Numpy i prakis af Henrik Sterner (hst@nextkbh.dk)

Numpy er et bibliotek til Python, som gør det muligt at arbejde med matricer og vektorer. Det er en forudsætning for at kunne arbejde med maskinelæring, da mange af de algoritmer vi skal arbejde med er baseret på matricer og vektorer.

Numpy tilbyder også en række funktioner til matematiske, statistiske og logiske operationer på arrays.

Om brugen af disse slides

- Disse slides forsøger at eksemplifere en lang række af de vigtige begreber i Numpy
- De må på ingen måde kopieres uden tilladelse fra Henrik Sterner
- De er lavet i markdown og kan derfor nemt konverteres til andre formater
- Ved brug af Visual Studio Code kan de konverteres til HTML, PDF, PowerPoint, Word, LaTeX og mange andre formater
- Slides er tilgængelige på github.com/henriksterner/IntelligenteSystemer/

Konvertere slides til andre formater

- Fra markdown til pdf ved brug af pandoc:

```
pandoc -t beamer -V theme:metropolis -o numpy.pdf numpy.md
```

- Fra markdown til word ved brug af pandoc:

```
pandoc -o numpy.docx numpy.md
```

Hvorfor bruge NumPy?

- NumPy er meget hurtigere end at bruge Python-lister til matematiske operationer. Det skyldes at NumPy forventer, at alle elementer i en liste har samme datatype, så den behøver ikke at tjekke typen af hvert element.
- Desuden lagres NumPy-arrays i hukommelsen som en sammenhængende blok, så det er hurtigere at tilgå elementer i en NumPy-array end i en Python-liste.
- I praksis er NumPy meget hurtigere end Python-lister, når vi arbejder med store datamængder (faktor 5 til 100)
- NumPy er et standardbibliotek i Python, så det er tilgængeligt i stort set alle Python-distributioner.

Installation af Numpy i Google Colab

Numpy er en del af Google Colab, så hvis du bruger Google Colab, så har du også Numpy. Hvis du ikke bruger Google Colab, så kan du installere Numpy ved at skrive følgende i en terminal:

```
pip install numpy
```

Installation af Numpy via Anaconda

Numpy er en del af Anaconda, så hvis du har installeret Anaconda, så har du også Numpy. Hvis du ikke har installeret Anaconda, så kan du installere Numpy ved at skrive følgende i en terminal:

```
pip install numpy
```

Numpy importeres på følgende måde:

```
import numpy as np
```

Teste om Numpy er installeret

- Åbn en terminal
- Skriv python
- Skriv import numpy as np
- Skriv print(np.version.version)
- Skriv exit()
- Hvis du får en versionsnummer, så er Numpy installeret korrekt
- Hvis du får en fejlmeddelelse, så er Numpy ikke installeret korrekt
- Hvis du får en fejlmeddelelse, så prøv at geninstallere Numpy

Teste hastigheden af Numpy arrays . . .

Vi kan teste hastigheden af Numpy arrays vs. Python lister ved at udføre følgende test:

```
import numpy as np
import time

# Test hastigheden af Numpy arrays
start = time.time()
a = np.arange(1000000)
b = np.arange(1000000)
c = a + b
end = time.time()
print(end - start)
```

Test hastigheden af Python lister

Her kan vi teste hastigheden af Python lister:

```
start = time.time()
a = list(range(1000000))
b = list(range(1000000))
c = []
for i in range(len(a)):
    c.append(a[i] + b[i])
end = time.time()
print(end - start)
```

Resultater af test

Resultater prøvet lokalt på min bærbare computer:

```
0.0049865245819091 8
0.1589999198913574 2
```

Dvs. Numpy arrays er ca. 37 gange hurtigere end Python lister i dette eksempel.

Arrays i numpy

Et array er en samling af elementer, som har samme datatype. Et array kan have en eller flere dimensioner. Et array med en dimension kaldes en vektor, et array med to dimensioner kaldes en matrix.

Definition af en vektor

En vektor er en samling af tal, som er ordnet efter en bestemt rækkefølge. En vektor kan repræsenteres som en kolonne eller en række. En vektor med n elementer kan repræsenteres som en n x 1 matrix eller en 1 x n matrix.

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Her er $v_1, v_2, \ldots, v_n$ reelle tal, som kaldes vektorens komponenter.

