatrix} \right\rangle = x^2 + y^2 = \|v\|_2^2,$$

dvs.

$$\|v\|_2 = \sqrt{\langle v, v \rangle}.$$

Med hensyn til parret  $(c, s) = (\cos(\theta), \sin(\theta))$ , der bestemmer vinklen  $\theta$ , bemærker at vi har

$$\langle e_0, w \rangle = \left\langle \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} c \\ s \end{bmatrix} \right\rangle = 1 \cdot c + 0 \cdot s = c.$$

Givet to enhedsvektorer  $w = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$ ,  $z = \begin{bmatrix} \cos(\varphi) \\ \sin(\varphi) \end{bmatrix}$  har vi

$$\begin{aligned} \langle w, z \rangle &= \left\langle \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}, \begin{bmatrix} \cos(\varphi) \\ \sin(\varphi) \end{bmatrix} \right\rangle \\ &= \cos(\theta) \cos(\varphi) + \sin(\theta) \sin(\varphi) \\ &= \cos(\varphi - \theta) \end{aligned}$$

når vi bruger trigonometriske identiteter. Generelt har vi

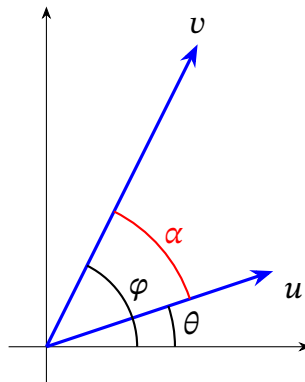
$$\langle u, v \rangle = \|u\|_2 \|v\|_2 \cos(\alpha)$$

hvor  $\alpha$  er vinklen mellem  $u$  og  $v$ , se figur 2.2.

Det indre produkt kan skrives på en anden måde hvis indfører transponerings operation. For en vektor  $u = \begin{bmatrix} a \\ b \end{bmatrix}$  er den *transponerede*

$$u^T = \begin{bmatrix} a & b \end{bmatrix},$$

## 2 PLANGEOMETRI: VEKTORER OG MATRICER



Figur 2.2: Vinklen mellem to vektorer.

en *rækkevektor*. Så sætter vi

$$u^T v = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = ax + by = \langle u, v \rangle.$$

Denne notation har den fordel, at vi kan skrive mange indre produkter på en gang: Givet en til vektor  $\tilde{u} = \begin{bmatrix} c \\ d \end{bmatrix}$  kan vi stable  $u^T$  og  $\tilde{u}^T$  ovenpå hinanden og danne matricen

$$A = \begin{bmatrix} u^T \\ \tilde{u}^T \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Vi kan nu stable deres indre produkter med  $v$ :

$$Av = \begin{bmatrix} u^T \\ \tilde{u}^T \end{bmatrix} v = \begin{bmatrix} u^T v \\ \tilde{u}^T v \end{bmatrix}, \quad (2.3)$$

dvs.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}. \quad (2.4)$$

Operationen  $v \mapsto Av$  tager så et punkt i planen  $(x, y)$  og transformerer den til et nyt punkt  $(ax + by, cx + dy)$ . Forskellige matricer  $A$  giver forskellige transformationer. For at forstå disse transformationer, er det en ide at kikke på hvad transformationer gør med flere punkter. Givet vektorer

$$v_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \quad v_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \quad v_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \quad \dots, \quad v_{n-1} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \end{bmatrix}$$



## 2.2 INDRE PRODUKT OG MATRIXTRANSFORMATIONER

kan disse samles i en matrix

$$[v_0 \mid v_1 \mid v_2 \mid \dots \mid v_{n-1}] = \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_{n-1} \\ y_0 & y_1 & y_2 & \dots & y_{n-1} \end{bmatrix}.$$

Så kan deres indre produkter med  $u$  samles til en rækkevektor

$$\begin{aligned} u^T [v_0 \mid v_1 \mid v_2 \mid \dots \mid v_{n-1}] &= [u^T v_0 \quad u^T v_1 \quad u^T v_2 \quad \dots \quad u^T v_{n-1}] \\ &= [a \quad b] \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_{n-1} \\ y_0 & y_1 & y_2 & \dots & y_{n-1} \end{bmatrix} \\ &= [ax_0 + by_0 \quad ax_1 + by_1 \quad ax_2 + by_2 \quad \dots \quad ax_{n-1} + by_{n-1}]. \end{aligned}$$

Tilsvarende kan vi samle effekten af transformation  $v \mapsto Av$  på  $v_0, \dots, v_{n-1}$  til en matrix med søjler  $Av_0, \dots, Av_{n-1}$

$$A[v_0 \mid v_1 \mid v_2 \mid \dots \mid v_{n-1}] = [Av_0 \mid Av_1 \mid Av_2 \mid \dots \mid Av_{n-1}] \quad (2.5)$$

dvs.

$$\begin{aligned} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_{n-1} \\ y_0 & y_1 & y_2 & \dots & y_{n-1} \end{bmatrix} \\ = \begin{bmatrix} ax_0 + by_0 & ax_1 + by_1 & ax_2 + by_2 & \dots & ax_{n-1} + by_{n-1} \\ cx_0 + dy_0 & cx_1 + dy_1 & cx_2 + dy_2 & \dots & cx_{n-1} + dy_{n-1} \end{bmatrix}. \end{aligned}$$

Lad os nu give nogle eksempler på sådanne transformationer.

### Skalering

$$S = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}, \quad r > 0.$$

Dette sender  $\begin{bmatrix} x \\ y \end{bmatrix}$  til  $\begin{bmatrix} rx \\ ry \end{bmatrix} = r \begin{bmatrix} x \\ y \end{bmatrix}$ , se figur 2.3.

Skalering med faktor  $r = 1$  ændrer ikke på punktet  $(x, y)$ ;  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$ .  
Matricen

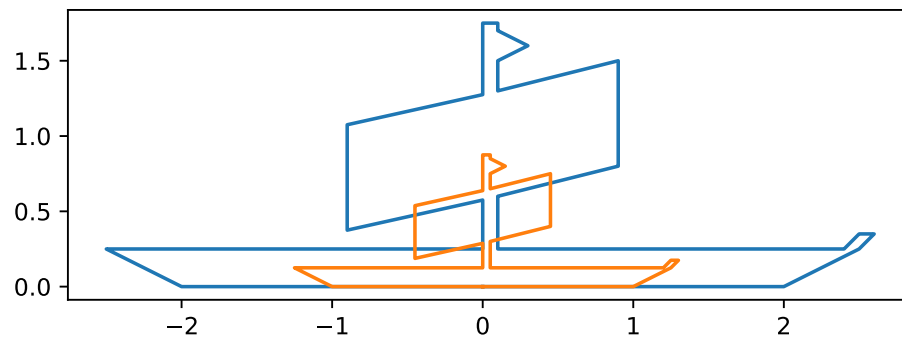
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

kaldes *identitetsmatricen*.

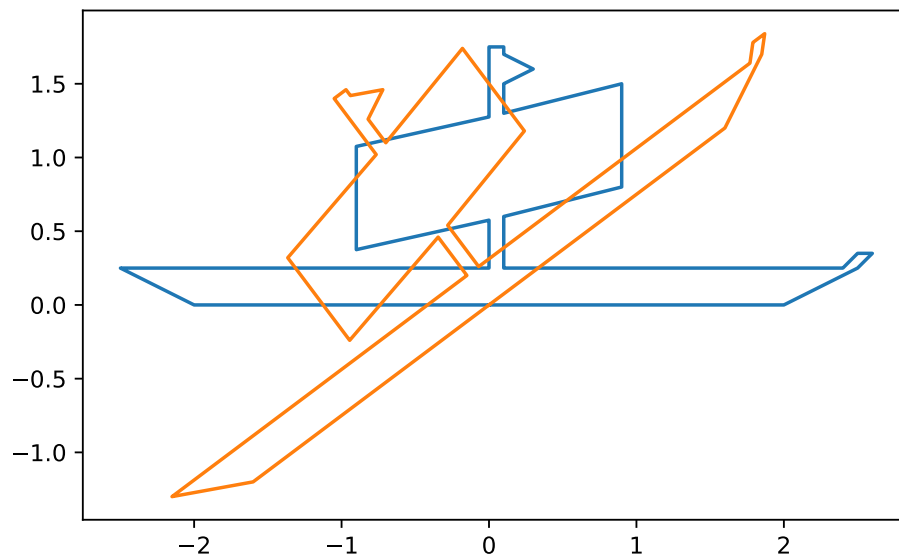
Ønsker man skalering med forskellige faktorer,  $r_1$  i  $x$ -retningen og  $r_2$  i  $y$ -retningen, bruger man matricen

$$\begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix}.$$

## 2 PLANGEOMETRI: VEKTORER OG MATRICER



Figur 2.3: Skalering med faktor  $r = 0,5$ .



Figur 2.4: Rotation med  $c = 0,8$ ,  $s = \sqrt{1 - c^2} = 0,6$ .

### Rotation

$$R = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \quad c^2 + s^2 = 1$$

Dette sender  $e_0$  til  $(c, s)$ , og  $e_1$  til  $(-s, c)$ . Det er en rotation igennem vinklen  $\theta$ , som opfylder  $\cos(\theta) = c$ ,  $\sin(\theta) = s$ , se figur 2.4. I mange anvendelser har vi ikke brug for at beregne vinklen  $\theta$ .

## 2.2 INDRE PRODUKT OG MATRIXTRANSFORMATIONER

*Bemærkning 2.1.* Hvis man skal bestemme  $\theta$ , så kan funktionen `np.arctan2` fra NumPy pakken bruges. For en vektor  $v = \begin{bmatrix} x \\ y \end{bmatrix}$  er vinklen  $\theta$  givet via `np.arctan2(y, x)`, pas på rækkefølgen!

```
>>> import numpy as np
>>> np.arctan2(0, 1)
0.0
>>> np.arctan2(1, 0)
1.5707963267948966
>>> np.arctan2(1, 1)
0.7853981633974483
```

Svaret er i radianer. Ønsker man at vide hvilke multiplum af  $\pi$  det er skal man dele med `np.pi`. F.eks.

```
>>> np.pi
3.141592653589793
>>> np.arctan2(1, 0)/np.pi
0.5
>>> np.arctan2(1, 1)/np.pi
0.25
```

svarende til at  $e_1 = (0, 1)$  har vinkel  $\pi/2$  fra  $e_0$  og at  $(1, 1)$  har vinkel  $\pi/4$ .  $\diamond$

### Spejling

$$M = \begin{bmatrix} c & s \\ s & -c \end{bmatrix}, \quad c^2 + s^2 = 1.$$

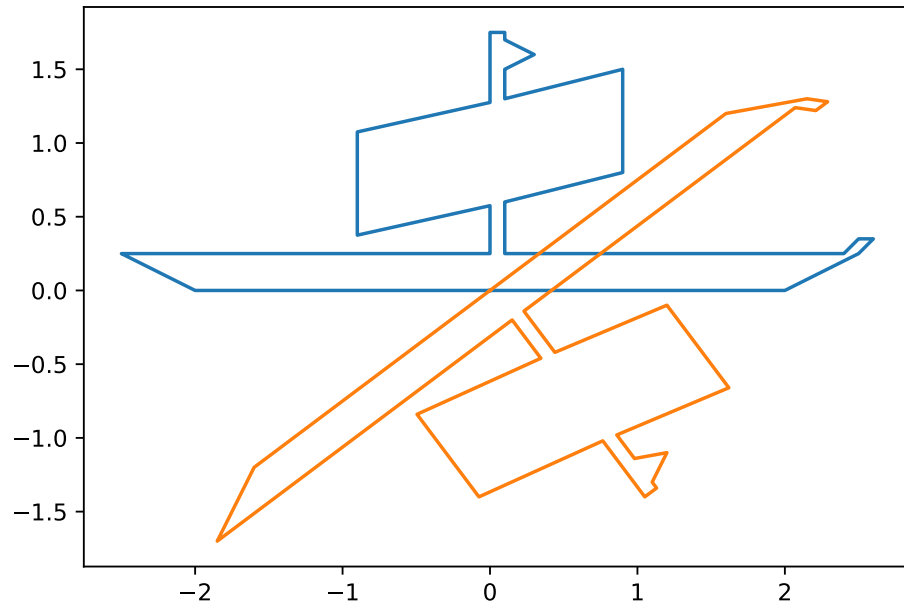
Bemærk fortegnsforskel fra rotationsmatricen; her ligger minusfortegnet på et element på diagonalen. Hvis  $(c, s) = (\cos \theta, \sin \theta)$ , så giver  $M$  en spejling i linjer igennem origo med vinkel  $\theta/2$  til  $x$ -aksen, se figur 2.5.

### Projektion

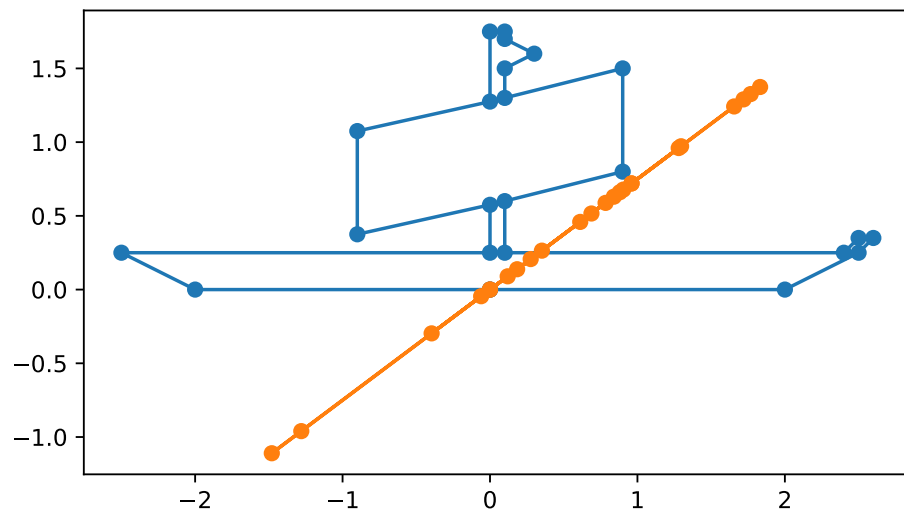
$$P = \begin{bmatrix} p^2 & pq \\ pq & q^2 \end{bmatrix}, \quad p^2 + q^2 = 1.$$

Dette sender alle punkter til punkter på linjen igennem origo og  $(p, q)$ , se figur 2.6. For  $q = 0$ , er  $P = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$  og  $P \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}$ , så  $y$ -koordinatet sættes til 0, i dette tilfælde.

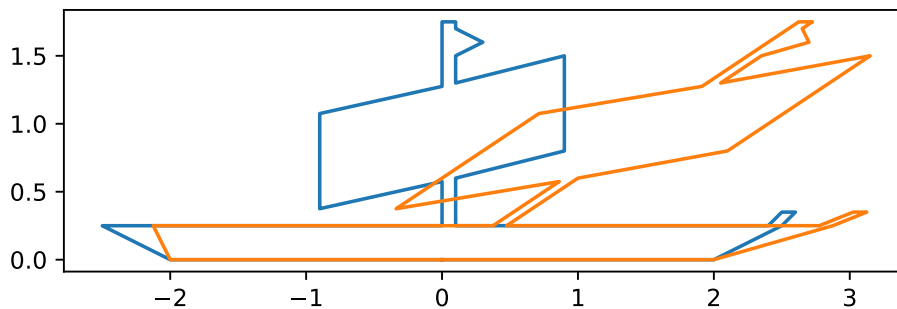
## 2 PLANGEOMETRI: VEKTORER OG MATRICER



Figur 2.5: Spejling med  $c = 0,8$ ,  $s = \sqrt{1 - c^2} = 0,6$ .



Figur 2.6: Projektion med  $p = 0,8$ ,  $q = \sqrt{1 - p^2} = 0,6$ .

Figur 2.7: Skævvridning med  $t = 1,5$ .**Skævvridning**

$$K = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix}$$

forskubber punkter til højre ved en størrelse bestemt af  $y$ -koordinatet:  $K \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x+ty \\ y \end{bmatrix}$ , se figur 2.6.

**2.3 Første tegninger**

Lad os se hvordan man kan lave tegninger som ovenfor med python. Vi bruger pakken `matplotlib.pyplot` samt `numpy`:

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

Lad os begynder med nogle koordinatpar, som f.eks.

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} 0,2 & 0,2 & -0,2 & 0,3 \\ -0,2 & 0,8 & 0,1 & 0,1 \end{bmatrix}.$$

Vi kan gemme  $x$  og  $y$  koordinater i to `np.array`:

```
>>> x = np.array([ 0.2, 0.2, -0.2, 0.3])
>>> y = np.array([-0.2, 0.8, 0.1, 0.1])
```

Nu tegner vi linjen, som begynder i  $(x_0, y_0)$  og kører videre igennem  $(x_1, y_1)$ ,  $(x_2, y_2)$  og hen til  $(x_3, y_3)$ .

## 2 PLANGEOMETRI: VEKTORER OG MATRICER

```
>>> fig, ax = plt.subplots()
>>> ax.set_aspect('equal')
>>> ax.plot(x, y)
[<matplotlib.lines.Line2D object at 0x12005f7c0>]
```

Afhængig af opsætningen, får man enten vist billedet straks eller man skal bede om at få den vist via

```
fig.show()
```

Lad os nu flytte vores tegning via en matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1,0 & 0,5 \\ 0,0 & 1,0 \end{bmatrix}.$$

Vi kan bruge formelen (2.4) næsten direkte; vores nye  $x$ -koordinater er  $ax+by = a*x+b*y$ , og den nye  $y$ -koordinater er  $cx+dy = c*x+d*y$ :

```
>>> a = 1.0
>>> b = 0.5
>>> c = 0.0
>>> d = 1.0
>>> fig, ax = plt.subplots()
>>> ax.set_aspect('equal')
>>> ax.plot(a*x+b*y, c*x+d*y)
[<matplotlib.lines.Line2D object at 0x12009fbe0>]
```

som vises frem på samme måde som ovenfor.

Figuren kan gemmes i en pdf fil via

```
fig.savefig('figure-name.pdf')
```

Man kan give tegningen en anden farve og/eller markere punkter ved f.eks.

```
ax.plot(a*x+b*y, c*x+d*y, color='r') #r = rød
ax.plot(a*x+b*y, c*x+d*y, marker='o') #o = cirkler/prikker
```

Plotgrænserne kan angives med

## 2.3 FØRSTE TEGNINGER

```
ax.set_xlim(-3.0, 3.0)  
ax.set_ylim(-2.0, 4.0)
```





# Kapitel 3

## Matricer og vektorer

### 3.1 Matricer

En *matrix* er et rektangulært skema af tal, som for eksempel

$$A = \begin{bmatrix} 3,1 & -2,1 & 4,0 & -1,6 \\ 7,2 & 3,6 & -2,7 & 11,3 \\ -5,6 & -1,2 & 3,5 & -17,2 \end{bmatrix}.$$

En matrix med  $m$  rækker og  $n$  søjler kaldes en  $(m \times n)$ -matrix. Tallene der kan bruges i matricen kaldes *skalarer*. Tallet der er placeret i række  $i$ , søjle  $j$ , kaldes den  $(i, j)$ te indgang i matricen; for en matrix  $A$  skriver vi  $a_{ij}$  eller  $A_{[i,j]}$  for denne indgang. Mht. konventionerne i python begynder nummerering ved  $i = 0, j = 0$ . Så i eksemplet overfor er  $A$  en  $(3 \times 4)$ -matrix, med  $a_{00} = 3,1$  og  $a_{13} = 11,3$ . Generelt, har vi

$$A = (a_{ij}) = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0,n-1} \\ a_{10} & a_{11} & \dots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1} & \dots & a_{m-1,n-1} \end{bmatrix}.$$

Mængden af alle  $(m \times n)$ -matricer med reelle indgange skrives  $\mathbb{R}^{m \times n}$ .

To matricer  $A, B$  er *lige med hinanden* hvis de har samme størrelse  $(m \times n)$  og samme indgange, dvs.  $a_{ij} = b_{ij}$  for alle  $i = 0, \dots, m-1, j = 0, \dots, n-1$ .

## 3.2 Matricer i python

Pakken NumPy har en fundamental indbygget objekt `ndarray` for matricer (og vektorer og tensorer). Matricer oprettes med kommandoen `np.array`

```
>>> import numpy as np
>>> a = np.array([[3.1, -2.1, 4.0, -1.6],
...               [7.2, 3.6, -2.7, 11.3],
...               [-5.6, -1.2, 3.5, -17.2]])
>>> a
array([[ 3.1, -2.1,  4. , -1.6],
       [ 7.2,  3.6, -2.7, 11.3],
       [-5.6, -1.2,  3.5, -17.2]])
```

(prikkerne `...` i begyndelsen af linjen skrives ikke ind, de viser bare at input-linjen begyndende med `>>>` fortsætter). Dette er en  $3 \times 4$  matrix

```
>>> a.shape
(3, 4)
```

I python siger man at `a` har 2 *akser*. Antallet af akser fås fra `ndim`

```
>>> a.ndim
2
```

Vi får adgang til enkelte indgange i `a` via `a[2, 3]` osv. Så de overnævnte indgange er

```
>>> a[0,0]
3.1
>>> a[1,3]
11.3
```

En hel søjle af `a` fås via `a[:, [2]]`, en række fås via `a[[1], :]`

```
>>> a[:, [2]] #søjle
array([[ 4. ],
       [-2.7],
```

## 3.2 MATRICER I PYTHON

```
[ 3.5]])  
>>> a[[1], :] #række  
array([[ 7.2,  3.6, -2.7, 11.3]])
```

Bemærk at der bruges ekstra parentes; glemmes disse får man resultater med kun én akse

```
>>> a[:, 2] #indgange i en søjle  
array([ 4. , -2.7,  3.5])  
>>> a[:, 2].ndim  
1  
>>> a[1, :] #indgange i en række  
array([ 7.2,  3.6, -2.7, 11.3])  
>>> a[1, :].ndim  
1
```

Indekserne begynder ved 0 og tæller op ad, men negative indekser kan bruges til at tælle ned fra den største værdi

```
>>> a[[-1], :] #sidste række  
array([[ -5.6,  -1.2,   3.5, -17.2]])  
>>> a[:, [-2]] #næstsidste søjle  
array([[ 4. ],  
       [-2.7],  
       [ 3.5]])
```

Elementerne i en ndarray har alle den samme datatype, dtype. Dette betyder at hvert element tager lige meget plads i computerens hukommelse, og gøre det nemmere og hurtigere for computeren til at finde et givent element. Vi har

```
>>> a.dtype  
dtype('float64')
```

Denne datatype bruges automatisk af python når mindst en indgang i a er givet som decimalbrøk. For at sikre, man får den rigtig datatype, kan man angive datatypen når matricen oprettes

### 3 MATRICER OG VEKTORER

```
>>> b = np.array([[1, 2],
...               [3, 7]], dtype='float')
>>> b
array([[1., 2.],
       [3., 7.]])
>>> b.dtype
dtype('float64')
```

En liste over forskellige datatypes NumPy kender som standard findes i begyndelsen af kapitel 4 af *NumPy user guide* (the NumPy community 2020). For det meste holder vi os til `float` typen. Lidt senere vil vi have brug for `complex` typen.

## 3.3 Heatmap plot

En nyttig metode for at visualisere en matrix er at farvelægge hver indgang med en farve bestemt af dens værdi. For dette kan man med fordel bruge `matshow` fra `matplotlib`.

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

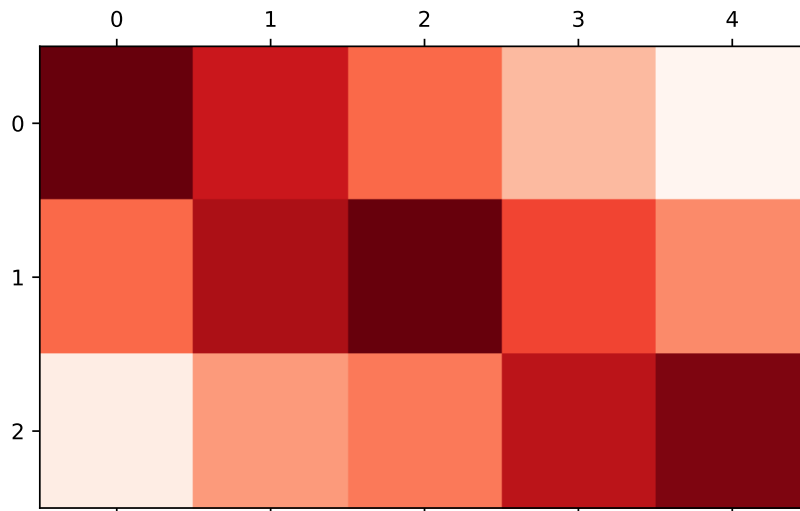
a = np.array([[1.0, 0.5, 0.0, -0.5, -1.0],
              [0.0, 0.7, 1.0, 0.2, -0.2],
              [-0.9, -0.3, -0.1, 0.6, 0.9]])

fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.matshow(a, cmap='Reds')
```

Som giver figur 3.1.

Argumentet `cmap` angiver hvilke farve skala der skal anvendes. Navne som `'Reds'`, `'Blues'` eller `'Greys'` bruge en stærk farve for høje værdier og hvid for de lavste værdier. Det kan være nyttig at tilføje oplysning om hvilke farver svarer til hvilke værdi, og det vises fint med en colorbar.

### 3.3 HEATMAP PLOT



Figur 3.1: En første heatmap plot.

```
fig, ax = plt.subplots()
ax.set_aspect('equal')
im = ax.matshow(a, cmap='Blues')
fig.colorbar(im, shrink=0.65)
```

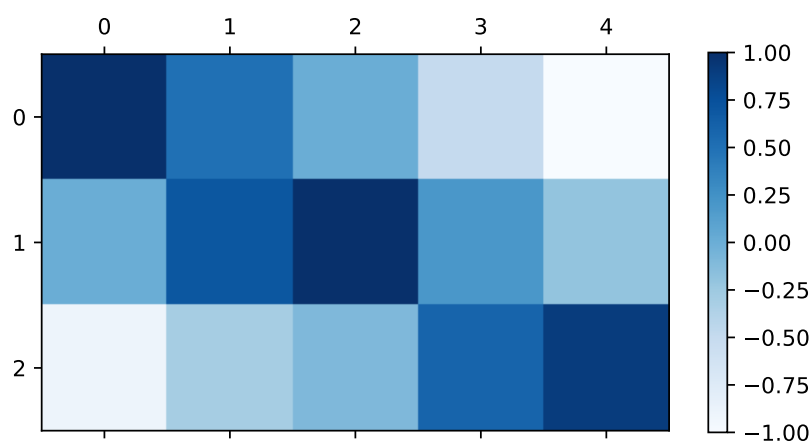
Størrelsen på colorbar styres af argumentet `shrink`, som skalere bjælken med den givne faktor.

Man kan justere hvilke værdier dækkes af farveskalaen, ved at sætte `clim`, f.eks. `clim = (-1.0, 1.0)` siger at farveskalaen skal bruges til at dække værdierne fra  $-1,0$  til  $1,0$ .

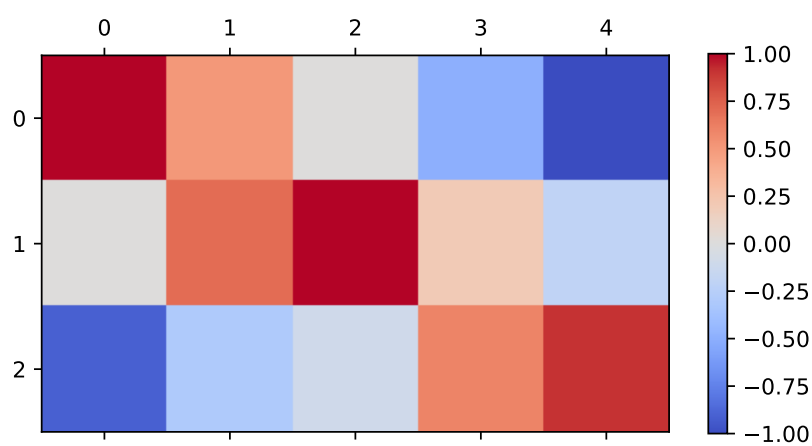
```
fig, ax = plt.subplots()
ax.set_aspect('equal')
im = ax.matshow(a, cmap='coolwarm', clim = (-1.0, 1.0))
fig.colorbar(im, shrink=0.65)
```

Her har vi brugt en farveskala der går fra blå til rød. Ved at bestemme grænser `clim = (-1.0, 1.0)` som er symmetrisk omkring  $0,0$ , opnås at negative værdier får en blå nuance, og positive værdier er rødlig nuance.

### 3 MATRICER OG VEKTORER



Figur 3.2: Heatmap med colorbar.



Figur 3.3: Heatmap med justeret farveskala.

### 3.4 Multiplikation med en skalar og matrixsum

Givet en matrix  $A \in \mathbb{R}^{m \times n}$  som i afsnit 3.1 og en skalar  $s \in \mathbb{R}$  defineres *multiplikation af  $A$  med en skalar  $s$*  som  $(m \times n)$ -matricen  $sA$  hvor  $s$  er ganget ind på alle indgange i  $A$  enkeltvis:

$$sA = s(a_{ij}) := (sa_{ij}) = \begin{bmatrix} sa_{00} & sa_{01} & \dots & sa_{0,n-1} \\ sa_{10} & sa_{11} & \dots & sa_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ sa_{m-1,0} & sa_{m-1,1} & \dots & sa_{m-1,n-1} \end{bmatrix}.$$

I python er operationen givet via operatoren `*`

```
>>> a
array([[ 3.1, -2.1,  4. , -1.6],
       [ 7.2,  3.6, -2.7, 11.3],
       [-5.6, -1.2,  3.5, -17.2]])
>>> s = 2.0
>>> s * a
array([[ 6.2, -4.2,  8. , -3.2],
       [14.4,  7.2, -5.4, 22.6],
       [-11.2, -2.4,  7. , -34.4]])
```

I tilfældet hvor  $s = 0$ , er  $sA = 0A = 0_{m \times n}$  *nulmatricen* i  $\mathbb{R}^{m \times n}$ , hvor alle indgange er 0. Hvis størrelsen af nulmatricen er underordnet, skriver vi blot 0 i stedet for  $0_{m \times n}$ . Funktionen `np.zeros()` giver en nulmatrix af en given størrelse; størrelsen specificeres som tuple, som f.eks. `(2, 3)`

```
>>> np.zeros((2, 3))
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Givet to matricer  $A, B$  af samme størrelse  $m \times n$ , defineres deres *matrixsum*  $A + B \in \mathbb{R}^{m \times n}$  ved at dens  $(i, j)$ -indgang er summen af de tilsvarende  $(i, j)$ -

### 3 MATRICER OG VEKTORER

indgange i  $A$  og  $B$ :

$$A + B = (a_{ij}) + (b_{ij}) := (a_{ij} + b_{ij})$$

$$= \begin{bmatrix} a_{00} + b_{00} & a_{01} + b_{01} & \dots & a_{0,n-1} + b_{0,n-1} \\ a_{10} + b_{10} & a_{11} + b_{11} & \dots & a_{1,n-1} + b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} + b_{m-1,0} & a_{m-1,1} + b_{m-1,1} & \dots & a_{m-1,n-1} + b_{m-1,n-1} \end{bmatrix}.$$

Dette gøres med almindelig plus operatoren  $+$  i python

```
>>> a
array([[ 3.1, -2.1,  4. , -1.6],
       [ 7.2,  3.6, -2.7, 11.3],
       [-5.6, -1.2,  3.5, -17.2]])
>>> b = np.array([[2.0, 1.0, 0.0, -1.0],
...               [2.0, 1.0, 0.0, -1.0],
...               [2.0, 1.0, 0.0, -1.0]])
>>> b
array([[ 2.,  1.,  0., -1.],
       [ 2.,  1.,  0., -1.],
       [ 2.,  1.,  0., -1.]])
>>> a + b
array([[ 5.1, -1.1,  4. , -2.6],
       [ 9.2,  4.6, -2.7, 10.3],
       [-3.6, -0.2,  3.5, -18.2]])
```

Tilsvarende er *differencen* af to  $(m \times n)$ -matricer

$$A - B = (a_{ij}) - (b_{ij}) := (a_{ij} - b_{ij})$$

og i python bruges almindelig minus  $-$ .

**Proposition 3.1.** *Skalar multiplikation og matrix sum har følgende regneregler:*

- (a)  $0A = 0_{m \times n}$  og  $1A = A$ , for  $0, 1 \in \mathbb{R}$ ,  $A \in \mathbb{R}^{m \times n}$ ,
- (b)  $A + 0_{m \times n} = A$ , for  $A, 0_{m \times n} \in \mathbb{R}^{m \times n}$ ,
- (c)  $(s + t)A = sA + tA$ , for  $s, t \in \mathbb{R}$ ,  $A \in \mathbb{R}^{m \times n}$ ,
- (d)  $s(A + B) = sA + sB$ , for  $s \in \mathbb{R}$ ,  $A, B \in \mathbb{R}^{m \times n}$ ,
- (e)  $A + B = B + A$ , for  $A, B \in \mathbb{R}^{m \times n}$ ,



- (f)  $A - B = A + (-1)B$ , for  $A, B \in \mathbb{R}^{m \times n}$ ,  
 (g)  $A + (B + C) = (A + B) + C$ , for  $A, B, C \in \mathbb{R}^{m \times n}$ .

*Bevis.* Alle resultater checkes ved at sammenligne tilsvarende indgange på begge sider.  $\square$

Skalarmultiplikation fra højre side defineres ved

$$As = (a_{ij})s := (a_{ij}s).$$

Da  $a_{ij}s = sa_{ij}$ , har vi

$$As = sA.$$

## 3.5 Transponering

Transponering er operationen  $A \rightarrow A^T$  på matricer der ombytter rækker og søjler. Hvis  $A \in \mathbb{R}^{m \times n}$ , så er  $A^T \in \mathbb{R}^{n \times m}$

```
>>> a
array([[ 3.1, -2.1,  4. , -1.6],
       [ 7.2,  3.6, -2.7, 11.3],
       [-5.6, -1.2,  3.5, -17.2]])
>>> a.T
array([[ 3.1,  7.2, -5.6],
       [-2.1,  3.6, -1.2],
       [ 4. , -2.7,  3.5],
       [-1.6, 11.3, -17.2]])
```

Mere visuelt har vi følgende

$$A = \begin{bmatrix} 3,1 & -2,1 & 4,0 & -1,6 \\ 7,2 & 3,6 & -2,7 & 11,3 \\ -5,6 & -1,2 & 3,5 & -17,2 \end{bmatrix} \mapsto \begin{bmatrix} 3,1 & 7,2 & -5,6 \\ -2,1 & 3,6 & -1,2 \\ 4,0 & -2,7 & 3,5 \\ -1,6 & 11,3 & -17,2 \end{bmatrix} = A^T$$

Matricen  $A$  bliver spejlet i diagonalen som består af de grønne indgange. I indeksnotation har vi

$$A = (a_{ij}) \mapsto A^T = (a_{ji}).$$

### 3.6 Vektorer

En *søjlevektor*  $v \in \mathbb{R}^n$  er en  $(n \times 1)$ -matrix, som vi skriver

$$v = (v_i) = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix}.$$

I tekst skriver vi  $v = (v_0, v_1, \dots, v_{n-1})$ . Da  $v$  er en  $(n \times 1)$ -matrix en matrix, vi kan bruge de operationerne i afsnit 3.4 til at få skalarmultiplikation og vektorsum

$$sv = s(v_i) := (sv_i) = \begin{bmatrix} sv_0 \\ sv_1 \\ \vdots \\ sv_{n-1} \end{bmatrix},$$

$$u + v = (u_i) + (v_i) := (u_i + v_i) = \begin{bmatrix} u_0 + v_0 \\ u_1 + v_1 \\ \vdots \\ u_{n-1} + v_{n-1} \end{bmatrix}.$$

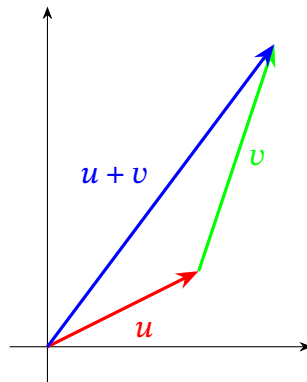
Som i kapitel 2 kan en vektor tænkes som en pil fra origo til punktet med koordinater  $(v_0, v_1, \dots, v_{n-1})$ . Vektorsum svarer til at sætte to piler efter hinanden, som i figur 3.4.

I python kan vi enten skriver en søjlevektor via sammen metoden for en  $(n \times 1)$ -matrix

```
>>> v = np.array([[3.0], [4.0], [-0.7], [6.2]])
>>> v
array([[ 3. ],
       [ 4. ],
       [-0.7],
       [ 6.2]])
```

eller vi omformer en array med en akse

```
>>> np.array([3.0, 4.0, -0.7, 6.2])[:, np.newaxis]
array([[ 3. ],
```



Figur 3.4: Vektorsum af  $u = (2, 1)$  og  $v = (1, 3)$ .

```
[ 4. ],
[-0.7],
[ 6.2]]])
```

Transponering af en søjlevektor  $v \in \mathbb{R}^n$  giver en *rækkevektor*  $v^T \in \mathbb{R}^{1 \times n}$ :

```
>>> v
array([[ 3. ],
       [ 4. ],
       [-0.7],
       [ 6.2]])
>>> v.T
array([[ 3. ,  4. , -0.7,  6.2]])
```



# Kapitel 4

## Matrix multiplikation

I afsnit 2.2 så vi hvordan man kan bruge en  $(2 \times 2)$ -matrix  $A$  til at flytte på vektorer  $v$  i  $\mathbb{R}^2$ , via  $v \mapsto Av$ , se (2.3). Ved at stille flere vektorer op efter hinanden fik så vi også (2.5) hvordan  $A$  flytter en  $(2 \times n)$ -matrix  $[v_0 \mid v_1 \mid v_2 \mid \dots \mid v_{n-1}]$  til  $[Av_0 \mid Av_1 \mid Av_2 \mid \dots \mid Av_{n-1}]$ . Lad os nu udvide disse processer til matricer af vilkårlig størrelse.

