Heli Hyttinen

Diskreetti matematiikka
Lineaarialgebra

2020

Kaakkois-Suomen
ammattikorkeakoulu

SISÄLLYS

1 LINEAARISESTI RIIPPUMATON JOUKKO S

Joukko S = {a, b}, missä a = (4, 1, 0) ja b = (0, 6, 1).
s(4, 1, 0) + k(0, 6, 1) = 0 pätee vain jos skalaarit s ja k ovat nollia (0), joten joukko S on lineaarisesti riippumaton. Jos toinen kertoimista olisi muu kuin nolla, jotta s(4, 1, 0) + k(0, 6, 1) = 0 pätisi, joukko olisi lineaarisesti riippuva.

s(4, 1, 0) + k(0, 6, 1) = (x, y, z), vaikka kertoimet olisivat mitä hyvänsä niin z = k ja jos k $\neq$ 0, niin z $\neq$ 0. Sekä s = $\frac{x}{4}$, joten jos s $\neq$ 0, niin x $\neq$ 0.

Esimerkiksi jos kertoimet olisivat s = 1 ja k = 0, niin (x, y, z) = (4, 1, 0), ei (0, 0, 0).
1(4, 1, 0) + 0(0, 6, 1)
= (4 + 0, 1 + 0, 0 + 0)
= (4, 1, 0)
Joten joukko S ei ole lineaarisesti riippuva.

2 S:N VEKTOREIDEN LINEAARIKOMBINAATIOT

Vektori c = (8, 20, 3) on joukon S lineaarikombinaatio. Skalaarit voitaisi selvittää seuraavasti.

s(4, 1, 0) + k(0, 6, 1) = (8, 20, 3)
s = $\frac{8}{4}$, eli 8 jaetaan 4, todeltaan s = 2. Ja k = $\frac{20 - s}{6}$, eli 20 vähennetään 2 ja erotus 18 jaetaan kuudella, todetaan k = 3.

Lasketaan:
2(4, 1, 0) + 3(0, 6, 1)
= (8 + 0, 2 + 18, 0 + 3)
= (8, 20, 3)

Lisäksi vektori d = (20, 17, 2) on joukon S lineaarikombinaatio. Skalaarit voitaisi selvittää seuraavasti.

s(4, 1, 0) + k(0, 6, 1) = (20, 17, 2)

s = $\frac{20}{4}$, eli 20 jaetaan 4, todetaan s = 5. Ja k = $\frac{17 - s}{6}$, eli 17 vähennetään 5 ja erotus 12 jaetaan kuudella, todetaan k = 2.

Lasketaan:

5(4, 1, 0) + 2(0, 6, 1)

= (20 + 0, 5 + 12, 0 + 2)

= (20, 17, 2)

3 AB

$\boldsymbol{A} = \begin{bmatrix} 4 & 0 & -3 \\ -1 & -2 & 3 \end{bmatrix}$ ja $\boldsymbol{b} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$

Matriisi $\mathbf{A}$ on kooltaan 2x3 ja matriisi $\mathbf{b}$ on 3x1, kertomalla $\mathbf{A}$:n rivit eli (4, 0, 3) ja (-1, -2, 3) $\mathbf{b}$:n sarakkeella eli (2, -1, 3) ja laskemalla tulot yhteen riveillä, saadaan 2x1 matriisi eli Ab.

$$\mathbf{Ab} = \begin{bmatrix} 4 \cdot 2 + 0 \cdot (-1) + (-3) \cdot 3 \\ -1 \cdot 2 + (-2) \cdot (-1) + 3 \cdot 3 \end{bmatrix}$$

$$= \begin{bmatrix} 8 + 0 - 9 \\ -2 + 2 + 9 \end{bmatrix}$$

$$= \begin{bmatrix} -1 \\ 9 \end{bmatrix}$$

Eli $\mathbf{Ab} = \begin{bmatrix} -1 \\ 9 \end{bmatrix}$

3.1 Ab Pythonilla

```
In [1]: import numpy as np
A = (4,0,-3), (-1,-2,3)
b = [2], [-1], [3]
A = np.array(A)
b = np.array(b)

#tulostetaan matriisit A ja b
print("A:\n",A)
print ("b:\n",b)
```

```
A:
 [[ 4  0 -3]
 [-1 -2  3]]
b:
 [[ 2]
 [-1]
 [ 3]]
```

```
In [2]: np.dot(A,b)
#kertoo A:n rivit b:n sarakkeella ja laskee tulot yhteen
```

```
Out[2]: array([[-1],
               [ 9]])
```

Kuva 1. Kuvakaappaus.