Ved $n = 2$ og $n = 3$ kan en vektor repræsenteres som en pil, hvor længden af pilen er lig med vektorens længde og retningen af pilen er lig med vektorens retning.

Eksempler på vektorer

En vektor i 2d:

$$\vec{v} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}$$

En vektor i 3d:

$$\vec{v} = \begin{bmatrix} 65 \\ 1 \\ -3 \end{bmatrix}$$

En vektor i 4d:

$$\vec{v} = \begin{bmatrix} -5 \\ 56 \\ 7 \\ 4 \end{bmatrix}$$

Vektorer i Python

I Python kan en vektor repræsenteres som en liste. For at kunne arbejde med vektorer i Python skal vi importere numpy-biblioteket.

```
import numpy as np
```

Vi kan nu repræsentere en vektor som en liste og konvertere den til et numpy array:

```
v = [7, 4]
v = np.array(v)
w = np.array([65, 1, -3])
u = np.array([-5, 56, 7, 4])
```

Udskrive vektoren

Vi kan nu udskrive vektoren:

```
print(v)
```

```
[7 4]
```

Vi kan udskrive vektorens komponenter:

```
print(v[0])
print(v[1])
```

```
7
4
```

Længden af en vektor i 2d

Længden af en vektor i 2d kan beregnes ved hjælp af Pythagoras' læresætning:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2}$$

I Python kan vi beregne længden af en vektor ved hjælp af funktionen `linalg.norm`:

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

```
8.06225774829855
```

Længden af en vilkårlig vektor

Længden af en vektor kan beregnes ved hjælp af Pythagoras' læresætning:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

I Python kan vi beregne længden af en vektor ved hjælp af funktionen `linalg.norm`:

```
print(np.linalg.norm(v))
print(np.linalg.norm(w))
print(np.linalg.norm(u))
```

```
8.06225774829855
65.00757532283 7
58.137767414994535
```

Addition af vektorer

To vektorer kan adderes ved at addere deres komponenter:

$$\vec{v} + \vec{w} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} + \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} v_1 + w_1 \\ v_2 + w_2 \\ \vdots \\ v_n + w_n \end{bmatrix}$$

I Python kan vi addere to vektorer ved at bruge operatoren `+`:

```
print(v + w)
```

```
[72  5 -3]
```

Subtraktion af vektorer

To vektorer kan subtraheres ved at subtrahere deres komponenter:

$$\vec{v} - \vec{w} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} - \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} v_1 - w_1 \\ v_2 - w_2 \\ \vdots \\ v_n - w_n \end{bmatrix}$$

I Python kan vi subtrahere to vektorer ved at bruge operatoren `-`:

```
print(v - w)
```

```
[-58   3   7]
```

Multiplikation af en vektor med en skalar

En vektor kan multipliceres med en skalar ved at multiplicere alle dens komponenter med skalarværdien:

$$a \cdot \vec{v} = a \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} a \cdot v_1 \\ a \cdot v_2 \\ \vdots \\ a \cdot v_n \end{bmatrix}$$

I Python kan vi multiplicere en vektor med en skalar ved at bruge operatoren `*`:

```
print(2 * v)
```

```
[14  8]
```

Multiplikation af to vektorer

To vektorer kan multipliceres ved at multiplicere deres komponenter:

$$\vec{v} \cdot \vec{w} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} v_1 \cdot w_1 \\ v_2 \cdot w_2 \\ \vdots \\ v_n \cdot w_n \end{bmatrix}$$

Dette kaldes også for skalarproduktet af $\vec{v}$ og $\vec{w}$. I Python kan vi multiplicere to vektorer ved at bruge funktionen `multiply`:

```
print(np.multiply(v, w))
```

```
[455   4 -12]
```

Division af en vektor med en skalar

En vektor kan divideres med en skalar ved at dividere alle dens komponenter med skalarværdien:

$$\frac{1}{a} \cdot \vec{v} = \frac{1}{a} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \frac{1}{a} \cdot v_1 \\ \frac{1}{a} \cdot v_2 \\ \vdots \\ \frac{1}{a} \cdot v_n \end{bmatrix}$$

I Python kan vi dividere en vektor med en skalar ved at bruge operatoren `/`:

```
print(v / 2)
```

```
[3.5 2. ]
```