### 4.1 Række-søjleprodukt

Vi begynder med et produkt for en rækkevektor med en søjlevektor. Hvis  $u^T \in \mathbb{R}^{1 \times n}$  og  $v \in \mathbb{R}^{n \times 1}$  er en rækkevektor og en søjlevektor *med det samme antal indgange* så er deres *række-søjleprodukt* givet ved

$$u^T v = [u_0 \quad u_1 \quad \dots \quad u_{n-1}] \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} = u_0 v_0 + u_1 v_1 + \dots + u_{n-1} v_{n-1}.$$

*Eksempel 4.1.* Et eksempel med  $n = 3$  er

$$\begin{bmatrix} 3 & -1 & 2 \end{bmatrix} \begin{bmatrix} 6 \\ 5 \\ -4 \end{bmatrix} = 3 \times 6 + (-1) \times 5 + 2 \times (-4) \\ = 18 - 5 - 8 = 5.$$

Δ

#### 4 MATRIX MULTIPLIKATION

*Eksempel 4.2.* For  $u^T = [1 \ 1 \ \dots \ 1] \in \mathbb{R}^{1 \times n}$  er

$$\begin{aligned} u^T v &= [1 \ 1 \ \dots \ 1] \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} = 1 \times v_0 + 1 \times v_1 + \dots + 1 \times v_{n-1} \\ &= v_0 + v_1 + \dots + v_{n-1} \end{aligned}$$

summen af indgangene i  $v$ . Ligeledes, for

$$w^T = (1/n)u^T = [1/n \ 1/n \ \dots \ 1/n] \in \mathbb{R}^{1 \times n}$$

er

$$w^T v = \frac{1}{n}(v_0 + v_1 + \dots + v_{n-1}),$$

som er middelværdien af indgangene i  $v$ . Δ

*Eksempel 4.3.* Betragt et polynomium

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1}.$$

Vi gemmer koefficienterne  $a_i$  i en rækkevektor  $u^T$ :

$$u^T = [a_0 \ a_1 \ \dots \ a_{n-1}] \in \mathbb{R}^{1 \times n}.$$

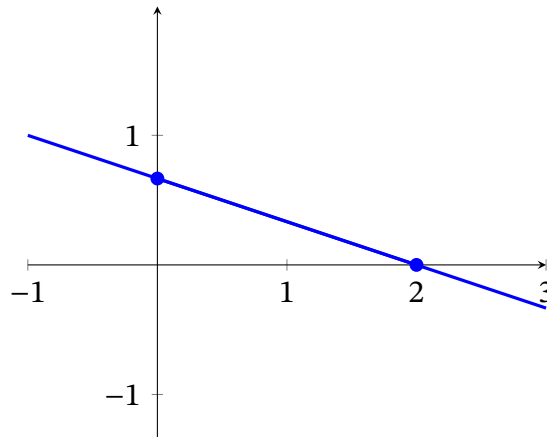
Givet et  $x_0$  kan vi betragte Vandermondevektoren

$$v = \begin{bmatrix} 1 \\ x_0 \\ x_0^2 \\ \vdots \\ x_0^{n-1} \end{bmatrix} \in \mathbb{R}^{n \times 1}.$$

Række-søjleproduktet af  $u^T$  med  $v$  er så

$$\begin{aligned} u^T v &= [a_0 \ a_1 \ \dots \ a_{n-1}] \begin{bmatrix} 1 \\ x_0 \\ x_0^2 \\ \vdots \\ x_0^{n-1} \end{bmatrix} \\ &= a_0 + a_1 x_0 + a_2 x_0^2 + \dots + a_{n-1} x_0^{n-1} = p(x_0) \end{aligned}$$

evaluering af  $p(x)$  i  $x = x_0$ . Δ

Figur 4.1: Linjen  $x + 3y = 2$ .

*Eksempel 4.4.* Givet en rækkevektor  $u^T = [a \ b] \in \mathbb{R}^{1 \times 2}$  som er ikke nul, og en værdi  $c \in \mathbb{R}$ , kan vi kikke på punkterne  $(x, y)$  således at  $v = \begin{bmatrix} x \\ y \end{bmatrix}$  opfylder  $u^T v = c$ . Skrives dette ud får vi

$$ax + by = c$$

som er ligningen for en ret linje i  $\mathbb{R}^2$ . F.eks. for  $(a, b) = (1, 3)$  og  $c = 2$  har vi

$$x + 3y = 2.$$

Sættes  $y = 0$ , fås  $x = 2$ , så denne linje skær  $x$ -aksen i punktet  $(2, 0)$ . Tilsvarende hvis vi sætter  $x = 0$  fås  $y = 2/3$ , så linjen går også igennem  $(0, 2/3)$ , se figur 4.1.

Tilsvarende for  $u^T = [a \ b \ c] \in \mathbb{R}^{1 \times 3}$ ,  $d \in \mathbb{R}$  og  $v = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^{3 \times 1}$ , er  $u^T v = d$  planen

$$ax + by + cz = d.$$

△

## 4.2 Matrixprodukt

## 4 MATRIX MULTIPLIKATION

**Definition 4.5.** For to matricer  $A \in \mathbb{R}^{m \times n}$  og  $B \in \mathbb{R}^{n \times r}$  defineres deres *matrixprodukt* til at være matricen  $AB \in \mathbb{R}^{m \times r}$  hvis  $(i, j)$ -indgang er række-søjleproduktet af rækken  $i$  fra  $A$  med søjlen  $j$  fra  $B$ . Dvs.  $C = AB$  har  $(i, j)$ -indgang

$$c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{i,n-1}b_{n-1,j}.$$

*Eksempel 4.6.* For et eksempel med  $m = 2$ ,  $n = 3$ ,  $r = 4$  tager vi

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 0 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{2 \times 3} \quad \text{og} \quad B = \begin{bmatrix} 6 & 1 & 0 & 0 \\ 5 & 1 & 1 & 2 \\ -4 & 1 & 0 & 2 \end{bmatrix} \in \mathbb{R}^{3 \times 4}.$$

Så er produktet  $AB$  givet ved

$$AB = \begin{bmatrix} 3 & -1 & 2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 6 & 1 & 0 & 0 \\ 5 & 1 & 1 & 2 \\ -4 & 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 4 & -1 & 2 \\ 1 & 2 & 1 & 4 \end{bmatrix}.$$

F.eks. indgangen nederst til højre er række-søjleproduktet

$$\begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix} = 0 \times 0 + 1 \times 2 + 1 \times 2 = 4.$$

Bemærk at for disse matricer er produktet  $BA$  ikke defineret:  $B \in \mathbb{R}^{3 \times 4}$  og  $A \in \mathbb{R}^{2 \times 3}$ , men  $4 \neq 2$ . △

I python er der naturligvis en funktion der vil beregne matrixproduktet for os, givet ved operatoren `@`. Så eksemplet ovenfor er

```
>>> import numpy as np
>>> a = np.array([[3.0, -1.0, 2.0],
...               [0.0, 1.0, 1.0]])
>>> b = np.array([[6.0, 1.0, 0.0, 0.0],
...               [5.0, 1.0, 1.0, 2.0],
...               [-4.0, 1.0, 0.0, 2.0]])
>>> a @ b
array([[ 5.,  4., -1.,  2.],
       [ 1.,  2.,  1.,  4.]])
```

Python er også enig at produktet  $BA$  er ikke defineret for disse to matricer



```

>>> b @ a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: matmul: Input operand 1 has a mismatch in its
  ↳ core dimension 0, with gufunc signature
  ↳ (n?,k),(k,m?)->(n?,m?) (size 2 is different from 4)

```

**Advarsel 4.7.** Matrix multiplikation er *ikke* givet ved  $*$ , man skal bruge python's  $@$  operator. !

Matrixproduktet kan fortolkes på forskellige måde. Bemærk først at hvis vi deler  $A$  op i dens  $m$  rækker, og  $B$  op i dens  $r$  søjler har vi tre forskellige måde at se på produktet:

$$\begin{aligned}
 AB &= \begin{bmatrix} a_0^T \\ a_1^T \\ a_2^T \\ \vdots \\ a_{m-1}^T \end{bmatrix} [b_0 \mid b_1 \mid b_2 \mid \dots \mid b_{r-1}] \\
 &= \begin{bmatrix} a_0^T b_0 & a_0^T b_1 & a_0^T b_2 & \dots & a_0^T b_{r-1} \\ a_1^T b_0 & a_1^T b_1 & a_1^T b_2 & \dots & a_1^T b_{r-1} \\ a_2^T b_0 & a_2^T b_1 & a_2^T b_2 & \dots & a_2^T b_{r-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1}^T b_0 & a_{m-1}^T b_1 & a_{m-1}^T b_2 & \dots & a_{m-1}^T b_{r-1} \end{bmatrix} \quad (4.1)
 \end{aligned}$$

$$= [Ab_0 \mid Ab_1 \mid Ab_2 \mid \dots \mid Ab_{r-1}] \quad (4.2)$$

$$= \begin{bmatrix} a_0^T B \\ a_1^T B \\ a_2^T B \\ \vdots \\ a_{m-1}^T B \end{bmatrix}. \quad (4.3)$$

## 4 MATRIX MULTIPLIKATION

Udtrykket (4.1) er selve definitionen på matrixprodukt, som en matrix af række-søjleprodukter.

I (4.2), ser vi at søjlerne af  $AB$  består af  $A$  ganget på søjlerne af  $B$ . Hvert søjle  $b_i$  af  $B$  er et punkt i  $\mathbb{R}^n$ , og  $Ab_i$  tager dette punkt og giver et nyt punkt i  $\mathbb{R}^m$ . Produktet  $AB$  er så effekten af transformationen  $v \mapsto Av$ ,  $\mathbb{R}^n \rightarrow \mathbb{R}^m$ , på søjlerne af  $B$ . For  $m = n = 2$ , er det dette syn vi havde på matrixprodukt i afsnit 2.2.

Her er et par eksempler i denne stil.

*Eksempel 4.8.* Lad

$$R_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \quad \text{med } c^2 + s^2 = 1.$$

Så er  $R_0$  en rotationsmatrix udvidet til en  $(3 \times 3)$ -matrix. På et punkt  $v = (x, y, z) \in \mathbb{R}^3$  er

$$R_0 v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ cy - sz \\ sy + cz \end{bmatrix},$$

som er igen et punkt i  $\mathbb{R}^3$ . Transformationen holder  $x$ -koordinatet fast, og udfører en rotation i  $(y, z)$ -planen.

Tilsvarende er

$$R_2 = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

en rotation i  $(x, y)$ -planen og

$$R_1 = \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix}$$

en rotation i  $(x, z)$ -planen. △

*Eksempel 4.9.* Betragt matricen

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

## 4.2 MATRIXPRODUKT

$P$  er en  $(2 \times 3)$ -matrix, så giver en transformation fra  $\mathbb{R}^3$  til  $\mathbb{R}^2$ . Mere præcist har vi for  $v = (x, y, z) \in \mathbb{R}^3$  at

$$Pv = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix},$$

dvs. at  $Pv$  består kun af de første to koordinater fra  $v$ . △

Ligeledes beskriver udtrykket (4.3) rækkerne i matrixproduktet  $AB$  som effekten af en transformation  $u^T \mapsto u^T B$ ,  $\mathbb{R}^{1 \times n} \rightarrow \mathbb{R}^{1 \times r}$ .

*Eksempel 4.10.* Følgende matrixprodukt

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & 0 & \dots & 0 \\ \vdots & & & \ddots & & \\ 1 & 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} v_0 \\ v_0 + v_1 \\ v_0 + v_1 + v_2 \\ \vdots \\ v_0 + v_1 + v_2 + \dots + v_{n-1} \end{bmatrix}$$

giver en løbende sum af indgangene i  $v$ . △




Især for matrix-vektorprodukter  $Ax \in \mathbb{R}^m$ , af  $A \in \mathbb{R}^{m \times n}$  og  $x \in \mathbb{R}^n$  er der to yderlige måde at anskue produktet. For det første, hvis vi betragte indgangene i  $x$  som variable og vælger et fast  $b \in \mathbb{R}^m$ , så er

$$Ax = b$$

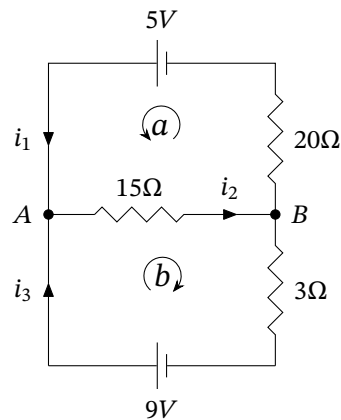
et lineært ligningssystem

$$\begin{array}{ccccccc} a_{00}x_0 + & a_{01}x_1 + \dots + & a_{0,n-1}x_{n-1} & = & b_0, \\ a_{10}x_0 + & a_{11}x_1 + \dots + & a_{1,n-1}x_{n-1} & = & b_1, \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m-1,0}x_0 + a_{m-1,1}x_1 + \dots + a_{m-1,n-1}x_{n-1} & = & b_{m-1}, \end{array}$$

bestående af  $m$  ligninger i de  $n$  variable  $x_0, x_1, \dots, x_{n-1}$ . Sådanne systemer optræder i mange forskellige sammenhæng.

*Eksempel 4.11.* Betragt det elektriske kredsløb givet i figur 4.2. Det består af to batterier  og tre elektriske modstander  forbundet som vist. Pilene  angiver den elektriske strøm i ledningen.

#### 4 MATRIX MULTIPLIKATION



Figur 4.2: Elektrisk kredsløb.

Den første lov af Kirchhoff fortæller at summen de elektriske strøm regnet med fortegn mod et knudepunkt er nul. Vores kreds har to knudepunkter markeret  $A$  og  $B$ . Kirchhoffs lov giver så to ligninger

$$\text{knudepunkt } A: \quad i_1 - i_2 + i_3 = 0,$$

$$\text{knudepunkt } B: \quad -i_1 + i_2 - i_3 = 0.$$

Kirchhoffs anden lov siger at sum af spændings fald over komponenter i hver delløkke af kredset er lige med spændingen fra batteriet, eller nul hvis der er ingen strømkilde i delkredsen. Spændingsfaldet  $V$  over en elektrisk modstand  $R$  er givet ved Ohms lov  $V = IR$ , hvor  $I$  er den elektriske strøm igennem komponenten. Vores figur har to lukkede delkredse, angivet som  $a$  og  $b$ , og vi har så

$$\text{løkke } a: \quad 20i_1 + 15i_2 = 5,$$

$$\text{løkke } b: \quad 15i_2 + 3i_3 = 9.$$

Vi har i alt 4 lineære ligninger i de 3 ubekendte  $i_1$ ,  $i_2$ ,  $i_3$ . Disse 4 ligninger kan skrives som matrixligningen

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 20 & 15 & 0 \\ 0 & 15 & 3 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 5 \\ 9 \end{bmatrix}.$$

△

### 4.3 REGNEREGLER FOR MATRIXPRODUKT

En anden måde at anskue produktet på, især når vi har en matrix-vektorprodukt  $Ax \in \mathbb{R}^m$ , af  $A \in \mathbb{R}^{m \times n}$  og  $x \in \mathbb{R}^n$ , er at dele  $A$  op i søjler og skrive  $x$  ud i koordinater

$$Ax = [v_0 \mid v_1 \mid v_2 \mid \dots \mid v_{n-1}] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = x_0 v_0 + x_1 v_1 + \dots + x_{n-1} v_{n-1}.$$

Dette er en *lineær kombination* af søjlerne  $v_0, v_1, \dots, v_{n-1}$  af  $A$ .

*Eksempel 4.12.* For

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{og} \quad x = \begin{bmatrix} -2 \\ 5 \\ 7 \end{bmatrix}$$

er

$$Ax = \begin{bmatrix} 3 & -1 & 2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 5 \\ 7 \end{bmatrix} = (-2) \begin{bmatrix} 3 \\ 0 \end{bmatrix} + 5 \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 7 \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

△

### 4.3 Regneregler for matrixprodukt

**Proposition 4.13.** *Matrixproduktet opfylder følgende regneregler i forhold til sum og skalarmultiplikation*

- (a)  $A(B + C) = AB + AC$ , for  $A \in \mathbb{R}^{m \times n}$ ,  $B, C \in \mathbb{R}^{n \times r}$ ,
- (b)  $(A + B)C = AC + BC$ , for  $A, B \in \mathbb{R}^{m \times n}$ ,  $C \in \mathbb{R}^{n \times r}$ ,
- (c)  $A(sB) = s(AB) = (sA)B$ , for  $s \in \mathbb{R}$ ,  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times r}$ . □

Især vigtigt er at

**Sætning 4.14.** For  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times r}$ ,  $C \in \mathbb{R}^{r \times p}$ ,

$$A(BC) = (AB)C.$$

*Bevis.* Skriv  $A = (a_{ij})$ ,  $B = (b_{jk})$ ,  $C = (c_{k\ell})$  for  $i \in \{0, \dots, m-1\}$ ,  $j \in \{0, \dots, n-1\}$ ,  $k \in \{0, \dots, r-1\}$ ,  $\ell \in \{0, \dots, p-1\}$ . Så er den  $(i, \ell)$ te indgang af  $A(BC)$  lige med

$$\sum_{j=0}^{n-1} a_{ij} \left( \sum_{k=0}^{r-1} b_{jk} c_{k\ell} \right).$$

#### 4 MATRIX MULTIPLIKATION

Men dette udtryk er lige med

$$\sum_{j=0}^{n-1} \sum_{k=0}^{r-1} a_{ij} (b_{jk} c_{k\ell}) = \sum_{j=0}^{n-1} \sum_{k=0}^{r-1} a_{ij} b_{jk} c_{k\ell} = \sum_{k=0}^{r-1} \sum_{j=0}^{n-1} a_{ij} b_{jk} c_{k\ell} = \sum_{k=0}^{r-1} \left( \sum_{j=0}^{n-1} a_{ij} b_{jk} \right) c_{k\ell},$$

som er netop den  $(i, \ell)$ te indgang af  $(AB)C$ . □

Specielt for  $v \in \mathbb{R}^r$  har vi

$$A(Bv) = (AB)v,$$

som siger at transformationen  $v \mapsto (AB)v$  er det samme som at transformere  $v$  under  $B$  og derefter transformere resultatet med  $A$ .

Fra de overnævnte egenskaber ser det ud til at matrix multiplikation ligner meget produkt operationen for almindelig tal, men der er nogle væsentlige forskelle.

**Advarsel 4.15.** Produkterne  $AB$  og  $BA$  er generelt ikke ens. !

Vi så endda i eksempel 4.6 at selvom  $AB$  er defineret, kan det være at  $BA$  ikke giver mening. Selv når begge produkter eksisterer er de oftest forskellige. Dette kan vi nemt se i python, det er ikke et problem der skyldes unøjagtighed.

```
>>> import numpy as np
>>> a = np.array([[1.0, 2.0],
...               [0.0, 1.0]])
>>> b = np.array([[1.0, 0.0],
...               [1.0, 2.0]])
>>> a @ b
array([[3., 4.],
       [1., 2.]])
>>> b @ a
array([[1., 2.],
       [1., 4.]])
```

**Advarsel 4.16.**  $AB = 0$  kan ske selvom  $A$  og  $B$  er ikke nulmatricer. !

```
>>> a = np.array([[1.0, 1.0],
...               [2.0, 2.0]])
```

```
>>> b = np.array([[1.0, 2.0],
...               [-1.0, -2.0]])
>>> a @ b
array([[0., 0.],
       [0., 0.]])
```

Som konsekvens har vi

**Advarsel 4.17.**  $CA = CB$  medfører *ikke* at  $A$  er lig med  $B$ , selvom  $C$  er ikke en nulmatrix. !

## 4.4 Identitetsmatricer

**Definition 4.18.** Den  $(n \times n)$ -identitetsmatrix er

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

Den  $(i, j)$ te indgang af  $I_n$  er 1 for  $i = j$ , og 0 for  $i \neq j$ .

**Proposition 4.19.** For  $A \in \mathbb{R}^{m \times n}$  er

$$I_m A = A = A I_n.$$

□

I python er identitetsmatricer givet ved `np.eye`:

```
>>> import numpy as np
>>> a = np.array([[1.3, -3.0, 0.7, 1.4],
...               [2.0, 4.2, -5.6, 6.2],
...               [1.5, -2.5, 3.2, 5.4]])
>>> np.eye(3) @ a
array([[ 1.3, -3. ,  0.7,  1.4],
       [ 2. ,  4.2, -5.6,  6.2],
       [ 1.5, -2.5,  3.2,  5.4]])
```

## 4 MATRIX MULTIPLIKATION

```
>>> a @ np.eye(4)
array([[ 1.3, -3. ,  0.7,  1.4],
       [ 2. ,  4.2, -5.6,  6.2],
       [ 1.5, -2.5,  3.2,  5.4]])
```

### 4.5 Ydre produkt

Vi begyndte kapitlet med række-søjleproduktet, som giver en skalar. Omvendt hvis vi har en søjle vektor  $v \in \mathbb{R}^{m \times 1}$  og en rækkevektor  $w^T \in \mathbb{R}^{1 \times n}$  kan vi danne matrixproduktet  $vw^T \in \mathbb{R}^{m \times n}$ . Resultatet kaldes det *ydre produkt* af  $v$  med  $w^T$ . I komponenter har vi

$$\begin{aligned} vw^T &= \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{m-1} \end{bmatrix} \begin{bmatrix} w_0 & w_1 & \dots & w_{n-1} \end{bmatrix} \\ &= \begin{bmatrix} v_0 w_0 & v_0 w_1 & \dots & v_0 w_{n-1} \\ v_1 w_0 & v_1 w_1 & \dots & v_1 w_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m-1} w_0 & v_{m-1} w_1 & \dots & v_{m-1} w_{n-1} \end{bmatrix} \in \mathbb{R}^{m \times n}. \end{aligned}$$

Vi kan bruge `np.matshow` til at plote nogle eksempler på ydre produkter. Resultaterne gives i figur 4.3.

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

v = np.array([0.0, 1.0, 0.0, 1.0])[:, np.newaxis]
wt = np.array([[1.0, 0.0, 1.0, 1.0]])
v @ wt

fig, ax = plt.subplots()
ax.matshow(v @ wt, cmap='Reds')
```



```
w2t = np.linspace(0, 1, 10)[np.newaxis, :]
fig, ax = plt.subplots()
ax.matshow(v @ w2t, cmap='Blues')
```

```
v2 = np.linspace(0, 1, 10)[:, np.newaxis]
fig, ax = plt.subplots()
ax.matshow(v2 @ w2t, cmap='coolwarm')
```

For det sidste eksempel bruger vi nogle tilfældige vektorer. Disse frembringes ved hjælp en tilfældighedsgenerator `rng = np.random.default_rng()` og derefter en funktionen som `rng.random()`, for en uniform fordeling på  $[0, 1)$ , eller `rng.standard_normal()`, for en standard normalfordeling:

```
# opstil en tilfældighedsgenerator
rng = np.random.default_rng()
# 5 tilfældige tal i [0.0, 1.0)
vr = rng.random(5)[:, np.newaxis]
print(vr)
```

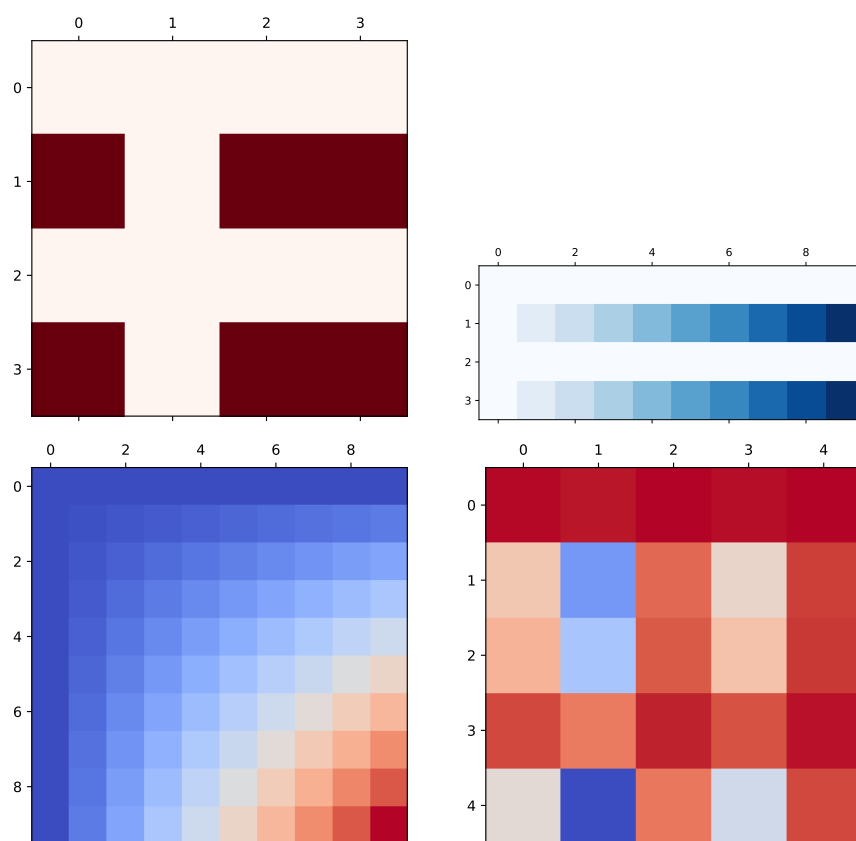
```
[[0.01674922]
 [0.78019725]
 [0.64407641]
 [0.17385816]
 [0.95814935]]
```

```
# 5 tilfældige tal normalfordelt med middelværdi 0
# og varians 1
wrt = rng.standard_normal(5)[np.newaxis, :]
print(wrt)
```

```
[[-0.69922005 -1.45703023 -0.25636653 -0.79844586 -0.12302635]]
```

```
fig, ax = plt.subplots()
ax.matshow(vr @ wrt, cmap='coolwarm')
```

## 4 MATRIX MULTIPLIKATION



Figur 4.3: Eksempler på ydre produkter.

# Kapitel 5

## Numerisk håndtering af matricer

### 5.1 Omkostninger ved matrixberegning

Matrixberegninger dækker over mange små aritmetiske operationer på indgangene. Det er derfor nødvendigt at have en idé om hvor mange ressourcer de beslaglægger ved computeren, og hvilke metoder er mindre ressourcekrævende end andre. Desuden har vi også set at hver operation med `float` risikerer at tilføje usikkerhed til resultatet, så det er godt at vide hvilke metoder bruger færre af disse operationer.

En simpel, men anvendelig, model, siger at hver af operationerne  $+$ ,  $-$ ,  $*$  og  $/$  på `floats` bruger den samme mængde tid eller ressource, kaldet en *flop*. Omkostninger ved en beregning gives så som et vist antal flops.

Som første eksempel lad os kikke på multiplikation af en vektor  $v \in \mathbb{R}^{n \times 1}$  med en skalar  $s \in \mathbb{R}$ . Ved beregning af  $u = sv$  udføres følgende operationer:

SKALAR-VEKTORPRODUKT( $s, v$ )

```
1 for  $i \in \{0, 1, \dots, n - 1\}$ :  
2      $u_i = s * v_i$   
3 return  $u$ 
```

Dvs. for hvert heltal  $i$  fra 0 til  $n - 1$ , udføres én `float` operation  $*$ . Da der er  $n$  tal i  $\{0, 1, \dots, n - 1\}$ , koster SKALAR-VEKTORPRODUKT( $s, v$ )  $n \times 1 = n$  flops.

For et række-søjleprodukt  $u^T v$ , hvor  $u^T \in \mathbb{R}^{1 \times n}$  og  $v \in \mathbb{R}^{n \times 1}$  har vi tilsvarende

## 5 NUMERISK HÅNDTERING AF MATRICER

RÆKKE-SØJLEPRODUKT( $u, v$ )

```

1   $c = 0$ 
2  for  $i \in \{0, 1, \dots, n-1\}$ :
3       $c = c + u_i * v_i$ 
4  return  $c$ 
```

Denne gang er der to **float**-operationer i linje 3: en gang  $+$  og en gang  $*$ . Vi får så at RÆKKE-SØJLEPRODUKT( $u, v$ ) koster  $2n$  flops.

Et matrixprodukt  $AB$ , for  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times r}$  består blot af  $m \times r$  række-søjleprodukter:

MATRIXPRODUKT( $A, B$ )

```

1  for  $i \in \{0, 1, \dots, m-1\}$ :
2      for  $j \in \{0, 1, \dots, r-1\}$ :
3           $c_{ij} = \text{RÆKKE-SØJLEPRODUKT}(A_{[i,:]}, B[:,j])$ 
4  return ( $c_{ij}$ )
```

Omkostningen er så  $mr \times 2n = 2mnr$  flops.

*Bemærkning 5.1.* Antag at  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times r}$  og  $C \in \mathbb{R}^{r \times p}$ . Vi har set at matematisk er  $(AB)C = A(BC)$ . Men hvis vi beregner omkostningerne ved beregning af de to udtryk ser vi en forskel. Beregningen af  $(AB)C$  koster

$$2mnr + 2mrp = 2(n+p)mr \text{ flops,}$$

hvorimod beregningen af  $A(BC)$  koster

$$2nrp + 2mnp = 2(m+r)np \text{ flops.}$$

Da disse to udtryk er ikke ens; der kan være stor forskel i antallet af flops.  $\diamond$

*Eksempel 5.2.* For  $A \in \mathbb{R}^{2000 \times 2}$ ,  $B \in \mathbb{R}^{2 \times 1000}$ ,  $C \in \mathbb{R}^{1000 \times 100}$  har vi

$$(AB)C : 2 \times (2 + 100) \times 2000 \times 1000 = 4,08 \cdot 10^8 \text{ flops,}$$

$$A(BC) : 2 \times (2000 + 1000) \times 2 \times 100 = 1,2 \cdot 10^6 \text{ flops,}$$

så der er en faktor 400 til forskel.  $\triangle$

Tabel 5.1 giver et overblik over omkostninger ved vektor- og matrixregneoperationer. Det skal noteres at disse vurderinger kan reduceres i nogle situationer, især hvis matricerne eller vektorerne har en bestemt struktur, f.eks. med mange indgange lige med 0.

operation				flops
vektorsum	$u + v$	$u, v \in \mathbb{R}^n$		$n$
skalar-vektorprodukt	$sv$	$s \in \mathbb{R}, v \in \mathbb{R}^n$		$n$
række-søjleprodukt	$u^T v$	$u^T \in \mathbb{R}^{1 \times n}, v \in \mathbb{R}^{n \times 1}$		$2n$
ydre produkt	$vw^T$	$v \in \mathbb{R}^{m \times 1}, w^T \in \mathbb{R}^{1 \times n}$		$mn$
matrixsum	$A + B$	$A, B \in \mathbb{R}^{m \times n}$		$mn$
skalar-matrixprodukt	$sA$	$s \in \mathbb{R}, A \in \mathbb{R}^{m \times n}$		$mn$
matrix-vektorprodukt	$Av$	$A \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^{n \times 1}$		$2mn$
matrixprodukt	$AB$	$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times r}$		$2mnr$

Tabel 5.1: Omkostninger ved vektor- og matrixberegninger.

Desuden for meget store vektorer og matricer er der også andre faktorer der skal tages i betragtning. En computer har typisk et begrænset mængde hukommelse som er hurtigt tilgængeligt (cache og RAM), så store store matricer skal nødvendigvis gemmes på en disk eller SSD. Men indhentning af data fra en disk eller SSD er væsentlige langsommere (f.eks. en ved faktor 1000) end fra cache eller RAM, så dette kan også påvirker hvor hurtigt en beregning kan udføres.

Overraskende nok findes der hurtige måde at gange matricer sammen end MATRIXPRODUKT ovenfor. For  $A, B \in \mathbb{R}^{n \times n}$  koster MATRIXPRODUKT( $A, B$ )  $2n^3$  flops. I 1969 fandt Volker Strassen en beregningsmetode der bruger højst  $c \times n^{2.807}$  flops, hvor  $c$  er en konstant uafhængigt af  $n$ . Strassens metode kan bruges effektivt på større matricer. Der findes andre nyere metoder hvis omkostning afhænger af en endnu mindre potens af  $n$ , men de har mest teoretisk interesse.

## 5.2 Pythons for-løkke

I de overstående eksempler har vi skrevet algoritmerne ikke i python med i generisk kode. Syntaksen er dog alligevel tæt på en konstruktion i python, som har den generelle form

```

1 for_variabel_in_objekt:
2     udfør_forskellige_trin
3     der_muligvis_bruger_variabel

```

## 5 NUMERISK HÅNDTERING AF MATRICER

Her har vi vist mellemrum eksplicit, da der er væsentligt at linjerne 2, 3, ..., der skal udføres i løkken, er rykket 4 mellemrum til højre i forhold til selve **for ... in ...** : linjen 1.

Objektet der typisk bruges i **for**-løkker er **range(...)**. For eksempel, skriver man **range(stop)** med stop et positivt heltal, fås rækken af heltal fra 0 til stop-1:

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

Man kan også skrive **range(start, stop)** for at angive en anden begyndelsesværdi end 0:

```
>>> for i in range(5, 10):
...     print(i)
...
5
6
7
8
9
```

Desuden kan man give en anden trinstørrelse via et tredje argument, som **range(start, stop, step)**:

```
>>> for i in range(0, 10, 2):
...     print(i)
...
0
2
4
```

```
6
8
```

F.eks. kan række-søjleproduktet ovenfor udregnes med følgende python kode

```
import numpy as np
rng = np.random.default_rng()
n = 20
ut = rng.random((1,n))
v = rng.random((n,1))
c = 0
for i in range(n):
    c = c + ut[0, i]*v[i, 0]
print(c)
```

som giver 4.727684605417684. Men NumPys egen @-operation er specielt skrevet til at gøre den type beregning mere effektivt med mere kompakt syntaks og kan tage højde for korrekt brug af float-operationer

```
c = (ut @ v)[0,0]
print(c)
```

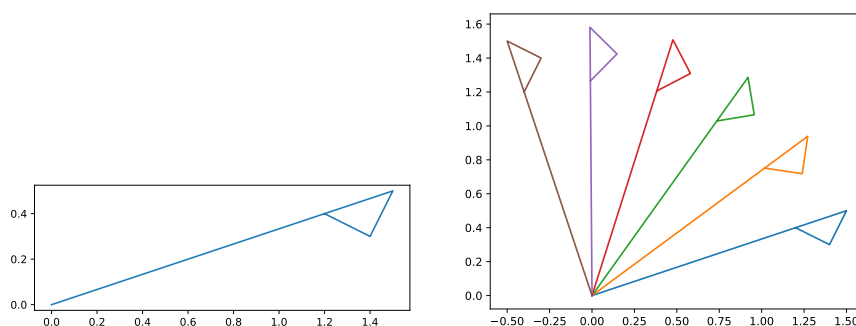
som giver 4.727684605417684. Generelt skal **for**-løkker ikke bruges til at erstatte operationer tilgængelig fra NumPy, men det kan være nyttigt når man afprøver nye beregningsmetode og i forbindelse med plot af figurer.

F.eks. hvis man vil illustrere effekten af gentagende brug af en rotationsmatrix på en simple figur. Vi begynder med en figur i planen angivet ved søjlerne af en  $(2 \times 4)$ -matrix points

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

points = np.array([[0.0, 1.5, 1.4, 1.2],
                  [0.0, 0.5, 0.3, 0.4]])
fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.plot(*points)
```

## 5 NUMERISK HÅNDTERING AF MATRICER



Figur 5.1: Oprindelig figur og dens rotationer.

som giver det første billede i figur 5.1. Vi har brugt lidt andet syntaks en tidligere, hvor en plot var typisk `ax.plot(x, y)`. I vores tilfælde er `x` og `y` rækkerne `points[0]` og `points[1]`, og `ax.plot(*points)` er en kort form for `ax.plot(points[0], points[1])`. Dvs. `*points` pakker `points` ud til `points[0]`, `points[1]`.

Nu stiller vi en rotationsmatrix op og bruger den til at drejer figuren 5 gange i en **for**-løkke:

```
c = np.cos(np.pi/10)
s = np.sin(np.pi/10)
R = np.array([[c, -s],
              [s,  c]])

fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.plot(*points)
for i in range(5):
    points = R @ points
    ax.plot(*points)
```

Resultatet er det andet billede i figur 5.1.



# Kapitel 6

## Lineære ligningssystemer

Vi har set tidligere hvordan en simpel elektrisk kreds giver anledning til et lineært ligningssystem. I dette kapitel kigger vi på nogle matematiske metoder for at forstå og løse sådanne systemer eksakt. Det skal bemærkes at udmiddelbart er disse metoder ikke så hensigt mæssig for numerisk beregning med `floats`, men der et par udsagn, som har væsentlig betydning for metoderne der bruges senere og som skal være på plads inden vi kan komme til det numeriske.

### 6.1 Elementære rækkeoperationer

Betragt det lineære ligningssystem

$$2x + 3y - 4z = 7, \quad (6.1a)$$

$$3x - 4y + z = -2, \quad (6.1b)$$

$$x + y + 2z = 3. \quad (6.1c)$$

Det er et system af 3 lineære ligninger i ubekendte  $x, y, z$ . Vi er interesseret i at finde alle løsninger  $(x, y, z)$  til systemet.

Vi kunne løse systemet ved f.eks. at isolere variabelen  $z$  i (6.1c) og derefter indsæt for  $z$  i (6.1a) og (6.1b). Vi får  $z = 3/2 - x/2 - y/2$  og (6.1a)–(6.1b) bliver

$$2x + 3y - 4\left(\frac{3}{2} - \frac{1}{2}x - \frac{1}{2}y\right) = 7,$$

$$3x - 4y + \left(\frac{3}{2} - \frac{1}{2}x - \frac{1}{2}y\right) = -2,$$

## 6 LINEÆRE LIGNINGSSYSTEMER

som forkortes til

$$4x + 5y = 13, \quad (6.2a)$$

$$\frac{5}{2}x - \frac{7}{2}y = -\frac{7}{2}. \quad (6.2b)$$

Processen kan så gentages ved at isolere  $y$  i (6.2b), og indsætte udtrykket i (6.2a). Der fås så én ligning med kun variabelen  $x$ , som kan løses for  $x$ , og derefter kan man finde først  $y$  og så  $z$  ved at indsætte den  $x$ -værdi.

En mere systematisk fremgangsmåde baserer sig på den følgende observation. Der er tre simple operationer vi kan udføre på system uden at ændre løsningsmængde, nemlig

- (I) byt om på rækkefølgen af ligninger,
- (II) gang en ligning igennem med en skalar som er ikke 0, eller
- (III) tilføj et multiplum af én ligning til en anden ligning.

Lad os skifte til matrixnotation for at lette arbejdet. Systemet (6.1a)–(6.1c) er ækvivalent med matrixligningen

$$\begin{bmatrix} 2 & 3 & -4 \\ 3 & -4 & 1 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \\ 3 \end{bmatrix}.$$

Vi kan samle koefficienterne i en *udvidet matrix*

$$\left[ \begin{array}{ccc|c} 2 & 3 & -4 & 7 \\ 3 & -4 & 1 & -2 \\ 1 & 1 & 2 & 3 \end{array} \right] \quad (6.3)$$

De tre elementære ligningsoperationer ovenfor er nu det samme som de følgende tre *elementære rækkeoperationer*

- (I)  $R_i \leftrightarrow R_j$  byt række  $i$  med række  $j$ ,
- (II)  $R_i \rightarrow sR_i$   $s \neq 0$  skalær  $R_i$  med en faktor  $s$ ,
- (III)  $R_i \rightarrow R_i + tR_j$   $j \neq i$  læg  $t$  gange række  $j$  til række  $i$ .