3.2 $\mathbf{bb^T}$

Kun $\boldsymbol{b} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$, saadaan $\mathbf{b}$:n transpoosi $\mathbf{b^T}$ muuttamalla $\mathbf{b}$:n rivit sarakkeiksi. Eli $\mathbf{b}^{\mathrm{T}} = [2 \quad -1 \quad 3]$. Nyt $\mathbf{b^T}$ on kooltaan 1x3 matriisi ja $\mathbf{b}$ on 3x1 matriisi, joten $\mathbf{bb^T}$ pitäisi saada neliömatriisi 3x3. Kerrotaan $\mathbf{b}$:n rivit $\mathbf{b^T}$:n sarakkeilla seuraavasti.

$$\mathbf{bb^T} = \begin{bmatrix} 2 \cdot 2 & 2 \cdot (-1) & 2 \cdot 3 \\ -1 \cdot 2 & -1 \cdot (-1) & -1 \cdot 3 \\ 3 \cdot 2 & 3 \cdot (-1) & 3 \cdot 3 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & -2 & 6 \\ -2 & 1 & -3 \\ 6 & -3 & 9 \end{bmatrix}$$

Eli $\mathbf{bb^T} = \begin{bmatrix} 4 & -2 & 6 \\ -2 & 1 & -3 \\ 6 & -3 & 9 \end{bmatrix}$

3.3 $\mathbf{bb^T}$ Pythonilla

```
In [1]: import numpy as np
        b = [2], [-1], [3]
        print(b)
        #tässä huomasin b:n olevan sama kuin 3-tupla
        #esittämällä matriisin b näin, transpoosi onnistuu alla

([2], [-1], [3])
```

```
In [2]: bt = np.transpose(b)
        print(bt)

[[ 2 -1  3]]
```

```
In [3]: np.dot(b, bt)
        #kertoo b:n rivit bt:n sarakkeilla, saadaan 3x3 matriisi

Out[3]: array([[ 4, -2,  6],
               [-2,  1, -3],
               [ 6, -3,  9]])
```

Kuva 2. Kuvakaappaus.

4 KOLMION SKAALAUS

Kolmion kärkipisteet ovat P_1 = (1, 1, 1), P_2 = (5, 0, 2) ja P_3 = (0, 4, 4).
Skalaari on s = 4,2.
Painopiste saadaan laskemalla kärkipisteiden vektoreiden alkiot yhteen järjestyksessä ja jakamalla summa kärkipisteiden määrällä. Painopisteen **c** koordinaatit lasketaan seuraavasti.

$\mathbf{c} = \frac{1}{3}(1 + 5 + 0), \frac{1}{3}(1 + 0 + 4), \frac{1}{3}(1 + 2 + 4)$
= (2 ; 1,67; 2,33)

Eli **c** = (2 ; 1,67; 2,33)

Seuraavaksi lasketaan kärkipisteen P_1 uusi sijainti skaalaamalla vektori skaalarilla 4,2 ja vähentämällä vektorin P_1 jokaisesta alkiosta puolikas painopisteen (eli vektorin **c**) ensimmäinen alkio.

P_1 = 4,2(1, 1, 1) – 0,5**c**
= (4,2 ; 4,2 ; 4,2) – (1 ; 1 ; 1)
= (3,2 ; 3,2 ; 3,2)

Eli uusi sijainti on P_1 = (3,2 ; 3,2 ; 3,2). Lasketaan seuraavaksi kärkipisteen P_2 uusi sijainti samalla tavalla, mutta vähennetään vektorin **c** toisen alkion puolikas vektorin P_2 alkioista.

P_2 = 4,2(5, 0, 2) – 0,5**c**
= (21 ; 0 ; 8,4) – (0,83 ; 0,83; 0,83)
= (20,17 ; -0,83 ; 7,57)

Eli uusi sijainti on P_2 = (20,17 ; -0,83 ; 7,57). Lasketaan seuraavaksi kärkipisteen P_3 uusi sijainti samalla tavalla, mutta vähennetään vektorin **c** kolmannen alkion puolikas vektorin P_3 alkioista.