Krydsproduktet af to vektorer

Krydsproduktet af to vektorer er en vektor, som står vinkelret på de to vektorer. Krydsproduktet af to vektorer er givet ved:

$$\vec{v} \times \vec{w} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} v_2 \cdot w_3 - v_3 \cdot w_2 \\ v_3 \cdot w_1 - v_1 \cdot w_3 \\ v_1 \cdot w_2 - v_2 \cdot w_1 \end{bmatrix}$$

I Python kan vi beregne krydsproduktet af to vektorer ved at bruge funktionen `cross`:

```
print(np.cross(v, w))
```

```
[  7  228 -261]
```

Prikproduktet mellem to vektorer

Prikproduktet mellem to vektorer er en skalar, som er givet ved:

$$\vec{v} \cdot \vec{w} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = v_1 \cdot w_1 + v_2 \cdot w_2 + \cdots + v_n \cdot w_n$$

I Python kan vi beregne prikket mellem to vektorer ved at bruge funktionen `dot`:

```
print(np.dot(v, w))
```

```
455
```

Tolkning af prikproduktet

Vi kan tolke prikproduktet som længden af den projektion af $\vec{v}$ på $\vec{w}$, som er parallel med $\vec{w}$:

$$\vec{v} \cdot \vec{w} = \|\vec{v}\| \cdot \|\vec{w}\| \cdot \cos(\theta)$$

hvor θ er vinklen mellem $\vec{v}$ og $\vec{w}$. Ved at isolere $\cos(\theta)$ får vi:

$$\cos(\theta) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \cdot \|\vec{w}\|}$$

Hvis θ er mindre end 90°, så er $\cos(\theta)$ positiv, og hvis θ er større end 90°, så er $\cos(\theta)$ negativ. Hvis θ er lig med 90°, så er $\cos(\theta)$ lig med 0.

Eksponering af en vektor

Eksponering af en vektor er en vektor, som er givet ved:

$$\exp(\vec{v}) = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

I Python kan vi beregne eksponeringen af en vektor ved at bruge funktionen `exp`:

```
print(np.exp(v))
```

```
[1096.63315843   54.59815003]
```

Logaritmen af en vektor

Logaritmen af en vektor er en vektor, som er givet ved:

$$\log(\vec{v}) = \begin{bmatrix} \log(v_1) \\ \log(v_2) \\ \vdots \\ \log(v_n) \end{bmatrix}$$

I Python kan vi beregne logaritmen af en vektor ved at bruge funktionen `log`:

```
print(np.log(v))
```

```
[1.94591015 1.38629436]
```

Sinus af en vektor

Sinus af en vektor er en vektor, som er givet ved:

$$\sin(\vec{v}) = \begin{bmatrix} \sin(v_1) \\ \sin(v_2) \\ \vdots \\ \sin(v_n) \end{bmatrix}$$

I Python kan vi beregne sinus af en vektor ved at bruge funktionen `sin`:

```
print(np.sin(v))
```

```
[0.6569866  -0.7568025 ]
```

Cosinus af en vektor

Cosinus af en vektor er en vektor, som er givet ved:

$$\cos(\vec{v}) = \begin{bmatrix} \cos(v_1) \\ \cos(v_2) \\ \vdots \\ \cos(v_n) \end{bmatrix}$$

I Python kan vi beregne cosinus af en vektor ved at bruge funktionen `cos`:

```
print(np.cos(v))
```

```
[ 0.75390225 -0.65364362]
```

Tangens af en vektor

Tangens af en vektor er en vektor, som er givet ved:

$$\tan(\vec{v}) = \begin{bmatrix} \tan(v_1) \\ \tan(v_2) \\ \vdots \\ \tan(v_n) \end{bmatrix}$$

I Python kan vi beregne tangens af en vektor ved at bruge funktionen `tan`:

```
print(np.tan(v))
```

```
[ 0.87144798 -1.15782128]
```

Indexing af vektorer i numpy

Vi kan indexere vektorer på samme måde som vi indexere lister i Python:

```
print(v[0])
print(v[1])
```

```
7
4
```

Slicing af vektorer i numpy

Vi kan slice vektorer på samme måde som vi slicer lister i Python ved brug af `:`. Herunder en generel formel for slicing af vektorer:

```
print(v[start:stop:step])
```