Disse operationer kan bruges på (6.3), f.eks. på den følgende måde. Vores mål

er at have 0-tal under diagonalen.

$$\begin{aligned}
 & \left[ \begin{array}{ccc|c} 2 & 3 & -4 & 7 \\ 3 & -4 & 1 & -2 \\ 1 & 1 & 2 & 3 \end{array} \right] \\
 & \sim_{R_0 \leftrightarrow R_2} \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 3 & -4 & 1 & -2 \\ 2 & 3 & -4 & 7 \end{array} \right] \sim_{R_1 \rightarrow R_1 - 3R_0} \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & -7 & -5 & -11 \\ 2 & 3 & -4 & 7 \end{array} \right] \\
 & \sim_{R_2 \rightarrow R_2 - 2R_0} \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & -7 & -5 & -11 \\ 0 & 1 & -8 & 1 \end{array} \right] \sim_{R_1 \leftrightarrow R_2} \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -8 & 1 \\ 0 & -7 & -5 & -11 \end{array} \right] \\
 & \sim_{R_2 \rightarrow R_2 + 7R_1} \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -8 & 1 \\ 0 & 0 & -61 & -4 \end{array} \right]
 \end{aligned} \tag{6.4}$$

Dette reduceret system kan du løses via *back substitution*: den sidste ligning siger  $-61z = -4$ , så  $z = 4/61$ , den anden ligning giver  $y = 1 + 8z = (61 + 32)/61 = 93/61$  og nu den første fører til  $x = 3 - y - 2z = (183 - 93 - 8)/61 = 82/61$ . Dvs.

$$x = \frac{82}{61}, \quad y = \frac{93}{61}, \quad z = \frac{4}{61}.$$

Som alternativ kan man forsætter med (6.4), og bruge yderligere rækkeoperationer til at få 1-tal på diagonalen og 0-tal ellers:

$$\begin{aligned}
 & \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -8 & 1 \\ 0 & 0 & -61 & -4 \end{array} \right] \\
 & \sim_{R_2 \rightarrow -R_2/61} \left[ \begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & -8 & 1 \\ 0 & 0 & 1 & 4/61 \end{array} \right] \sim_{R_0 \rightarrow R_0 - R_1} \left[ \begin{array}{ccc|c} 1 & 0 & 10 & 2 \\ 0 & 1 & -8 & 1 \\ 0 & 0 & 1 & 4/61 \end{array} \right] \\
 & \sim_{R_0 \rightarrow R_0 - 10R_2} \left[ \begin{array}{ccc|c} 1 & 0 & 0 & 82/61 \\ 0 & 1 & -8 & 1 \\ 0 & 0 & 1 & 4/61 \end{array} \right] \sim_{R_1 \rightarrow R_1 + 8R_2} \left[ \begin{array}{ccc|c} 1 & 0 & 0 & 82/61 \\ 0 & 1 & 0 & 93/61 \\ 0 & 0 & 1 & 4/61 \end{array} \right]
 \end{aligned} \tag{6.5}$$

## 6.2 Rækkeoperationer i python

Lad os se hvordan sådanne rækkeoperationer kan udføres i python. Husk at  $a[i, :]$  er den  $i$ te række i  $a$ . Rækkeoperation (II)  $R_i \rightarrow sR_i$  kan så udføres

operation	python
(I) $R_i \leftrightarrow R_j$	<code>a[ [i,j], :] = a[ [j,i], :]</code>
(II) $R_i \rightarrow sR_i (s \neq 0)$	<code>a[i, :] *= s</code>
(III) $R_i \rightarrow R_i + tR_j (j \neq i)$	<code>a[i, :] += t*a[j, :]</code>

Tabel 6.1: Rækkeoperationer i python.

via `a[i, :] = s * a[i, :]`. Men der er en nyttig forkortelse `*=`, som også hjælper med at undgå slåfejl: `x *= s` er det samme som `x = x * s`. Vi kan derfor lave rækkeoperation (II), som

```
a[i, :] *= s
```

Tilsvarende for operation (III) kan vi bruge `x += y` til som forkortelse af `x = x + y`, så operation (III)  $R_i \rightarrow R_i + tR_j$  udføres via

```
a[i, :] += t*a[j, :]
```

For rækkeoperation (I) er der en nyttig udvidet indekserings notation: `a[ [i,j], :]` er nemlig rækker *i* og *j* fra a ub i den givne rækkefølge. Så operation (I)  $R_i \leftrightarrow R_j$  kan laves via

```
a[ [i,j], :] = a[ [j,i], :]
```

Disse rækkeoperationer er opsummeret i tabel 6.1.

Vi kan bruge disse operationer til at lave reduktionen (6.3) i python. Først lad os se hvordan man danner den udvidede matrix ved hjælp af `np.hstack`. Kommandoen `np.hstack([a, b])` giver en ny array med søjler fra *a* efterfulgt af søjler fra *b*.

```
>>> import numpy as np
>>> a = np.array([[2.0, 3.0, -4.0],
...              [3.0, -4.0, 1.0],
...              [1.0, 1.0, 2.0]])
>>> a
array([[ 2.,  3., -4.],
       [ 3., -4.,  1.]])
```

## 6.2 RÆKKEOPERATIONER I PYTHON

```
    [ 1.,  1.,  2.]])
>>> b = np.array([7.0, -2.0, 3.0])[:, np.newaxis]
>>> b
array([[ 7.],
       [-2.],
       [ 3.]])
>>> aub = np.hstack([a, b])
>>> aub
array([[ 2.,  3., -4.,  7.],
       [ 3., -4.,  1., -2.],
       [ 1.,  1.,  2.,  3.]])
```

Derefter bruger vi rækkeoperationerne fra tabel 6.1:

```
>>> aub
array([[ 2.,  3., -4.,  7.],
       [ 3., -4.,  1., -2.],
       [ 1.,  1.,  2.,  3.]])
>>> aub[ [0,2], :] = aub[ [2,0], :]
>>> aub
array([[ 1.,  1.,  2.,  3.],
       [ 3., -4.,  1., -2.],
       [ 2.,  3., -4.,  7.]])
>>> aub[1, :] += -3*aub[0, :]
>>> aub
array([[ 1.,  1.,  2.,  3.],
       [ 0., -7., -5., -11.],
       [ 2.,  3., -4.,  7.]])
>>> aub[2, :] += -2*aub[0, :]
>>> aub
array([[ 1.,  1.,  2.,  3.],
       [ 0., -7., -5., -11.],
       [ 0.,  1., -8.,  1.]])
>>> aub[ [1,2], :] = aub[ [2,1], :]
>>> aub
array([[ 1.,  1.,  2.,  3.],
       [ 0.,  1., -8.,  1.],
       [ 0., -7., -5., -11.]])
```

```
>>> aub[2, :] += 7*aub[1, :]
>>> aub
array([[ 1.,  1.,  2.,  3.],
       [ 0.,  1., -8.,  1.],
       [ 0.,  0., -61., -4.]])
```

Man kan forsætte i samme stil hvis man vil udføre operationerne i (6.5).

### 6.3 Echelonform

Generelt kan man ikke reducere en koefficientmatrix til en form med 1-tal på diagonalen, men man kan komme tæt på. Følgende matrix er i det der kaldes »echelonform«:

$$\begin{bmatrix} 1 & 0 & 2,3 & -1,2 & 0,5 \\ 0 & 0 & 1 & 3,2 & -2,2 \\ 0 & 0 & 0 & 1 & 0,3 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (6.6)$$

**Definition 6.1.** En matrix er i *echelonform* hvis alle de følgende tre krav er opfyldte:

- (a) Den første ikke nul indgang i hver række er lige med 1. Indgangen kaldes et *pivotelement*.
- (b) Pivot elementer ligger længere til højre i efterfølgende rækker.
- (c) Nulrækker kommer til sidst.

I vores matrix er pivotelementerne fremhævet nedenfor

$$\begin{bmatrix} \boxed{1} & 0 & 2,3 & -1,2 & 0,5 \\ 0 & 0 & \boxed{1} & 3,2 & -2,2 \\ 0 & 0 & 0 & \boxed{1} & 0,3 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

og vi ser at alle indgange under trappestien dannet af pivotelementer er 0.

**Proposition 6.2.** *Enhver matrix kan reduceres til echelonform via rækkeoperationer.*

## 6.4 LØSNING VIA RÆKKEOPERATIONER

*Bevis.* Givet en matrix  $A \in \mathbb{R}^{m \times n}$ , find den første søjle som er ikke  $0_{m \times 1}$  og vælg et element i denne søjle som er forskellig fra 0. Brug rækkeoperation (I) til at flytte dette element til den første række. Så har vi en matrix af formen

$$\begin{bmatrix} 0 & \dots & 0 & \times & * & \dots & * \\ 0 & \dots & 0 & * & * & \dots & * \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & * & * & \dots & * \end{bmatrix},$$

hvor  $\times$  betegner et element forskelligt fra 0, og  $*$  angiver vilkårlige tal.

Brug rækkeoperation (I) til at få et 1-tal i position  $\times$ . Hvis  $m = 1$  har vi nu en matrix i echelonform og vi er færdig.

Hvis  $m > 1$ , kan vi antage at vi har vist at alle  $((m-1) \times n)$  kan reduceres til echelonform.

Brug nu rækkeoperation (III) til at få en ny matrix med alle elementer i søjlen under 1-tallet lige med 0:

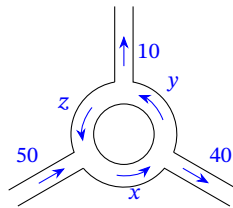
$$\begin{bmatrix} 0 & \dots & 0 & 1 & * & \dots & * \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & * & \dots & * \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & * & \dots & * \end{bmatrix} = \begin{bmatrix} u^T \\ \cdot \\ B \\ \cdot \end{bmatrix}. \quad (6.7)$$

Matricen  $B$  under den punkterede linje har  $B \in \mathbb{R}^{(m-1) \times n}$ , så  $B$  kan rækkereduceres til echelonform. Bemærk at pivotelementer fra  $B$  ligger til højre fra pivotelementet 1 i rækken  $u^T$  i (6.7). Så ved at reducere  $B$  til echelonform får vi en echelonform for  $A$ .  $\square$

## 6.4 Løsning af lineære ligningssystemer via rækkeoperationer

For matricen (6.6) i echelonform, er det tilsvarende system af lineære ligninger af formen

$$\begin{aligned} x_0 + 2,3x_2 - 1,2x_3 + 0,5x_4 &= 1,2, \\ x_2 + 3,2x_3 - 2,2x_4 &= 0,7, \\ x_3 + 0,3x_4 &= -0,2, \\ 0 &= 0, \end{aligned}$$



Figur 6.1: Rundkørsel.

hvor vi har indsat nogle tilfældige værdier på den højre side. I dette system ser vi at hvis vi giver  $x_4$ , og det usynlige  $x_1$ , vilkårlige værdier, så kan vi bestemme  $x_3$ ,  $x_2$  og til sidst  $x_1$  via back substitution. Vi siger at variablerne  $x_1, x_4$  er *frie*, og at  $x_0, x_2, x_3$  er *bundne*. Systemet har så uendelig mange løsninger  $(x_0, x_1, x_2, x_3, x_4)$ , da  $x_1, x_4$  kan tage hvilket som helst reelle værdier. Bemærk at de bundne variabler svar netop til pivotelementerne, så

- (a) antallet af bundne variabler er antallet af pivotelementer, og
- (b) antallet af frie variabler er antallet af søjler i koefficientmatricen minus antallet af pivotelementer.

*Eksempel 6.3.* Betragt rundkørslen i figur 6.1. På de ensrettede tilsluttende veje er antallet af biler per time målt som angivet. Antallet  $x, y, z$  af biler per time på de forskellige deler af selve rundkørslen opfylder

$$z + 50 = x, \quad y + 40 = x \quad \text{og} \quad z + 10 = y.$$

Dette omskrives til systemet

$$\begin{aligned} x - z &= 50, \\ x - y &= 40, \\ y - z &= 10. \end{aligned}$$

Den tilsvarende udvidede matrix er

$$\begin{aligned} \left[ \begin{array}{ccc|c} 1 & 0 & -1 & 50 \\ 1 & -1 & 0 & 40 \\ 0 & 1 & -1 & 10 \end{array} \right] &\xrightarrow{R_1 \rightarrow R_1 - R_0} \left[ \begin{array}{ccc|c} 1 & 0 & -1 & 50 \\ 0 & -1 & 1 & -10 \\ 0 & 1 & -1 & 10 \end{array} \right] \\ &\xrightarrow{R_2 \rightarrow R_2 + R_1} \left[ \begin{array}{ccc|c} 1 & 0 & -1 & 50 \\ 0 & -1 & 1 & -10 \\ 0 & 0 & 0 & 0 \end{array} \right] &\xrightarrow{R_1 \rightarrow -R_1} \left[ \begin{array}{ccc|c} 1 & 0 & -1 & 50 \\ 0 & 1 & -1 & 10 \\ 0 & 0 & 0 & 0 \end{array} \right] \end{aligned}$$



## 6.4 LØSNING VIA RÆKKEOPERATIONER

Systemet er konsistent. Pivotelementer er søjler 0 og 1, så  $z$  er fri, og  $x, y$  er bundne. Den generelle løsning til systemet er

$$x = 50 + z, \quad y = 10 + z, \quad z \in \mathbb{R}.$$

Realistiske løsninger skal dog være positive så vi må begrænse os til  $z \geq 0$ .  $\Delta$

Generelt givet et lineært ligningssystem

$$\begin{aligned} a_{00}x_0 + a_{01}x_1 + \cdots + a_{0,n-1}x_{n-1} &= b_0, \\ a_{10}x_0 + a_{11}x_1 + \cdots + a_{1,n-1}x_{n-1} &= b_1, \\ &\vdots \\ a_{m-1,0}x_0 + a_{m-1,1}x_1 + \cdots + a_{m-1,n-1}x_{n-1} &= b_{m-1}, \end{aligned}$$

skriver vi det i matrixform

$$Ax = b$$

med  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$  givne, og  $x \in \mathbb{R}^n$  ukendt. Vi danner så den udvidede matrix

$$[A \mid b]$$

og reducerer til echelonform

$$[C \mid d]$$

Vores oprindelig lineære ligningssystem har de samme løsninger som systemet

$$Cx = d. \tag{6.8}$$

For et generelt system er der tre muligheder

(a)  $[C \mid d]$  har en række af formen

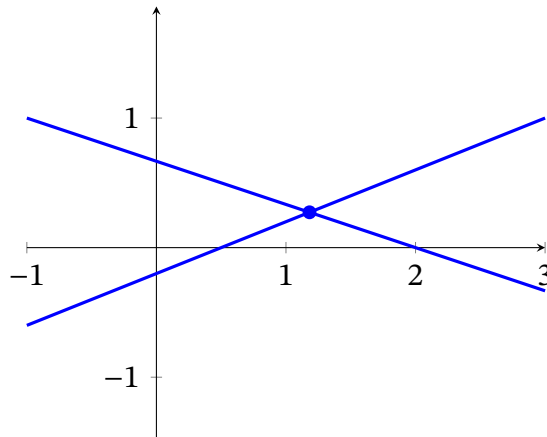
$$[0 \quad 0 \quad \cdots \quad 0 \mid 1]$$

Dette betyder at den tilsvarende ligning siger  $0 = 1$ , som er en modstrid. Så har systemet  $Ax = b$  ingen løsninger. Systemet siges at være *inkonsistent*.

(b) Systemet er konsistent og et af de følgende to tilfælde gælder:

- (i) Antallet af pivotelementer er lige med antallet af søjler. Så er der ingen frie variabler, alle variabler er bundne og systemet har en entydig løsning.
- (ii) Der er strengt færre pivotelementer end søjler. Så giver de andre søjler frie variabler og der er uendelige mange løsninger til systemet. Systemet er *underbestemt*.

## 6 LINEÆRE LIGNINGSSYSTEMER



Figur 6.2: To linjer i planen, som skærer.

Vi har derfor vist at:

**Sætning 6.4.** *Et lineært ligningssystem enten*

- (a) *er en inkonsistent, eller*
- (b) (i) *har præcis én løsninger, eller*  
(ii) *har uendelig mange løsninger bestemt af frie variable.* □

**Bemærkning 6.5.** For et konsistent system af  $m$  ligninger i  $n$  variable, med  $m < n$ , er der altid frie variable og så uendelig mange løsninger. ◇

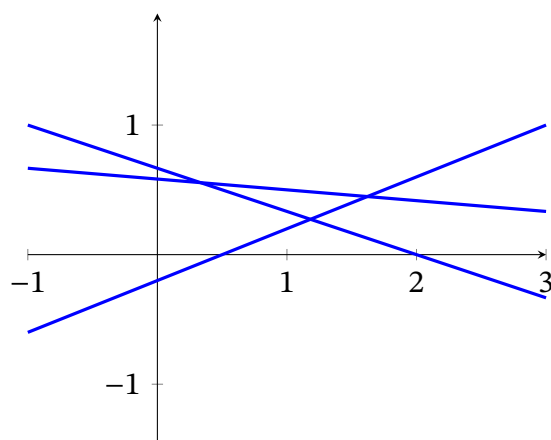
**Eksempel 6.6.** For systemer i to variable kan vi tegne os frem til de forskellige udfald. En lineær ligning  $ax + by = c$  beskriver er en linje i planen, som vi så i eksempel 4.4. Løsningen på et system af sådanne ligninger er koordinaterne på punkter der ligger på alle de beskrevne linjer i planen, dvs. koordinater på fælles skæringspunkter.

Systemet

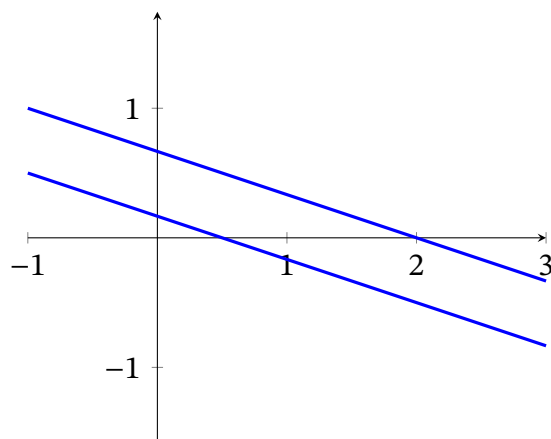
$$\begin{aligned}x + 3y &= 2, \\ 2x - 5y &= 1\end{aligned}$$

vises i figur 6.2, og det ses at der er netop ét skæringspunkt, dvs. en entydig løsning på systemet.

## 6.4 LØSNING VIA RÆKKEOPERATIONER



Figur 6.3: Tre linjer i planen, som har intet fællespunkt.



Figur 6.4: To parallelle linjer i planen har intet fællespunkt.

Derimod for systemet

$$\begin{aligned}x+3y &= 2, \\2x-5y &= 1, \\x+6y &= 5,\end{aligned}$$

er tegningen, som i figur 6.3, ses at tre linjer har ingen fællespunkt, så systemet har ingen løsning og er inkonsistent.

## 6 LINEÆRE LIGNINGSSYSTEMER

En anden situation uden fællesløsninger

$$\begin{aligned}x+3y &= 2, \\ 2x+6y &= 1.\end{aligned}$$

Som det ses i figur 6.4, er det to linjer parallelle, så har ingen skæringspunkt. Her er systemet inkonsistent. Men hvis vi justerer konstanterne på højre siden til

$$\begin{aligned}x+3y &= 2, \\ 2x+6y &= 4,\end{aligned}$$

så beskriver begge ligninger den samme linje i planen, og alle punkter på linjen er løsninger på systemet. Dvs. dette er et systemet med uendelige mange løsninger. △

# Kapitel 7

## Matrixinvers

For reelle tal er det meget nyttigt at man kan dele med et tal  $s$ , som er ikke nul. Dette er det sammen som at gange med den reciprokke  $1/s$ . For matricer er der en tilsvarende matrix, den inverse, som er især nyttig i eksakte regnestykker. Men modsat reelle tal er der mange matricer der ikke tillader en invers. For det første skal matricen være kvadratisk (definition 7.1 nedenfor), og for kvadratiske matricer er der ydeligere krav, der skal opfyldes.

### 7.1 Egenskaber af den inverse matrix

**Definition 7.1.** En matrix  $A$  er *kvadratisk* hvis den har det samme antal rækker som søjler. Dvs. en kvadratisk matrix er et  $A \in \mathbb{R}^{n \times n}$ .

Bemærk at identitetsmatricen  $I_n$  er kvadratisk og for alle  $A \in \mathbb{R}^{n \times n}$  har vi

$$I_n A = A = A I_n.$$

**Sætning 7.2.** Lad  $A \in \mathbb{R}^{n \times n}$  være en kvadratisk matrix. Hvis der findes en matrix  $A^{-1} \in \mathbb{R}^{n \times n}$  således at

$$A^{-1} A = I_n \tag{7.1}$$

så (a) er  $A$  invertibel, (b) er  $A^{-1}$  den eneste matrix der opfylder (7.1), og (c)

$$A A^{-1} = I_n. \tag{7.2}$$

Omvendt, hvis en matrix  $A^{-1}$  opfylder (7.2), så er den entydigt bestemt og opfylder (7.1).

## 7 MATRIXINVERS

Beviset for dette resultat gives senere i kapitlet, se afsnit 7.2. Her giver vi nogle eksempler og fokuserer på nogle af egenskaberne for  $A^{-1}$ .

For  $A \in \mathbb{R}^{2 \times 2}$  er der en simpel formel for  $A^{-1}$ . Matricen

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

har invers

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \quad \text{når } ad - bc \neq 0.$$

Dette kan bekræftes direkte ved beregning af matrixproduktet (7.1):

$$\begin{aligned} A^{-1}A &= \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} da - bc & db - bd \\ -ca + ac & -cb + ad \end{bmatrix} \\ &= \frac{1}{ad - bc} \begin{bmatrix} ad - bc & 0 \\ 0 & ad - bc \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2. \end{aligned}$$

Ligningen (7.2) kan bekræftes ved en tilsvarende udregning. Bemærk betingelsen  $ad - bc \neq 0$  for eksistens af den inverse i disse tilfælde. Tallet  $ad - bc$  kaldes determinanten af  $A \in \mathbb{R}^{2 \times 2}$ .

Givet et lineært ligningssystem

$$Ax = b$$

med  $A \in \mathbb{R}^{n \times n}$  kvadratisk og invertibel, kan man bruge inversen til at løse systemet. Vi ganger begge sider igennem med  $A^{-1}$ , til at få

$$A^{-1}Ax = A^{-1}b. \quad (7.3)$$

Men  $A^{-1}A = I_n$ , så  $A^{-1}Ax = I_n x = x$ , og ligning (7.3) er blot

$$x = A^{-1}b.$$

Dvs. vi har fundet den entydige løsning til systemet. Dette er ofte af teoretisk interesse; i en numerisk sammenhæng er det generelt bedst at undgå den inverse matrix og bruge andre metoder for sådanne systemer, som ikke er så følsom over for unøjagtigheder.

*Eksempel 7.3.* Betragt systemet

$$\begin{aligned} 2x + 3y &= 1, \\ x + 2y &= 2. \end{aligned}$$

## 7.1 EGENSKABER AF DEN INVERSE MATRIX

I matrixform er dette

$$\begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Koefficientmatricen har  $ad - bc = 2 \times 2 - 3 \times 1 = 1$ , som er forskellig fra nul, så matricen er invertibel. Vi får så

$$\begin{aligned} \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \frac{1}{2 \times 2 - 3 \times 1} \begin{bmatrix} 2 & -3 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \frac{1}{1} \begin{bmatrix} 2 \times 1 - 3 \times 2 \\ -1 \times 1 + 2 \times 2 \end{bmatrix} \\ &= \begin{bmatrix} -4 \\ 3 \end{bmatrix}. \end{aligned}$$

Dvs.  $x = -4$ ,  $y = 3$ . △

Transformationen  $b \mapsto A^{-1}b$  er den omvendte transformation af  $x \mapsto Ax$ . Så f.eks. den inverse til en rotation i  $\mathbb{R}^2$  er den samme rotation i den modsatte retning. Som matricer har rotationsmatricen

$$R = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \quad c^2 + s^2 = 1$$

invers

$$R^{-1} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}.$$

Bemærk at i dette tilfælde fås  $R^{-1}$  fra  $R$  via simpel ombytning af indgange. Generelt er beregning af den inverse væsentlig mere kompliceret, og resultatet kan være svært at kontrollere. Senere vil vi have særlig interesse for matricer hvor inversen er nemt tilgængelig.

Der er en simpel formel for den invers af et produkt af matricer:

**Proposition 7.4.** Hvis  $A, B \in \mathbb{R}^{n \times n}$  er invertibel, så er  $AB$  invertibel med invers

$$(AB)^{-1} = B^{-1}A^{-1}.$$

*Bevis.* Det er nok at regne

$$(B^{-1}A^{-1})(AB) = B^{-1}(A^{-1}A)B = B^{-1}I_n B = B^{-1}B = I_n,$$

så  $B^{-1}A^{-1}$  er den inverse til  $AB$ . □

Matrixinvers kan beregnes i python via `np.linalg.inv`. Her kan vi begynde at ane nogle af de numeriske problemer med den inverse.

## 7 MATRIXINVERS

```
>>> import numpy as np

>>> a = np.array([[1.0,          1.0],
...               [1.0000000001, 1.0]])
>>> a
array([[1., 1.],
       [1., 1.]])
>>> np.linalg.inv(a)
array([[ -9.99999916e+08,  9.99999916e+08],
       [ 9.99999917e+08, -9.99999916e+08]])
```

Vi ser at selvom denne matrix har alle indgange af størrelse 1, har den inverse indgange af størrelsesorden  $10^9$ . Dette stammer fra at i beregningen af den inverse kommer vi til at gange igennem med

$$\frac{1}{ad - bc} = \frac{1}{1.0 - 1.0000000001} = \frac{1}{-0.0000000001} = -10,0 \cdot 10^8.$$

Et eksempel af en anden type er

```
>>> n = 100
>>> a = np.triu(2*np.eye(n) - np.ones((n,n)))
>>> a
array([[ 1., -1., -1., ..., -1., -1., -1.],
       [ 0.,  1., -1., ..., -1., -1., -1.],
       [ 0.,  0.,  1., ..., -1., -1., -1.],
       ...,
       [ 0.,  0.,  0., ...,  1., -1., -1.],
       [ 0.,  0.,  0., ...,  0.,  1., -1.],
       [ 0.,  0.,  0., ...,  0.,  0.,  1.]])
>>> np.linalg.inv(a)[0,-1]
3.1691265005705735e+29
```

Matricen  $a$  har  $-1$  i alle pladser over diagonalen,  $1$  på diagonalen, og  $0$  under diagonalen. Så alle indgange har størrelse højst  $1$ . Men den inverse har en indgang der er størrelsesorden  $10^{29}$ . Faktisk indeholder den inverse matricer elementer med alle størrelsesordner mellem  $1$  og  $10^{29}$ .



## 7.2 Eksistens af den inverse

**Definition 7.5.** En kvadratisk matrix  $A \in \mathbb{R}^{n \times n}$  er *invertibel* hvis der findes et  $B \in \mathbb{R}^{n \times n}$  således at

$$BA = I_n = AB. \quad (7.4)$$

**Lemma 7.6.** Matricen  $B$  i (7.4) er entydig.

*Bevis.* Antag at  $C \in \mathbb{R}^{n \times n}$  opfylder Hvis vi har (7.4) og

$$CA = I_n = AC.$$

Så har vi

$$B = BI_n = B(AC) = (BA)C = I_n C = C,$$

dvs.  $C = B$ . Så  $B$  er entydig.  $\square$

Når vi har dette resultat, giver det god mening at skrive  $A^{-1} = B$  og at kalde  $A^{-1}$  den *inverse* til  $A$ .

**Lemma 7.7.** Hvis  $A \in \mathbb{R}^{n \times n}$  er invertibel, så har det lineære ligningssystem

$$Ax = 0, \quad x \in \mathbb{R}^n \quad (7.5)$$

kun én løsning, nemlig  $x = 0$ .

*Bevis.* Bemærk først at  $x = 0$  er en løsning til ligning, da  $A0 = 0$ .

Nu lad os gange  $A^{-1}$  på (7.5) fra den venstre side. Vi får

$$A^{-1}Ax = 0.$$

Men  $A^{-1}A = I_n$ , så  $0 = A^{-1}Ax = I_n x = x$ , dvs.  $x = 0 \in \mathbb{R}^n$  er den eneste løsning.  $\square$

*Bemærkning 7.8.* Beviset for lemma 7.7 bruger kun ligningen  $A^{-1}A = I_n$ . Så for vilkårlig  $A$  hvis  $B$  er en matrix således at  $BA = I_n$ , kan vi også konkludere at ligningen  $Ax = 0$  har kun  $x = 0$  som løsning.  $\diamond$

Hvis ligningssystemet  $Ax = 0$  har kun én løsning, så indeholder hver søjle ét pivotelement. For  $A \in \mathbb{R}^{n \times n}$ , betyder dette at  $A$  har echelonform

$$\begin{bmatrix} 1 & * & * & \cdots & * \\ 0 & 1 & * & \cdots & * \\ 0 & 0 & 1 & & * \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & & 1 \end{bmatrix}. \quad (7.6)$$

## 7 MATRIXINVERS

Nu kan vi bruge yderlige rækkeoperationer af type (III) på (7.6) til at sætte alle elementer  $*$  til 0, og dermed reducerer (7.6) til identitetsmatricen  $I_n$ . I andre ord,  $A$  kan række reduceres til  $I_n$ .

En bemærkelsesværdig ting er at rækkeoperationer kan realises via matrix-multiplikation via bestemte matricer.

**Definition 7.9.** En *elementær matrix*  $E$  er resultatet af en elementær rækkeoperation på  $I_n$ .

*Eksempel 7.10.* For  $n = 2$ , lad os kikke på et eksempel på hver type rækkeoperation.

Type (I): er den eneste mulighed  $R_0 \leftrightarrow R_1$ . Den giver den elementære matrix

$$E = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Type (II): rækkeoperationen  $R_1 \rightarrow sR_1$  har elementær matrix

$$E = \begin{bmatrix} 1 & 0 \\ 0 & s \end{bmatrix}.$$

Type (III): rækkeoperationen  $R_1 \rightarrow R_1 + tR_0$  svarer til matricen

$$E = \begin{bmatrix} 1 & 0 \\ t & 1 \end{bmatrix}.$$

△

**Proposition 7.11.** At udføre en elementære rækkeoperation på en  $(m \times n)$ -matrix  $A$  er det samme som at tage matrixproduktet  $EA$ , hvor  $E \in \mathbb{R}^{m \times m}$  er den elementære matrix svarende til den pågældende rækkeoperation.

*Bevis.* Hvis  $C$  har rækker  $u_0^T, u_1^T, \dots, u_{n-1}^T$ , så har produktet  $CA$  rækker  $u_0^T A, u_1^T A, \dots, u_{n-1}^T A$ . Det følger at udførelse af en elementær rækkeoperation på  $CA$  er det samme som at udføre den samme elementær rækkeoperation på  $C$  og derefter gange fra højre med  $A$ . Ved at tage  $C = I_n$ , fås resultatet.  $\square$

Bemærk at elementære matricer er invertible: deres invers er blot den elementære matrix svarende til den omvendte rækkeoperation.

*Bevis for sætning 7.2.* Hvis  $A^{-1}A = I_n$ , så giver lemma 7.7 at det lineære ligningssystem  $Ax = 0$  har kun én løsning, nemlig  $x = 0$ . Det følger at  $A$  kan rækkereduceres til  $I_n$ . Men det er det samme som at sige at der er elementære matricer  $E_0, E_1, \dots, E_{r-2}, E_{r-1}$  således at

$$E_{r-1}E_{r-2} \dots E_1E_0A = I_n.$$

Ganges denne ligning fra venstre med først  $E_{r-1}^{-1}$ , så  $E_{r-2}^{-1}$ , osv., fås

$$A = E_0^{-1}E_1^{-1} \dots E_{r-2}^{-1}E_{r-1}^{-1}.$$

Men den højre side er invertibel med invers  $E_{r-1}E_{r-2} \dots E_1E_0$ . Så  $A$  selv er en invertibel matrix, og  $AA^{-1} = I_n$ .

For det omvendte resultat, bruger vi bare at  $AA^{-1} = I_n$  fortæller at matricen  $B = A^{-1}$  har en matrix  $B^{-1} = A$  således at  $B^{-1}B = I_n$ . Det overstående argument viser at  $B$  er invertibel, med invers  $B^{-1} = A$ . Det siger at  $B^{-1}B = I_n = BB^{-1}$ , som er  $AA^{-1} = I_n = A^{-1}A$ , og vi har at  $A$  er invertibel.  $\square$

## 7.3 Beregning af den inverse

Beviset for sætning 7.2 fører også til en metode for at beregne  $A^{-1}$  via elementære rækkeoperationer. Vi så at  $A^{-1} = E_{r-1}E_{r-2} \dots E_1E_0$ , men dette er resultatet af at udføre på  $I_n$  de rækkeoperationer, som reducerer  $A$  til  $I_n$ . Vi kan så beregne  $A^{-1}$  ved at starte med den udvidede matrix

$$[A \mid I_n]$$

og derefter rækkereducerer den til

$$[I_n \mid A^{-1}].$$

*Eksempel 7.12.* For matricen

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 1 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

## 7 MATRIXINVERS

beregner vi inversen

$$\begin{aligned}
 & \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array} \right] \\
 & \sim_{R_1 \rightarrow R_1 + R_0} \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 3 & 4 & 1 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{array} \right] \sim_{R_2 \rightarrow R_2 - R_0} \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 3 & 4 & 1 & 1 & 0 \\ 0 & -2 & -1 & -1 & 0 & 1 \end{array} \right] \\
 & \sim_{R_1 \rightarrow R_1 + R_2} \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 3 & 0 & 1 & 1 \\ 0 & -2 & -1 & -1 & 0 & 1 \end{array} \right] \sim_{R_2 \rightarrow R_2 + 2R_1} \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 3 & 0 & 1 & 1 \\ 0 & 0 & 5 & -1 & 2 & 3 \end{array} \right] \\
 & \sim_{R_0 \rightarrow R_0 - 2R_1} \left[ \begin{array}{ccc|ccc} 1 & 0 & -3 & 1 & -2 & -2 \\ 0 & 1 & 3 & 0 & 1 & 1 \\ 0 & 0 & 5 & -1 & 2 & 3 \end{array} \right] \sim_{R_2 \rightarrow \frac{1}{5}R_2} \left[ \begin{array}{ccc|ccc} 1 & 0 & -3 & 1 & -2 & -2 \\ 0 & 1 & 3 & 0 & 1 & 1 \\ 0 & 0 & 1 & -\frac{1}{5} & \frac{2}{5} & \frac{3}{5} \end{array} \right] \\
 & \sim_{R_0 \rightarrow R_0 + 3R_2} \left[ \begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{2}{5} & -\frac{4}{5} & \frac{1}{5} \\ 0 & 1 & 3 & 0 & 1 & 1 \\ 0 & 0 & 1 & -\frac{1}{5} & \frac{2}{5} & \frac{3}{5} \end{array} \right] \sim_{R_1 \rightarrow R_1 - 3R_2} \left[ \begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{2}{5} & -\frac{4}{5} & \frac{1}{5} \\ 0 & 1 & 0 & \frac{3}{5} & -\frac{1}{5} & -\frac{4}{5} \\ 0 & 0 & 1 & -\frac{1}{5} & \frac{2}{5} & \frac{3}{5} \end{array} \right].
 \end{aligned}$$

Fra dette aflæser vi at

$$A^{-1} = \begin{bmatrix} \frac{2}{5} & -\frac{4}{5} & -\frac{1}{5} \\ \frac{3}{5} & -\frac{1}{5} & -\frac{4}{5} \\ -\frac{1}{5} & \frac{2}{5} & \frac{3}{5} \end{bmatrix}.$$

Δ

# Kapitel 8

## Ortogonalitet og projektioner

### 8.1 Standard indre produkt

For vektorer  $u, v \in \mathbb{R}^n$ , definerer vi deres standard *indre produkt* til at være

$$\begin{aligned}\langle u, v \rangle &= u^T v \\ &= \begin{bmatrix} u_0 & u_1 & \dots & u_{n-1} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} \\ &= u_0 v_0 + u_1 v_1 + \dots + u_{n-1} v_{n-1}.\end{aligned}\tag{8.1}$$

Vi siger at  $u, v$  er *ortogonal*, eller at  $v$  står *vinkelret* på  $u$ , og skriver  $u \perp v$ , hvis

$$\langle u, v \rangle = 0.$$

*Eksempel 8.1.* For  $u = (1, -2, 3)$  og  $v = (0, 5, 7)$  har vi

$$\langle u, v \rangle = 1 \times 0 + (-2) \times 5 + 3 \times 7 = 0 - 10 + 21 = 11,$$

så de står ikke vinkelret på hinanden. Tilgængæld har vi for  $w = (2, 1, 0)$  og  $w' = (3, 0, -1)$  er

$$\langle u, w \rangle = 1 \times 2 + (-2) \times 1 + 3 \times 0 = 0,$$

$$\langle u, w' \rangle = 1 \times 3 + (-2) \times 0 + 3 \times 1 = 0,$$

så  $w$  og  $w'$  står vinkelret på  $u$ . Bemærk at  $w'$  er ikke proportionelt med  $w$ , så der kan være forskellige retninger der er vinkelret på en given vektor  $u$ .  $\triangle$

## 8 ORTOGONALITET OG PROJEKTIONER

I python kan disse regnes i formen  $u^T v$  via @, men det giver en  $(1 \times 1)$ -matrix:

```
>>> import numpy as np
>>> u = np.array([1.0, -2.0, 3.0])[:, np.newaxis]
>>> v = np.array([0.0, 5.0, 7.0])[:, np.newaxis]
>>> u.T @ v
array([[11.]])
```

Så for at få en skalar skal vi tage indgangen  $[0, 0]$ :

```
>>> (u.T @ v)[0, 0]
11.0
```

Som praktisk alternativ er der funktionen `np.vdot`, som giver direkte en skalar og kræver ingen transponering:

```
>>> np.vdot(u, v)
11.0
```

**Lemma 8.2.** *Det indre produkt har de følgende regneegenskaber, for  $u, v, w \in \mathbb{R}^n$  og  $a, b \in \mathbb{R}$ :*

- (a)  $\langle u, v \rangle = \langle v, u \rangle$ ,
- (b)  $\langle au + bw, v \rangle = a\langle u, v \rangle + b\langle w, v \rangle$ ,
- (c)  $\langle u, av + bw \rangle = a\langle u, v \rangle + b\langle u, w \rangle$ ,
- (d)  $\langle v, v \rangle \geq 0$ ,
- (e) for  $v \neq 0$  er  $\langle v, v \rangle > 0$ .

*Bevis.* For del (a), bruger vi slutformlen i (8.1) til at få

$$\begin{aligned}\langle u, v \rangle &= u_0 v_0 + u_1 v_1 + \cdots + u_{n-1} v_{n-1} \\ &= v_0 u_0 + v_1 u_1 + \cdots + v_{n-1} u_{n-1} \\ &= \langle v, u \rangle,\end{aligned}$$

da  $u_0 v_0 = v_0 u_0$ , osv.

For dele (b) og (c) brug bare at  $\langle u, v \rangle = u^T v$  og de tilsvarende relationer for matrixmultiplikation.

For del (d), beregner vi

$$\langle v, v \rangle = v_0^2 + v_1^2 + \cdots + v_{n-1}^2.$$

## 8.1 STANDARD INDRE PRODUKT

Men  $v_k^2 \geq 0$ , så  $\langle v, v \rangle$  er sum af led større end eller lige med 0, og dermed er  $\langle v, v \rangle \geq 0$ . For del (e), hvis  $v \neq 0$ , så er der mindst en indgang  $v_k$  med  $v_k \neq 0$ . Men så har vi  $v_k^2 > 0$ , og  $\langle v, v \rangle \geq v_k^2 > 0$ , som påstået.  $\square$

Vi sætter

$$\|v\|_2 = \sqrt{\langle v, v \rangle} = \sqrt{v_0^2 + v_1^2 + \cdots + v_{n-1}^2} \quad (8.2)$$

til at være **2-normen** af  $v \in \mathbb{R}^n$ , og siger at  $v$  er en **enhedsvektor** hvis  $\|v\|_2 = 1$ .