P_3 = 4,2(0, 4, 4) – 0,5**c**
= (0 ; 16,8 ; 16,8) – (1,17 ; 1,17 ; 1,17)
= (-1,17 ; 15,63 ; 15,63)

Eli uusi sijainti P_3 = (-1,17 ; 15,63 ; 15,63).

Kolmion uudet kärkipisteet skaalattuna luvulla 4,2 ovat

P_1 = (3,2 ; 3,2 ; 3,2),

P_2 = (20,17 ; -0,83 ; 7,57),

P_3 = (-1,17 ; 15,63 ; 15,63).

4.1 Kolmion skaalaus Pythonilla

In [1]:
```
import numpy as np

#kärkipisteet
p1=(1,1,1)
p2=(5,0,2)
p3=(0,4,4)

p1=np.array(p1)
p2=np.array(p2)
p3=np.array(p3)
print("Kärkipisteet:\np1: ",p1,"\np2: ",p2,"\np3: ",p3)

#painopisteen koordinaatit erikseen
c1=p1[:1]+p2[:1]+p3[:1]
c2=p1[1:2]+p2[1:2]+p3[1:2]
c3=p1[2:3]+p2[2:3]+p3[2:3]
c1=c1/3
c2=c2/3
c3=c3/3
print("\nPainopisteen c koordinaatit:\n",c1, c2, c3)
```

```
Kärkipisteet:
p1:  [1 1 1]
p2:  [5 0 2]
p3:  [0 4 4]

Painopisteen c koordinaatit:
 [2.] [1.66666667] [2.33333333]
```

In [2]:
```
#puolet painopisteen koordinaateista
c1=c1/2
c2=c2/2
c3=c3/2

#skaalaus
p1=4.2*p1-c1
p2=4.2*p2-c2
p3=4.2*p3-c3
print(p1, p2, p3)
```

```
[3.2 3.2 3.2] [20.16666667 -0.83333333  7.56666667] [-1.16666667 15.63333333 1
5.63333333]
```

```
In [3]: p1=np.round_(p1, decimals = 2)
        p2=np.round_(p2, decimals = 2)
        p3=np.round_(p3, decimals = 2)
        print("Kärkipisteiden uudet sijainnit:\np1:",p1,"\np2: ",p2,"\np3: ",p3)

Kärkipisteiden uudet sijainnit:
p1: [3.2 3.2 3.2]
p2:  [20.17 -0.83  7.57]
p3:  [-1.17 15.63 15.63]
```

Kuva 4. Kuvakaappaus.

5 GAUSSIN ELIMOINTIALGORITMI

Ratkaisen seuraavan yhtälöryhmän matriisien ja Gaussin elimointialgortimin avulla.

$$\begin{cases} 2x_1 - 5x_2 = 1 \\ 3x_1 + 2x_2 = 11 \end{cases}$$

Ensimmäiseksi sijoitetaan luvut matriisiin ja nimetään ensimmäinen rivi R1 ja toinen rivi R2. Tarkoituksena on eliminoida ensimmäisen rivin toinen luku ja toisen rivin ensimmäinen luku. Sekä saada vastakkaiset luvut luvuksi 1. Rivien viimeiset luvut tulevat lopulta olemaan muuttujat x_1 ja x_2, eli $\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & x_2 \end{bmatrix}$.

$$\begin{matrix} \text{R1} \\ \text{R2} \end{matrix} \begin{bmatrix} 2 & -5 & 1 \\ 3 & 2 & 11 \end{bmatrix}$$

Ensimmäisellä kierroksella kerrotaan toisen rivin ensimmäisellä luvulla ensimmäinen rivi ja toisen rivin kertojana käytetään ensimmäisen rivin ensimmäistä lukua negatiivisena.

$$\xrightarrow[\text{-2 R2} \longrightarrow \text{R2}]{\text{3 R1} \longrightarrow \text{R1}} \quad \begin{matrix} \text{R1} \\ \text{R2} \end{matrix} \begin{bmatrix} 6 & -15 & 3 \\ -6 & -4 & -22 \end{bmatrix}$$

Koska käytettiin negatiivista kertojaa toisella rivillä, saadaan toisella kierroksella toisen rivin ensimmäinen luku eliminoitua lisäämällä toiseen riviin ensimmäinen rivi.

$$\xrightarrow{\text{R1 + R2} \longrightarrow \text{R2}} \quad \begin{matrix} \text{R1} \\ \text{R2} \end{matrix} \begin{bmatrix} 6 & -15 & 3 \\ 0 & -19 & -19 \end{bmatrix}$$

Kolmannella kierroksella jaetaan toinen rivi luvulla -19 ja saadaan toisen rivin toiseksi ja kolmanneksi luvuksi 1.