- start: startindeks
- stop: stopindeks
- step: skridt

Eksempel på slicing af vektorer

Givet vektoren:

```
v = np.array([7, 4, 2, 6, 8, 9, 1, 3, 5])
```

Så kan vi slice vektoren på følgende måde:

```
print(v[0:3]) # [7 4 2]
print(v[3:6]) #  [6 8 9]
print(v[6:9]) # [1 3 5]
print(v[0:9:3]) # [7 6 1]
```

: i slicing af vektorer

Hvis vi udelader startindeks, så sættes startindeks til 0:

```
print(v[:3]) # [7 4 2]
```

Hvis vi udelader stopindeks, så sættes stopindeks til længden af vektoren:

```
print(v[3:]) # [6 8 9 1 3 5]
```

Hvis vi udelader skridt, så sættes skridt til 1:

```
print(v[0:9:1]) # [7 4 2 6 8 9 1 3 5]
print(v[0:9]) # [7 4 2 6 8 9 1 3 5]
```

Broadcasting i numpy

Numpy understøtter broadcasting, som betyder at vi kan udføre operationer på vektorer med forskellige længder. Hvis vi eksempelvis vil lægge en vektor til en skalar, så vil numpy automatisk kopiere vektoren, så den får samme længde som skalarværdien:

```
print(v + 2) # [ 9  6  4  8 10 11  3  5  7]
```

Hvis vi vil lægge to vektorer sammen, så skal de have samme længde:

```
print(v + np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])) # [ 8  6
```

Vigtige funktioner på vektorer i numpy - del I

- np.sum: summerer alle elementer i en vektor
- np.mean: beregner gennemsnittet af alle elementer i en vektor
- np.std: beregner standardafvigelsen af alle elementer i en vektor
- np.min: finder det mindste element i en vektor
- np.max: finder det største element i en vektor

Eksempel på vigtige funktioner på vektorer i numpy - del I

```
v=np.array([7, 4, 2, 6, 8, 9, 1, 3, 5])
print(np.sum(v)) # 45
print(np.mean(v)) # 5.0
print(np.std(v)) # 2.581988897471611
print(np.min(v)) # 1
print(np.max(v)) # 9
```

Vigtige funktioner på vektorer i numpy - del II

- np.argmin: finder indekset for det mindste element i en vektor
- np.argmax: finder indekset for det største element i en vektor
- np.sort: sorterer alle elementer i en vektor
- np.unique: finder alle unikke elementer i en vektor
- np.abs: finder absolutværdien af alle elementer i en vektor
- np.round: runder alle elementer i en vektor

Eksempler på vigtige funktioner på vektorer i numpy - del II

```
v=np.array([7, 4, 2, 6, 8, 9, 1, 3, 5])
print(np.argmin(v)) # 6
print(np.argmax(v)) # 5
print(np.sort(v)) # [1 2 3 4 5 6 7 8 9]
print(np.unique(v)) # [1 2 3 4 5 6 7 8 9]
print(np.abs(v)) # [7 4 2 6 8 9 1 3 5]
w = np.array([7.2, 4.4, 2.6, 6.8, 8.9, 9.1, 1.3, 3.5, 5.7])
print(np.round(w)) # [7. 4. 3. 7. 9. 9. 1. 4. 6.]
```

Viktige funktioner på vektorer i numpy - del III

- np.where: finder indekset for alle elementer i en vektor, som opfylder en bestemt betingelse
- np.concatenate: samler to vektorer til en vektor
- np.append: tilføjer et element til en vektor
- np.delete: fjerner et element fra en vektor
- np.insert: indsætter et element i en vektor
- np.copy: kopierer en vektor
- np.reshape: ændrer formen på en vektor

Eksempel på vigtige funktioner på vektorer i numpy - del III

```
v= np.array([7, 4, 2, 6, 8, 9, 1, 3, 5])
w = np.array([7.2, 4.4, 2.6, 6.8, 8.9, 9.1, 1.3, 3.5, 5.7])
print(np.where(v > 5)) # (array([0, 3, 4, 5], dtype=int64),
print(np.concatenate((v, w))) # [7.  4.  2.  6.  8.  9.  1
print(np.append(v, 10)) # [ 7  4  2  6  8  9  1  3  5 10]
print(np.delete(v, 0)) # [4 2 6 8 9 1 3 5]
print(np.insert(v, 0, 10)) # [10  7  4  2  6  8  9  1  3  5
print(np.copy(v)) # [7 4 2 6 8 9 1 3 5]
print(np.reshape(v, (3, 3))) # [[7 4 2] [6 8 9] [1 3 5]]
```