I python kan  $\|v\|_2$  beregnes via `np.linalg.norm(v)`:

```
>>> import numpy as np
>>> v = np.array([1.0, -2.0, 3.0])
>>> np.linalg.norm(v)
3.7416573867739413
>>> np.sqrt(v[0]**2 + v[1]**2 + v[2]**2)
3.7416573867739413
```

**Proposition 8.3** (Pythagoras sætning). *For  $u, v$  ortogonal er*

$$\|u + v\|_2^2 = \|u\|_2^2 + \|v\|_2^2.$$

*Bevis.* Vi beregner dette kun ved brug af regnereglerne fra lemma 8.2.

$$\begin{aligned} \|u + v\|_2^2 &= \langle u + v, u + v \rangle = \langle u, u + v \rangle + \langle v, u + v \rangle && \text{lemma 8.2(b),} \\ &= \langle u, u \rangle + \langle u, v \rangle + \langle v, u \rangle + \langle v, v \rangle && \text{lemma 8.2(c),} \\ &= \|u\|_2^2 + 2\langle u, v \rangle + \|v\|_2^2 && \text{lemma 8.2(a).} \end{aligned}$$

Men  $u, v$  er ortogonal, så  $\langle u, v \rangle = 0$ , og vi har resultatet.  $\square$

**Lemma 8.4.** *Hvis  $v$  er en vektor in  $\mathbb{R}^n$  med  $v \neq 0$ , så er*

$$w = \frac{1}{\|v\|_2} v$$

*en enhedsvektor.*

*Bevis.* Vi begreger

$$\|w\|_2^2 = \langle w, w \rangle = \frac{1}{\|v\|_2^2} \langle v, v \rangle = 1,$$

da  $\langle v, v \rangle = \|v\|_2^2$ .  $\square$

## 8.2 Vinkel mellem vektorer

**Definition 8.5.** For  $u, v \in \mathbb{R}^n$  forskellig fra 0 siger vi at *vinklen* mellem  $u$  og  $v$  er  $0 \leq \theta \leq \pi$  med

$$\cos \theta = \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2}.$$

For  $u \perp v$  har vi  $\langle u, v \rangle = 0$ , så  $\theta = \pi/2 = 90^\circ$ , som stemmer overens med vores almindelig vinkelmål.

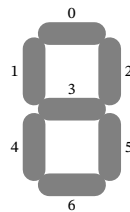
*Eksempel 8.6.* Vektorerne  $u = (1, -2, 3)$  og  $v = (0, 5, 7)$  fra eksempel 8.1 er ikke parallelle. De har  $\langle u, v \rangle = 11$ ,  $\|u\|_2 = \sqrt{1^2 + (-2)^2 + 3^2} = \sqrt{14}$ ,  $\|v\|_2 = \sqrt{0^2 + 5^2 + 7^2} = \sqrt{74}$ . Så

$$\cos \theta = \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2} = \frac{11}{\sqrt{14}\sqrt{74}} \approx \frac{11}{32,2} = 0,342,$$

som giver  $\theta \approx \cos^{-1}(0,342) = 1,22 \text{ rad} = 70,0^\circ$ . △

Lad os bemærke at for  $u \neq 0 \neq v$  har vi at  $u, v$  er parallelle, dvs.  $u = sv$  for et  $s \in \mathbb{R}$ , kun hvis  $u/\|u\|_2 = \pm v/\|v\|_2$ . Det sidste er det samme som at vinklen  $\theta$  er enten 0 eller  $\pi$ .

*Eksempel 8.7.* Et digitalt ur viser cifre ved at oplyse nogle elementer, der er arrangeret som nedenfor:



Standard cifre kan gemmes i en vektor som har et 1-tal for hvert segment der er oplyst, og 0-tal ellers. F.eks.

$$v_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \begin{array}{c} \text{[Diagram of digit 0 with all segments lit in red]} \end{array} \quad v_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{array}{c} \text{[Diagram of digit 8 with segments 1, 2, 3, 4, 5, 6 lit in red]} \end{array} \quad v_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \begin{array}{c} \text{[Diagram of digit 9 with segments 0, 1, 2, 3, 4, 5, 6 lit in red]} \end{array} \quad \dots$$



En dag viser uret



(8.3)

Hvilke cifre ligner dette mest?

Vi kan danne en matrix med søjler repræsenterende cifre 0 til 9:

```
>>> import numpy as np
>>> v = np.empty((7,10), dtype=float)
>>> v[:,0] = np.array([1,1,1,0,1,1,1])
>>> v[:,1] = np.array([0,0,1,0,0,1,0])
>>> v[:,2] = np.array([1,0,1,1,1,0,1])
>>> v[:,3] = np.array([1,0,1,1,0,1,1])
>>> v[:,4] = np.array([0,1,1,1,1,0,0])
>>> v[:,5] = np.array([1,1,0,1,0,1,1])
>>> v[:,6] = np.array([0,1,0,1,1,1,1])
>>> v[:,7] = np.array([1,0,1,0,0,1,0])
>>> v[:,8] = np.array([1,1,1,1,1,1,1])
>>> v[:,9] = np.array([1,1,1,1,0,1,0])
>>> v
array([[1., 0., 1., 1., 0., 1., 0., 1., 1., 1.],
       [1., 0., 0., 0., 1., 1., 1., 0., 1., 1.],
       [1., 1., 1., 1., 1., 0., 0., 1., 1., 1.],
       [0., 0., 1., 1., 1., 1., 1., 0., 1., 1.],
       [1., 0., 1., 0., 1., 0., 1., 0., 1., 0.],
       [1., 1., 0., 1., 0., 1., 1., 1., 1., 1.],
       [1., 0., 1., 1., 0., 1., 1., 0., 1., 0.]])
```

(8.3) repræsenteres så af vektoren

```
>>> u = np.array([0,1,0,1,1,1,0])[:, np.newaxis]
>>> u
array([[0],
       [1],
       [0],
       [1],
```

## 8 ORTOGONALITET OG PROJEKTIONER

```
[1],  
[1],  
[0]])
```

Vi kan beregne indre produkterne mellem søjlerne af  $v$  og  $u$  ved

```
>>> v.T @ u  
array([[3.],  
       [1.],  
       [2.],  
       [2.],  
       [3.],  
       [3.],  
       [4.],  
       [1.],  
       [4.],  
       [3.]])
```

Dette tæller hvor mange tændte elementer  $u$  har til fælles med hvert cifre, og vi ser den har 4 tændte elementer til fælles med 6 og 8.

Hvis vi skalærer søjlerne af  $v$  og vektoren  $u$  til enhedsvektor, kan vi i stedet beregne cosinus til vinklerne mellem  $u$  og hvert søjle via matrix-vektorprodukt

```
>>> vn = np.empty_like(v)  
>>> for i in range(10):  
...     vn[:, i] = v[:, i]/np.linalg.norm(v[:, i])  
...  
>>> un = u/np.linalg.norm(u)  
  
>>> cosines = vn.T @ un  
>>> cosines  
array([[0.61237244],  
       [0.35355339],  
       [0.4472136 ],  
       [0.4472136 ],  
       [0.75       ],  
       [0.67082039],  
       [0.89442719],
```

```
[0.28867513],
[0.75592895],
[0.67082039]])
```

Her bruges `np.empty_like(v)` til at danne en ny matrix med samme antal rækker og søjler som `v`. Fra resultatet ser vi at cosinus til vinklen til 6-tallet er størst, så dette er den bedste kandidat for det korrekte cifre.  $\triangle$

For at definitionen på vinklen giver mening har vi brug for at udtrykket for  $\cos \theta$  ligger mellem  $-1$  og  $1$ . Men dette følger fra:

**Sætning 8.8** (Cauchy-Schwarz ulighed). *Givet  $u, v \in \mathbb{R}^n$ , gælder*

$$|\langle u, v \rangle| \leq \|u\|_2 \|v\|_2, \quad (8.4)$$

*med lighed hvis og kun hvis  $u, v$  er parallelle.*

*Bevis.* Bemærk først at hvis  $u = 0$  eller  $v = 0$ , så er begge sidder af ulighed 0 og der er intet at vise. Vi derfor kikker på tilfældet hvor  $u \neq 0 \neq v$ . For  $a, b \in \mathbb{R}$  betragter vi  $\|au + bv\|_2^2$ . Vi har

$$\begin{aligned} 0 &\leq \|au + bv\|_2^2 = \langle au + bv, au + bv \rangle \\ &= a^2 \|u\|_2^2 + 2ab \langle u, v \rangle + b^2 \|v\|_2^2. \end{aligned} \quad (8.5)$$

Det følger at

$$-2ab \langle u, v \rangle \leq a^2 \|u\|_2^2 + b^2 \|v\|_2^2. \quad (8.6)$$

Vælg nu  $\varepsilon \in \{\pm 1\}$ ,  $a = -\varepsilon \|v\|_2$  og  $b = \|u\|_2$ . Så bliver (8.6) til

$$2\varepsilon \|u\|_2 \|v\|_2 \langle u, v \rangle \leq 2 \|u\|_2^2 \|v\|_2^2.$$

Vi deler med  $2\|u\|_2 \|v\|_2$ , som er strengt positivt, da  $u \neq 0 \neq v$ , til at få

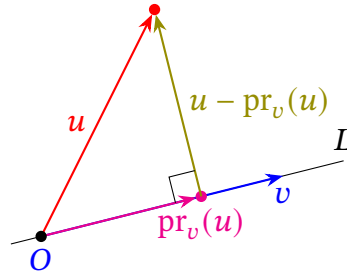
$$\varepsilon \langle u, v \rangle \leq \|u\|_2 \|v\|_2 \quad \text{for både } \varepsilon = +1 \text{ og } \varepsilon = -1.$$

Dette er det samme som (8.4).

Hvornår får vi lighed? Hvis  $u = 0$  eller  $v = 0$ , er der intet at vise:  $u, v$  er parallelle og vi har lighed. For  $u \neq 0 \neq v$ , får vi lighed netop når vi har lighed i (8.5) for et af vores særlig valg af  $a$  og  $b$ , dvs. netop når

$$0 = \|- \varepsilon \|v\|_2 u + \|u\|_2 v\|_2^2$$

for et  $\varepsilon \in \{\pm 1\}$ . Men dette sker kun for  $-\varepsilon \|v\|_2 u + \|u\|_2 v = 0$ , dvs.  $u/\|u\|_2 = \varepsilon v/\|v\|_2$ , som siger at  $u, v$  er parallelle.  $\square$



Figur 8.1: Projektion på en linje.

### 8.3 Projektion på en linje

Givet en vektor  $v \in \mathbb{R}^n$ , som er ikke nul, kan vi betragte den rette linje  $L$  igennem origo i retningen  $v$ :

$$L = \{sv \mid s \in \mathbb{R}\}.$$

Givet en anden vektor  $u \in \mathbb{R}^n$ , kan vi spørge hvilket punkt på  $L$  er tættest på  $u$ . Afstanden mellem  $u$  og  $sv$  er  $\|u - sv\|_2$ . At minimere  $\|u - sv\|_2$  er det samme som at minimere  $\|u - sv\|_2^2$ . Vi har

$$\|u - sv\|_2^2 = \langle u - sv, u - sv \rangle = \|u\|_2^2 - 2s\langle u, v \rangle + s^2\|v\|_2^2 =: p(s),$$

som er et andengradspolynomium i  $s$ . Toppunktet bestemmes via

$$0 = p'(s) = -2\langle u, v \rangle + 2s\|v\|_2^2,$$

så ligger ved  $s = \langle u, v \rangle / \|v\|_2^2$ . Dvs. punktet på  $L$ , som ligger tættest på  $u$ , er

$$\text{pr}_v(u) = \frac{\langle u, v \rangle}{\|v\|_2^2} v.$$

Vi kalder  $\text{pr}_v(u)$  *projektionen* af  $u$  langs  $v$ .

Det skal bemærkes at  $u - \text{pr}_v(u)$  er vinkelret på  $v$ , sammenlign med figur 8.1:

$$\begin{aligned} \langle u - \text{pr}_v(u), v \rangle &= \langle u, v \rangle - \langle \text{pr}_v(u), v \rangle = \langle u, v \rangle - \left\langle \frac{\langle u, v \rangle}{\|v\|_2^2} v, v \right\rangle \\ &= \langle u, v \rangle - \frac{\langle u, v \rangle}{\|v\|_2^2} \langle v, v \rangle = \langle u, v \rangle - \langle u, v \rangle \\ &= 0, \end{aligned}$$

### 8.3 PROJEKTION PÅ EN LINJE

da  $\langle v, v \rangle = \|v\|_2^2$ . Det følger fra Pythagoras sætning at

$$\|u\|_2^2 = \|\text{pr}_v(u)\|_2^2 + \|u - \text{pr}_v(u)\|_2^2,$$

så afstanden fra  $u$  til dens projektion er

$$\|u - \text{pr}_v(u)\|_2 = \sqrt{\|u\|_2^2 - \|\text{pr}_v(u)\|_2^2}.$$

*Eksempel 8.9.* Betragt  $v = (1, 2) \in \mathbb{R}^2$  og  $u = (2, 3)$ . Vi har  $\langle u, v \rangle = 2 \times 1 + 3 \times 2 = 8$ ,  $\|v\|_2^2 = 1^2 + 2^2 = 5$ , giver

$$\text{pr}_v(u) = \frac{8}{5} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 8/5 \\ 16/5 \end{bmatrix}.$$

Afstanden fra  $u$  til  $L$  er så

$$\begin{aligned} \sqrt{\|u\|_2^2 - \|\text{pr}_v(u)\|_2^2} &= \sqrt{(2^2 + 3^2) - ((8/5)^2 + (16/5)^2)} \\ &= \sqrt{13 - (64/5)} = \sqrt{1/5}. \end{aligned}$$

△

Observer også at for et punkt  $sv$  på  $L$  er projektionen punktet selv:

$$\text{pr}_v(sv) = sv,$$

og derved har vi

$$\text{pr}_v(\text{pr}_v(u)) = \text{pr}_v(u). \quad (8.7)$$

Projektion kan skrives som en matrix multiplikation

$$\text{pr}_v(u) = \frac{1}{\|v\|_2^2} v \langle v, u \rangle = \frac{1}{\|v\|_2^2} v v^T u = P u$$

for matricen

$$P = \frac{1}{\|v\|_2^2} v v^T.$$

Ligning (8.7) siger at

$$P^2 = P.$$

Vi har desuden at

$$P^T = P,$$

dvs. at  $P$  er også en *symmetrisk* matrix.

## 8 ORTOGONALITET OG PROJEKTIONER

*Eksempel 8.10.* For  $v = (1, 2) \in \mathbb{R}^2$ , har vi

$$P = \frac{1}{5} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}.$$

△

Som eksempel i python har vi

```
>>> import numpy as np
>>> v = np.array([1.0, -2.0, 3.0])[:, np.newaxis]
>>> normsq = np.vdot(v, v)
>>> p = (1/normsq) * v @ v.T
>>> p
array([[ 0.07142857, -0.14285714,  0.21428571],
       [-0.14285714,  0.28571429, -0.42857143],
       [ 0.21428571, -0.42857143,  0.64285714]])
```

som er symmetrisk

```
>>> np.all(p.T == p)
True
```

og opfylder  $P^2 = P$  inden for machine epsilon

```
>>> np.allclose(p @ p, p, atol = np.finfo(float).eps)
True
```

### 8.4 Ortogonalitet

Lad os nu kikke på et større antal vektorer.

**Definition 8.11.** En samling vektorer  $v_0, v_1, \dots, v_{k-1}$  er *ortogonal* hvis

$$\langle v_i, v_j \rangle = 0, \quad \text{for alle } i \neq j.$$

Hvis alle vektorerne  $v_0, v_1, \dots, v_{k-1}$  er desuden enhedsvektor, så siger vi at samlingen er *ortonormal*.

*Eksempel 8.12.* Samlingen

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix}$$

er ortogonal, men ingen af vektorerne har længde 1, så samlingen er ikke ortonormal.  $\triangle$

*Eksempel 8.13.* Standardvektorerne  $e_0, e_1, \dots, e_{n-1}$  i  $\mathbb{R}^n$  er ortonormal.  $\triangle$

*Eksempel 8.14.* For  $c^2 + s^2 = 1$  er vektorerne

$$\begin{bmatrix} c \\ s \end{bmatrix}, \quad \begin{bmatrix} -s \\ c \end{bmatrix}$$

ortonormal. Det samme gælder parret

$$\begin{bmatrix} c \\ s \end{bmatrix}, \quad \begin{bmatrix} s \\ -c \end{bmatrix}.$$

$\triangle$

Bemærk betingelsen for at  $v_0, v_1, \dots, v_{k-1}$  er ortonormal er det samme som

$$\langle v_i, v_j \rangle = \begin{cases} 1, & \text{for } i = j, \\ 0, & \text{for } i \neq j. \end{cases}$$

Ofte er det nemmere at finde vektorer der er ortogonal end ortonormal. Men så forskelligt er to begreber ikke:

**Lemma 8.15.** Hvis  $v_0, v_1, \dots, v_{k-1}$  er ortogonal og alle vektorer er forskellige fra nul, så er

$$\frac{1}{\|v_0\|_2} v_0, \frac{1}{\|v_1\|_2} v_1, \dots, \frac{1}{\|v_{k-1}\|_2} v_{k-1}$$

en ortonormal samling.

*Bevis.* Vi har  $\langle v_i, v_j \rangle = 0$  for  $i \neq j$ . Desuden er  $\langle v_i, v_i \rangle = \|v_i\|_2^2 > 0$ , da  $v_i \neq 0$ . Resultatet følger nu fra lemma 8.4, og

$$\left\langle \frac{1}{\|v_i\|_2} v_i, \frac{1}{\|v_j\|_2} v_j \right\rangle = \frac{1}{\|v_i\|_2 \|v_j\|_2} \langle v_i, v_j \rangle = 0, \quad \text{for } i \neq j.$$

$\square$

## 8 ORTOGONALITET OG PROJEKTIONER

*Eksempel 8.16.* Vektorerne  $u = (1, -2, 3)$ ,  $w = (3, 0, -1)$  er ortogonal, som beregnet i eksempel 8.1. Men

$$\langle u, u \rangle = 1^2 + (-2)^2 + 3^2 = 14 \neq 1,$$

så de er ikke ortonormal. Til gengæld får vi en ortonormal samling ved at skalære disse vektorer til enhedsvektorer, dvs.

$$\frac{u}{\|u\|_2} = \frac{1}{\sqrt{14}} \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}, \quad \frac{w}{\|w\|_2} = \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 0 \\ -1 \end{bmatrix}$$

er ortonormal. Δ

Lad  $v_0, v_1, \dots, v_{k-1}$  være en ortogonal samling af vektorer. En styrke ved sådan en samling er det er nemt at dekomponere andre vektor som kombinationer af  $v_0, v_1, \dots, v_{k-1}$ , uden at løse lineære ligningssystemer.

**Proposition 8.17.** *Hvis*

$$u = x_0 v_0 + x_1 v_1 + \dots + x_{k-1} v_{k-1} \tag{8.8}$$

hvor  $v_0, v_1, \dots, v_{k-1}$  er ortogonal, og ingen er 0, så er

$$x_0 = \frac{\langle u, v_0 \rangle}{\|v_0\|_2^2}, \quad x_1 = \frac{\langle u, v_1 \rangle}{\|v_1\|_2^2}, \quad \dots, \quad x_{k-1} = \frac{\langle u, v_{k-1} \rangle}{\|v_{k-1}\|_2^2}. \tag{8.9}$$

Desuden er

$$\|u\|_2^2 = x_0^2 \|v_0\|_2^2 + x_1^2 \|v_1\|_2^2 + \dots + x_{k-1}^2 \|v_{k-1}\|_2^2. \tag{8.10}$$

Den sidste ligning kendes som [Parsevals identitet](#).

*Bevis.* Lad os tage det indre produkt af (8.8) med  $v_i$ . Så får vi

$$\begin{aligned} \langle u, v_i \rangle &= \langle x_0 v_0 + x_1 v_1 + \dots + x_{k-1} v_{k-1}, v_i \rangle \\ &= x_0 \langle v_0, v_i \rangle + x_1 \langle v_1, v_i \rangle + \dots + x_{k-1} \langle v_{k-1}, v_i \rangle. \end{aligned}$$

Det eneste led på den højre side, som er ikke nul, er leddet  $x_i \langle v_i, v_i \rangle = x_i \|v_i\|_2^2$ . Det følger at  $x_i = \langle u, v_i \rangle / \|v_i\|_2^2$ , som påstået.



Tilsvarende har vi

$$\begin{aligned}\|u\|_2^2 &= \langle x_0 v_0 + \cdots + x_{k-1} v_{k-1}, x_0 v_0 + \cdots + x_{k-1} v_{k-1} \rangle \\ &= \sum_{i,j=0}^{k-1} x_i x_j \langle v_i, v_j \rangle.\end{aligned}$$

Men  $\langle v_i, v_j \rangle$  er kun forskellig fra nul for  $i = j$  og dens værdi er  $\|v_i\|_2^2$ , så

$$\|u\|_2^2 = \sum_{i=0}^{k-1} x_i^2 \|v_i\|_2^2,$$

som ønsket.  $\square$

*Eksempel 8.18.* Lad  $v_0, v_1, v_2$  være vektorerne fra eksempel 8.12. Vi har  $\|v_0\|_2^2 = 3$ ,  $\|v_1\|_2^2 = 8$  og  $\|v_2\|_2^2 = 6$ . For  $u = (0, 1, 0)$ , har vi så

$$\begin{aligned}u &= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \frac{-2}{8} \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix} + \frac{-1}{6} \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix} \\ &= \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \frac{1}{4} \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix} - \frac{1}{6} \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix}.\end{aligned}$$

$\triangle$

Givet (8.8) og (8.9) med  $v_i, i = 0, \dots, k-1$ , ortogonal og ingen 0, kan vi skrive den højre side af (8.8), som

$$\text{pr}_{v_0}(u) + \text{pr}_{v_1}(u) + \cdots + \text{pr}_{v_{k-1}}(u).$$

Husk at  $\text{pr}_v(u) = Pu$  for  $P = (1/\|v\|_2^2)vv^T$ . Vi har så at den højre side af (8.8) er

$$\begin{aligned}&\frac{\langle v_0, u \rangle}{\|v_0\|_2^2} v_0 + \frac{\langle v_1, u \rangle}{\|v_1\|_2^2} v_1 + \cdots + \frac{\langle v_{k-1}, u \rangle}{\|v_{k-1}\|_2^2} v_{k-1} \\ &= \left( \frac{1}{\|v_0\|_2^2} v_0 v_0^T + \frac{1}{\|v_1\|_2^2} v_1 v_1^T + \cdots + \frac{1}{\|v_{k-1}\|_2^2} v_{k-1} v_{k-1}^T \right) u.\end{aligned}$$

**Definition 8.19.** For  $v_0, v_1, \dots, v_{k-1}$  ortogonal, med alle  $v_i \neq 0$ , er *projektionen langs  $v_0, v_1, \dots, v_{k-1}$*  givet ved  $Pu$ , hvor  $P$  er matricen

$$P = \frac{1}{\|v_0\|_2^2} v_0 v_0^T + \frac{1}{\|v_1\|_2^2} v_1 v_1^T + \cdots + \frac{1}{\|v_{k-1}\|_2^2} v_{k-1} v_{k-1}^T. \quad (8.11)$$

**Proposition 8.20.** For  $P$  som i definition 8.19 gælder

(a)  $P^2 = P$  og

(b)  $P^T = P$ .

Desuden for vilkårlig  $w \in \mathbb{R}^n$  har vi

(c)  $w - Pw \perp v_i$  for alle  $i$ , og

(d)  $\|w\|_2^2 = \|Pw\|_2^2 + \|w - Pw\|_2^2$ .

Det følger at  $u = Pw$  er vektoren af formen (8.8), som er tættest på  $w$ .

*Bevis.* For at vise  $P^2 = P$  er det nok at bemærke at

$$(v_i v_i^T)(v_j v_j^T) = v_i(v_i^T v_j)v_j^T = \langle v_i, v_j \rangle v_i v_j^T = \begin{cases} 0, & \text{for } i \neq j, \\ \|v_i\|_2^2 v_i v_i^T, & \text{for } i = j. \end{cases}$$

Så følger resultatet ved at gange  $P^2$  ud.

Ligningen  $P^T = P$  følger fra at  $v_i v_i^T$  er symmetrisk.

Observer at  $\langle Pw, v_i \rangle = \langle v_i, Pw \rangle = v_i^T Pw$  og at  $v_i^T v_j v_j^T = \langle v_i, v_j \rangle v_j^T$ . Så har vi

$$\langle w - Pw, v_i \rangle = \langle w, v_i \rangle - v_i^T Pw = \langle w, v_i \rangle - \frac{1}{\|v_i\|_2^2} \|v_i\|_2^2 v_i^T w = \langle w, v_i \rangle - \langle v_i, w \rangle = 0.$$

Identiteten for  $\|w\|_2^2$  følger nu fra Pythagoras sætning.

Antag at  $y = y_0 v_0 + \dots + y_{k-1} v_{k-1}$  er ikke lige med  $Pw$ . Så er  $Pw - y \neq 0$  og lemma 8.2(e) giver  $\|Pw - y\|_2^2 > 0$ . Da  $w - Pw$  er vinkelret på hver  $v_i$ , er  $w - Pw$  vinkelret på både  $y$  og  $Pw$ , og så vinkelret på  $Pw - y$ . Det følger fra Pythagoras at

$$\begin{aligned} \|w - y\|_2^2 &= \|(w - Pw) + (Pw - y)\|_2^2 = \|w - Pw\|_2^2 + \|Pw - y\|_2^2 \\ &> \|w - Pw\|_2^2. \end{aligned}$$

Dvs. at afstanden fra  $w$  til  $y$  er større end afstanden fra  $w$  til  $Pw$ . □

**Eksempel 8.21.** Lad  $v_0 = (1, -2, 3)$ ,  $v_1 = (3, 0, -1)$  være det ortogonale par  $u$ ,  $w$  fra eksempel 8.1. Vi kan beregne

$$P \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \frac{-1}{14} \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} + \frac{3}{10} \begin{bmatrix} 3 \\ 0 \\ -1 \end{bmatrix} = \frac{1}{35} \begin{bmatrix} 29 \\ 5 \\ -3 \end{bmatrix}.$$

△

*Eksempel 8.22.* Lad os forsøge at approksimere funktionen  $y = e^x$  for  $-1 \leq x \leq 1$  med polynomier af grad højst 2. Vi vil gøre dette numerisk i python.

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

Vi start med funktioner 1,  $x$  og  $x^2$  evalueret ved 100 punkter jævnt fordelt over  $[-1, 1]$ :

```
n = 100
x = np.linspace(-1, 1, 100)

# funktion 1
v0 = np.ones(100)[:, np.newaxis]
# funktion x
v1 = x[:, np.newaxis]
# funktion x**2
u2 = (x**2)[:, np.newaxis]
```

$v_0$  svarer til 1,  $v_1$  til  $x$  og  $u_2$  til  $x^2$ . Vi ser at  $v_0$  er stort set vinkelret på  $v_1$

```
print(np.vdot(v0, v1))
```

8.215650382226158e-15

og at  $v_1$  er næsten vinkelret på  $u_2$

```
print(np.vdot(v1, u2))
```

7.216449660063518e-15

men  $u_2$  er ikke vinkelret på  $v_0$

```
print(np.vdot(v0, u2))
```

34.006734006734014

Sæt  $v_2$  til at være  $u_2$  minus projektionen af  $u_2$  på  $v_0$

## 8 ORTOGONALITET OG PROJEKTIONER

```
v2 = u2 - np.vdot(u2, v0) / np.vdot(v0, v0) * v0
```

(Vi har ikke brug for at trække projektionen på  $v_1$ , da  $u_2$  og  $v_0$  er begge vinkelret på  $v_1$ .) Så er  $v_2$  stort set vinkelret på  $v_0$  og  $v_1$

```
print(np.vdot(v0, v2))
print(np.vdot(v1, v2))
```

```
-1.1102230246251565e-15
4.884981308350689e-15
```

og vi kan bruge  $v_0, v_1, v_2$  som ortogonal samling. Disse funktioner er tegnet i figur 8.2.

```
fig, ax = plt.subplots()
ax.plot(x, v0[:, 0], label='v0')
ax.plot(x, v1[:, 0], label='v1')
ax.plot(x, v2[:, 0], label='v2')
ax.legend()
```

Lad  $u$  være repræsentationen af eksponentialfunktionen

```
u = np.exp(x)[:, np.newaxis]
```

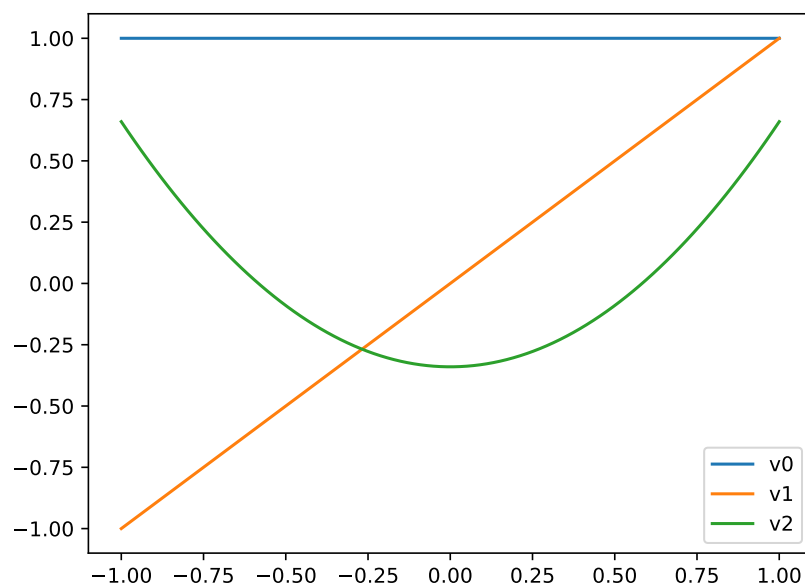
Dens projektion er så

```
u_proj = (1/np.vdot(v0, v0) * v0 @ (v0.T @ u)
          + 1/np.vdot(v1, v1) * v1 @ (v1.T @ u)
          + 1/np.vdot(v2, v2) * v2 @ (v2.T @ u))
```

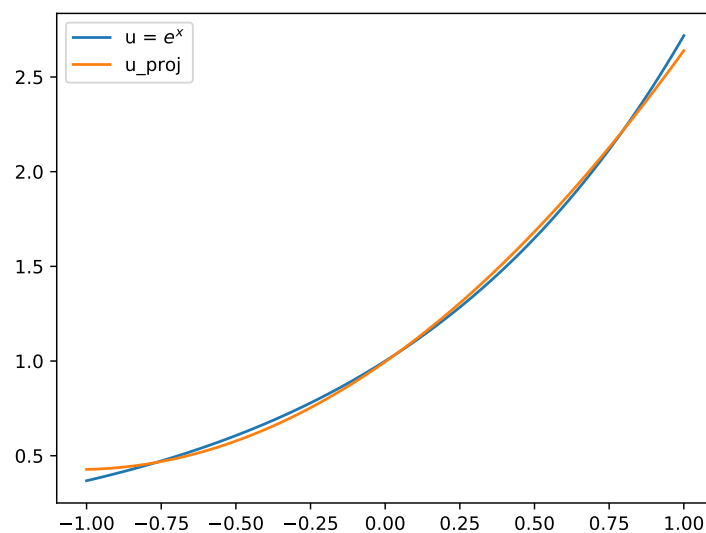
En plot af  $u$  og  $u_{\text{proj}}$  giver figur 8.3. Vi ser at  $u_{\text{proj}}$  ligger meget tæt på  $u$  over hele intervallet.

```
fig, ax = plt.subplots()
ax.plot(x, u[:, 0], label='u =  $e^{\{x\}}$ ')
ax.plot(x, u_proj[:, 0], label='u_proj')
ax.legend()
```

## 8.4 ORTOGONALITET

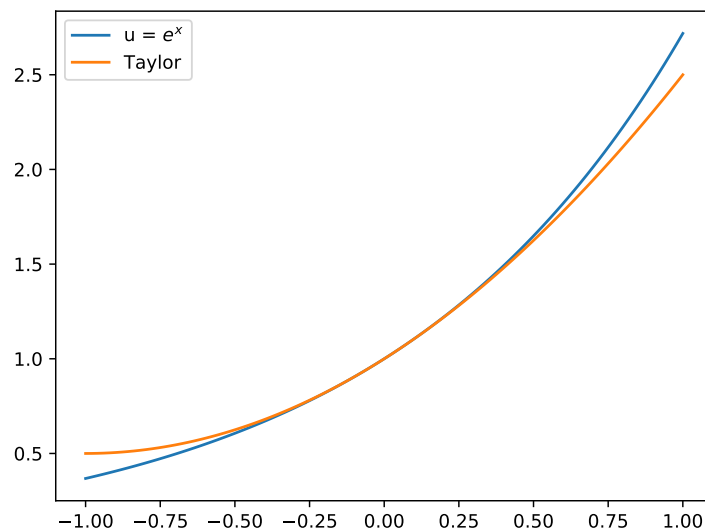


Figur 8.2: Den ortogonale samling  $v_0$ ,  $v_1$ ,  $v_2$  af polynomier af højst anden grad.



Figur 8.3: Numerisk approksimation af  $y = e^x$ ,  $-1 \leq x \leq 1$ .

## 8 ORTOGONALITET OG PROJEKTIONER



Figur 8.4: Approximation af  $y = e^x$ ,  $-1 \leq x \leq 1$ , via dens anden ordens Taylor udvikling er kun godt tæt på  $x = 0$ .

Dette er i modsætning til den anden ordens Taylor udviklingen

$$e^x \approx 1 + x + \frac{1}{2}x^2$$

som kun er en god tilnærmelse til  $u$  tæt på  $x = 0$ , se figur 8.4.

```
fig, ax = plt.subplots()
ax.plot(x, u[:, 0], label='u =  $e^{\mathbf{x}}$ ')
ax.plot(x, (1. + x + (1/2.)*x**2), label='Taylor')
ax.legend()
```

Δ

# Kapitel 9

## Ortogonal matricer

Ortogonal og ortonormale samlinger af vektorer er tæt forbundne med ortogonale matricer. Desuden er ortogonale matricer det foretrukne værktøj for numerisk arbejde.

### 9.1 Definition og første eksempler

**Definition 9.1.** En kvadratisk matrix  $A \in \mathbb{R}^{n \times n}$  er *ortogonal* hvis

$$A^T A = I_n.$$

*Eksempel 9.2.* For  $n = 2$  er enhver rotationsmatrix ortogonal. Nemlig, for  $c^2 + s^2 = 1$  har vi

$$R^T R = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}^T \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} c^2 + s^2 & -cs + sc \\ -sc + cs & s^2 + c^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Tilsvarende gælder for spejlingsmatricer

$$M^T M = \begin{bmatrix} c & s \\ s & -c \end{bmatrix} \begin{bmatrix} c & s \\ s & -c \end{bmatrix} = \begin{bmatrix} c^2 + s^2 & 0 \\ 0 & c^2 + s^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Δ

*Eksempel 9.3.* Lad  $u \in \mathbb{R}^n$  være en enhedsvektor. Så er

$$A = I_n - 2uu^T$$

## 9 ORTOGONALE MATRICER

er en ortogonal matrix: det ydre produkt  $uu^T$  er symmetrisk, og vi har

$$\begin{aligned} A^T A &= (I_n - 2uu^T)(I_n - 2uu^T) = I_n - 4uu^T + 4(uu^T)(uu^T) \\ &= I_n - 4uu^T + 4u(u^T u)u^T = I_n, \end{aligned}$$

da  $u^T u = \langle u, u \rangle = 1$ .

For eksempel for  $n = 4$ , er  $u = (1/2, -1/2, 1/2, -1/2)$  en enhedsvektor, så

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - 2 \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix} \begin{bmatrix} 1/2 & -1/2 & 1/2 & -1/2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1/2 & -1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & -1/2 & 1/2 \end{bmatrix} \\ &= \begin{bmatrix} 1/2 & 1/2 & -1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 & 1/2 \end{bmatrix} \end{aligned}$$

en ortogonal matrix. △

## 9.2 Egenskaber

Ortogonale matricer er netop dem, der bevarer det indre produkt. Dette medfører at ortogonale matricer også bevarer norm, afstand og vinkel, da disse er defineret ud fra det indre produkt. Så enhver figur bevarer dens form og størrelse under en transformation givet ved en ortogonal matrix.

For at vise at ortogonale matricer bevarer det indre produkt, har vi brug for at vide hvordan transponering påvirker matrixprodukter.

**Proposition 9.4.** For  $B \in \mathbb{R}^{m \times n}$  og  $C \in \mathbb{R}^{n \times p}$  har vi

$$(BC)^T = C^T B^T. \quad (9.1)$$

Observér at  $C^T \in \mathbb{R}^{p \times n}$ , og  $B^T \in \mathbb{R}^{n \times m}$ , så  $C^T B^T$  er det eneste produkt via generelt kan danne ud fra  $C^T$  og  $B^T$ . Desuden er  $(BC)^T$  og  $C^T B^T$  begge matricer i  $\mathbb{R}^{p \times m}$ .



*Bevis.* Husk at  $BC$  har  $(i, j)$ -indgang  $\sum_{k=0}^{n-1} b_{ik}c_{kj}$ . Så dette er den  $(j, i)$ te indgang af  $(BC)^T$ . Men  $C^T B^T$  har  $(j, i)$ -indgang

$$\sum_{k=0}^{n-1} (C^T)_{jk} (B^T)_{ki} = \sum_{k=0}^{n-1} c_{kj} b_{ik} = ((BC)^T)_{ji},$$

som påstået.  $\square$

**Proposition 9.5.** For  $A \in \mathbb{R}^{n \times n}$  ortogonal gælder

$$\langle Au, Av \rangle = \langle u, v \rangle, \quad \text{for alle } u, v \in \mathbb{R}^n. \quad (9.2)$$

Specielt gælder  $\|Au\|_2 = \|u\|_2$  for alle  $u \in \mathbb{R}^n$ .

Omvendt, hvis  $A \in \mathbb{R}^{n \times n}$  opfylder (9.2), så er  $A$  en ortogonal matrix.

*Bevis.* For  $A$  ortogonal har vi, ved brug af (9.1),

$$\langle Au, Av \rangle = (Au)^T (Av) = u^T A^T Av = u^T I_n v = u^T v,$$

så (9.2) holder.