$\frac{1}{-19}$ R2 ⟶ R2 ⟶ R1 $\begin{bmatrix} 6 & -15 & 3 \\ 0 & 1 & 1 \end{bmatrix}$ R2

Neljännellä kierroksella lisätään ensimmäiseen riviin toinen rivi kerrottuna luvulla 15, jotta saadaan ensimmäisen rivin toinen luku eliminoitua.

R1 + 15R2 ⟶ R1 ⟶ R1 $\begin{bmatrix} 6 & 0 & 18 \\ 0 & 1 & 1 \end{bmatrix}$ R2

Viimeisellä kierroksella jaetaan ensimmäinen rivi luvulla 6, jotta saadaan ensimmäisen rivin ensimmäinen luku luvuksi 1.

$\frac{1}{6}$ R1 ⟶ R1 ⟶ R1 $\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \end{bmatrix}$ R2

Nyt matriisin molempien rivien viimeiset luvut näyttävät muuttujat x_1 ja x_2.

Eli x_1= 3 ja x_2= 1. Kokeillaan laskea.

2(3) - 5(1)
= 6 – 5
= 1

3(3) + 2(1)
= 9 + 2
= 11

Vaikuttaa toimivan, eli:

$\begin{cases} 2x_1 - 5x_2 = 1 \\ 3x_1 + 2x_2 = 11 \end{cases}$ = $\begin{cases} 2(3) - 5(1) = 1 \\ 3(3) + 2(1) = 11. \end{cases}$

5.1 Gaussin elimointialgoritmi Pythonilla

Kuvissa 5 ja 6 on tehty sama Pythonilla numpy-kirjaston avulla. Pythonilla sijoitin rivit omiin matriiseihinsa ja kuvassa 6 ilmoitetaan muuttujien x_1 ja x_2 arvot.

In [2]:
```
import numpy as np
R1 = np.array([[2,-5,1]])
R2 = np.array([[3,2,11]])
print('R1:\n',R1,'\nR2:\n',R2)

#---ensimmäinen kierros
#3R1 on uusi R1
R1 = 3*R1
#-2R2 on uusi R2
R2 = -2*R2
print('\nensimmäinen kierros\nR1:\n',R1,'\nR2:\n',R2)

#---toinen kierros
#R1 + R2 on uusi R2
R2 = R1[0:] + R2
print('\ntoinen kierros\nR1:\n',R1,'\nR2:\n',R2)

#---kolmas kierros
#-19 / R2 on uusi R2
R2[...,1:] = R2[...,1:] /-19
print('\nkolmas kierros\nR1:\n',R1,'\nR2:\n',R2)

#---neljäs kierros
#R1 + 15R2 on uusi R1
R1 = R1 + 15*R2[0:]
print('\nneljäs kierros\nR1:\n',R1,'\nR2:\n',R2)

#---viimeinen kierros
#6 / R1 on uusi R1
R1 = R1/6
print('\nviimeinen kierros\nR1:\n',R1,'\nR2:\n',R2)
```

```
R1:
 [[ 2 -5  1]]
R2:
 [[ 3  2 11]]

ensimmäinen kierros
R1:
 [[  6 -15   3]]
R2:
 [[ -6  -4 -22]]
```

```
toinen kierros
R1:
 [[  6 -15   3]]
R2:
 [[  0 -19 -19]]

kolmas kierros
R1:
 [[  6 -15   3]]
R2:
 [[0 1 1]]

neljäs kierros
R1:
 [[ 6  0 18]]
R2:
 [[0 1 1]]

viimeinen kierros
R1:
 [[1. 0. 3.]]
R2:
 [[0 1 1]]
```

```
In [3]: x1 = R1[...,2:]
        x2 = R2[...,2:]
        print('muuttuja x1: ',x1,'\nmuuttuja x2: ',x2)
```

```
muuttuja x1:  [[3.]]
muuttuja x2:  [[1]]
```

Kuva 6. Kuvakaappaus.