Transponering af vektorer

En vektor kan transponeres ved at ændre dens form fra en n x 1 matrix til en 1 x n matrix. Dvs. en matrix med en række og n kolonner til en matrix med n rækker og en kolonne. Herunder en matematisk definition af transponering af vektorer:

$$\vec{v}^T = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}^T = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

I Python kan vi transponere en vektor ved at bruge funktionen `transpose`:

```
print(np.transpose(v))
```

```
[7 4 2 6 8 9 1 3 5]
```

Definition af en matrix

En matrix er en samling af tal, som er ordnet i rækker og kolonner. En matrix med m rækker og n kolonner kan repræsenteres som en m x n matrix.

$$\vec{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

Indexing af matricer i numpy

Betragt 3x3 matricen, A bestående af 3 rækker og 3 kolonner med tallene 1 til 9:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Vi kan indexere matricer på samme måde som vi indexere lister i Python:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A[0, 0]) # 1
print(A[0, 1]) # 2
print(A[0, 2]) # 3
#...
print(A[2, 0]) # 7
print(A[2, 1]) # 8
print(A[2, 2]) # 9
```

Slice af matricer i numpy

Vi kan slice matricer på samme måde som vi slicer lister i Python ved brug af `:`. Herunder en generel formel for slicing af matricer:

```
print(A[start:stop:step, start:stop:step])
```

- start: startindeks
- stop: stopindeks
- step: skridt

Eksempel på slicing af matricer

Betragt 3x3 matricen, A bestående af 3 rækker og 3 kolonner med tallene 1 til 9:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A[0:2, 0:2]) # [[1 2] [4 5]]
print(A[0:2, 1:3]) # [[2 3] [5 6]]
print(A[1:3, 0:2]) # [[4 5] [7 8]]
print(A[1:3, 1:3]) # [[5 6] [8 9]]
```

: i slicing af matricer

Hvis vi udelader startindeks, så sættes startindeks til 0:

```
print(A[:2, :2]) # [[1 2] [4 5]]
```

Hvis vi udelader stopindeks, så sættes stopindeks til længden af vektoren:

```
print(A[1:, 1:]) # [[5 6] [8 9]]
```

Hvis vi udelader skridt, så sættes skridt til 1:

```
print(A[0:3:1, 0:3:1]) # [[1 2 3] [4 5 6] [7 8 9]]
print(A[0:3, 0:3]) # [[1 2 3] [4 5 6] [7 8 9]]
```

Broadcasting i numpy

Numpy understøtter broadcasting, som betyder at vi kan udføre operationer på matricer med forskellige længder. Hvis vi eksempelvis vil lægge en matrix til en skalar, så vil numpy automatisk kopiere matricen, så den får samme længde som skalarværdien:

```
print(A + 2) # [[ 3  4  5] [ 6  7  8] [ 9 10 11]]
```

Hvis vi vil lægge to matricer sammen, så skal de have samme længde:

```
print(A + np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])) # [
```

Vigtige funktioner på matricer i numpy - del I

- np.sum: summerer alle elementer i en matrix
- np.mean: beregner gennemsnittet af alle elementer i en matrix
- np.std: beregner standardafvigelsen af alle elementer i en matrix
- np.min: finder det mindste element i en matrix
- np.max: finder det største element i en matrix
- np.argmin: finder indekset for det mindste element i en matrix
- np.argmax: finder indekset for det største element i en matrix
- np.sort: sorterer alle elementer i en matrix

Vigtige funktioner på matricer i numpy - del II

- np.unique: finder alle unikke elementer i en matrix
- np.abs: finder absolutværdien af alle elementer i en matrix
- np.round: runder alle elementer i en matrix
- np.where: finder indekset for alle elementer i en matrix, som opfylder en bestemt betingelse
- np.concatenate: samler to matricer til en matrix
- np.append: tilføjer et element til en matrix
- np.delete: fjerner et element fra en matrix
- np.insert: indsætter et element i en matrix
- np.copy: kopierer en matrix
- np.reshape: ændrer formen på en matrix