Omvendt hvis (9.2) gælder, så er den  $(i, j)$ te indgang af  $A^T A$  givet ved

$$(A^T A)_{ij} = e_i^T (A^T A) e_j = \langle Ae_i, Ae_j \rangle = \langle e_i, e_j \rangle = \begin{cases} 1, & \text{for } i = j, \\ 0, & \text{for } i \neq j, \end{cases}$$

som er den  $(i, j)$ te indgang af  $I_n$ . Så  $A^T A = I_n$  og  $A$  er en ortogonal matrix.  $\square$

*Bemærkning 9.6.* Numerisk er ortogonale matricer meget nyttig. Antag at en vektor  $v \in \mathbb{R}^n$  har en fejl givet ved en vektor  $w \in \mathbb{R}^n$  med  $\|w\|_2 < c$ , dvs.  $v$  repræsenteres af  $v+w$ . For  $A$  ortogonal har vi at  $Av$  repræsenteres af  $A(v+w) = Av + Aw$ , og fejlen er så  $Aw$ . Da  $A$  er ortogonal har vi  $\|Aw\|_2 = \|w\|_2$ , så fejlen i  $Av$  opfylder den samme estimering  $\|Aw\|_2 < c$ , som den oprindelig fejl  $w$ .  $\diamond$

Desuden har ortogonale matricer andre pæne egenskaber. Specielt kan deres invers beregnes ved simpel ombytning af indgangerne.

**Proposition 9.7.** (a) Hvis  $A$  er ortogonal, så er  $A$  invertibel med invers  $A^T$ .

(b) For  $A, B \in \mathbb{R}^{n \times n}$  ortogonale, er  $AB$  ortogonal.

*Bevis.* Del (a) følger direkte fra  $A^T A = I_n$  og sætning 7.2.

For del (b), bruger vi (9.1):

$$(AB)^T (AB) = (B^T A^T) (AB) = B^T (A^T A) B = B^T I_n B = B^T B = I_n,$$

så  $AB$  er ortogonal.  $\square$

### 9.3 Ortonormale vektorer og ortogonale matricer

Givet en samling vektorer  $v_0, v_1, \dots, v_{k-1}$  i  $\mathbb{R}^n$  kan vi danne en matrix  $V$  med disse vektorer som søjler

$$V = [v_0 \mid v_1 \mid \dots \mid v_{k-1}] \in \mathbb{R}^{n \times k}.$$

Matricen

$$G = V^T V \in \mathbb{R}^{k \times k}$$

kaldes *Grammatricen* for  $v_0, \dots, v_{k-1}$ , efter Jørgen Pedersen Gram (1850–1916). Vi har

$$\begin{aligned} G = V^T V &= \begin{bmatrix} v_0^T \\ v_1^T \\ \vdots \\ v_{k-1}^T \end{bmatrix} [v_0 \mid v_1 \mid \dots \mid v_{k-1}] \\ &= \begin{bmatrix} v_0^T v_0 & v_0^T v_1 & \dots & v_0^T v_{k-1} \\ v_1^T v_0 & v_1^T v_1 & \dots & v_1^T v_{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k-1}^T v_0 & v_{k-1}^T v_1 & \dots & v_{k-1}^T v_{k-1} \end{bmatrix}, \end{aligned}$$

dvs. den  $(i, j)$ te indgang i  $G$  er det indre produkt mellem  $v_i$  og  $v_j$ :

$$G = (g_{ij}) = (\langle v_i, v_j \rangle).$$

Det følger at

**Lemma 9.8.** Samlingen  $v_0, v_1, \dots, v_{k-1}$  i  $\mathbb{R}^n$  er ortonormal hvis og kun hvis Grammatricen  $G = V^T V \in \mathbb{R}^{k \times k}$  er lige med identitetsmatricen  $I_k$ :

$$G = V^T V = I_k.$$

□

**Sætning 9.9.** En ortonormal samling  $v_0, v_1, \dots, v_{k-1}$  i  $\mathbb{R}^n$  har højst  $k = n$  vektorer.

### 9.3 ORTONORMALE VEKTORER OG ORTOGONALE MATRICER

*Bevis.* Hvis  $k \geq n$ , så betragter vi de første  $n$  vektorer og deres tilhørende matrix

$$A = [v_0 \mid v_1 \mid \dots \mid v_{n-1}] \in \mathbb{R}^{n \times n}.$$

Matricen  $A$  er kvadratisk, og lemma 9.8 giver at  $A^T A = I_n$ , så  $A$  er ortogonal. Det følger fra proposition 9.7(a) at  $A$  er invertibel med invers  $A^T$ . Lad  $u \in \mathbb{R}^n$  være en vektor, som står vinkelret på alle de  $n$  vektorer  $v_0, v_1, \dots, v_{n-1}$ , dvs.  $v_i^T u$  for  $i = 0, \dots, n-1$ . Da er

$$A^T u = \begin{bmatrix} v_0^T \\ v_1^T \\ \vdots \\ v_{n-1}^T \end{bmatrix} u = \begin{bmatrix} v_0^T u \\ v_1^T u \\ \vdots \\ v_{n-1}^T u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Men  $A^T$  er invertibel, så lemma 7.7 giver at ligningssystemet  $A^T u = 0$  har kun én løsning, nemlig  $u = 0$ . Så  $u = 0$ , og dermed kan den ortonormale samling  $v_0, v_1, \dots, v_{n-1}$  ikke tilføjes flere enhedsvektorer.  $\square$

En ortonormal samling i  $\mathbb{R}^n$  med netop  $n$  vektorer kaldes en *ortonormal basis* for  $\mathbb{R}^n$ .

Lemma 9.8 giver straks det følgende resultat.

**Proposition 9.10.** Søjlerne af en ortogonal matrix  $A \in \mathbb{R}^{n \times n}$  udgør en ortonormal basis for  $\mathbb{R}^n$ .  $\square$

*Eksempel 9.11.* Fra en rotationsmatrix  $R = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}$ ,  $c^2 + s^2 = 1$ , har vi at

$$\begin{bmatrix} c \\ s \end{bmatrix}, \quad \begin{bmatrix} -s \\ c \end{bmatrix}$$

er en ortonormal basis for  $\mathbb{R}^2$ .

Eksempel 9.3 giver at

$$\begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}, \quad \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ -1/2 \end{bmatrix}, \quad \begin{bmatrix} -1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}, \quad \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

er en ortonormal basis for  $\mathbb{R}^4$ .  $\triangle$

## 9.4 Householdermatrix

Lad os skrive eksempel 9.3 på en lidt mere generel måde.

**Definition 9.12.** En *Householdermatrix* er en kvadratisk matrix  $H \in \mathbb{R}^{n \times n}$ , som er ortogonal og har formen

$$H = I_n - svv^T$$

for et  $v \in \mathbb{R}^n$  og et  $s \in \mathbb{R}$ . Vektoren  $v$  kaldes den *Householdervektor*.

Hvis  $v \neq 0$  og  $s \neq 0$ , har vi

$$s = \frac{2}{\|v\|_2^2}. \quad (9.3)$$

Dette følger fra først at bemærke at  $H$  er symmetrisk, og så regnestykket

$$\begin{aligned} I_n &= H^T H = (I_n - svv^T)(I_n - svv^T) = I_n - 2svv^T + s^2 v(v^T v)v^T \\ &= I_n + s((-2 + s\|v\|_2^2)vv^T), \end{aligned}$$

som giver  $s\|v\|_2^2 = 2$  og dermed (9.3). Bemærk at vi har altid  $s \geq 0$ .

**Proposition 9.13.** Enhver Householdermatrix  $H = I_n - svv^T$  opfylder

- (a)  $H^T = H$ ,
- (b)  $H^2 = I_n$ ,
- (c)  $Hv = -v$ , hvis  $s \neq 0$ , og
- (d)  $Hw = w$  for alle  $w \perp v$ .

Det vil sige at  $H$  er en spejlingsmatrix, der sender  $v$  til  $-v$  og fastholder alle vektorer, som står vinkelret på  $v$ . Se figur 9.1.

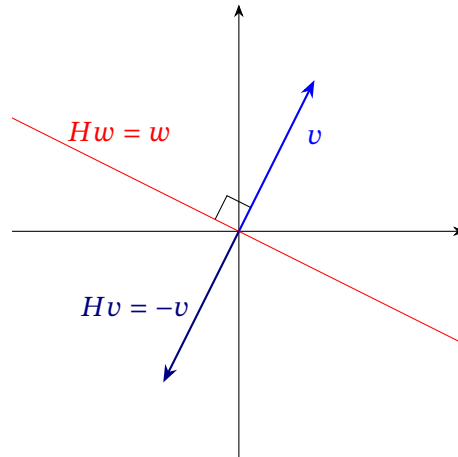
*Bevis.* Vi har allerede bemærket at  $H^T = H$ , og så er  $H^2 = H^T H$ , som vi har vist er lige med  $I_n$ . For de sidste to dele, først har vi

$$Hv = (I_n - svv^T)v = v - sv(v^T v) = v - s\|v\|_2^2 v = v - 2v = -v,$$

og for  $w \perp v$  har vi  $v^T w = 0$ , så

$$Hw = w - sv(v^T w) = w - 0 = w,$$

som påstået. □



Figur 9.1: Effekten af en Householdermatrix.

I mange situationer, f.eks. løsning af ligningssystemer, vil vi gerne flytte en given vektor til en med mange indgange lige med 0. Householdermatricer er rigtig til god til dette. Faktisk kan vi flytte til et multiplum af  $e_0 = (1, 0, \dots, 0)$ . For numerisk sikkerhed er det bedst at være forsigtig med fortegnet af dette multiplum. I det næste resultat træffer vi det rigtige valg.

**Proposition 9.14.** Givet en enhedsvektor  $u = (u_0, u_1, \dots, u_{n-1}) \in \mathbb{R}^n$ , sæt

$$\varepsilon = \begin{cases} -1, & \text{hvis } u_0 \geq 0, \\ +1, & \text{hvis } u_0 < 0. \end{cases}$$

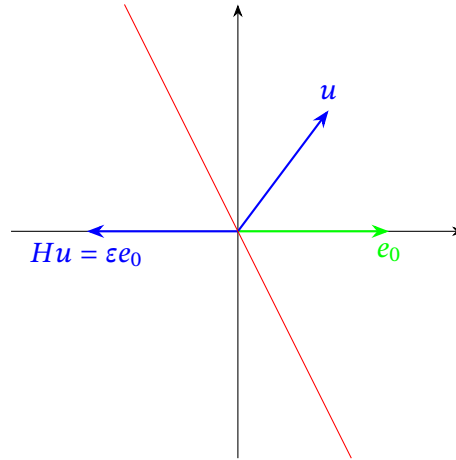
Så findes der en Householdermatrix  $H = I_n - svv^T$  med

$$He_0 = \varepsilon u, \quad v = \begin{bmatrix} 1 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} \quad \text{og} \quad s = 1 + |u_0|.$$

*Bevis.* Vi skal finde et passende  $v$ . Først har vi  $He_0 = e_0 - sv(v^T e_0)$ , men  $v^T e_0 = 1$ , så  $He_0 = e_0 - sv$ . Ligningen  $He_0 = \varepsilon u$ , giver  $e_0 - sv = \varepsilon u$ , som vi omskriver til

$$sv = e_0 - \varepsilon u.$$

## 9 ORTOGONALE MATRICER



Figur 9.2: Spejling af en enhedsvektor  $u$  til  $\varepsilon e_0$  via en Householdermatrix.

Kikkes på den 0te indgang har vi

$$s = 1 - \varepsilon u_0 = 1 + |u_0| \geq 1.$$

Sæt nu

$$v = \frac{1}{s}(e_0 - \varepsilon u). \quad (9.4)$$

Beregningen  $\|v\|_2^2 = (1 + 2\varepsilon \langle e_0, u \rangle + \|u\|_2^2)/s^2 = (1 - 2|u_0| + 1)/s^2 = 2(1 + |u_0|)/s^2 = 2/s$ , bekræfter at  $s = 2/\|v\|_2^2$  og dermed at  $H$  er en Householdermatrix.  $\square$

Husk at  $H^2 = I_n$ , så ligningen  $He_0 = \varepsilon u$  giver

$$Hu = \varepsilon e_0 = \begin{bmatrix} \varepsilon \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

*Bemærkning 9.15.* Fortegnet  $\varepsilon$  er valgt netop således at vi får  $s \geq 1$ . Da  $u$  er en enhedsvektor har vi  $|u_0| \leq 1$ , så vi har også  $s \leq 2$ . Det betyder at i ligningen (9.4) at der er ikke numeriske problemer med at dele med  $s$ .  $\diamond$

*Eksempel 9.16.* Vi ønsker at bestemme en Householdermatrix  $H$  der flytter  $x = (1, 2, 3)$  til et multiplum af  $e_0$ .

## 9.4 HOUSEHOLDERMATRIX

Sæt  $u = x/\|x\|_2 = x/\sqrt{14}$ . Så er  $u_0 = 1/\sqrt{14}$ ,  $\varepsilon = -1$ ,  $s = 1 + (1/\sqrt{14}) = (1 + \sqrt{14})/\sqrt{14}$  og

$$v = \frac{\sqrt{14}}{1 + \sqrt{14}} \begin{bmatrix} 1 + 1/\sqrt{14} \\ 2/\sqrt{14} \\ 3/\sqrt{14} \end{bmatrix} = \frac{1}{1 + \sqrt{14}} \begin{bmatrix} \sqrt{14} + 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2/(1 + \sqrt{14}) \\ 3/(1 + \sqrt{14}) \end{bmatrix}.$$

Vi kan lave dette beregning i python og bekræfter effekten af den frembragte Householdertransformation:

```
>>> import numpy as np
>>> x = np.array([1.0, 2.0, 3.0])[:, np.newaxis]
>>> u = x / np.linalg.norm(x)
>>> u[0,0]
0.2672612419124244
>>> eps = -1 if u[0,0] >=0 else 1
>>> eps
-1
>>> s = 1 + np.abs(u[0,0])
>>> s
1.2672612419124243
>>> v = (-eps/s) * u
>>> v[0,0] = 1
>>> v
array([[1.          ],
       [0.42179344],
       [0.63269017]])
>>> Hx = x - s * v @ (v.T @ x)
>>> Hx
array([[ -3.74165739e+00],
       [-4.44089210e-16],
       [-4.44089210e-16]])
>>> float(np.linalg.norm(x))
3.7416573867739413
```

△

Bemærk at vi implementer multiplikation med  $H = I_n - svv^T$  på et  $x$ , som

$$Hx = x - sv(v^T x)$$

## 9 ORTOGONALE MATRICER

for at spare på antallet af flops. Desuden behøver vi hverken at konstruere eller gemme matricen  $H$ , men kan nøjes med at gemme skalaren  $s$  og vektoren  $v$ . (Da  $v_0 = 1$  kan dette plads i  $v$  bruges til at gemme  $s$ .) Dette sparer hukommelsen. Det kan være nyttigt at oprette en funktion, som udfører Householdertransformationen givet af  $s$  og  $v$  på en anden vektor  $x$ .

```
import numpy as np

def House(s, v, x):
    return x - s * v @ (v.T @ x)
```

Dette bruges på følgende måde

```
rng = np.random.default_rng()
v = rng.random((10,1))
s = 2 / np.vdot(v,v)

x = rng.random((10,1))
Hx = House(s, v, x)
print(Hx)
```

```
[[-0.06035591]
 [-0.13861355]
 [-1.18021827]
 [ 0.72321294]
 [-0.43307961]
 [-1.05966842]
 [-0.65859451]
 [ 0.29906795]
 [-0.69418729]
 [ 0.44216777]]
```

Vi kan bekræfte at  $H^2x = x$  og at  $H$  afbilder  $v$  til  $-v$  indenfor machine epsilon:

```
print(np.allclose(House(s, v, Hx), x,
                  atol = np.finfo(float).eps))
print(np.allclose(House(s, v, v), -v,
                  atol = np.finfo(float).eps))
```



True  
True

## 9.5 Udvidelse til ortonormal basis

**Sætning 9.17.** Givet en enhedsvektor  $u \in \mathbb{R}^n$ , så findes der en ortonormal basis  $v_0, v_1, \dots, v_{n-1}$  for  $\mathbb{R}^n$  med  $v_0 = u$ .

*Bevis.* Definér  $\varepsilon$  og  $H$  som i proposition 9.14. Da er  $He_0 = \varepsilon u$  og  $H$  er en ortogonal matrix. Matricen  $A = \varepsilon H$  er også ortogonal, da  $\varepsilon^2 = 1$ , og har  $u$  som 0te søjle. Vores ortonormal basis fås nu som søjlerne  $v_0, \dots, v_{n-1}$  af  $A$ .  $\square$

**Korollar 9.18.** For  $u_0, u_1, \dots, u_{k-1}$  ortonormal i  $\mathbb{R}^n$  kan den udvides til en ortonormal basis  $u_0, \dots, u_{k-1}, \dots, u_{n-1}$  for  $\mathbb{R}^n$ .

*Bevis.* Da ortonormale baser er det sammen som søjlerne af en ortogonal matrix, se proposition 9.10, er det nok at finde en ortogonal matrix  $A$  med søjler 0 til  $k-1$  givet ved  $u_0$  til  $u_{k-1}$ .

Brug proposition 9.14 til at finde  $\varepsilon$  og en Householdermatrix  $H$  med  $He_0 = \varepsilon u_0$ . Sæt  $B = \varepsilon H$ , som er ortogonal og har  $u_0$  som 0te søjle. Vi har  $u_0 = Be_0$  og  $e_0 = Bu_0$ . For  $i > 0$  har vi også at  $\langle Bu_i, e_0 \rangle = \langle Bu_i, Bu_0 \rangle = \langle u_i, u_0 \rangle = 0$ . Dette siger at den 0te indgang i  $Bu_i$  er 0, så

$$Bu_i = \begin{bmatrix} 0 \\ w_i \end{bmatrix}, \quad i = 1, \dots, k-1$$

for vektorer  $w_1, \dots, w_{k-1} \in \mathbb{R}^{n-1}$ . Da  $\langle w_i, w_j \rangle = \langle Bu_i, Bu_j \rangle = \langle u_i, u_j \rangle$ , er  $w_1, \dots, w_{k-1}$  ortonormal i  $\mathbb{R}^{n-1}$ . Per induktion, findes der en ortogonal matrix  $C \in \mathbb{R}^{(n-1) \times (n-1)}$  med  $C = [w_1 \mid \dots \mid w_{k-1} \mid \dots]$ . Dannes matricen

$$D = \left[ \begin{array}{c|c} 1 & 0 \\ \hline 0 & C \end{array} \right] \in \mathbb{R}^{n \times n}$$

er  $D$  ortogonal, og

$$\begin{aligned} D &= [e_0 \mid Bu_1 \mid \dots \mid Bu_{k-1} \mid \dots] \\ &= [Bu_0 \mid Bu_1 \mid \dots \mid Bu_{k-1} \mid \dots] \\ &= B[u_0 \mid u_1 \mid \dots \mid u_{k-1} \mid \dots]. \end{aligned}$$

## 9 ORTOGONALE MATRICER

Matricen  $A = B^T D$  er ortogonal og vi har

$$[u_0 \mid u_1 \mid \dots \mid u_{k-1} \mid \dots] = B^T D = A,$$

som ønsket.

□

## Kapitel 10

# Singulærværdidekomponering

En vektor  $v \in \mathbb{R}^2$  i planen beskrives fint af længden  $\|v\|_2$  og vinklen  $\theta$  fra  $x$ -aksen (så længe  $v \neq 0$ ). Som vi har set før kan denne vinkeloplysning erstattes af enhedsvektoren  $\begin{bmatrix} c \\ s \end{bmatrix}$ ,  $c^2 + s^2 = 1$ , med  $c = \cos(\theta)$ ,  $s = \sin(\theta)$ . Husk at  $\begin{bmatrix} c \\ s \end{bmatrix} = v/\|v\|_2$ .

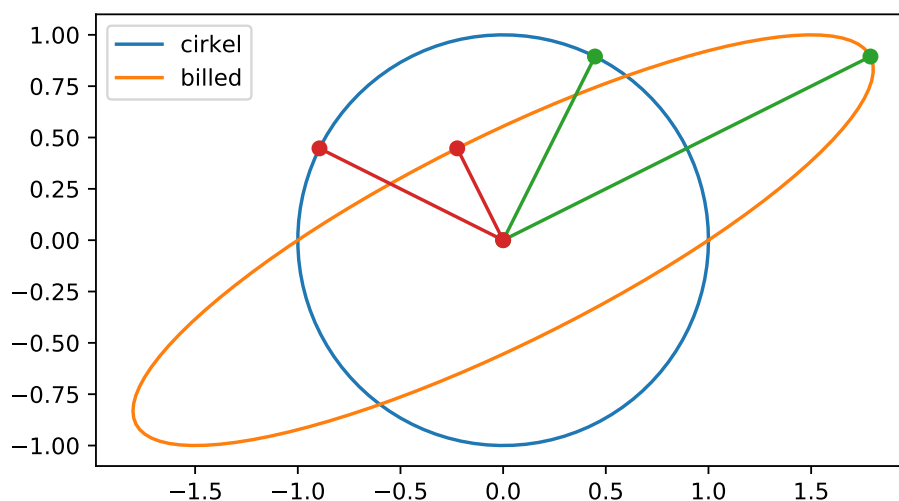
I højere dimensioner,  $v \in \mathbb{R}^n$ ,  $v \neq 0$ , beskrives tilsvarende ved dens længde  $\sigma = \|v\|_2$  og enhedsvektoren  $u = v/\|v\|_2$ . Betragtes  $v$  som en matrix  $v \in \mathbb{R}^{n \times 1}$ , har vi

$$v = \sigma u = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{bmatrix} \begin{bmatrix} \sigma \\ 1 \end{bmatrix}$$

hvor  $\begin{bmatrix} \sigma \\ 1 \end{bmatrix}$  og  $\begin{bmatrix} 1 \end{bmatrix}$  er  $(1 \times 1)$ -matricer. Dette er eksempel på en (tynd) singulærværdidekomponering (SVD) af en matrix. Tallet  $\sigma$  er en singulærværdi; vektorerne  $u \in \mathbb{R}^n$  og  $\begin{bmatrix} 1 \end{bmatrix} \in \mathbb{R}^1$  venstre- og højresingulærvektorer.

En tilsvarende dekomponering findes for vilkårlige matricer  $A \in \mathbb{R}^{m \times n}$ , og giver os god oplysning om deres struktur. Desuden findes der effektive og præcise metoder til at beregne singulærværdidekomponering på, så det er blevet til et vigtigt numerisk værktøj.

Singulærværdierne gemmer på oplysning om  $A$  der er relevant for at vurdere pålideligheden af f.eks. løsninger af lineære ligningssystemer  $Ax = b$ , og dekomponeringen kan bruges til approksimering af  $A$  i komprimeret form. Brug af singulærværdier og singulærvektorer er også meget relevant for analyse af datamængder: et kraftigt ofte brugt dataværktøj er principalkomponent



Figur 10.1: Billedet af en cirkel efter matrixmultiplikation med  $A \in \mathbb{R}^{2 \times 2}$ .

analyse; singulærværdidekomponering udføre dette analyse, og mere, på en numerisk mere pålidelig måde.

## 10.1 Singulærværdier i dimension 2

Lad os begynde med at betragte en  $(2 \times 2)$ -matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Vi kan tegne billedet af enhedscirklen  $(\cos(\theta), \sin(\theta))$  under transformationen  $x \mapsto Ax$ . Et typisk resultat vises i figur 10.1.

Vi ser at dette billede af cirklen er altid ellipseformet, uanset hvilke matrix  $A$  vi bruger. (Nogle gange vil denne ellipse være mast sammen til et linjestykke eller et enkelt punkt).

En ellipse har to akser: en stor akse og en lille akse. Halvdelen af længden af disse to akser er netop det vi kalder singulærværdierne  $\sigma_0$  og  $\sigma_1$  for  $A$ . Det to akser står vinkelret på hinanden og peger i retninger  $u_0$  og  $u_1$ , hvor vi tager  $u_i$  til at være enhedsvektorer.

## 10.1 SINGULÆRVÆRDIER I DIMENSION 2

Det er et overraskende faktum at der findes enhedsvektorer  $v_0$  og  $v_1$  i  $\mathbb{R}^2$ , som står vinkelret på hinanden, således at

$$Av_0 = \sigma_0 u_0, \quad Av_1 = \sigma_1 u_1. \quad (10.1)$$

Dette er en uventet da en vilkårlig matrix  $A$  bevarer ikke vinkler mellem tilfældige vektorer.

Lad os samle  $u_0, u_1$  og  $v_0, v_1$  til  $(2 \times 2)$ -matricer

$$U = [u_0 \mid u_1] \quad \text{og} \quad V = [v_0 \mid v_1].$$

Så er  $U$  og  $V$  ortogonale matricer, da deres søjler er enhedsvektorer, der står vinkelret på hinanden. Sættes

$$u_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \quad u_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}.$$

kan vi omskrive (10.1) til

$$\begin{aligned} AV &= [Av_0 \mid Av_1] = [\sigma_0 u_0 \mid \sigma_1 u_1] = \begin{bmatrix} x_0 \sigma_0 & x_1 \sigma_1 \\ y_0 \sigma_0 & y_1 \sigma_1 \end{bmatrix} \\ &= \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix} \begin{bmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{bmatrix} = U\Sigma, \end{aligned} \quad (10.2)$$

hvor

$$\Sigma = \begin{bmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{bmatrix}.$$

Da  $V$  er en ortogonal matrix, er den invertibel med invers  $V^T$ . Ganges (10.2) på den højre side med  $V^{-1} = V^T$ , fås

$$A = U\Sigma V^T. \quad (10.3)$$

Dette kaldes singulærværdidekomponeringen (SVD) af  $A$ .

Denne SVD kan beregnes i python via `np.linalg.svd`. Funktionen `np.linalg.svd` giver tre objekter: (a) matricen  $U$ , (b) en ndarray med en akse, som indholder singulærværdierne  $\sigma_0, \sigma_1$ , og (c) den transponerede matrix  $V^T$ .

```
>>> import numpy as np

>>> a = np.array([[1.0,  7.0],
```

```

... [1.0, -1.0]])

>>> u, s, vt = np.linalg.svd(a)
>>> s
array([7.12310563, 1.12310563])
>>> u
array([[ -0.99250756,  0.12218326],
       [ 0.12218326,  0.99250756]])
>>> vt
array([[ -0.12218326, -0.99250756],
       [ 0.99250756, -0.12218326]])

```

Disse kan bruges til at rekonstruere  $a$

```

>>> u @ np.diag(s) @ vt
array([[ 1.,  7.],
       [ 1., -1.]])

```

Her danner  $\text{np.diag}(s)$  en diagonal matrix med indgange fra  $s$  sat langs diagonalen.

(10.3) kan også skrives via ydre produkter. Sæt

$$v_0 = \begin{bmatrix} a_0 \\ b_0 \end{bmatrix}, \quad v_1 = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix}.$$

Så er (10.3)

$$\begin{aligned}
 U\Sigma V^T &= [\sigma_0 u_0 \mid \sigma_1 u_1] \begin{bmatrix} v_0^T \\ v_1^T \end{bmatrix} \\
 &= \begin{bmatrix} x_0 \sigma_0 & x_1 \sigma_1 \\ y_0 \sigma_0 & y_1 \sigma_1 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} = \begin{bmatrix} x_0 \sigma_0 a_0 + x_1 \sigma_1 a_1 & x_0 \sigma_0 b_0 + x_1 \sigma_1 b_1 \\ y_0 \sigma_0 a_0 + y_1 \sigma_1 a_1 & y_0 \sigma_0 b_0 + y_1 \sigma_1 b_1 \end{bmatrix} \\
 &= \sigma_0 \begin{bmatrix} x_0 a_0 & x_0 b_0 \\ y_0 a_0 & y_0 b_0 \end{bmatrix} + \sigma_1 \begin{bmatrix} x_1 a_1 & x_1 b_1 \\ y_1 a_1 & y_1 b_1 \end{bmatrix} \\
 &= \sigma_0 \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \end{bmatrix} + \sigma_1 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \begin{bmatrix} a_1 & b_1 \end{bmatrix} \\
 &= \sigma_0 u_0 v_0^T + \sigma_1 u_1 v_1^T.
 \end{aligned}
 \tag{10.4}$$

## 10.2 SVD generelt

**Sætning 10.1.** Enhver matrix  $A \in \mathbb{R}^{m \times n}$  har en [singulærværdidekomponering](#)

$$A = U \Sigma V^T \quad (10.5)$$

med  $U \in \mathbb{R}^{m \times m}$  og  $V \in \mathbb{R}^{n \times n}$  ortogonale matricer, med  $\Sigma \in \mathbb{R}^{m \times n}$  en diagonal matrix

$$\Sigma = \text{diag}(\sigma_0, \dots, \sigma_{k-1}) = \begin{bmatrix} \sigma_0 & 0 & 0 & \dots \\ 0 & \sigma_1 & 0 & \dots \\ \vdots & & \ddots & \vdots \end{bmatrix}, \quad k = \min\{m, n\}, \quad (10.6)$$

og med  $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{k-1} \geq 0$ .

Tallene  $\sigma_i$  kaldes [singulærværdier](#) for  $A$ . Søjlerne af  $U$  er [venstresingulærvektorer](#); søjlerne af  $V$  er [højresingulærvektorer](#).

Beviset gives i afsnit 10.4. Her vil vi fokusere på fortolkningen og anvendelser.

*Eksempel 10.2.* For  $A \in \mathbb{R}^{2 \times 3}$  har dens SVD formen

$$A = [u_0 \mid u_1] \begin{bmatrix} \sigma_0 & 0 & 0 \\ 0 & \sigma_1 & 0 \end{bmatrix} \begin{bmatrix} v_0^T \\ v_1^T \\ v_2^T \end{bmatrix}, \quad u_i \in \mathbb{R}^{2 \times 1}, v_j \in \mathbb{R}^{3 \times 1}.$$

Dette kan skrives via ydre produkter, som vi gjorde for  $(2 \times 2)$ -matricer i (10.4), og giver

$$A = \sigma_0 u_0 v_0^T + \sigma_1 u_1 v_1^T.$$

Bemærk at dette form tager ikke vektoren  $v_2$  i brug. Det svarer til matrixproduktet

$$A = [u_0 \mid u_1] \begin{bmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{bmatrix} \begin{bmatrix} v_0^T \\ v_1^T \end{bmatrix},$$

som kaldes en tynd SVD. △

Generelt er (10.5) det samme som

$$A = \sigma_0 u_0 v_0^T + \sigma_1 u_1 v_1^T + \dots + \sigma_{k-1} u_{k-1} v_{k-1}^T, \quad (10.7)$$

hvor igen  $k = \min\{m, n\}$ . Der er sørget for at  $u_i$  og  $v_j$  er enhedsvektorer, og at faktorerne  $\sigma_i$  er aftagende. Derfor er de led, der bidrager mest til  $A$ , dem der

kommer først i (10.7). Sagt med andre ord, er den største del af oplysningen i  $A$  indeholdt i de første led af (10.7).

I en del applikationer kan man nøjes med at approksimere  $A$  med de første  $r$  led i (10.7),  $r \leq k$ . Vi kalder

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

en *forkortet SVD* med  $r$  led.

## 10.3 Eksempler på SVD

### 10.3.1 Punkter i planen

Betragt 1000 punkter i  $\mathbb{R}^2$ , som til venstre i Figur 10.2.

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

rng = np.random.default_rng()

n = 1000
a = (np.array([[1.0, 5.0], [1.0, -1.0]])
      @ rng.standard_normal((2,n)))

print(np.abs(a).max())
```

19.33054098070929

```
fig, ax = plt.subplots()

ax.set_xlim(-20, 20)
ax.set_ylim(-20, 20)
ax.set_aspect('equal')

ax.plot(*a, 'o', markersize = 2)
```

Hvis vi beregner en fuld SVD, får vi en  $V^T$ , som er meget stor.



```
u, s, vt = np.linalg.svd(a)
print(u.shape, s.shape, vt.shape)
```

```
(2, 2) (2,) (1000, 1000)
```

I stedet for beregner vi en tynd SVD ved hjælp af tilføjesen `full_matrices=False` til `np.linalg.svd`:

```
u, s, vt = np.linalg.svd(a, full_matrices=False)
print(u.shape, s.shape, vt.shape)
```

Vi ser at den første singulærværdi er væsentlig større end den anden

```
print(s)
```

```
[158.26027918  38.0284055 ]
```

Denne singulærværdi har venstresingulærvektor, som er den først søjle af `u`

```
print(u)
```

```
[[-0.98666151  0.16278536]
 [ 0.16278536  0.98666151]]
```

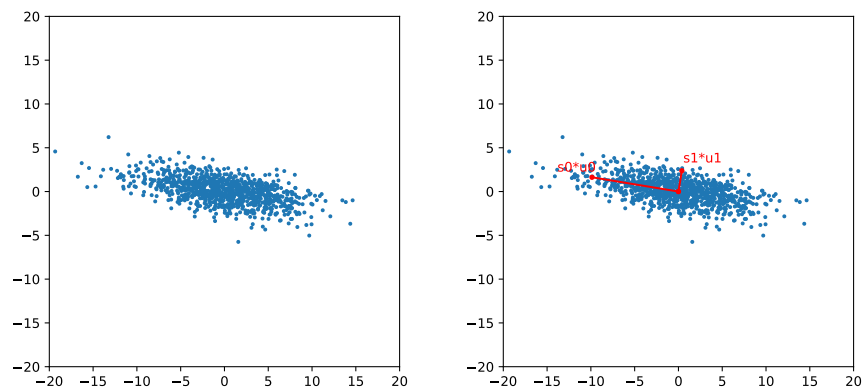
```
origo = np.zeros((2,1))
scale = 2/np.sqrt(n)
tscale = 1.4*scale

fig, ax = plt.subplots()

ax.set_xlim(-20, 20)
ax.set_ylim(-20, 20)
ax.set_aspect('equal')

ax.plot(*a, 'o', markersize = 2)
```

## 10 SINGULÆRVÆRDIDekomponering



Figur 10.2: SVD af punktsamling i planen.

```
ax.plot(*(np.hstack([origo, u[:, [0]]*s[0]*scale])),  
        'red', marker='.')  
ax.text(*u[:, [0]]*s[0]*tscale, 's0*u0', color='red')  
  
ax.plot(*(np.hstack([origo, u[:, [1]]*s[1]*scale])),  
        'red', marker='.')  
ax.text(*u[:, [1]]*s[1]*tscale, 's1*u1', color='red')
```

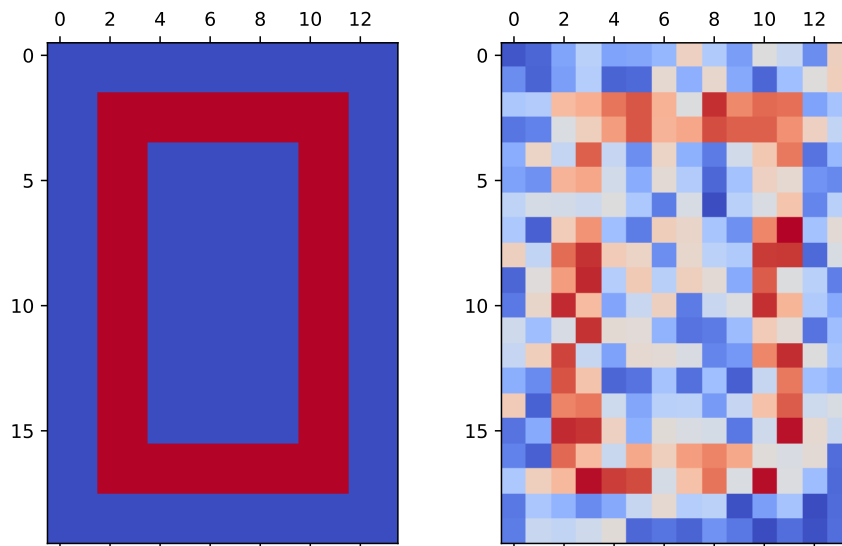
Figur 10.2 demonstrerer at de venstresingulærvektorer giver retningerne hvor variationen af punkterne er hhv. størst og mindst.

### 10.3.2 Billede med støj

For det næste eksempel, lad os danne en et simpelt billede af cifret 0:

```
import matplotlib.pyplot as plt  
import numpy as np  
  
a = np.zeros((20,14))  
a[2, 2:12] = 1  
a[3, 2:12] = 1  
a[16, 2:12] = 1  
a[17, 2:12] = 1  
a[2:18, 2] = 1
```

### 10.3 EKSEMPLER PÅ SVD



Figur 10.3: Billedet af cifret 0 samt en version med tilføjet støj.

```
a[2:18, 3] = 1
a[2:18, 10] = 1
a[2:18, 11] = 1

fig, ax = plt.subplots()
ax.matshow(a, cmap='coolwarm')
```

Dette giver billedet til venstre i figur 10.3.

Lad os tilføje støj til dette billede. Bemærk at støjniveauet er ret højt i forhold til den oprindelige matrix, se billedet til højre i figur 10.3.

```
rng = np.random.default_rng()
a += 1.5 * rng.random(a.shape)

fig, ax = plt.subplots()
ax.matshow(a, cmap='coolwarm')
```

Vi beregner en tynd SVD og kikker på singularværdierne

```
u, s, vt = np.linalg.svd(a, full_matrices=False)
np.set_printoptions(linewidth = 60)
print(np.round(s,3))
```

```
[19.912  4.377  3.388  2.579  2.235  1.894  1.833  1.689
  1.421  1.358  1.135  0.926  0.738  0.702]
```

Vi ser at de første to til tre værdier er en del større end resten, så vi kan forvente bedste støjreduktion ved at bruge kun disse værdier. Lad os danne en funktion som angiver SVD for  $a$  forkortet til  $r$  led.

```
def svdapprox(u, s, vt, r):
    "Givet svd u, s, vt angiv forkortelse til niveau r."
    return u[:, :r] @ np.diag(s[:r]) @ vt[:r, :]
```

Vi kan nu plotte alle disse forkortede SVD

```
fig, axs = plt.subplots(3, 4)
for r, (i,j) in enumerate(np.ndindex(axs.shape)):
    axs[i,j].matshow(svdapprox(u, s, vt, r), cmap='coolwarm')
plt.tight_layout() # justerer placering af hver subplot
```

Som kan ses i figur 10.4 giver den anden og tredje forkortede SVD gode gengivelse af det oprindelige billede

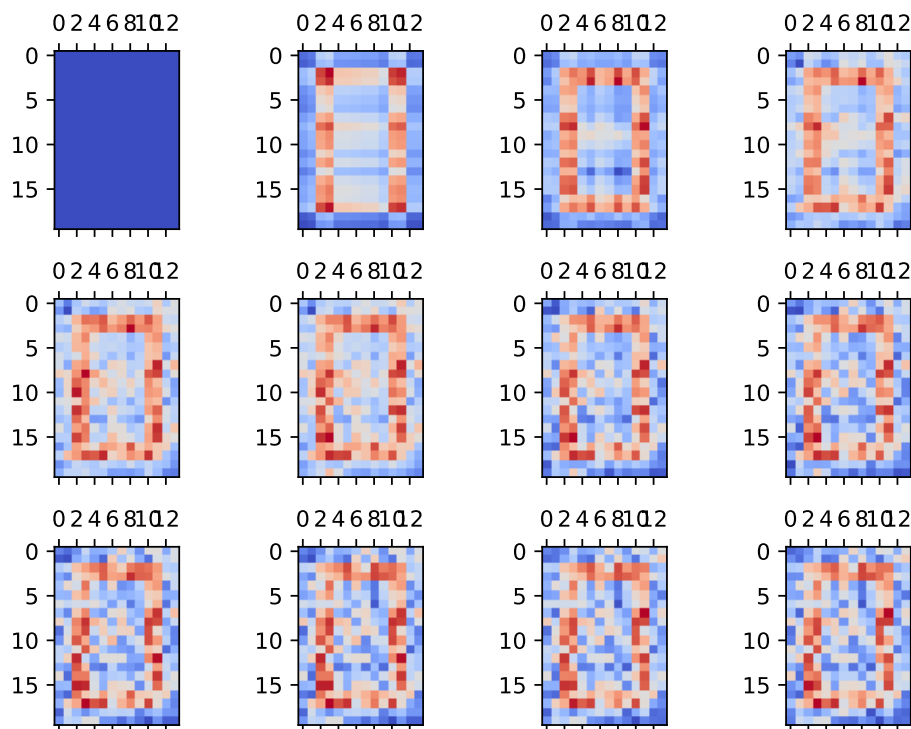
### 10.3.3 Komprimering af et billede

Lad os arbejde med et billede taget med mobiltelefonen, se figur 10.5, af domkirken i Uppsala. Billedfiler fra mobiltelefon er typisk i .jpg format, men man kan også arbejde med andre formatter, som .png.

Når vi har en .jpg kan billedfilen læses ind via `plt.imread` og vises via `imshow`:

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

### 10.3 EKSEMPLER PÅ SVD



Figur 10.4: SVD af billede med støj.