6 ORTOGONAALIPROJEKTIO PIENIMMÄN NELIÖSUMMAN MENETELMÄLLÄ

$\mathbf{R}^3$:n vektorit $\boldsymbol{a_1}$ = (4,6,0) ja $\boldsymbol{a_2}$ = (0,6,3) määräävät kaksiulotteisen aliavaruuden (tason), johon määrään vektorin **b** = (8,4,3) ortogonaaliprojektion käyttämällä pienimmän neliösumman menetelmää Pythonin numpy-kirjaston avulla.

Aluksi sijoitetaan $\boldsymbol{a_1}$ ja $\boldsymbol{a_2}$ vektorit matriisiin **A**, josta saadaan 3x2 matriisi eli $m \, x \, n$ matriisi, jossa m on suurempi kuin n, eli rivejä on enemmän kuin sarakkeita.

```
In [1]: import numpy as np
        #sijoitetaan vektorit a1 ja a2 matriisiin A
        A = np.array([[4,0],[6,6],[0,3]])
        print('Matriisi A on 3x2 matriisi\n',A)
        #lineaarisesti riippumattomien sarakkeiden maksimimäärä
        print('Rangi on ',np.linalg.matrix_rank(A))

Matriisi A on 3x2 matriisi
 [[4 0]
 [6 6]
 [0 3]]
Rangi on  2
```

Kuva 7. Kuvakaappaus.

Kuvassa 7 selvitettiin myös, että **A**:n sarakkeet (n) ovat lineaarisesti riippumattomia, sillä $rank(\boldsymbol{A}) = n$. Eli rangi on sama kuin sarakkeiden määrä (2). Seuraavaksi sijoitetaan vektori **b** omaan matriisiinsa (kuva 8).

```
In [3]: #vektori b:n transpoosi, jotta saadaan 3x1 matriisi
        b = np.array([[8,4,3]]).T
        print(b)

[[8]
 [4]
 [3]]
```

Kuva 8. Kuvakaappaus.

b' = **A**x ja $\mathbf{A}^T\mathbf{A}x = \mathbf{A}^T\mathbf{b}$, jossa $x = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$, koska **A**:n sarakkeet ovat lineaarisesti riippumattomat.

Seuraavaksi (kuva 9) luodaan **A**:n transpoosin ja **A**:n tulosta neliömatriisi $\mathbf{A}^T\mathbf{A}$. josta saadaan käänteismatriisi $(\mathbf{A}^T\mathbf{A})^{-1}$.

```
In [5]: #koska A on m x n matriisi, jossa m on suurempi kuin n ja rank = n, niin
        #matriisilla AtA on käänteismatriisi
        #AtA eli A:n transpoosin ja A:n tulo
        AtA = A.T @ A
        print('AtA:\n',AtA)
```

```
AtA:
 [[52 36]
 [36 45]]
```

```
In [6]: #AtA:n käänteismatriisi iAtA
        iAtA = np.linalg.inv(AtA)
        print('iAtA:\n',iAtA)
```

```
iAtA:
 [[ 0.04310345 -0.03448276]
 [-0.03448276  0.04980843]]
```

Kuva 9. Kuvakaappaus.

Kuvassa 10 lasketaan ensin **A**:n transpoosin ja **b**:n tulo, ja sen jälkeen niiden tulon ja $\mathbf{A}^T\mathbf{A}$:n käänteismatriisin tulo eli $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$, joka on x. Jonka jälkeen voidaan laskea pienimmän neliösumman ratkaisu **b'**, eli **A**:n ja x:n tulo.

```
In [7]: x = iAtA @ ((A.T) @ b)
        #A:n ja x:n tulo eli pienimmän neliösumman ratkaisu
        b_ = A @ x
        print('pienimmän neliösumman ratkaisu on\n',b_)
```

```
pienimmän neliösumman ratkaisu on
 [[ 5.10344828]
 [ 5.93103448]
 [-0.86206897]]
```

Kuva 10. Kuvakaappaus

Ortogonaaliprojektio $\mathbf{b}'$ on $\mathbf{R}^3$:n vektori, sillä taso sijaitsee $\mathbf{R}^3$:ssa.

$$\mathbf{b}' = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{b} \approx \begin{bmatrix} 5{,}1 \\ 5{,}93 \\ -0{,}86 \end{bmatrix}$$