np.sum på matricer

np.sum kan bruges til at beregne summen af alle elementer i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.sum(A)) # 45
print(np.sum(A, axis=0)) # [12 15 18]
print(np.sum(A, axis=1)) # [ 6 15 24]
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.mean på matricer

np.mean kan bruges til at beregne gennemsnittet af alle elementer i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.mean(A)) # 5.0
print(np.mean(A, axis=0)) # [4. 5. 6.]
print(np.mean(A, axis=1)) # [2. 5. 8.]
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.std på matricer

np.std kan bruges til at beregne standardafvigelsen af alle elementer i en matrix. Standardafvigelsen er et mål for spredningen af elementerne i en matrix. Herunder en matematisk definition af standardafvigelsen:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.std(A)) # 2.581988897471611
print(np.std(A, axis=0)) # [2.44948974 2.44948974 2.449489
print(np.std(A, axis=1)) # [0.81649658 0.81649658 0.816496
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.min på matricer

np.min kan bruges til at finde det mindste element i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.min(A)) # 1
print(np.min(A, axis=0)) # [1 2 3]
print(np.min(A, axis=1)) # [1 4 7]
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.max på matricer

np.max kan bruges til at finde det største element i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.max(A)) # 9
print(np.max(A, axis=0)) # [7 8 9]
print(np.max(A, axis=1)) # [3 6 9]
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.argmin på matricer

np.argmin kan bruges til at finde indekset for det mindste element i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.argmin(A)) # 0
print(np.argmin(A, axis=0)) # [0 0 0]
print(np.argmin(A, axis=1)) # [0 0 0]
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.argmax på matricer

np.argmax kan bruges til at finde indekset for det største element i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.argmax(A)) # 8
print(np.argmax(A, axis=0)) # [2 2 2]
print(np.argmax(A, axis=1)) # [2 2 2]
```

np.sort på matricer

np.sort kan bruges til at sortere alle elementer i en matrix:

```
A = np.array([[3, 2, 1], [6, 5, 4], [9, 8, 7]])
print(np.sort(A)) # [[1 2 3] [4 5 6] [7 8 9]]
print(np.sort(A, axis=0)) # [[3 2 1] [6 5 4] [9 8 7]]
print(np.sort(A, axis=1)) # [[1 2 3] [4 5 6] [7 8 9]]
```

axis =0 betyder at vi finder det mindste element i hver kolonne.
axis = 1 betyder at vi finder det mindste element i hver række.

np.unique på matricer

np.unique kan bruges til at finde alle unikke elementer i en matrix:

```
A = np.array([[3, 2, 1], [6, 5, 4], [9, 8, 7]])
print(np.unique(A)) # [1 2 3 4 5 6 7 8 9]
```

np.abs på matricer

np.abs kan bruges til at finde absolutværdien af alle elementer i en matrix:

```
A = np.array([[3, -2, 1], [-6, 5, -4], [9, -8, 7]])
print(np.abs(A)) # [[3 2 1] [6 5 4] [9 8 7]]
```

np.round på matricer

np.round kan bruges til at runde alle elementer i en matrix:

```
A = np.array([[3.2, -2.4, 1.6], [-6.8, 5.9, -4.1], [9.3, -8
print(np.round(A)) # [[ 3. -2.  2.] [-7.  6. -4.] [ 9. -8.
```

np.where på matricer

np.where kan bruges til at finde indekset for alle elementer i en matrix, som opfylder en bestemt betingelse:

```
A = np.array([[3, -2, 1], [-6, 5, -4], [9, -8, 7]])
print(np.where(A > 0)) # (array([0, 0, 1, 1, 2, 2], dtype=
```

np.concatenate på matricer

np.concatenate kan bruges til at samle to matricer til en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[7, 8, 9], [10, 11, 12]])
print(np.concatenate((A, B))) # [[ 1  2  3] [ 4  5  6] [ 7
```

np.append på matricer

np.append kan bruges til at tilføje et element til en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.append(A, 7)) # [1 2 3 4 5 6 7]
```

np.delete på matricer

np.delete kan bruges til at fjerne et element fra en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.delete(A, 0)) # [2 3 4 5 6]
```

np.insert på matricer

np.insert kan bruges til at indsætte et element i en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.insert(A, 0, 7)) # [7 1 2 3 4 5 6]
```

np.copy på matricer

np.copy kan bruges til at kopiere en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.copy(A)) # [[1 2 3] [4 5 6]]
```