```
a = plt.imread('upsala-img.jpg')  
  
fig, ax = plt.subplots()  
ax.imshow(a)
```

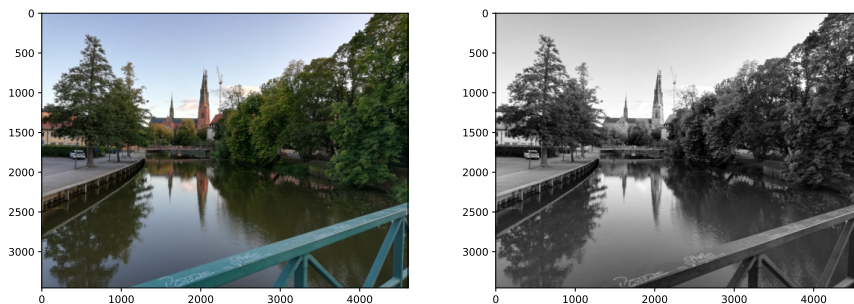
Dette giver en ndarray med shape

```
print(a.shape)
```

(3456, 4608, 3)

Den sidste indeks svarer til tal for rød, grøn og blå farver, i denne rækkefølge. (For en fil i .png format vil der også være et sidste tal for gennemsigtigheden.) Billedet her har  $3456 \times 4608 = 15925248$  pixels, dvs.  $3456 \times 4608 / 2^{20} = 15.1875$  megapixels.

## 10 SINGULÆRVÆRDIDekomponering



Figur 10.5: Uppsala domkirke.

Typisk får vi nok med at kun kikke på den ene farve. Her vælger vi rød og får en almindelig matrix  $b$

```
b = a[:, :, 0]
print(b.shape)
```

(3456, 4608)

som kan plottes i sort/hvid.

```
fig, ax = plt.subplots()
ax.imshow(b, cmap='Greys_r')
```

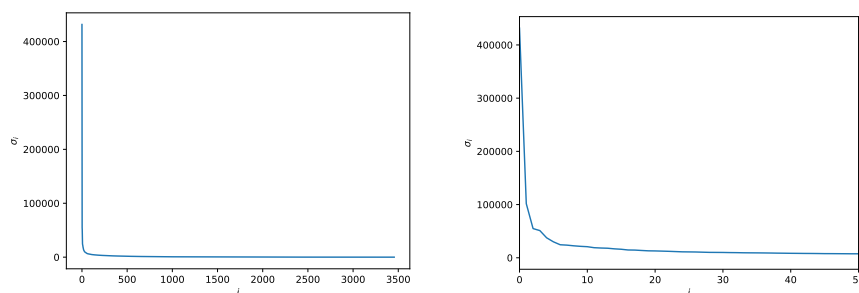
Lad os nu beregne den tynde SVD af  $b$ . Dette tager lidt tid, men mindre end 1 min på min maskine. Vi kan derefter plotte singulærværdierne, se figur 10.6.

```
u, s, vt = np.linalg.svd(b, full_matrices=False)

fig, ax = plt.subplots()
ax.set_xlabel(r'$i$')
ax.set_ylabel(r'$\sigma_{i}$')
ax.plot(s)
```

Vi ser at mange singulærværdier er meget tæt på 0. Vi kikker lidt nærmere på en plot af de første værdier.

### 10.3 EKSEMPLER PÅ SVD



Figur 10.6: Singulærværdier for billedet af domkirken.

```
fig, ax = plt.subplots()
ax.set_xlim(0, 50)
ax.set_xlabel(r'$i$')
ax.set_ylabel(r'$\sigma_{i}$')
ax.plot(s)
```

Til orientering kan vi se på nogle konkrete singulærværdier

```
print(np.round(s[[0, 2, 10, 50, 100]], 3))
```

```
[431654.686  54864.817  20826.709   7494.8    5049.833]
```

Vi kan forkorte SVD ved forskellige niveauer, og to af disse vises i figur 10.7.

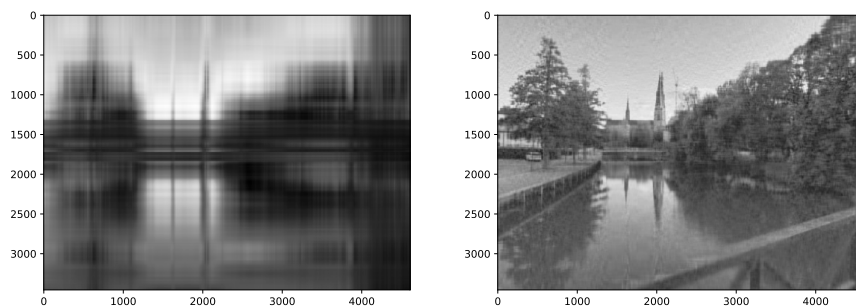
```
def svdapprox(u, s, vt, r):
    "Givet svd u, s, vt angiv trunkering til niveau r."
    return u[:, :r] @ np.diag(s[:r]) @ vt[:r, :]
```

```
fig, ax = plt.subplots()
ax.imshow(svdapprox(u, s, vt, 3), cmap='Greys_r')
```

```
fig, ax = plt.subplots()
ax.imshow(svdapprox(u, s, vt, 50), cmap='Greys_r')
```

En forkortning med kun 50 singulærværdier giver en god gengivelse af originalen. Dette er også en komprimering.

## 10 SINGULÆRVÆRDIDekomponering



Figur 10.7: SVD komprimeringer til hhv. 3 og 50 led.

```
print('Bytes i a: ', a.nbytes)
print('Bytes i b: ', b.nbytes)
komprimeret_size = (u[:, :50].nbytes + s[:50].nbytes +
                    vt[:50, :].nbytes)
print('Bytes for det komprimerede billed: ',
      komprimeret_size)
print('Relativ størrelse af det komprimerede til b')
print('i procent: ', komprimeret_size / b.nbytes * 100)
```

```
Bytes i a: 47775744
Bytes i b: 15925248
Bytes for det komprimerede billed: 3226000
Relativ størrelse af det komprimerede til b
i procent: 20.257141364454732
```

Data for den forkortede SVD med 50 led fylder kun 2,5 % af størrelsen af det oprindelige billede

### 10.4 Eksistens af SVD

Bevis for sætning 10.1. Idet  $V$  er ortogonal er (10.5) det samme som

$$AV = U\Sigma.$$



Hvis  $V$  har søjler  $v_0, \dots, v_{n-1}$ ,  $U$  har søjler  $u_0, \dots, u_{m-1}$  og  $\Sigma$  er som i (10.6), er denne ligning det samme som

$$[Av_0 \mid Av_1 \mid \dots] = [\sigma_0 u_0 \mid \sigma_1 u_1 \mid \dots]. \quad (10.8)$$

Så vores opgave er at finde ortonormale samlinger  $v_0, \dots, v_{n-1}$  og  $u_0, \dots, u_{m-1}$  samt skalarer  $\sigma_0, \dots, \sigma_{k-1}$  således at (10.8) gælder.

Betragt funktionen

$$f_0(v) = \|Av\|_2^2, \quad \text{for } \|v\|_2 = 1.$$

Dette har en maksimumsværdi  $s_0 = \|Av_0\|_2^2$  der opnås ved en enhedsvektor  $v_0$ . Da  $s_0 \geq 0$ , kan vi skrive  $\sigma_0 = \sqrt{s_0} = \|Av_0\|_2 \geq 0$ . Hvis  $\sigma_0 = 0$ , har vi  $A = 0$ , og kan tage  $\Sigma = 0$ ,  $U = I_m$ ,  $V = I_n$ . Når  $\sigma_0 > 0$  sætter vi  $u_0 = Av_0/\sigma_0 = Av_0/\|Av_0\|_2$ , som er en enhedsvektor. Så har vi

$$Av_0 = \sigma_0 u_0,$$

som er den ønskede ligning for den 0te søjle i (10.8).

Vi vil nu gerne gentage dette argument, men kun med vektorer ortogonale på  $v_0$ . For at gøre dette har vi brug for den følgende observation:

$v_0$  er et maksimumspunkt for  $f_0$  på enhedsvektorer medfører at  $\langle u_0, Av \rangle = 0$  for alle  $v$  ortogonal på  $v_0$ .

Dette kan ses på følgende måde. Det er nok at bekræfte udgagnet for  $v$ , der er en enhedsvektorer ortogonal på  $v_0$ . Så er

$$w(t) = \cos(t)v_0 + \sin(t)v$$

en enhedsvektor for alle  $t \in \mathbb{R}$ . Funktionen  $h(t) = f_0(w(t))$ , opfylder at  $h(0) = f_0(v_0) = s_0$ , så  $h(0) \geq h(t)$  for alle  $t$ , og specielt er  $h'(0) = 0$ . Men

$$\begin{aligned} h(t) &= f_0(w(t)) = \|\cos(t)Av_0 + \sin(t)Av\|_2^2 \\ &= \cos^2(t)\|Av_0\|_2^2 + 2\cos(t)\sin(t)\langle Av_0, Av \rangle + \sin^2(t)\|Av\|_2^2. \end{aligned}$$

Så

$$\begin{aligned} 0 &= h'(0) \\ &= (-2\cos(t)\sin(t)\|Av_0\|_2^2 + 2(\cos^2(t) - \sin^2(t))\langle Av_0, Av \rangle \\ &\quad + 2\sin(t)\cos(t)\|Av\|_2^2) \Big|_{t=0} \\ &= 2\langle Av_0, Av \rangle. \end{aligned}$$

Derfor er  $\langle Av_0, Av \rangle = 0$  for alle  $v$  ortogonal på  $v_0$ . Da  $u_0$  er proportionelt med  $Av_0$ , har vi så vist at

$$Av \perp u_0, \quad \text{for alle } v \perp v_0.$$

Vi kan nu gentage argumenterne ovenfor ved at kikke på maksimumsværdierne af

$$f_1(v) = \|Av\|_2^2, \quad \text{for } v \perp v_0 \text{ med } \|v\|_2 = 1,$$

og derefter

$$f_2(v) = \|Av\|_2^2, \quad \text{for } v \perp v_0, v_1 \text{ med } \|v\|_2 = 1,$$

osv. og stopper når  $f_r$  er identisk 0. Vi får  $v_0, \dots, v_{r-1}, u_0, \dots, u_{r-1}$  og  $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{r-1} > 0$  med

$$Av_j = \sigma_j u_j$$

samt

$\sigma_j^2 = \|Av_j\|_2^2$  er maksimalværdien af  $\|Av\|_2^2$  over enhedsvektorer  $v$  ortogonal på  $v_0, \dots, v_{j-1}$

for  $j = 0, \dots, r-1$ . Nu udvid til ortonormale baser  $v_0, \dots, v_{n-1}, u_0, \dots, u_{m-1}$ , ved hjælp af korollar 9.18, og sæt  $\sigma_r = \dots = \sigma_{k-1} = 0$ , til at få (10.8) og dermed (10.5), som ønsket.  $\square$

# Kapitel 11

## Konditionstal

Vi har set at fejl kan opstå fra forskellige kilder: fejl i den oprindelige data, fejl ved repræsentation i computeren, fejl i beregningsoperationer. Konditionstal giver et praj om om hvor meget fejl sendes videre igennem beregninger og om hvor følsomt problemet er ovenfor fejl.

### 11.1 Konditionstal for differentiable funktioner

Betragt funktionen der beregner  $f(x) = x^2$ . Hvis  $x$  ændres til  $x + h$ , så ændres  $f(x)$  til  $f(x + h) = (x + h)^2 = x^2 + 2xh + h^2$ . Ændringen i  $f$  er så  $f(x + h) - f(x) = 2xh + h^2 = h(2x + h)$ . Vi er interesseret i hvordan denne differens opfører sig for  $h$  tæt på 0. Man definerer det *absolutte konditionstal* til at være

$$\hat{\kappa}(x) = \lim_{h \rightarrow 0} \frac{|f(x + h) - f(x)|}{|h|} = |f'(x)|.$$

Så for vores funktion  $f(x) = x^2$  er  $\hat{\kappa}(x) = |2x| = 2|x|$ . Dette siger at en lille ændring i  $x$  fører til en ændring i  $f(x)$ , som er  $2|x|$  større.

Vi har set tidligere at især i forbindelse med *float* er det relevant at beregne relative fejl. Tilsvarende kan vi definere det relative konditionstal, eller blot *konditionstallet*  $\kappa(x)$  til at være forholdet mellem de relative ændringer af  $f(x)$  og  $x$ . Dette giver følgende definition for konditionstallet:

$$\begin{aligned} \kappa(x) &= \lim_{h \rightarrow 0} \left( \frac{|f(x + h) - f(x)|}{|f(x)|} \bigg/ \frac{|h|}{|x|} \right) = \lim_{h \rightarrow 0} \left( \frac{|f(x + h) - f(x)|}{|h|} \frac{|x|}{|f(x)|} \right) \\ &= \frac{|x||f'(x)|}{|f(x)|}, \end{aligned}$$

## 11 KONDITIONSTAL

forudsat at  $f(x) \neq 0 \neq x$ .

For funktionen  $f(x) = x^2$  har vi  $\kappa(x) = |x| |2x| / |x^2| = 2$ . Dvs. en lille relative fejl i  $x$  afspejles i en relativ fejl, der er dobbelt stort, i  $x^2 = f(x)$ .

*Eksempel 11.1.*  $f(x) = \sqrt{x}$ ,  $x > 0$ , har konditionstal

$$\kappa(x) = |x| \left| \frac{d}{dx} \sqrt{x} \right| \bigg/ |\sqrt{x}| = |x| \left| \frac{1}{2\sqrt{x}} \right| \bigg/ |\sqrt{x}| = \left| \frac{\sqrt{x}}{2} \right| \bigg/ |\sqrt{x}| = \frac{1}{2}.$$

△

*Eksempel 11.2.*  $f(x) = e^x$  har konditionstal

$$\kappa(x) = |x| \left| \frac{d}{dx} e^x \right| \bigg/ |e^x| = |x| e^x / e^x = |x|.$$

Så beregning af  $e^x$  for  $x$  stort kan forventes at have en stor relativ fejl. △

Konditionstal af størrelsesorden 1 eller 10 er til at tåle; vi mister få betydende cifre i beregning af svaret. Omvendt betyder et konditionstal af størrelsesorden  $10^6$ , at en relativ fejl i  $x$  forstørres voldsomt, og dette er ofte uacceptabel.

Når vi arbejder med vektorer, skal vi justere lidt på de overnævnte definitioner. Først skal vi erstatte numerisk værdi med normen af vektoren. For  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  erstattes den afledede med gradienten

$$(\nabla f)_x = \left( \frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_{n-1}} \right).$$

For en differentiabel  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ , defineres *konditionstallet* i  $x$  til at være

$$\kappa(x) = \frac{\|x\|_2 \|(\nabla f)_x\|_2}{|f(x)|}.$$

*Eksempel 11.3.* Betragt funktionen  $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ , som beregner differensen,

$$f(x_0, x_1) = x_1 - x_0$$

af tallene  $x_0$  og  $x_1$ . Gradienten af  $f$  er

$$(\nabla f)_x = \left( \frac{\partial(x_1 - x_0)}{\partial x_0}, \frac{\partial(x_1 - x_0)}{\partial x_1} \right) = (-1, 1).$$

## 11.1 KONDITIONSTAL FOR DIFFERENTIABLE FUNKTIONER

Konditionstallet for  $f$  beregnes til at være

$$\kappa(x) = \frac{\|x\|_2 \|(-1, 1)\|_2}{|x_1 - x_0|} = \sqrt{2} \frac{\sqrt{x_0^2 + x_1^2}}{|x_1 - x_0|}. \quad (11.1)$$

Vi ser at konditionstallet er stort for  $x_0$  tæt på  $x_1$ . Dette understøtter det vi har tidligere set (f.eks. eksempel 1.4) at differenceoperationen kan føre til stor fejl når tallene er tæt på hinanden.

Lad os kikke lidt nærmere på dette eksempel, for at understøtte definitionen på  $\kappa(x)$ .

Hvis  $(x_0, x_1)$  ændres med en vektor  $(h_0, h_1)$ , er ændringen i  $f$ :

$$\begin{aligned} f(x_0 + h_0, x_1 + h_1) - f(x_0, x_1) &= ((x_1 + h_1) - (x_0 + h_0)) - (x_1 - x_0) \\ &= h_1 - h_0. \end{aligned}$$

Vi kan skrive det sidste som et indre produkt

$$f(x_0 + h_0, x_1 + h_1) - f(x_0, x_1) = h_1 - h_0 = \left\langle \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} h_0 \\ h_1 \end{bmatrix} \right\rangle.$$

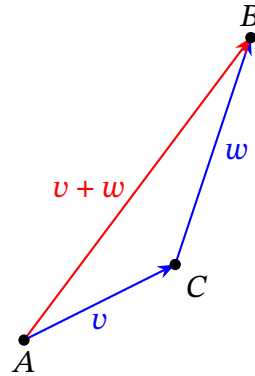
Bemærk at vektoren  $(-1, 1)$  er  $(\nabla f)_x$ . Nu giver Cauchy-Schwarz uligheden at den absolutte fejl i  $f$  ved  $x$  er

$$\begin{aligned} |f(x_0 + h_0, x_1 + h_1) - f(x_0, x_1)| \\ = \left| \left\langle \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} h_0 \\ h_1 \end{bmatrix} \right\rangle \right| &\leq \left\| \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_2 \left\| \begin{bmatrix} h_0 \\ h_1 \end{bmatrix} \right\|_2 = \sqrt{2} \|h\|_2. \end{aligned}$$

Bemærk at Cauchy-Schwarz uligheden sikrer også at der er lighed for visse  $h$ : nemlig de  $h$ , som er parallel med  $(-1, 1)$ . Så den største mulige fejl er givet ved den højre side  $\sqrt{2}\|h\|_2$ . Beregner vi forholdet mellem den relative fejl i  $f$  og den relative ændring i  $x$ , får vi

$$\frac{|f(x+h) - f(x)|}{|f(x)|} \bigg/ \frac{\|h\|_2}{\|x\|_2} = \frac{|f(x+h) - f(x)|}{\|h\|_2} \frac{\|x\|_2}{|f(x)|} \leq \sqrt{2} \frac{\sqrt{x_0^2 + x_1^2}}{|x_1 - x_0|},$$

og denne grænse opnås for visse  $h$ . Denne maksimale værdi er netop konditionstallet  $\kappa(x)$  i (11.1).  $\triangle$



Figur 11.1: Trekantsuligheden.

## 11.2 Matrixnorm

Vi ønsker at udforske tilsvarende ændringer i relation til lineære ligningssystemer  $Ax = b$ . Det kræver dog at vi har et norm-begreb for matricen  $A$ .

**Definition 11.4.** En *norm* på  $\mathbb{R}^n$  er en funktion  $\|\cdot\|: \mathbb{R}^n \rightarrow \mathbb{R}$  med

- (N1)  $\|v\| \geq 0$  for alle  $v \in \mathbb{R}^n$ , og  $\|v\| = 0$  kun for  $v = 0$ ,
- (N2)  $\|sv\| = |s|\|v\|$  for alle  $s \in \mathbb{R}$ ,
- (N3)  $\|v + w\| \leq \|v\| + \|w\|$ .

Vi har set at 2-normen  $\|\cdot\|_2 = \sqrt{\langle \cdot, \cdot \rangle}$  opfylder (N1) og (N2). Betingelse (N3) kaldes *trekantsuligheden*, da den siger at i figur 11.1 er vejen direkte fra  $A$  til  $B$  kortere end turen langs de andre to sider af trekanten via punktet  $C$ .

Trekantsuligheden gælder for  $\|\cdot\|_2$ , da

$$\begin{aligned} \|v + w\|_2^2 &= \langle v + w, v + w \rangle = \langle v, v \rangle + \langle v, w \rangle + \langle w, v \rangle + \langle w, w \rangle \\ &= \|v\|_2^2 + 2\langle v, w \rangle + \|w\|_2^2 \\ &\leq \|v\|_2^2 + 2\|v\|_2\|w\|_2 + \|w\|_2^2 \\ &= (\|v\|_2 + \|w\|_2)^2, \end{aligned}$$

hvor vi har brugt Cauchy-Schwarz uligheden (8.4).

Der findes andre normer på  $\mathbb{R}^n$ . To første eksempler er *1-normen*

$$\|v\|_1 = \|(v_0, v_1, \dots, v_{n-1})\|_1 = |v_0| + |v_1| + \dots + |v_{n-1}|$$

og maksimumsnormen eller  $\infty$ -normen

$$\|v\|_{\infty} = \|(v_0, v_1, \dots, v_{n-1})\|_{\infty} = \max\{|v_0|, |v_1|, \dots, |v_{n-1}|\}.$$

Numerisk kan det være hurtige at beregne  $\|v\|_1$  eller  $\|v\|_{\infty}$  end  $\|v\|_2$ . Desuden indeholder disse normer sammenlignelig oplysning om  $v$ , da for  $v \in \mathbb{R}^n$  gælder

$$\|v\|_{\infty} \leq \|v\|_2 \leq \|v\|_1 \leq \sqrt{n}\|v\|_2 \leq n\|v\|_{\infty}.$$

Som navnene antyder er de overnævnte normer en del af en større familie. For hvert tal  $p$  med  $1 \leq p \leq \infty$  er  $p$ -normen givet ved

$$\|v\|_p = \|(v_0, v_1, \dots, v_{n-1})\|_p = \left( \sum_{i=0}^{n-1} |v_i|^p \right)^{1/p}$$

For matricer  $A \in \mathbb{R}^{m \times n}$  indfører vi en matrix eller operator  $2$ -norm på følgende måde

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2. \quad (11.2)$$

Dvs.  $\|A\|_2$  måler den største skalering af længden af  $x$  når vi ganger med  $A$ . At (11.2) definerer en norm, følger direkte fra egenskaberne af  $\|\cdot\|_2$  for vektorer. Ligning (11.2) er det samme som

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}, \quad (11.3)$$

som ofte gives som definition. Dette ækvivalens gælder, da

$$\frac{\|Ax\|_2}{\|x\|_2} = \frac{1}{\|x\|_2} \|Ax\|_2 = \left\| \frac{1}{\|x\|_2} Ax \right\|_2 = \left\| A \left( \frac{1}{\|x\|_2} x \right) \right\|_2 = \left\| A \left( \frac{x}{\|x\|_2} \right) \right\|_2$$

og  $x/\|x\|_2$  er enhedsvektor. Omskrivningen (11.3) giver den nyttige

$$\|Ax\|_2 \leq \|A\|_2 \|x\|_2, \quad \text{for alle } x \in \mathbb{R}^n. \quad (11.4)$$

Man kan definere en tilsvarende  $p$ -norm for  $A$  ved  $\|A\|_p = \max_{\|x\|_p=1} \|Ax\|_p$ , men vi vil holdes os mest til  $\|A\|_2$ .

**Lemma 11.5.** For  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times p}$  gælder

$$\|AB\|_2 \leq \|A\|_2 \|B\|_2.$$

## 11 KONDITIONSTAL

*Bevis.* Det følger fra (11.4) at

$$\|ABx\|_2 = \|A(Bx)\|_2 \leq \|A\|_2 \|Bx\|_2 \leq \|A\|_2 \|B\|_2 \|x\|_2.$$

Så for  $\|x\|_2 = 1$ , gælder

$$\|ABx\|_2 \leq \|A\|_2 \|B\|_2.$$

Det følger at maksimumsværdien for  $\|ABx\|_2$  over  $\|x\|_2 = 1$  er højst  $\|A\|_2 \|B\|_2$ , men det er netop udsagnet  $\|AB\|_2 \leq \|A\|_2 \|B\|_2$ , der ønskes.  $\square$

Dette 2-norm for  $A$  er faktisk noget vi har set før:

**Lemma 11.6.** Hvis  $A \in \mathbb{R}^{m \times n}$  har singulærværdidekomponering  $A = U\Sigma V^T$ , med singulærværdier  $\sigma_0 \geq \dots \geq \sigma_{k-1} \geq 0$ ,  $k = \min\{m, n\}$ , så er 2-normen af  $A$  givet ved

$$\|A\|_2 = \sigma_0,$$

hvor  $\sigma_0$  er den største singulærværdi.

*Bevis.* Matricen  $U \in \mathbb{R}^{m \times m}$  er ortogonal, så  $\|Uy\|_2 = \|y\|_2$  for alle  $y \in \mathbb{R}^m$ , se proposition 9.5. Vi har så

$$\|Ax\|_2 = \|U\Sigma V^T x\|_2 = \|\Sigma V^T x\|_2. \quad (11.5)$$

For at bestemme  $\|A\|_2$  skal vi så

$$\text{maksimere (11.5) over alle } x \text{ med } \|x\|_2 = 1. \quad (11.6)$$

Men  $V^T$  er også ortogonal, så bevarer  $\|\cdot\|_2$  og er invertibel. Ved at skrive  $w = V^T x$ , får vi alle  $w$  med  $\|w\|_2 = 1$ , fra de forskellige  $x$  med  $\|x\|_2 = 1$ . Det giver at (11.6) er det sammen som at

$$\text{maksimere } \|\Sigma w\|_2 \text{ over alle } w \text{ med } \|w\|_2 = 1. \quad (11.7)$$

Bemærk nu at for  $w = (w_0, \dots, w_{n-1})$  er

$$\begin{aligned} \|\Sigma w\|_2 &= \sqrt{\sigma_0^2 w_0^2 + \dots + \sigma_{k-1}^2 w_{k-1}^2} \\ &\leq \sqrt{\sigma_0^2 (w_0^2 + \dots + w_{n-1}^2)} = \sigma_0 \|w\|_2 \end{aligned}$$

og vi har lighed for  $w = e_0$ . Det følger at

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \max_{\|w\|_2=1} \|\Sigma w\|_2 = \sigma_0,$$

som ønsket.  $\square$



## 11.3 Konditionstal og lineære ligningssystemer

Lad os nu betragte et lineært ligningssystem med samme antal ligninger, som variabler:

$$Ax = b$$

for  $x, b \in \mathbb{R}^n$ ,  $A \in \mathbb{R}^{n \times n}$ . Lad os desuden antage at den kvadratiske matrix  $A$  er invertibel, så for hvert  $b$  er der en entydig løsning  $x = A^{-1}b$ . Der er flere forskellige måder at beregne et konditionstal for systemet, med det viser sig at alle kontrolleres af [konditionstallet for  \$A\$](#)

$$\kappa(A) = \|A\|_2 \|A^{-1}\|_2.$$

**Proposition 11.7.** *Konditionstallet for en invertibel matrix  $A \in \mathbb{R}^{n \times n}$ , er givet ved*

$$\kappa(A) = \sigma_0 / \sigma_{n-1} \quad (11.8)$$

hvor  $\sigma_0 \geq \dots \geq \sigma_{n-1} > 0$  er singularværdierne for  $A$ .

*Bevis.* Lemma 11.6 viser netop at  $\|A\|_2 = \sigma_0$ . Skriver vi  $A$ 's singularværdidekomponering, som  $A = U\Sigma V^T$ , har vi for  $A$  invertibel at  $A^{-1} = V\Sigma^{-1}U^T$ . Men

$$\Sigma^{-1} = \begin{bmatrix} \sigma_0 & 0 & \dots & 0 \\ 0 & \sigma_1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & \sigma_{n-1} \end{bmatrix}^{-1} = \begin{bmatrix} 1/\sigma_0 & 0 & \dots & 0 \\ 0 & 1/\sigma_1 & & 0 \\ \vdots & & \ddots & \\ 0 & 0 & & 1/\sigma_{n-1} \end{bmatrix}.$$

Det følger at  $A^{-1}$  har singularværdier  $1/\sigma_{n-1} \geq \dots \geq 1/\sigma_0$ , og den singularværdidekomponering af  $A^{-1}$  er

$$A^{-1} = [v_{n-1} \mid \dots \mid v_1 \mid v_0] \begin{bmatrix} 1/\sigma_{n-1} & 0 & \dots & 0 \\ & \ddots & & \vdots \\ 0 & & 1/\sigma_1 & 0 \\ 0 & \dots & 0 & 1/\sigma_1 \end{bmatrix} \begin{bmatrix} u_{n-1}^T \\ \vdots \\ u_1^T \\ u_0^T \end{bmatrix}$$

hvor  $u_i$  og  $v_j$  er søjlerne af  $U$  hhv.  $V$ . Vi har så at  $\|A^{-1}\|_2 = 1/\sigma_{n-1}$ , og resultatet følger.  $\square$

## 11 KONDITIONSTAL

Lad os først betragte funktionen  $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$

$$f(x) = Ax,$$

for en given invertibel matrix  $A$ . Ændrer vi  $x$  til  $x+h$ , ændres  $f(x)$  til  $f(x+h) = A(x+h) = Ax + Ah$ . Den relative ændring i  $f$  i forhold til den for  $x$  har størrelse

$$\begin{aligned} \frac{\|f(x+h) - f(x)\|_2}{\|f(x)\|_2} & \bigg/ \frac{\|h\|_2}{\|x\|_2} = \frac{\|Ah\|_2}{\|Ax\|_2} \bigg/ \frac{\|h\|_2}{\|x\|_2} \\ & = \frac{\|Ah\|_2}{\|h\|_2} \frac{\|x\|_2}{\|Ax\|_2}. \end{aligned}$$

Dens maksimal værdi er så

$$\kappa(x) = \|A\|_2 \frac{\|x\|_2}{\|Ax\|_2},$$

som opnås for  $h$  parallelt med  $v_0$ . Når vi lader  $x$  variere, kan vi skrive  $b = Ax$ , så  $x = A^{-1}b$ , og har

$$\frac{\|x\|_2}{\|Ax\|_2} = \frac{\|A^{-1}b\|_2}{\|b\|_2}$$

som har maksimumsværdi  $\|A^{-1}\|_2$ . Det følger at

$$\max_{x \neq 0} \kappa(x) = \|A\|_2 \|A^{-1}\|_2 = \kappa(A), \quad (11.9)$$

med maksimum opnået for  $x$  parallelt med  $v_{n-1}$ .

*Eksempel 11.8.* Betragt ligningssystemet

$$\begin{aligned} 7x_0 + 3x_1 &= 5, \\ 9x_0 + 4x_1 &= -1. \end{aligned}$$

Den tilhørende matrix er

$$A = \begin{bmatrix} 7 & 3 \\ 9 & 4 \end{bmatrix},$$

hvis konditionstal kan beregnes i python:

```
>>> import numpy as np
>>> a = np.array([[7.0, 3.0],
...               [9.0, 4.0]])
>>> np.linalg.cond(a)
154.99354811853624
```

### 11.3 KONDITIONSTAL OG LINEÆRE LIGNINGSSYSTEMER

Tages  $x$  til at være vektoren  $v_1$  fra SVD for  $A$ , og  $h$  til at være proportionelt med  $v_0$ , ser vi at ændring i  $Ax$  til  $A(x + h)$  er ret stort

```
>>> u, s, vt = np.linalg.svd(a)
>>> s
array([12.44964048,  0.0803236 ])
>>> x = vt[[1], :].T
>>> x
array([[ -0.40157455],
       [ 0.91582634]])
>>> a @ x
array([[ -0.06354287],
       [ 0.04913435]])
>>> h = 0.1 * vt[[0], :].T
>>> h
array([[ -0.09158263],
       [ -0.04015746]])
>>> a @ (x+h)
array([[ -0.82509367],
       [ -0.93573917]])
>>> ((np.linalg.norm(a @ h)/np.linalg.norm(h))
...  * (np.linalg.norm(x)/np.linalg.norm(a @ x)))
154.99354811853738
```

Dette er det størst muligt relativ fejl. Vælges  $x$  og  $h$  tilfældigt er fejlen typisk mindre

```
>>> rng = np.random.default_rng()
>>> x = rng.standard_normal((a.shape[1],1))
>>> a @ x
array([[ 7.80936741],
       [ 9.95032752]])
>>> h = rng.normal(0.0, 0.1, (a.shape[1],1))
>>> a @ (x+h)
array([[ 8.07986711],
       [10.29589109]])
>>> a @ (x+h)
array([[ 8.07986711],
```

## 11 KONDITIONSTAL

```
[10.29589109]])  
>>> ((np.linalg.norm(a @ h)/np.linalg.norm(h))  
... * (np.linalg.norm(x)/np.linalg.norm(a @ x)))  
1.10358421726309
```

Faktisk er fejlen mindst for  $x$  parallelt med  $v_0$  og  $h$  parallelt med  $v_1 = v_{n-1}$ .  $\Delta$

Hvis vi er interesseret i hvordan løsninger  $x$  til  $Ax = b$  ændrer sig når  $b$  varierer, kan vi bruge den samme analyse, men for funktionen  $b \mapsto x = A^{-1}b$ . Dvs.  $A$  udskiftes med  $A^{-1}$  ovenfor. Men så er resultatet det samme udtryk (11.9); men rollerne af  $v_0$  og  $v_{n-1}$  spilles nu af  $u_{n-1}$  og  $u_0$ .

Desuden hvis man holder  $b$  fast, og lad  $A$  varierer, dvs. betragter funktionen  $A \mapsto x = A^{-1}b$ , kan man også vise at ændringerne styres også af (11.9).

For en generel matrix  $A \in \mathbb{R}^{m \times n}$  kan man godt definere konditionstallet til at være  $\kappa(A) = \sigma_0/\sigma_{k-1} \in [0, \infty]$ , jævnfør (11.8). Nogle af de overnævnte resultater kan så overføres til denne situation.

# Kapitel 12

## Generelle vektorrum

### 12.1 Vektorrum

Indtil videre har vi kun betragtet vektorer som elementer i  $\mathbb{R}^n$ . Men i det forrige kapitel, så vi at matricer i  $\mathbb{R}^{m \times n}$  kan også håndteres på en lignende måde. Både  $\mathbb{R}^n$  og  $\mathbb{R}^{m \times n}$  er eksempler på vektorrum, men der er andre objekter der kan med fordel behandles på en tilsvarende måde.

**Definition 12.1.** Et *vektorrum* består af en mængde  $V$ , hvis elementer vi kalder *vektorer*. Der er to operationer defineret. Den første er en sum operation  $u + v \in V$  for alle  $u, v \in V$ , som opfylder

- (V1)  $u + v = v + u$ , for alle  $u, v \in V$ ,
- (V2)  $(u + v) + w = u + (v + w)$ , for alle  $u, v, w \in V$ ,
- (V3) der findes  $0 \in V$  således at  $v + 0 = v$  for alle  $v \in V$ ,
- (V4) for hver  $v \in V$  findes der  $-v \in V$  således at  $v + (-v) = 0$ .

Den anden operation er multiplikation med en skalar  $sv \in V$  defineret for alle skalar  $s$  og alle  $v \in V$ , med følgende egenskaber

- (V5)  $s(u + v) = su + sv$ ,
- (V6)  $(s + t)v = sv + tv$ ,
- (V7)  $(st)v = s(tv)$ ,
- (V8)  $1v = v$ ,

for alle skalarer  $s$  og  $t$ , og alle  $u, v \in V$ .

*Eksempel 12.2.*  $V = \mathbb{R}^n = \mathbb{R}^{n \times 1}$  med vektorsum og skalar multiplikation som vi har brugt tidligere opfylder dette. Den eneste der skal bemærkes er elementet 0

i del (V3) er vektoren

$$0 = 0_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n.$$

△

*Eksempel 12.3.* Rummet af matricer  $V = \mathbb{R}^{m \times n}$  er ligeledes et vektorrum. Denne gang er nulelementet matricen  $0_{m \times n}$ . Bemærk at rummet af matricer har yderligere matrixmultiplikation, som en ekstra operation ud over de to operationer i definition 12.1. △

*Eksempel 12.4.* Lad  $V$  bestå af alle polynomier af grad højst 2. Et typisk element i  $V$  er

$$p(x) = a_0 + a_1x + a_2x^2$$

med  $a_0, a_1, a_2 \in \mathbb{R}$ . Vi kan definere en sum operationer for  $p(x)$  og  $q(x) = b_0 + b_1x + b_2x^2$  ved

$$(p + q)(x) = (a_0 + b_0) + (a_1 + b_1)x + (a_2 + b_2)x^2.$$

Så kan man tjekke at dele (V1) til (V4) er opfyldt med nulpolynomiet givet ved

$$0(x) = 0 + 0x + 0x^2.$$

Med skalarmultiplikation givet ved

$$(sp)(x) = sa_0 + (sa_1)x + (sa_2)x^2,$$

er de resterende punkter i definition 12.1 opfyldt, og vi får et vektorrum. △

*Eksempel 12.5.* Eksempel 12.4 kan udvides til polynomier af grad højst  $n$ , eller til alle polynomier, på tilsvarende måde. △

**Lemma 12.6.** *Lad  $V$  være et vektorrum og lad  $v \in V$ .*

(a) *Så kan nulelementet beregnes som  $0 = 0v$ .*

(b) *Elementet  $-v$  er givet som  $-v = (-1)v$ .*

*Begge elementer er entydigt bestemt.*

*Bevis.* Først er 0 entydig: hvis  $v + u = v$ , så har vi  $0 = v - v = (v + u) - v = u$ , dvs.  $u = 0$ . Nu gælder  $v + 0v = 1v + 0v = (1 + 0)v = 1v = v$ , så  $0v = 0$ .

Elementet  $-v$  er entydig, da  $v + w = 0$  giver  $-v = -v + 0 = -v + (v + w) = (-v + v) + w = 0 + w = w$ , så  $w = -v$ . Vi beregner nu  $v + (-1)v = 1v + (-1)v = (1 - 1)v = 0v = 0$ , så  $(-1)v = -v$ . □

*Eksempel 12.7.* Lad  $V$  bestå af alle differentiable funktioner  $f: [a, b] \rightarrow \mathbb{R}$ . Vi sætter  $f + g$  til at være funktionen givet ved

$$(f + g)(x) = f(x) + g(x), \quad \text{for alle } x \in [a, b],$$

og definerer skalarmultiplikation ved

$$(sf)(x) = s(f(x)), \quad \text{for alle } x \in [a, b].$$

Bemærk at både  $f + g$  og  $sf$  er differentiable funktioner. Vi får et vektorrum, med nulvektor funktionen  $0(x) = 0$ , for alle  $x \in [a, b]$ .  $\Delta$

## 12.2 Andre skalarer

Andre eksempler på vektorrum gives ved at bruge andre typer skalarer. Indtil videre har vores skalarer været reelle tal  $s \in \mathbb{R}$ . Men vektorrum kan dannes med andre koefficienter. Tænkes der på hvordan vi har løst lineære ligningssystemer, er det væsentlig at vi kan tage sum og differens af skalarer og at vi kan gange og dele med dem. Vigtigst for os vil være at vi kan bruge komplekse tal.

De *komplekse tal* betegnes  $\mathbb{C}$ . Et element  $z \in \mathbb{C}$  skrives

$$z = x + iy$$

hvor  $x, y \in \mathbb{R}$  og  $i$  er et symbol, med  $i^2 = -1$ . Vi kalder  $x = \operatorname{Re}(z)$  den *reelle* del af  $z$ , og  $y = \operatorname{Im}(z)$  den *imaginære* del. Beregning med komplekse tal udføres netop som for reelle tal blot med den ekstra relation at  $i^2 = -1$ .

*Eksempel 12.8.* For  $z = 1 + 2i$ ,  $w = -2 + 3i$  har vi

$$z + w = (1 + 2i) + (-2 + 3i) = 1 + (-2) + i(2 + 3) = -1 + 5i$$

og

$$\begin{aligned} zw &= (1 + 2i)(-2 + 3i) = 1 \times (-2) + 1 \times 3i + 2i \times (-2) + (2i) \times (3i) \\ &= -2 + 3i - 4i + 6i^2 = -2 - i - 6 = -8 - i. \end{aligned}$$

$\Delta$

For  $z = x + iy$ ,  $w = s + it$  får vi så

$$z + w = (x + iy) + (s + it) = (x + s) + i(y + t), \quad (12.1)$$

$$zw = (x + iy)(s + it) = (xs - yt) + i(xt + ys). \quad (12.2)$$

## 12 GENERELLE VEKTORRUM

(12.1) er bare vektorsum i  $\mathbb{R}^2$  oversat til denne ny notation. (12.2) kommer fra at gange  $(x + iy)(s + it)$  ud på almindeligvis samt reglen  $i^2 = -1$  som i eksemplet.

Vi kan godt regne med komplekse tal i python (her er python en del bedre end andre programmeringssprog). Vi skal dog gå opmærksom på at python bruger  $j$  for tallet  $i$ , i tråd med god traditioner for elektriske ingeniører:

```
>>> import numpy as np
>>> z = 1.0 + 2.0j
>>> type(z)
<class 'complex'>
>>> np.real(z)
1.0
>>> np.imag(z)
2.0
>>> w = -2.0 + 3.0j
>>> z + w
(-1+5j)
>>> z * w
(-8-1j)
```

Man skal være opmærksom på at der skal stå et tal direkte foran  $j$ , dvs. uden mellemrum og uden  $*$ .

Man kan også dele med komplekse tal, som er forskellig fra  $0 = 0 + 0i$ . Opskriften er

$$\frac{1}{z} = \frac{1}{x + iy} = \frac{x - iy}{x^2 + y^2}. \quad (12.3)$$

Ved direkte udregning kan man tjekke at  $z(1/z) = 1$ :

$$\begin{aligned} z \frac{1}{z} &= (x + iy) \frac{x - iy}{x^2 + y^2} = \frac{1}{x^2 + y^2} (x + iy)(x - iy) \\ &= \frac{1}{x^2 + y^2} (x^2 - ixy + ixy - i^2 y^2) = \frac{x^2 + y^2}{x^2 + y^2} = 1. \end{aligned}$$

*Eksempel 12.9.* For  $z = 1 + 2i$  har vi

$$\frac{1}{z} = \frac{1}{1 + 2i} = \frac{1 - 2i}{1^2 + 2^2} = \frac{1 - 2i}{5} = \frac{1}{5} - \frac{2}{5}i,$$

som python kan bekræfte



```

>>> z = 1.0 + 2.0j
>>> 1/z
(0.2-0.4j)
>>> z * (1/z)
(1+0j)

```

△

Vi kan også arbejde med lineære ligningssystemer hvis koefficienter er komplekse tal på præcis den samme måde med rækkeoperationer, som vi gjorde tidligere.

*Eksempel 12.10.* Systemet

$$\begin{aligned} 2x + 3y - 4z &= 7 - i, \\ 3ix - (4 + i)y + z &= -2, \\ x + y + (2 + i)z &= 3 + i \end{aligned}$$

kan løses via rækkeoperationer. F.eks. i python

```

>>> import numpy as np
>>> a = np.array([[2.0, 3.0, -4.0],
...              [3.0j, -(4.0+1.0j), 1.0],
...              [1.0, 1.0, 2.0+1.0j]])
>>> a.dtype
dtype('complex128')
>>> b = np.array([7.0-1.0j, -2.0, 3.0+1.0j])[:, np.newaxis]
>>> aub = np.hstack([a, b])
>>> aub
array([[ 2.+0.j,  3.+0.j, -4.+0.j,  7.-1.j],
       [ 0.+3.j, -4.-1.j,  1.+0.j, -2.+0.j],
       [ 1.+0.j,  1.+0.j,  2.+1.j,  3.+1.j]])
>>> aub[ [0,2], :] = aub[ [2,0], :]
>>> aub[1, :] += -3.0j * aub[0, :]
>>> aub[2, :] += -2.0 * aub[0, :]
>>> aub
array([[ 1.+0.j,  1.+0.j,  2.+1.j,  3.+1.j],
       [ 0.+0.j, -4.-4.j,  4.-6.j,  1.-9.j],

```

## 12 GENERELLE VEKTORRUM

```
[ 0.+0.j, 1.+0.j, -8.-2.j, 1.-3.j]])
>>> aub[ [1,2], :] = aub[ [2,1], :]
>>> aub[0, :] += - aub[1, :]
>>> aub[2, :] += (4.0 + 4.0j) * aub[1, :]
>>> aub
array([[ 1. +0.j,  0. +0.j, 10. +3.j,  2. +4.j],
       [ 0. +0.j,  1. +0.j, -8. -2.j,  1. -3.j],
       [ 0. +0.j,  0. +0.j, -20.-46.j, 17.-17.j]])
>>> aub[2, :] *= 1/(-20.0-46.0j)
>>> aub[0, :] += -(10.0+3.0j) * aub[2, :]
>>> aub[1, :] += (8.0+2.0j) * aub[2, :]
>>> aub
array([[ 1.          +0.j          ,  0.          +0.j          ,
        0.          +0.j          ,  1.58108108-0.98648649j],
       [ 0.          +0.j          ,  1.          +0.j          ,
        0.          +0.j          ,  1.51351351+0.91891892j],
       [-0.          +0.j          , -0.          +0.j          ,
        1.          +0.j          ,  0.17567568+0.44594595j]])
>>> print('x = ', aub[0, -1])
x = (1.581081081081081-0.9864864864864868j)
>>> print('y = ', aub[1, -1])
y = (1.5135135135135136+0.9189189189189189j)
>>> print('z = ', aub[2, -1])
z = (0.17567567567567569+0.44594594594594594j)
>>> # Tjek af løsning
>>> v = aub[:, [-1]]
>>> a @ v - b
array([[ -8.88178420e-16-8.88178420e-16j],
       [ 1.11022302e-15-5.55111512e-16j],
       [ 0.00000000e+00-4.44089210e-16j]])
>>> np.allclose(a @ v, b, atol = np.finfo(float).eps)
True
```

Δ

Formlen (12.3) indeholder to væsentlig operationer på  $z = x + iy$ . Den første er den *konjugerede*

$$\bar{z} = \overline{x + iy} = x - iy.$$

```
>>> z = 1.0 + 2.0j
>>> np.conj(z)
(1-2j)
```

Den anden den *numeriske værdi*

$$|z| = |x + iy| = \sqrt{x^2 + y^2},$$

som er blot normen af vektoren  $(x, y)$  i  $\mathbb{R}^2$ .

```
>>> import numpy as np
>>> z = 1.0 + 2.0j
>>> np.abs(z)
2.23606797749979
>>> np.sqrt(np.real(z)**2+np.imag(z)**2)
2.23606797749979
```

Vi kan så omskrive formlen (12.3), som

$$\frac{1}{z} = \frac{\bar{z}}{|z|^2}$$

Bemærk at  $|z|^2 = x^2 + y^2 = (x + iy)(x - iy) = z\bar{z}$ . Vi har også

$$|zw| = |z||w|$$

for alle komplekse tal  $z$  og  $w$ . Konjugation opfylder

$$\overline{\bar{z} + \bar{w}} = z + w, \quad \overline{zw} = \bar{z}\bar{w}.$$

## 12.3 Underrum

En nem og væsentlig kilde til vektorrum er visse delmængde af kendte vektorrum.

**Definition 12.11.** Lad  $V$  være et vektorrum. En ikke-tom delmængde  $W \subset V$  er et *underrum* hvis sum og skalarmultiplikation sender element af  $W$  til elementer  $W$ . Dvs.

## 12 GENERELLE VEKTORRUM

- (a)  $W$  er ikke tom,
- (b)  $v, w \in W$  medfører  $v + w \in W$ ,
- (c)  $w \in W$  og  $s$  en skalar medfører  $sw \in W$ .

Bemærk at nulvektoren  $0$  må nødvendigvis ligge i  $W$ , da for et  $w \in W$  har vi  $0 = 0w \in W$ . Det kan godt tjekkes at et hvert underrum opfylder kravene i definition 12.1, så selv et vektorrum.

*Eksempel 12.12.* Givet  $v \in \mathbb{R}^n$  med  $v \neq 0$ , er linjen  $L$  igennem  $v$  og origo, mængden

$$L = \{tv : t \in \mathbb{R}\}.$$

Dette mængde indeholder  $v$  selv, så er ikke tom. To elementer i  $L$  har formen  $t_0v$  og  $t_1v$ , så deres sum er  $(t_0v) + (t_1v) = (t_0 + t_1)v$ , som ligger i  $L$ . Til sidst er  $s(tv) = (st)v$  i  $L$ . Så  $L$  er et underrum af  $\mathbb{R}^n$ , og dermed et vektorrum.  $\Delta$

Dette eksempel kan udvides til planer osv. på den følgende måde.

**Definition 12.13.** Rummet *udspændt* af vektorer  $v_0, v_1, \dots, v_{k-1} \in V$  er

$$\begin{aligned} \text{Span}\{v_0, v_1, \dots, v_{k-1}\} \\ = \{x_0v_0 + x_1v_1 + \dots + x_{k-1}v_{k-1} \mid x_0, x_1, \dots, x_{k-1} \text{ skalarer}\}. \end{aligned}$$

Vektorerne

$$x_0v_0 + x_1v_1 + \dots + x_{k-1}v_{k-1}$$

er netop alle de mulige lineære kombinationer af  $v_0, \dots, v_{k-1}$ .

*Eksempel 12.14.*  $xy$ -planen i  $\mathbb{R}^3$  består af alle vektorer

$$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix}.$$

Men

$$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = xe_0 + ye_1.$$

Så  $xy$ -planen er netop  $\text{Span}\{e_0, e_1\}$ .  $\Delta$

**Definition 12.15.** Lad  $A \in \mathbb{R}^{m \times n}$ . Så er *søjlerummet*  $S(A)$  af  $A$  rummet udspændt af dens søjler.

Eksempel 12.16. Matricen

$$A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

har søjler

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}.$$

Så søjlerummet består af alle vektorer af formen

$$x_0 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + x_1 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_0 + 2x_2 \\ x_1 + x_2 \\ 0 \end{bmatrix}. \quad (12.4)$$

Nar  $x_i$  varierer får vi alle vektorer af formen

$$\begin{bmatrix} s \\ t \\ 0 \end{bmatrix},$$

dvs. i dette tilfælde er søjlerummet  $S(A)$  planen i  $\mathbb{R}^3$  udspændt af  $e_0$  og  $e_1$ .

Bemærk at

$$Ax = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_0 + 2x_2 \\ x_1 + x_2 \\ 0 \end{bmatrix},$$

som er det samme som resultatet i (12.4). Så søjlerummet  $S(A)$  består netop af alle vektorer der kan fås som  $Ax$  for  $x \in \mathbb{R}^3$ .  $\triangle$

Den sidste bemærkning giver anledning til:

**Proposition 12.17.** For  $A \in \mathbb{R}^{m \times n}$  er søjlerummet

$$S(A) = \{Ax \mid x \in \mathbb{R}^n\}.$$

Specielt har ligningssystemet

$$Ax = b$$

en løsning hvis og kun hvis  $b$  ligger i søjlerummet  $S(A)$  af  $A$ .

*Bevis.* Skriv

$$A = [v_0 \mid v_1 \mid \dots \mid v_{n-1}].$$

Så har vi

$$Ax = [v_0 \mid v_1 \mid \dots \mid v_{n-1}] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = x_0 v_0 + x_1 v_1 + \dots + x_{n-1} v_{n-1}.$$

Det følger at de eneste vektor der kan skrives i formen  $Ax$  er netop lineære kombinationer af søjlerne af  $A$ , dvs. elementer af søjlerummet  $S(A)$  for  $A$ .  $\square$

Hvis vi har betragter singulærværdidekomponering i ydre produktform

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

så har vi  $\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{k-1} \geq 0$ . Lad  $r \leq k$  være så at  $\sigma_{r-1}$  er den sidste singulærværdi som er ikke nul. Så er  $A$  givet ved

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

Beregner vi  $Ax$ , får vi

$$Ax = \sigma_0 \langle v_0, x \rangle u_0 + \sigma_1 \langle v_1, x \rangle u_1 + \dots + \sigma_{r-1} \langle v_{r-1}, x \rangle u_{r-1},$$

som er en lineær kombination af  $u_0, u_1, \dots, u_{r-1}$ . Ved at lade  $x$  løbe over  $v_0, v_1, \dots, v_{r-1}$ , ser vi at vi får alle sådanne lineære kombinationer på denne måde. Det betyder at

$$S(A) = \text{Span}\{u_0, u_1, \dots, u_{r-1}\}.$$

*Eksempel 12.18.* Betragt  $v = (1, 0, -1) \in \mathbb{R}^3$ . Sæt

$$W = v^\perp = \{w \in \mathbb{R}^3 \mid \langle v, w \rangle = 0\}.$$

Så er  $W$  et underrum: (a)  $\langle v, 0 \rangle = 0$  giver  $0 \in W$ , så  $W$  er ikke tom, (b)  $u, w \in W$  siger  $\langle v, u \rangle = 0 = \langle v, w \rangle$ , men så er  $\langle v, u + w \rangle = \langle v, u \rangle + \langle v, w \rangle = 0 + 0 = 0$ , som giver  $u + w \in W$ , (c) for  $w \in W$  og  $s \in \mathbb{R}$  har vi  $\langle v, sw \rangle = s \langle v, w \rangle = 0$ , så  $sw \in W$ .  $\triangle$

Vi vil møde flere eksempler på underrum efterhånden.

# Kapitel 13

## Indre produkter

### 13.1 Definitioner og eksempler

Vi har allerede mødt det standard indre produkt på  $\mathbb{R}^n$ , og set nogle af dens egenskaber i lemma 8.2. Dette kan udvides udmiddelbart på den følgende måde.

**Definition 13.1.** Et *indre produkt* på et vektorrum  $V$  med enten reelle eller komplekse skalarer er skalarer  $\langle u, v \rangle$  for alle  $u, v \in V$  således at

- (I1)  $\langle u, v \rangle = \overline{\langle v, u \rangle}$ ,
- (I2)  $\langle su + tw, v \rangle = s\langle u, v \rangle + t\langle w, v \rangle$ ,
- (I3)  $\langle v, v \rangle \in \mathbb{R}$  og  $\langle v, v \rangle \geq 0$ ,
- (I4) for  $v \neq 0$  er  $\langle v, v \rangle > 0$ ,

for alle  $u, v, w \in V$  og alle  $s, t$  skalar.

Vi får fra del (I1) og del (I2) at

$$\langle u, sv + tw \rangle = \overline{s}\langle u, v \rangle + \overline{t}\langle u, w \rangle. \quad (13.1)$$

Som tidligere, siger vi at  $u$  og  $v$  er *ortogonal* hvis  $\langle u, v \rangle = 0$ .

*Bemærkning 13.2.* Hvis vores skalarer er reelle tal så har konjugation ingen effekt og kan droppes fra det ovenstående.  $\diamond$

*Bemærkning 13.3.* Sætters  $\|v\| = \sqrt{\langle v, v \rangle}$ , fås en *norm* på  $V$ , som opfylder det der svarer til definition 11.4, dvs.

- (N1)  $\|v\| \geq 0$  for alle  $v \in V$ , og  $\|v\| = 0$  kun for  $v = 0$ ,
- (N2)  $\|sv\| = |s|\|v\|$  for alle  $s$  skalar,
- (N3)  $\|v + w\| \leq \|v\| + \|w\|$ .

### 13 INDRE PRODUKTER

(Beviset for del (N3) kræver dog Cauchy-Schwarz uligheden nedenfor.) Vi kalder  $\|\cdot\|$  *normen induceret* af  $\langle \cdot, \cdot \rangle$ .

I situationen hvor vores skalarer er reelle kan man definere *vinklen* mellem  $u$  og  $v$  stort set som i definition 8.5, dvs.

$$\cos \theta = \frac{\langle u, v \rangle}{\|u\| \|v\|},$$

når  $u \neq 0 \neq v$ . Igen er det Cauchy-Schwarz uligheden, som sikrer at dette giver mening.  $\diamond$

*Eksempel 13.4.* På  $\mathbb{C}^n$  er det standard indre produkt givet ved

$$\langle u, v \rangle = u_0 \overline{v_0} + u_1 \overline{v_1} + \cdots + u_{n-1} \overline{v_{n-1}}. \quad (13.2)$$

Så

$$\begin{aligned} \left\langle \begin{bmatrix} 1+i \\ 2-i \end{bmatrix}, \begin{bmatrix} -1+i \\ 1+i \end{bmatrix} \right\rangle &= (1+i) \overline{(-1+i)} + (2-i) \overline{(1+i)} \\ &= (1+i)(-1-i) + (2-i)(1-i) \\ &= -1-i-i-i^2 + 2-2i-i+i^2 \\ &= 1-5i. \end{aligned}$$

Bemærk at

$$\langle u, v \rangle = \overline{v}^T u,$$

da  $u_j \overline{v_j} = \overline{v_j} u_j$ .

Den tilhørende norm er

$$\|v\|_2 = \|v\| = \sqrt{|v_0|^2 + |v_1|^2 + \cdots + |v_{n-1}|^2}.$$

Så

$$\left\| \begin{bmatrix} 1+3i \\ 2-4i \end{bmatrix} \right\|_2 = \sqrt{((1)^2 + 3^2) + (2^2 + (-4)^2)} = \sqrt{1+9+4+16} = \sqrt{30}.$$

I python kan `np.vdot` godt bruges til at beregne disse indre produkter, men man skal passe på rækkefølgen:

det indre produkt mellem komplekse  $u$  og  $v$  gives som `np.vdot(v, u)`, med  $u$  og  $v$  i modsat rækkefølge.



Funktionen `np.linalg.norm` virker, som det skal.

```
>>> import numpy as np
>>> u = np.array([1.0 + 1.0j, 2.0 - 1.0j])[:, np.newaxis]
>>> v = np.array([-1.0 + 1.0j, 1.0 + 1.0j])[:, np.newaxis]
>>> np.vdot(v, u)
(1-5j)
>>> w = np.array([1.0 + 3.0j, 2.0 - 4.0j])[:, np.newaxis]
>>> np.linalg.norm(w)
5.477225575051661
>>> np.vdot(w, w)
(30+0j)
>>> np.sqrt(np.vdot(w, w))
(5.477225575051661+0j)
```

△

*Eksempel 13.5.* På  $\mathbb{R}^n$  givet reelle tal  $w_0, \dots, w_{n-1} > 0$  er

$$\langle u, v \rangle := w_0 u_0 v_0 + w_1 u_1 v_1 + \dots + w_{n-1} u_{n-1} v_{n-1}$$

et indre produkt. Dette kan bruges når forskellige koordinatretninger måler forskellige type størrelse, f.eks. med forskellige enheder. Vi kalder dette et *vægtet indre produkt*.

På  $\mathbb{C}$  er det tilsvarende udtryk

$$\langle u, v \rangle := w_0 u_0 \overline{v_0} + w_1 u_1 \overline{v_1} + \dots + w_{n-1} u_{n-1} \overline{v_{n-1}}.$$

△

*Eksempel 13.6.* På  $V = \mathbb{R}^{m \times n}$ , kan vi definere

$$\langle A, B \rangle = \langle (a_{ij}), (b_{ij}) \rangle = \sum_{i=1}^m \sum_{j=1}^n a_{ij} b_{ij}.$$

Dette definerer et indre produkt, og den tilhørende norm kaldes *Frobeniusnormen*

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n (a_{ij})^2}.$$

Der skal understreges at  $\|A\|_2 \neq \|A\|_F$  generelt. F.eks. har

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\|A\|_2 = \sigma_0 = 2, \text{ men } \|A\|_F = \sqrt{2^2 + 1^2} = \sqrt{5}. \quad \Delta$$

*Eksempel 13.7.* Lad  $V$  være vektorrummet, som består af kontinuerte funktioner  $f: [a, b] \rightarrow \mathbb{R}$ ,  $b > a$ . Det  *$L^2$ -indre produkt* er givet ved

$$\langle f, g \rangle = \int_a^b f(x)g(x) dx.$$

Det væsentlige her er at  $\langle f, f \rangle = \int_a^b f(x)^2 dx \geq 0$ . Hvis  $f$  er ikke identisk 0, så er der et punkt  $x_0 \in [a, b]$  hvor  $f(x_0) \neq 0$ . Da  $f$  er kontinuert er der et interval  $[c, d] \subset [a, b]$ ,  $d > c$ , omkring  $x_0$  med  $f(x)^2 \geq f(x_0)^2/2$  for  $c \leq x \leq d$ . Det følger at  $\langle f, f \rangle \geq \int_c^d f(x_0)^2/2 dx > 0$ , så del (I4) er opfyldt.

F.eks. på  $[0, 1]$  har vi for funktionerne  $f(x) = x$ ,  $g(x) = x^2$  at

$$\langle f, g \rangle = \int_0^1 x \times x^2 dx = \int_0^1 x^3 dx = \left[ \frac{1}{4}x^4 \right]_0^1 = \frac{1}{4} - 0 = \frac{1}{4}.$$

Givet  $w: [a, b] \rightarrow \mathbb{R}$  kontinuert med  $w(x) > 0$ , for alle  $x$ , kan man definere et *vægtet indre produkt* ved

$$\langle f, g \rangle = \int_a^b f(x)g(x)w(x) dx.$$

Δ

*Eksempel 13.8.* For  $V$  vektorrummet af kontinuerte  $f: [a, b] \rightarrow \mathbb{C}$  har man  *$L^2$ -indre produkt*

$$\langle f, g \rangle = \int_a^b f(x)\overline{g(x)} dx.$$

Δ

Mange af vores begreber og resultater for den standard indre produkt gælder også for vilkårlige indre produkter. Specielt siger vi at  $u, v$  er ortogonal hvis  $\langle u, v \rangle = 0$ . Vi har

**Proposition 13.9.** *Lad  $\langle \cdot, \cdot \rangle$  være et indre produkt på et vektorrum  $V$  med skalarer  $\mathbb{R}$  eller  $\mathbb{C}$ . Sæt  $\|v\| = \sqrt{\langle v, v \rangle}$  til at være den inducerede norm. De følgende udsagn gælder.*

### 13.1 DEFINITIONER OG EKSEMPLER

- (a) (Pythagoras) For  $u, v$  ortogonal, er  $\|u + v\|^2 = \|u\|^2 + \|v\|^2$ .  
 (b) (Cauchy-Schwarz) For alle  $u, v \in V$ ,

$$|\langle u, v \rangle| \leq \|u\| \|v\|,$$

med lighed kun hvis  $u$  og  $v$  er parallelle, dvs. enten  $v = 0$  eller  $u = sv$  for en skalar  $s$ .

*Bevis.* For Pythagoras sætningen gentager vi beviset for proposition 8.3 med  $\|\cdot\|_2$  erstattet af  $\|\cdot\|$ :  $\langle u, v \rangle = 0$  giver  $\langle v, u \rangle = \overline{\langle u, v \rangle} = 0$ , så for  $u, v$  ortogonal har vi

$$\begin{aligned} \|u + v\|_2^2 &= \langle u + v, u + v \rangle = \langle u, u + v \rangle + \langle v, u + v \rangle && \text{definition 13.1(I2),} \\ &= \langle u, u \rangle + \langle u, v \rangle + \langle v, u \rangle + \langle v, v \rangle && \text{ligning (13.1),} \\ &= \|u\|_2^2 + \|v\|_2^2. \end{aligned}$$

For Cauchy-Schwarz uligheden, bemærk først at der intet at vise hvis  $v = 0$ , da  $\langle u, 0 \rangle = \langle u, 0 \times 0 \rangle = 0 \langle u, 0 \rangle = 0$ , og  $\|v\| = 0$ .

Hvis  $v \neq 0$ , betagter vi

$$w = u - tv, \quad \text{med } t = \langle u, v \rangle / \|v\|^2.$$

Når  $u = sv$  for en skalar  $s$ , har vi  $\langle u, v \rangle = s \langle v, v \rangle = s \|v\|^2$ , og så  $t = s$  og  $w = sv - tv = 0$ . Generelt er

$$\begin{aligned} 0 &\leq \|w\|^2 = \langle u - tv, u - tv \rangle \\ &= \langle u, u \rangle - \langle tv, u \rangle - \langle u, tv \rangle + \langle tv, tv \rangle \\ &= \|u\|^2 - t \langle v, u \rangle - \bar{t} \langle u, v \rangle + |t|^2 \|v\|^2 \\ &= \|u\|^2 - t \overline{\langle u, v \rangle} - \bar{t} \langle u, v \rangle + |t|^2 \|v\|^2 \\ &= \|u\|^2 - \frac{1}{\|v\|^2} \langle u, v \rangle \overline{\langle u, v \rangle} - \frac{1}{\|v\|^2} \overline{\langle u, v \rangle} \langle u, v \rangle + \frac{1}{\|v\|^4} |\langle u, v \rangle|^2 \|v\|^2 \\ &= \|u\|^2 - \frac{1}{\|v\|^2} |\langle u, v \rangle|^2 - \frac{1}{\|v\|^2} |\langle u, v \rangle|^2 + \frac{1}{\|v\|^2} |\langle u, v \rangle|^2 \\ &= \|u\|^2 - \frac{1}{\|v\|^2} |\langle u, v \rangle|^2. \end{aligned}$$

Vi har så

$$|\langle u, v \rangle|^2 \leq \|u\|^2 \|v\|^2.$$

Tages kvadratroden, fås Cauchy-Schwarz uligheden. Vi har lighed kun når  $w = 0$ , dvs. kun når  $u = tv$ , så  $u$  er parallel med  $v$ .  $\square$

## 13.2 Ortogonale samlinger og tilnærmelse

Med vores nye definitioner, kan vi arbejde med ortogonale og ortonormale samlinger vektorer stort set som førhen. Specielt gælder proposition 8.17: hvis  $v_0, \dots, v_{k-1}$  er ortogonal og  $u \in \text{Span}\{v_0, \dots, v_{k-1}\}$ , så har vi

$$u = x_0 v_0 + \dots + x_{k-1} v_{k-1} \quad \text{med } x_j = \frac{\langle u, v_j \rangle}{\|v_j\|^2}, \quad j = 0, 1, \dots, k-1.$$

Tilsvarende er projektionen  $P$  på  $W = \text{Span}\{v_0, v_1, \dots, v_{k-1}\}$ , når  $v_0, v_1, \dots, v_{k-1}$  er ortogonal, og ingen er 0, givet ved

$$P(u) = \frac{\langle u, v_0 \rangle}{\|v_0\|^2} v_0 + \frac{\langle u, v_1 \rangle}{\|v_1\|^2} v_1 + \dots + \frac{\langle u, v_{k-1} \rangle}{\|v_{k-1}\|^2} v_{k-1}. \quad (13.3)$$

Vektoren  $w = P(u)$  er den vektor i  $W$ , som er tættest på  $u$ , dvs. har  $\|u - w\|$  mindst muligt. Beviset er det samme, som vi har givet i  $\mathbb{R}^n$ , for projektion langs  $v_0, v_1, \dots, v_{k-1}$ .

*Eksempel 13.10.* Lad os betragte rummet  $V$  af kontinuerte funktioner  $[0, \pi] \rightarrow \mathbb{R}$  udstyret med det  $L^2$ -indre produkt. Vi påstår at

$$1, \cos(x), \cos(2x), \dots, \cos((k-1)x)$$

er en ortogonal samling i  $V$ . Dette gælder da for  $n > 0$

$$\langle 1, \cos nx \rangle = \int_0^\pi \cos(nx) dx = \left[ \frac{1}{n} \sin(nx) \right]_0^\pi = \frac{1}{n} (\sin(n\pi) - \sin(0)) = 0,$$

så 1 er vinkelret på de andre funktioner. Desuden har vi for  $m \neq n$  at

$$\begin{aligned} \langle \cos(mx), \cos(nx) \rangle &= \int_0^\pi \cos(mx) \cos(nx) dx \\ &= \frac{1}{2} \int_0^\pi \cos((m+n)x) + \cos((m-n)x) dx \\ &= \frac{1}{2} \left[ \frac{1}{m+n} \sin((m+n)x) + \frac{1}{m-n} \sin((m-n)x) \right]_0^\pi = 0. \end{aligned}$$

Så funktionerne er vinkelret på hinanden, som påstået.

Bemærk at

$$\|1\|^2 = \int_0^\pi 1^2 dx = \pi,$$

og

$$\|\cos(nx)\|^2 = \int_0^\pi (\cos(nx))^2 dx = \frac{1}{2} \int_0^\pi (\cos(2nx) + 1) dx = \frac{\pi}{2}.$$

Lad os beregne projektionen af funktionen  $1 - x$  på

$$\text{Span}\{1, \cos(x), \cos(2x), \dots, \cos((k-1)x)\}.$$

Vi har

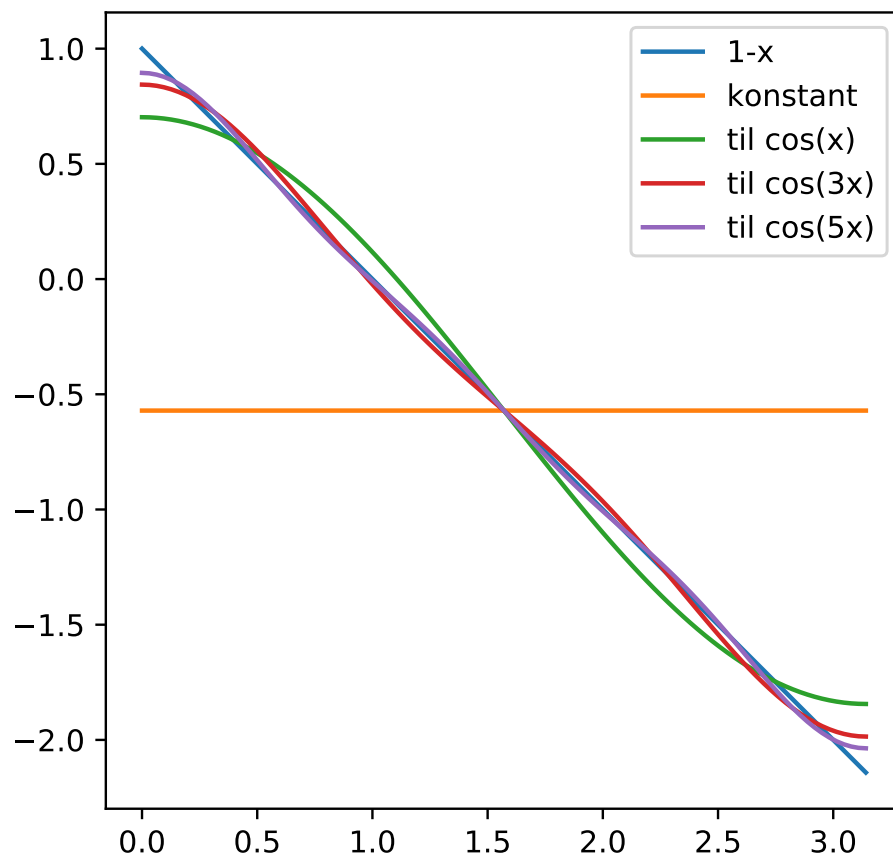
$$\langle 1 - x, 1 \rangle = \int_0^\pi (1 - x)1 dx = \left[ x - \frac{1}{2}x^2 \right]_0^\pi = \pi - \frac{1}{2}\pi^2 = \frac{\pi}{2}(2 - \pi)$$

og, for  $n > 0$ ,

$$\begin{aligned} \langle 1 - x, \cos(nx) \rangle &= \int_0^\pi \cos(nx) - x \cos(nx) dx = 0 - \int_0^\pi x \cos(nx) dx \\ &= -\left[ \frac{1}{n} x \sin(nx) \right]_0^\pi + \int_0^\pi \frac{1}{n} \sin(nx) dx \\ &= 0 + \left[ -\frac{1}{n^2} \cos(nx) \right]_0^\pi = -\frac{1}{n^2}(-\cos(n\pi) + 1) \\ &= \frac{1}{n^2}((-1)^n - 1) = \begin{cases} -\frac{2}{n^2}, & \text{for } n \text{ ulige,} \\ 0, & \text{for } n \text{ lige.} \end{cases} \end{aligned}$$

Sættes ind i (13.3), får vi

$$\begin{aligned} P(1 - x) &= \frac{\langle 1 - x, 1 \rangle}{\|1\|^2} 1 + \frac{\langle 1 - x, \cos(x) \rangle}{\|\cos(x)\|^2} \cos(x) + \frac{\langle 1 - x, \cos(2x) \rangle}{\|\cos(2x)\|^2} \cos(2x) \\ &\quad + \dots + \frac{\langle 1 - x, \cos((k-1)x) \rangle}{\|\cos((k-1)x)\|^2} \cos((k-1)x) \\ &= \frac{\pi(2 - \pi)/2}{\pi} 1 + \frac{2/1^2}{\pi/2} \cos(x) + 0 + \frac{2/3^2}{\pi/2} \cos(3x) + 0 \\ &\quad + \frac{2/5^2}{\pi/2} \cos(5x) + \dots + \frac{2/r^2}{\pi/2} \cos(rx) \\ &= \frac{1}{2}(2 - \pi) + \frac{4}{\pi} \cos(x) + \frac{4}{9\pi} \cos(3x) \\ &\quad + \frac{4}{25\pi} \cos(5x) + \dots + \frac{4}{r^2\pi} \cos(rx), \end{aligned} \tag{13.4}$$

Figur 13.1: Fourier cosinus tilnærmelse for  $1 - x$ .

hvor  $r$  er det største ulige tal  $\leq k - 1$ . Dette kaldes en *Fourier cosinus udvikling* af funktionen  $1 - x$ . Billedfilformatet jpeg er baseret på en diskret udgave af Fourier cosinus rækkeudvikling.

△

*Eksempel 13.11.* Fourier rækker generelt kommer fra at betragte kontinuerte funktioner  $[-\pi, \pi] \rightarrow \mathbb{R}$ . Her er

$$1, \cos(x), \sin(x), \cos(2x), \sin(2x), \dots, \cos((k-1)x), \sin((k-1)x) \quad (13.5)$$

ortogonal med

$$\|1\|^2 = 2\pi$$

og

$$\|\cos(nx)\|^2 = \pi = \|\sin(nx)\|^2.$$

Man kan beregne en tilnærmelse til andre kontinuerte funktioner på  $[-\pi, \pi]$  ved at projicere på rummet udspændt af (13.5):

$$\begin{aligned} P(f) = & a_0 + c_1 \cos(x) + b_1 \sin(x) \\ & + c_2 \cos(2x) + b_2 \sin(2x) \\ & + c_3 \cos(3x) + b_3 \sin(3x) \\ & + \cdots + c_{k-1} \cos((k-1)x) + b_{k-1} \sin((k-1)x), \end{aligned}$$

med

$$\begin{aligned} a_0 &= \frac{\langle f, 1 \rangle}{\|1\|^2} = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) dx, \\ c_m &= \frac{\langle f, \cos(mx) \rangle}{\|\cos(mx)\|^2} = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(mx) dx \\ b_m &= \frac{\langle f, \sin(mx) \rangle}{\|\sin(mx)\|^2} = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(mx) dx. \end{aligned}$$

I det næste afsnit vil vi se hvordan, udregning af denne type kan udføres numerisk i python. △

## 13.3 Numerisk integration

Hvis vi vil regne eksempler som 13.10 i python, skal vi have styr på hvordan vi kan beregne integraler numerisk.

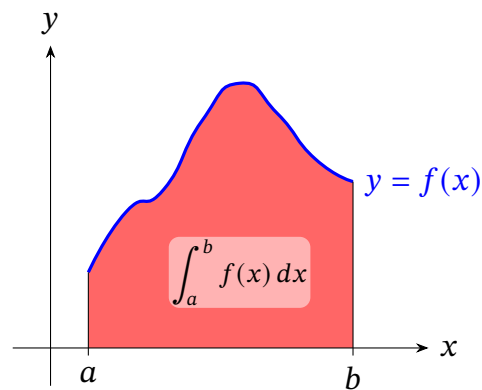
Givet en kontinuert funktion  $f: [a, b] \rightarrow \mathbb{R}$  er integralet  $\int_a^b f(x) dx$  arealet under dens graf, se figur 13.2. Der er forskellige måder vi kan lave en tilnærmelse til dette areal.

Lad  $x_0, x_1, \dots, x_{n-1}$  være  $n$  punkter ligelig fordelt fra  $a = x_0$  til  $b = x_{n-1}$ . Sådant en samling får vi fra `np.linspace(a, b, n)`. For hvert  $x_j$  kan vi betragte rektanglet  $R$  med venstre side ved  $x_j$ , højre side ved  $x_{j+1}$ , og med højde  $f(x_j)$ , se figur 13.3. Rektanglet har areal

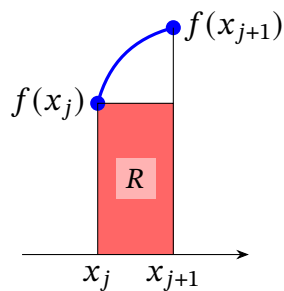
$$hf(x_j)$$

hvor  $h = x_{j+1} - x_j$ . Vi har sådan et rektangel for hvert venstre start punkt  $x_0, x_1, \dots, x_{n-2}$ , da rektanglet til højre fra  $x_{n-1}$  ligger uden for det område vi

### 13 INDRE PRODUKTER



Figur 13.2: Integral.



Figur 13.3: Rektanglet fra den venstre funktionsværdi.

arbejder med. En mulig tilnærmelse til integralet er summen af arealerne på disse rektangler:

$$\begin{aligned} & hf(x_0) + hf(x_1) + \cdots + hf(x_{n-2}) \\ &= h(f(x_0) + f(x_1) + \cdots + f(x_{n-2})). \end{aligned}$$

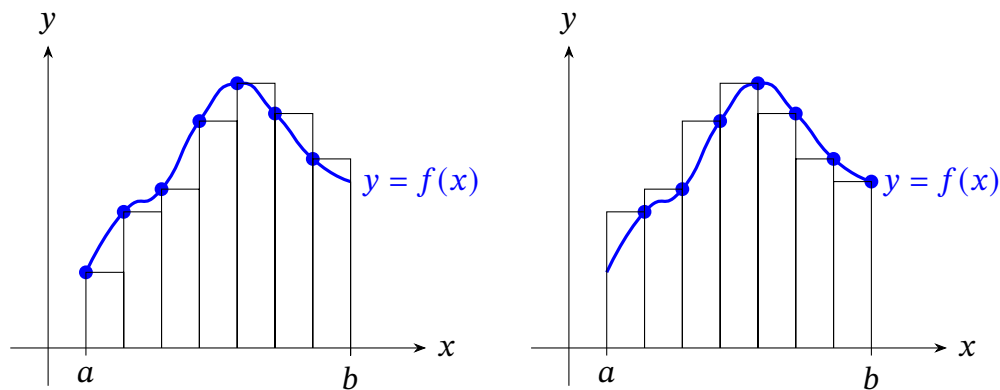
Værdien  $h$  kan regnes, som

$$h = \frac{1}{n-1}(b-a),$$

da vi har delt  $[a, b]$  op i  $(n-1)$ -delintervaller. Ved brug af `np.linspace()` er det ikke nødvendigt at beregne  $h$ , man kan bede `np.linspace()` at returnere det ved hjælp af `np.linspace(a, b, n, retstep=True)`. F.eks. tager vi



### 13.3 NUMERISK INTEGRATION



Figur 13.4: Venstre- og højre-tilnærmelse til integral.

funktionen  $f(x) = x^3$  på  $[0, 2]$ , hvor vi ved at  $\int_0^2 x^3 dx = [x^4/4]_0^2 = 4$ , kan vi regne i python med  $n = 100$ :

```
>>> import numpy as np
>>> n = 100
>>> x, h = np.linspace(0, 2, n, retstep=True)
>>> h * np.sum((x**3)[: -1])
3.919600040812163
```

Dette er lidt mindre end 4, da funktionen er voksende og alle rektangler ligger under grafen.

Som alternativ kunne man brug funktionsværdien i den højre endepunkt af intervallet  $[x_j, x_{j+1}]$ , se figur 13.4, og får tilnærmelsen

$$h(f(x_1) + f(x_2) + \dots + f(x_{n-1}))$$

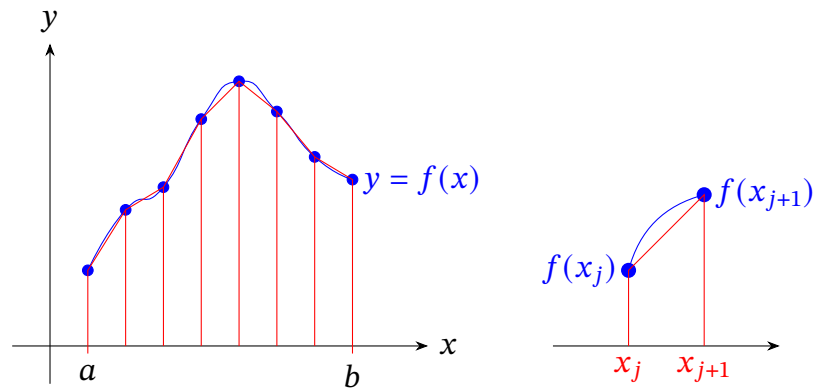
```
>>> h * np.sum((x**3)[1:])
4.081216202428324
```

Denne gang, for funktionen  $f(x) = x^3$ , får vi et svar der lidt for stort.

Et kompromis er at erstatte rektanglet med trapezet, som i figur 13.5. Dette trapez har areal

$$h \frac{1}{2} (f(x_j) + f(x_{j+1})).$$

### 13 INDRE PRODUKTER



Figur 13.5: Trapez-tilnærmelse til integral.

Dette giver trapez-tilnærmelsen til integralet

$$\frac{1}{2}h(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + f(x_{n-1})). \quad (13.6)$$

Denne regel er implementeret i python som `np.trapz()`, som kan bruges på følgende måde

```
>>> np.trapz(x**3, dx=h)
4.000408121620244
```

som er væsentligt tættere på den rigtige værdi for integralet.

Generelt kan det vises at denne trapezregel (13.6) på funktioner, som har et andet afledte, beregner integraler  $\int_a^b f(x) dx$  inden for en fejl af

$$\frac{1}{12}h^3(b-a) \max\{|f''(x)| \mid a \leq x \leq b\}.$$

Trapezformlen (13.6) kommer fra at erstattet  $f$  med linjestykker. Man kan i stedet for bruge højre ordens polynomier. Andengradspolynomier giver anledning til Simpsons regel

$$\int_a^b f(x) dx \approx \frac{1}{3}h(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 2f(x_{n-3}) + 4f(x_{n-2}) + f(x_{n-1})) \quad (13.7)$$

for  $n$  ulig. Denne gang er fejlen i beregning af integralet højst

$$\frac{1}{90}h^4(b-a)\max\{|f''''(x)| \mid a \leq x \leq b\}.$$

Lad os anvender trapezreglen for beregner eksempel 13.10 numerisk i python.