np.reshape på matricer

np.reshape kan bruges til at ændre formen på en matrix:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.reshape(A, (3, 2))) # [[1 2] [3 4] [5 6]]
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.reshape(B, (9, 1))) # [[1] [2] [3] [4] [5] [6] [7
C = np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]]])
print(np.reshape(C, (9, 1))) # [[1] [2] [3] [4] [5] [6] [7
```

np.flatten på matricer

np.flatten kan bruges til at ændre formen på en matrix til en vektor:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.flatten(A)) # [1 2 3 4 5 6]
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.flatten(B)) # [1 2 3 4 5 6 7 8 9]
C= np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]]])
print(np.flatten(C)) # [1 2 3 4 5 6 7 8 9]
```

Transponering af matricer

En matrix kan transponeres ved at ændre dens form fra en m x n matrix til en n x m matrix. Dvs. en matrix med m rækker og n kolonner til en matrix med n rækker og m kolonner. Herunder en matematisk definition af transponering af matricer:

$$A^T = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

Eksempel på transponering af matricer

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.transpose(A)) # [[1 4] [2 5] [3 6]]
B= np.array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]]])
print(np.transpose(B)) # [[[1 4 7] [2 5 8] [3 6 9]]]
```

Matrix multiplikation

Matrix multiplikation er en operation, som kan udføres på to matricer. Hvis vi har to matricer, A og B, så kan vi udføre matrix multiplikation på følgende måde:

$$A \cdot B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots \\ c_{21} & c_{22} & \cdots \\ \vdots & \vdots & \ddots \\ c_{m1} & c_{m2} & \cdots \end{bmatrix}$$

Hvor:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Eksempel på matrix multiplikation

I vektor notation kan vi skrive matrix multiplikation på følgende måde:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 & 1 \cdot 8 + 2 \cdot 10 + 3 \cdot 12 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11 & 4 \cdot 8 + 5 \cdot 10 + 6 \cdot 12 \end{bmatrix} = \begin{bmatrix}$$

I Python kan vi udføre matrix multiplikation ved at bruge funktionen `dot`:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[7, 8], [9, 10], [11, 12]])
print(np.dot(A, B)) # [[ 58  64] [139 154]]
```

Matrix multiplikation med vektorer

Hvis vi har en matrix, A, og en vektor, v, så kan vi udføre matrix multiplikation på følgende måde:

$$A \cdot \vec{v} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

Hvor:

$$c_i = \sum_{k=1}^{n} a_{ik} \cdot v_k$$

Eksempel på matrix multiplikation med vektorer

I vektor notation kan vi skrive matrix multiplikation med vektorer på følgende måde:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 8 + 3 \cdot 9 \\ 4 \cdot 7 + 5 \cdot 8 + 6 \cdot 9 \end{bmatrix} = \begin{bmatrix} 50 \\ 122 \end{bmatrix}$$

I Python kan vi udføre matrix multiplikation med vektorer ved at bruge funktionen `dot`:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
v = np.array([7, 8, 9])
print(np.dot(A, v)) # [ 50 122]
```

Matrix multiplikation med skalarer

Hvis vi har en matrix, A, og en skalar, s, så kan vi udføre matrix multiplikation på følgende måde:

$$s \cdot A = s \cdot \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} s \cdot a_{11} & s \cdot a_{12} & \cdots & s \cdot a_{1n} \\ s \cdot a_{21} & s \cdot a_{22} & \cdots & s \cdot a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s \cdot a_{m1} & s \cdot a_{m2} & \cdots & s \cdot a_{mn} \end{bmatrix}$$

Eksempel på matrix multiplikation med skalarer

I vektor notation kan vi skrive matrix multiplikation med skalarer på følgende måde:

$$2 \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 2 & 2 \cdot 3 \\ 2 \cdot 4 & 2 \cdot 5 & 2 \cdot 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix}$$

I Python kan vi udføre matrix multiplikation med skalarer ved at bruge funktionen `dot`:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(np.dot(2, A)) # [[ 2  4  6] [ 8 10 12]]
```

Operationer på matricer

- Addition: `+`
- Subtraktion: `-`
- Multiplikation: `*`
- Division: `/`
- Potens: `**`
- Modulo: `%`
- Determinant: `linalg.det`
- Norm: `linalg.norm`