```
>>> import numpy as np
>>> n = 100
>>> x, h = np.linspace(0, np.pi, 100, retstep=True)
```

Vores ortogonale funktioner er 1,  $\cos(x)$ ,  $\cos(2x)$ , osv. Den første gemmer vi i

```
>>> konstant = np.ones_like(x)
```

de andre kan bare regnes som  $\text{np.cos}(m*x)$ . Lad os bekræfter at et par af disse er ortogonal. Indre produktet er  $\langle f, g \rangle = \int_0^\pi f(x)g(x) dx$ , så vi kan bruge  $\text{np.trapz}()$  på  $f*g$ :

```
>>> def indre_produkt(f, g, h):
...     return np.trapz(f * g, dx=h)
...
>>> indre_produkt(konstant, np.cos(3*x), h)
-2.0816681711721685e-17
>>> indre_produkt(np.cos(2*x), np.cos(3*x), h)
-2.7755575615628914e-17
>>> indre_produkt(np.cos(5*x), np.cos(3*x), h)
-9.020562075079397e-17
```

som viser at de er ortogonale inden for machine epsilon. Vi har også beregnet  $\|1\|^2 = \pi$ ,  $\|\cos(mx)\|^2 = \pi/2$ , som kan også bekræftes i python

```
>>> def norm_sq(f, h):
...     return indre_produkt(f, f, h)
...
>>> norm_sq(konstant, h) - np.pi
-4.440892098500626e-16
>>> norm_sq(np.cos(x), h) - np.pi/2
```

### 13 INDRE PRODUKTER

```
-2.220446049250313e-16
>>> norm_sq(np.cos(4*x), h) - np.pi/2
0.0
```

Dette er igen korrekt inden for machine epsilon.

I eksempel 13.10 beregnede vi projektionerne af  $1 - x$  på de første  $k$  basis vektorer. Lad os indføre en funktion der gøre dette for os

```
>>> def proj(f, k, x, h):
...     out = (indre_produkt(f, konstant, h)
...           / norm_sq(konstant, h)
...           * konstant)
...     for m in range(1, k, 2):
...         out += (indre_produkt(f, np.cos(m*x), h)
...                / norm_sq(np.cos(m*x), h)
...                * np.cos(m*x))
...     return out
... 
```

Plotter man  $1-x$ , og  $\text{proj}(1-x, k, x, h)$ , for  $k = 1, 2, 4, 6$  fås figur 13.1. Bemærk at

```
>>> np.allclose(proj(1-x, 3, x, h), proj(1-x, 2, x, h),
...             atol = np.finfo(float).eps)
True
```

som bekræfter at  $\cos(2x)$  bidrager ikke til denne projektion, se formlen (13.4). Sådanne bidrag falder væk da indre produkterne  $\langle 1 - x, \cos(mx) \rangle$  er nul for  $m$  lige:

```
>>> for m in range(2, 7, 2):
...     print(indre_produkt(1-x, np.cos(m*x), h))
... 
```

```
-5.551115123125783e-17
-5.551115123125783e-17
-2.7755575615628914e-17
```

## Kapitel 14

# Ortogonale samlinger: klassisk Gram-Schmidt

Vi har brugt ortogonale samlinger for at finde tilnærmelser til vektorer og funktioner. Her vil vi undersøge forskellige metode for at konstruere sådanne samlinger.

### 14.1 Den klassiske Gram-Schmidt proces

Lad  $V$  være et vektorrum med indre produkt  $\langle \cdot, \cdot \rangle$ . Husk at projektion på linjen udspændt af  $v \neq 0$  er givet ved

$$\text{pr}_v(u) = \frac{\langle u, v \rangle}{\|v\|^2} v.$$

Vektoren

$$u - \text{pr}_v(u)$$

er derefter vinkelret på  $v$ . Vi har også set at for  $v_0, \dots, v_{k-1}$  ortogonal, med ingen 0, er projektionen til  $\text{Span}\{v_0, \dots, v_{k-1}\}$  givet ved

$$\text{pr}_{v_0, \dots, v_{k-1}}(u) = \frac{\langle u, v_0 \rangle}{\|v_0\|^2} v_0 + \dots + \frac{\langle u, v_{k-1} \rangle}{\|v_{k-1}\|^2} v_{k-1}$$

og at  $u - \text{pr}_{v_0, \dots, v_{k-1}}(u)$  er ortogonal på  $v_0, \dots, v_{k-1}$ .

Vi kan bruge dette til at opbygge en samling ortogonale vektorer nogenlunde på følgende måde. Input er vektorer  $u_0, u_1, \dots, u_{k-1} \in V$ .

## 14 ORTOGONALE SAMLINGER: KLASSISK GRAM-SCHMIDT

KLASSISK GRAM-SCHMIDT PROCES—SKITSE( $u_0, u_1, \dots, u_{k-1}$ )

- 1 Sæt  $w_0 = u_0$ ,  $v_0 = c_0 w_0$  for et valgfrit  $c_0 \neq 0$ .
- 2 Sæt  $w_1 = u_1 - \text{pr}_{v_0}(u_1)$ ,  $v_1 = c_1 w_1$ ,  $c_1 \neq 0$ .
- 3 Sæt  $w_2 = u_2 - \text{pr}_{v_0, v_1}(u_2)$ ,  $v_2 = c_2 w_2$ ,  $c_2 \neq 0$ .
- ...
- 4 Sæt  $w_r = u_r - \text{pr}_{v_0, \dots, v_{r-1}}(u_r)$ ,  $v_r = c_r w_r$ ,  $c_r \neq 0$ .
- ...
- 5 **return**  $v_0, v_1, \dots, v_{k-1}$ , en ortogonal samling,  
der udspænder det samme rum som  $u_0, u_1, \dots, u_{k-1}$ .

Vælges  $c_r = 1/\|w_r\|$  i hvert trin, så bliver  $v_0, v_1, \dots, v_{k-1}$  ortonormal.

*Eksempel 14.1.* I  $\mathbb{R}^4$  med det standard indre produkt betragt

$$u_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad u_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad u_2 = \begin{bmatrix} -1 \\ 3 \\ 1 \\ -1 \end{bmatrix}.$$

Vi sætter

$$v_0 = w_0 = u_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

Bemærk  $\|v_0\|_2^2 = 1^2 + 1^2 + 0^2 + 0^2 = 2$ . Vi har  $\langle u_1, v_0 \rangle = 2 \times 1 + 1 \times 1 + 1 \times 0 + 1 \times 0 = 3$ .  
Så

$$\begin{aligned} w_1 &= u_1 - \text{pr}_{v_0}(u_1) \\ &= u_1 - \frac{\langle u_1, v_0 \rangle}{\|v_0\|_2^2} v_0 = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \frac{3}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 2 \\ 2 \end{bmatrix}. \end{aligned}$$

Lad os gange i gennem med  $c_1 = 2$  for at fjerne brøken og sætte

$$v_1 = 2w_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \\ 2 \end{bmatrix},$$

## 14.1 DEN KLASSISKE GRAM-SCHMIDT PROCES

som har  $\|v_2\|_2^2 = 1 + 1 + 4 + 4 = 10$ . Vi regner så at

$$\begin{aligned} w_2 &= u_2 - \text{pr}_{v_0, v_1}(u_2) \\ &= u_2 - \frac{\langle u_2, v_0 \rangle}{\|v_0\|_2^2} v_0 - \frac{\langle u_2, v_1 \rangle}{\|v_1\|_2^2} v_1 = \begin{bmatrix} -1 \\ 3 \\ 1 \\ -1 \end{bmatrix} - \frac{-1+3}{2} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \frac{-1-3+2-2}{10} \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \\ &= \begin{bmatrix} -1 \\ 3 \\ 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \frac{2}{5} \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} = \frac{1}{5} \begin{bmatrix} -8 \\ 8 \\ 9 \\ -1 \end{bmatrix}. \end{aligned}$$

Sættes  $v_2 = 5w_2$ , fås at

$$v_0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad v_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \\ 2 \end{bmatrix}, \quad v_2 = \begin{bmatrix} -8 \\ 8 \\ 9 \\ -1 \end{bmatrix}$$

er en ortogonal samling, som udspænder  $\text{Span}\{u_0, u_1, u_2\}$  i  $\mathbb{R}^4$ .

Det tilsvarende regnestykke i python er

```
>>> import numpy as np

>>> u0 = np.array([1.0, 1.0, 0.0, 0.0])[:, np.newaxis]
>>> u1 = np.array([2.0, 1.0, 1.0, 1.0])[:, np.newaxis]
>>> u2 = np.array([-1.0, 3.0, 1.0, -1.0])[:, np.newaxis]

>>> def proj_på(v, u):
...     return np.vdot(v, u) / np.vdot(v, v) * v
...
>>> v0 = u0
>>> v0
array([[1.],
       [1.],
       [0.],
       [0.]])
>>> w1 = u1 - proj_på(v0, u1)
>>> v1 = 2*w1
```

```

>>> v1
array([[ 1.],
       [-1.],
       [ 2.],
       [ 2.]])
>>> w2 = u2 - proj_på(v0, u2) - proj_på(v1, u2)
>>> v2 = 5*w2
>>> v2
array([[ -8.],
       [  8.],
       [  9.],
       [-1.]])

```

△

Ved beskrivelsen af den klassiske Gram-Schmidt proces ovenfor, brugte vi ordet »nogenlunde«. Grunden til dette er at vi har brug for at  $w_r$ , og dermed  $v_r$ , er ikke nul i hvert trin. Dette er tilfældet hvis og kun hvis  $u_0, u_1, \dots, u_{k-1}$  opfylder:

**Definition 14.2.** En samling vektorer  $u_0, u_1, \dots, u_{k-1}$  i  $V$  er *lineært uafhængig* hvis den eneste løsning på ligningen

$$s_0 u_0 + s_1 u_1 + \dots + s_{k-1} u_{k-1} = 0 \quad (14.1)$$

med  $s_i$  skalarer er  $s_0 = 0 = s_1 = \dots = s_{k-1}$ .

*Bemærkning 14.3.* En måde at tjekke denne betingelse på er at køre algoritmen ovenfor: Hvis  $w_r = 0$  for et  $r$ , så er  $u_r = \text{pr}_{u_0, u_1, \dots, u_{r-1}}(u_r)$ . Men denne projektion er en lineær kombination  $s_0 u_0 + s_1 u_1 + \dots + s_{r-1} u_{r-1}$  af  $u_0, \dots, u_{r-1}$ . Sættes  $s_r = -1$  og  $s_i = 0$  for  $i > r$ , fås en løsning til (14.1) med  $s_r \neq 0$ . Så samlingen er ikke lineært uafhængig.

Omvendt hvis  $u_0, u_1, \dots, u_{k-1}$  er lineært uafhængig, så er ingen  $u_r$  en lineær kombination af  $u_0, \dots, u_{r-1}$ . Specielt er  $u_r$  ikke lige med  $\text{pr}_{u_0, u_1, \dots, u_{r-1}}(u_r)$ , så  $w_r \neq 0$  for alle  $r$ . ◇

*Eksempel 14.4.* Lad  $u_0, u_1, \dots, u_{k-1} \in \mathbb{R}^n$  og betragt matricen

$$A = [u_0 \mid u_1 \mid \dots \mid u_{k-1}] \in \mathbb{R}^{n \times k}$$



## 14.1 DEN KLASSISKE GRAM-SCHMIDT PROCES

hvis søjler er  $u_r, r = 0, \dots, k-1$ . Så er  $u_0, u_1, \dots, u_{k-1}$  lineært uafhængig hvis og kun hvis ligningen

$$Ax = 0, \quad x \in \mathbb{R}^k$$

har kun én løsningen, nemlig  $x = 0$ . Dette gælder fordi  $Ax = x_0 u_0 + x_1 u_1 + \dots + x_{k-1} u_{k-1}$ , så ligning (14.1) er  $Ax = 0$  efter udskiftning af variable. Reduceres  $A$  til echelonform får vi så at  $u_0, u_1, \dots, u_{k-1}$  er lineært uafhængig hvis alle søjler i  $A$  indeholder et pivotelement.

F.eks. vektorerne  $u_0 = (1, 0, 2)$ ,  $u_1 = (0, 2, 1)$ ,  $u_2 = (1, 2, 3)$  er ikke lineært uafhængig da

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix} \sim_{R_2 \rightarrow R_2 - 2R_0} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 2 \\ 0 & 1 & 1 \end{bmatrix} \sim_{R_1 \leftrightarrow R_2} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 2 & 2 \end{bmatrix} \sim_{R_2 \leftrightarrow R_2 - 2R_1} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

har kun pivot elementer i de første 2 søjler, og ikke i den sidste.  $\Delta$

*Eksempel 14.5.* For  $a < b$ , betragt vektorrummet af differentiable funktioner  $f: [a, b] \rightarrow \mathbb{R}$ . Så er polynomierne

$$1, x, x^2, \dots, x^{k-1}$$

lineært uafhængig i  $V$ . For at vise dette, antag at

$$p(x) = s_0 1 + s_1 x + s_2 x^2 + \dots + s_{k-1} x^{k-1} = 0(x) = 0, \quad \text{for alle } a \leq x \leq b.$$

Betragt den  $(k-1)$ 'te afledte  $d^{k-1}p/dx^{k-1}$ , som er nødvendigvis 0. Vi får

$$0 = \frac{d^{k-1}p}{dx^{k-1}} = (k-1)! s_{k-1},$$

som giver  $s_{k-1} = 0$ . Kikker man derefter på den  $(k-2)$ 'te afledte, får man  $s_{k-2}$ , og fortsætter man i på denne måde, ser man at  $s_r = 0$  for alle  $r$ .

Bruger man den klassiske Gram-Schmidt på  $1, x, x^2, \dots$  for forskellige indre produkter får man nogle kendte ortogonale systemer af polynomier:

(a) *Legendre polynomier*  $[a, b] = [-1, 1]$ ,  $\langle f, g \rangle = \int_{-1}^1 f(x)g(x) dx$ , giver

$$P_0(x) = 1,$$

$$P_1(x) = x,$$

$$P_2(x) = 3x^2 - 2,$$

$$P_3(x) = 5x^3 - 3x.$$

(b) *Chebyshev polynomier*  $[a, b] = [-1, 1]$ ,

$$\langle f, g \rangle = \int_{-1}^1 \frac{f(x)g(x)}{\sqrt{1-x^2}} dx$$

giver

$$T_0(x) = 1,$$

$$T_1(x) = x,$$

$$T_2(x) = 2x^2 - 1,$$

$$T_3(x) = 4x^3 - 3x,$$

$$T_4(x) = 8x^4 - 8x^2 + 1.$$

Disse og flere andre polynomier af denne type er indbygget i NumPy.

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

from numpy.polynomial import Polynomial as P
from numpy.polynomial import Legendre as L
from numpy.polynomial import Chebyshev as T

x, h = np.linspace(-1, 1, 100, retstep=True)

fig, ax = plt.subplots()
for i in range(6):
    ax.plot(x, L.basis(i)(x))
```

```
fig, ax = plt.subplots()
for i in range(6):
    ax.plot(x, T.basis(i)(x))
```

Vi får koefficienter for de givne polynomier vist via

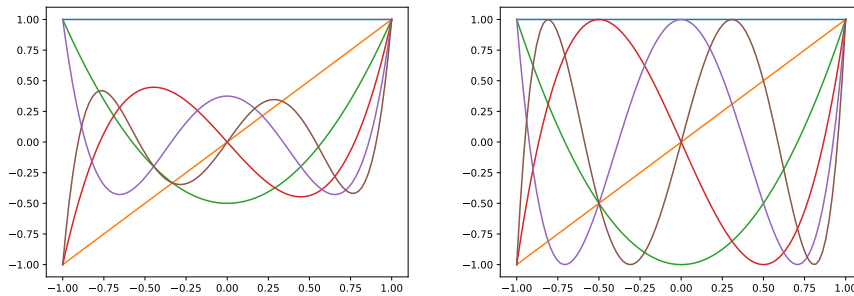
```
np.polynomial.set_default_printstyle('ascii')
print(T.basis(4).convert(kind=P))
```

```
1.0 + 0.0 x**1 - 8.0 x**2 + 0.0 x**3 + 8.0 x**4
```

som passer med vores udtryk for  $T_4(x)$ .

△

## 14.2 KLASSISK GRAM-SCHMIDT I $\mathbb{R}^n$



Figur 14.1: Legendre og Chebychev polynomier.

## 14.2 Klassisk Gram-Schmidt i $\mathbb{R}^n$

Vores klassisk Gram-Schmidt starter med  $u_0, u_1, \dots, u_{k-1}$  lineært uafhængig. Lad os indføre lidt andet notation i tilfældet hvor vi konstruere en ortonormal samling  $v_0, v_1, \dots, v_{k-1}$ . I hvert trin danne vi

$$w_j = u_j - \text{pr}_{v_0, \dots, v_{j-1}}(u_j) = u_j - r_{0j}v_0 - \dots - r_{j-1,j}v_{j-1}. \quad (14.2)$$

Da  $\langle v_i, w_j \rangle = 0$  og  $\|v_i\|_2 = 1$ , kan vi tage indre produktet af  $v_i$  med (14.2), som dermed giver  $r_{ij} = \langle v_i, u_j \rangle = v_i^T u_j$ . For at få en ortonormal samling, sætter vi så

$$v_j = \frac{1}{r_{jj}} w_j, \quad \text{hvor } r_{jj} = \|w_j\|_2. \quad (14.3)$$

Alt i alt har vi den følgende algoritme

KLASSISK GRAM-SCHMIDT( $u_0, u_1, \dots, u_{k-1}$ )

```

1  for  $j \in \{0, 1, \dots, k-1\}$ :
2       $w_j = u_j$ 
3      for  $i \in \{0, 1, \dots, j-1\}$ :
4           $r_{ij} = v_i^T u_j$ 
5           $w_j = w_j - r_{ij}v_i$ 
6       $r_{jj} = \|w_j\|_2$ 
7       $v_j = w_j / r_{jj}$ 
```

Givet  $v_i$  og  $r_{ij}$  kan vi omskrive (14.2) og (14.3) til at få

$$\begin{aligned} u_0 &= r_{00}v_0, \\ u_1 &= r_{01}v_0 + r_{11}v_1, \\ u_2 &= r_{02}v_0 + r_{12}v_1 + r_{22}v_2, \\ &\vdots \\ u_{k-1} &= r_{0,k-1}v_0 + r_{1,k-1}v_1 + \cdots + r_{k-1,k-1}v_{k-1}. \end{aligned} \tag{14.4}$$

Dette kan skrives i matrixform, som

$$\begin{aligned} A &= [u_0 \mid u_1 \mid u_2 \mid \cdots \mid u_{k-1}] \\ &= [v_0 \mid v_1 \mid v_2 \mid \cdots \mid v_{k-1}] \begin{bmatrix} r_{00} & r_{01} & r_{02} & \cdots & r_{0,k-1} \\ 0 & r_{11} & r_{12} & \cdots & r_{1,k-1} \\ 0 & 0 & r_{22} & \cdots & r_{2,k-1} \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & & r_{k-1,k-1} \end{bmatrix} \\ &= QR. \end{aligned} \tag{14.5}$$

Her er  $A \in \mathbb{R}^{n \times k}$ , matricen  $Q \in \mathbb{R}^{n \times k}$  har ortonormale søjler  $v_0, \dots, v_{k-1}$ , og  $R \in \mathbb{R}^{k \times k}$  er øvre triangulær. En dekomponering af denne form kaldes en *tynd QR-dekomponering* af  $A$ . En *fuld QR-dekomponering* fås ved at udvide  $R$  med 0-tal til en matrix i  $\mathbb{R}^{n \times k}$ , og at tilføje søjler til  $Q$ , f.eks. ved hjælp af korollar 9.18, så vi får en ortogonal matrix i  $\mathbb{R}^{n \times n}$ .

Hvis  $u_0, u_1, \dots, u_{k-1}$  er lineært uafhængig, giver den klassiske Gram-Schmidt proces os en tynd QR-dekomponering med  $r_{ii} > 0$  for alle  $i$ .

Hvis  $A$  er en  $n \times k$ -matrix, med  $n \geq k$ , hvis søjler er ikke lineært uafhængig, så kan vi køre den klassiske Gram-Schmidt proces, men når et  $w_j$  er 0, kan vi bare vælge  $v_j$  til være en tilfældig enhedsvektor vinkelret på  $v_0, v_1, \dots, v_{j-1}$ . På denne måde får vi en QR-dekomponering for  $A$ , men nogle  $r_{ii}$  kan være 0.

Dette giver

**Proposition 14.6.** *Alle matrixer  $A \in \mathbb{R}^{n \times k}$ , for  $n \geq k$ , tillader en tynd og end fuld QR-dekomponering.*

*Hvis søjlerne af  $A$  er lineært uafhængig og  $r_{ii} > 0$  for alle  $i$ , så er denne tynd QR-dekomponering entydig.*  $\square$

*Bemærkning 14.7.* Som vi vil se i det næste kapitel er der numeriske problemer med den klassiske Gram-Schmidt proces. Vi vil give andre metoder, der er numerisk mere hensigtsmæssig.  $\diamond$

## 14.2 KLASSISK GRAM-SCHMIDT I $\mathbb{R}^n$

*Advarsel 14.8.* Numpy har en indbygget funktion `np.linalg.qr`. Den producerer ikke en QR-dekomponering af den overstående type, da den tillader  $r_{ii} < 0$ . !



# Appendiks A

## Det græske alfabet

$\alpha$	A	alpha	$\nu$	N	nu
$\beta$	B	beta	$\xi$	$\Xi$	xi
$\gamma$	$\Gamma$	gamma	$o$	O	omicron
$\delta$	$\Delta$	delta	$\pi$	$\Pi$	pi
$\varepsilon$	E	epsilon	$\rho$	R	rho
$\zeta$	Z	zeta	$\sigma$	$\Sigma$	sigma
$\eta$	H	eta	$\tau$	T	tau
$\theta$	$\Theta$	theta	$\upsilon$	Y	ypsilon
$\iota$	I	jota	$\varphi$	$\Phi$	phi
$\kappa$	K	kappa	$\chi$	X	chi
$\lambda$	$\Lambda$	lambda	$\psi$	$\Psi$	psi
$\mu$	M	mu	$\omega$	$\Omega$	omega





# Appendiks B

## Python

Programmeringssproget python kan hentes fra

<https://www.python.org>

Vær opmærksom på at der er nogle væsentlige forskel mellem python version 2.\* og 3.\*. Vi bruger python 3, mere præcist

```
>>> import sys
>>> print(sys.version)
3.9.2 (default, Feb 22 2021, 19:12:30)
[Clang 12.0.0 (clang-1200.0.32.29)]
```

Nogle computer har python 2 installeret i forvejen, men ikke python 3. I sådanne tilfælde bliver man nødt til at installere python 3 selv.

Desuden bruger vi pakker fra SciPy samlingen, som findes ved

<https://www.scipy.org>

Specielt NumPy og Matplotlib

```
>>> import numpy
>>> print(numpy.__version__)
1.20.1
>>> import matplotlib
>>> print(matplotlib.__version__)
3.3.4
```

## B PYTHON

Installationsvejledning findes ved samlingens hjemmeside.

For at skrive python filer er det nyttigt at have jupyter lab som kan hentes fra

<https://jupyter.org>

I jupyter lab kan man oprette notebooks, som kombinerer kode og almindelig tekst.

Der findes nogle distributioner som anaconda,

<https://www.anaconda.com>

der inkluderer python og alle de overnævnte tillægspakker.

# Bibliografi

The NumPy community (29. jun. 2020). *NumPy user guide*. Vers. release 1.19.0.  
214 s. URL: <https://www.scipy.org/docs.html>.



# Python indeks

- (differens)
  - array, 32
  - float, 2, 3
- # (kommentar), 10
- () (tupel), 31
- \* (produkt)
  - float, 2, 3
  - skalar-array, 22, 31
- \* (unpack), 56
- \*\* (potens)
  - float, 3
- \*=, 60
- + (sum)
  - array, 32
  - float, 2, 3
- +=, 60
- / (kvotient)
  - float, 2, 3
- // (floor divide), 6
- :
- indeksering, 26
- = (sæt et variabel), 10
- @ (matrixprodukt), 40

## A

- all, 86
- allclose, 86
- arccos, 80
- arctan2, 19

- array, 21, 26

## C

- Chebyshev, 162
- clim, 29
- cmap, 28
- color
  - plot, 22
- colorbar, 28
- complex, 136
- cond
  - linalg.cond, 130
- conj, 139
- convert
  - polynomial, 162

## D

- def, 104
- default\_rng, 49
- diag, 110
- dtype, 27
  - float, 27

## E

- e
  - i float, 5
- empty, 81
- empty\_like, 82
- enumerate, 116
- exp, 7, 92

## PYTHON INDEXES

### F

`finfo` (float information), 6  
`float`, 5  
    `dtype`, 27  
`for`, 53, 56  
`from`, 162

### H

`hstack`, 60

### I

`if...else`, 103  
`imag`  
    `complex`, 136  
`import`, 6  
`imread`, 116  
`imshow`, 116  
`indexing`, 27  
    `ndarray`, 26  
`inv`  
    `linalg.inv`, 71

### J

`j`, 136

### L

`label`  
    `plot`, 92  
`legend`, 92  
`Legendre`, 162  
`linalg`  
    `linalg.cond`, 130  
    `linalg.inv`, 71  
    `linalg.norm`, 79, 145  
    `linalg.qr`, 165  
    `linalg.svd`, 109, 113, 118  
`linspace`, 48  
    `retstep`, 152

### M

`marker`  
    `plot`, 22  
`markersize`, 112  
`matplotlib`, 21  
`matshow`, 28, 48, 116

### N

`nbytes`  
    `ndarray`, 119  
`ndarray`, 26  
`ndim`, 26  
`ndindex`, 116  
`norm`  
    `linalg.norm`, 79, 145  
`normal`, *see* `rng`  
`np`, *see* `numpy`  
`numpy`, 6

### O

`ones`, 72, 91  
`ones_like`, 155

### P

`pi`, 19  
`plot`, 21  
`polynomial`  
    `Chebyshev`, 162  
    `convert`, 162  
    `Legendre`, 162  
    `Polynomial`, 162  
`print`, 49, 54

### Q

`qr`  
    `linalg.qr`, 165

### R

`random`, 49

`range`, 54

`real`

`complex`, 136

`rng`, se også `default_rng`

`normal`, 131

`random`, 49

`standard_normal`, 49

## S

`savefig`, 22

`set_aspect`, 21

`set_printoptions`, 115

`set_xlabel`, 118

`set_xlim`, 22

`set_ylabel`, 118

`set_ylim`, 22

`seterr`, 9

`shape`

`ndarray`, 26

`show`

`billede`, 116

`matrix`, 28

`plot`, 22

`shrink`, 29

`sqrt`, 10

`standard_normal`, se `rng`

`subplots`, 21, 116

`svd`

`linalg.svd`, 109, 113, 118

`svdapprox`, 116, 119

## T

`T` (transponering), 33

`text`

`subplot`, 113

`tight_layout`, 116

`trapz`, 154

`triu`, 72

## V

`vdot`, 104, 145

## Z

`zeros`, 31





# Indeks

1-norm, 126

2-norm, 79

kompleks, 144

matrix, 127

$\infty$ -norm, 127

## A

akse, 26

andengradsligning, 10, 11

## B

back substitution, 59

basis

ortonormal, 99

bundne variabler, 64

## C

cache, 53

Cauchy-Schwarz ulighed, 83

Chebyshev polynomier, 162

## D

determinant, 70

differens

float, 3

## E

echelonform, 62

eksponent, 5

eksponentialfunktion, 7

elektrisk

kredsløb, 43

elementær

matrix, 74

elementære

operationer på ligninger, 58

rækkeoperationer, 58

enhedsvektor, 14, 79

epsilon

machine, 6

## F

F1, se float, aksiomer

F2, se float, aksiomer

farvelægning

af en matrix, 28

farveskala, 28

grænser, 29

fejl, 3

relativ, 4

float, 5

aksiomer, 9

flop, 51

flydende-komma repræsentation, 5

fordeling

standard normal-, 49

uniform-, 49

for-løkke, 51, 53

Fourier cosinus række, 150

## INDEKS

Fourier række, 150  
frie variabler, 64  
Frobeniusnorm, 145

### G

Grammatrix, 98  
Gram-Schmidt  
    klassisk, 157, 163

### H

heatmap plot, 28  
Householder  
    matrix, 100  
    vektor, 100

### I

identitetsmatrix, 17, 47, 69  
imaginær del, 135  
indeksering, 26, 27  
    matrix, 25  
    negative, 27  
indgang, 25  
indre produkt, 15, 77, 143  
     $L^2$ -, 146  
    over  $\mathbb{C}$ , 144  
    vægtet, 145  
     $L^2$ -, 146  
induceret norm, 144  
integration  
    numerisk, 151  
invers  
    af produkt, 71  
invers matrix, 69, 73  
     $2 \times 2$ , 70  
    via rækkeoperationer, 75  
invertibel, 69, 73

### K

Kirchhoffs lov, 44

Kirchhoffs lov, 43  
kompleks tal, 135  
    imaginær del, 135  
    konjugerede, 138  
    numerisk værdi, 139  
    reel del, 135  
konditionstal, 123, 124  
    absolut, 123  
    for en invertibel matrix, 129  
    for en matrix, 132  
konjugerede  
    kompleks tal, 138  
kredsløb  
    elektrisk, 43  
kvadratisk matrix, 69  
kvotient  
    float, 3

### L

$L^2$ -indre produkt, 146  
Legendre polynomier, 161  
ligningssystem  
    lineær, 43  
ligningsystem  
    lineær, 70  
lineær  
    kombination, 45  
    ligningssystem, 43, 70, 73  
lineær kombination, 140  
lineært  
    uafhængig, 160  
linje  
    ret, 39

### M

machine epsilon, 6  
maksimumsnorm, 127  
mantis, 5

matplotlib, 21  
 matrix, 25  
      $(m \times n)$ -, 25  
     difference, 32  
     elementær, 74  
     Gram, 98  
     Householder, 100  
     identitets-, 17, 47, 69  
     invers, 69, 70, 73  
         via rækkeoperationer, 75  
     kvadratisk, 69  
     lighed, 25  
     nul-, 31  
     ortogonal, 95  
     produkt, se matrixprodukt  
     projektion, 85, 89  
     rotation, 95  
     søjlerum, 140  
     spejling, 95  
     sum, 31, 32  
     symmetrisk, 85  
     udvidet, 58  
 matrixprodukt, 40, 52  
     Strassen, 53  
 matrix-vektorprodukt, 45  
 multiplikation  
     med en skalar, 31–34, 133  
  
**N**  
 nøjagtighed  
     af float, 6  
 norm, 14, 126, 143  
      $\infty$ -, 127  
     1-, 126  
     2-, 79, 144  
     2- for matrix, 127  
     Frobenius-, 145  
     induceret af indre produkt, 143

    maksimums-, 127  
     operator, 127  
      $p$ -, 127  
 normalfordeling  
     standard, 49  
 nulmatrix, 31  
 numerisk værdi, 139  
  
**O**  
 Ohms lov, 44  
 operatornorm, 127  
 ortogonal  
     matrix, 95  
     vektorer, 77, 86, 143  
 ortogonale polynomier  
     Chebychev, 162  
     Legendre, 161  
 ortonormal  
     basis, 99  
     vektorer, 86  
 overflow, 7  
  
**P**  
 parallel, 80  
 Parsevals identitet, 88  
 $\pi$ , 19  
 pil, 13, 34  
 pivotelement, 62  
 plan, 39  
 plot  
     heatmap, 28  
 $p$ -norm, 127  
 polynomium, 38  
 potens  
     float, 3  
 prikprodukt, 15  
 produkt  
     float, 3

## INDEKS

- indre, 77, 143
- matrix, 40, 52
- matrix-vektor, 45
- række-søjle, 37, 51
- skalar-vektor, 51
- ydre, 48

- projektion, 89
  - matrix, 85, 89
  - på en linje, 84
- Pythagoras sætning, 79

## Q

- QR-dekomponering, 164
  - fuld, 164
  - tynd, 164

## R

- $\mathbb{R}^{m \times n}$ , 25
- RAM, 53
- reel del, 135
- relativ fejl, 4
- ret linje, 39
- række
  - af en matrix, 26
- rækkeoperationer, 58
  - i python, 60
- række-søjleprodukt, 51
- rækkevektor, 16, 35
- rotation, 95

## S

- $S(A)$ , 140
- Simpsons regel, 154
- singulærvektor, 111
  - højre-, 111
  - venstr-, 111
- singulærværdi, 107, 111
- singulærværdidekomponering, 107, 109, 111

- forkortet, 112

- tynd, 107, 111, 115, 118

- ydre produkt, 110, 111

- skalar, 25, 133

- skalarmultiplikation, se multiplikation, med en skalar

- skalar-vektorprodukt, 51

- søjle

- af en matrix, 26

- søjlerum, 140

- søjlevektor, 34

- Span, 140

- spejling, 95

- SSD, 53

- Strassen

- matrixprodukt, 53

- sum

- float, 3

- matrix, 31, 32

- vektor, 34

- SVD, se singulærværdidekomponering

- symmetrisk matrix, 85

## T

- tilfældighedsgenerator, 49

- transponering

- af produkt, 96

- matrix, 33

- vektor, 15

- trapezregel, 154

- trekantsulighed, 126

- tupel, 31

## U

- udspændt, 140

- udvidet matrix, 58

- underflow, 7

underrum, 139  
uniformfordeling, 49

## V

Vandermode  
vektor, 38  
vektor, 34, 133  
enheds-, 79  
parallelle, 80  
række, 35

søjle, 34  
sum, 34  
vektorum, 133  
vinkel, 80, 144  
vinkelret, 77  
vægtet indre produkt, 145

## Y

ydre produkt, 48