

# Foils for Chapter 2 in linear algebra and numerical methods

Øyvind Ryan

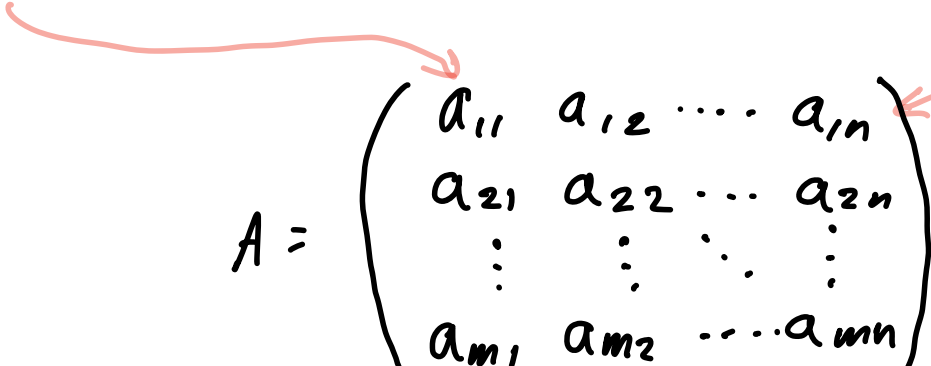
2026-08-20

# Matrices and matrix multiplication

I en matrise listes komponentene i to retninger: vertikalt  
horisontalt.

Matriser skrives med store bokstaver, komponentene i små.

$a_{ij}$  er komponenten i  $A$  fra rad  $i$  og søyle  $j$ .



angir start/slutt på matrise

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Matrisen over er  $m \times n$  ( $m$  rader,  $n$  søyler)

Alle rader har like mange komponenter, alle søyler har like  
mange komponenter.

$A$  er kvadratisk når  $m=n$ .

radvektor: Et horisontalt segment i  $A$ :  $(a_{11} \ a_{12} \ \dots \ a_{1n})$

søylevektor: — vertikalt —————

$$\begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix}$$

# Creating matrices with Python

```
A = np.array([[1,2,3]])  
B = np.array([[1],[2],[3]])
```

- ▶  $A$  is a row vector (a  $1 \times 3$ -matrix),
- ▶  $B$  a column vector (a  $3 \times 1$ -matrix).

Rows and columns will be indexed from 0 and upwards, as with vectors:

```
A[0,1]
```

2

```
B[1,0]
```

2

The  $3 \times 4$ -matrix  $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ :

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

The dimensions of  $A$ :

```
m, n = np.shape(A)  
m, n
```

(3, 4)

- ▶  $a_{i,:}$  means the row vector obtained from row  $i$  in  $A$ ,
- ▶  $a_{:,j}$  the column vector obtained from column  $j$  in  $A$ .

```
A[i,:]  
A[:,j]
```

- ▶ Square brackets indicate the part of the matrix we want.
- ▶ Colon stands for “all components”.

## Extracting a *sub-matrix* of a matrix:

To extract all components from row 1 and 3, which also are in column 1, 3, and 4:

```
A[np.ix_([0,2],[0,2,3])]
```

```
array([[ 1,  3,  4],  
       [ 9, 11, 12]])
```

Note the differences in the following two calls:

```
print(A[1,:])
```

```
[5 6 7 8]
```

```
print(A[[1],:])
```

```
[[5 6 7 8]]
```

The latter should be used if the result is to be used in matrix computations.

# The transpose

The *transpose* of the  $m \times n$ -matrix  $A$  is the  $n \times m$ -matrix with component  $(i, j)$  being  $a_{j, i}$ . We write  $A^T$  for the transpose of  $A$ .

$$\text{For } A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \text{ we get } A^T = \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix}.$$

Matrices are easily transposed as follows:

A.T

```
array([[ 1,  5,  9],  
       [ 2,  6, 10],  
       [ 3,  7, 11],  
       [ 4,  8, 12]])
```

If  $A = A^T$ ,  $A$  is called *symmetric*.

# Properties of the transpose

$$(A + B)^T = A^T + B^T$$

$$(A - B)^T = A^T - B^T$$

$$(A^T)^T = A$$

$$(sA)^T = sA^T$$

$$\begin{aligned} ((A+B)^T)_{ij} &= (A+B)_{ji} \\ &= a_{ji} + b_{ji} \end{aligned}$$

$$\begin{aligned} (A^T + B^T)_{ij} &= (A^T)_{ij} + (B^T)_{ij} \\ &= a_{ji} + b_{ji} \end{aligned}$$

# Some useful matrices

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix} \text{ er diagonal}$$

► 0: The matrix with all zeros

► 1: The matrix with all ones.

► `diag(a)`: The diagonal matrix with the vector `a` on the diagonal.

►  $I_n$ : The diagonal matrix with ones on the diagonal  
*Identitetsmatrisen*

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ diagonal, med enere p\u00e5 diagonalen}$$

These can be obtained as follows:

```
np.zeros((m,n)) # Creates an mxn matrix of zeros
np.ones((m,n))  # Creates an mxn matrix of ones
np.diag(a)      # Creates a diagonal matrix with the components
                # of a on the diagonal.
np.eye(n)       # Creates an nxn identity matrix
```

# Operations that give meaning both for vectors and matrices:

```
np.max(A,axis=0)
```

```
array([ 9, 10, 11, 12])
```

```
np.min(A,axis=0)
```

```
array([1, 2, 3, 4])
```

```
np.sum(A,axis=0)
```

```
array([15, 18, 21, 24])
```

```
np.prod(A,axis=0)
```

```
array([ 45, 120, 231, 384])
```

Difference from vector version: operations are performed column by column.

Matrix addition- and subtraction are defined component-wise.

Hvis  $A$  og  $B$  er begge  $m \times n$ , så definerer vi

$$A + B = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

$A - B$  :  $m \times n$ -matrisen med  $ij$ -komponent lik  $a_{ij} - b_{ij}$

$tA$  : matrise med komponenter  $t a_{ij}$

## Example

Compute  $3 \begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix} + 2 \begin{pmatrix} -1 & 2 \\ -2 & 1 \end{pmatrix}$ .

$$3 \begin{pmatrix} 1 & 4 \\ 2 & 1 \end{pmatrix} + 2 \begin{pmatrix} -1 & 2 \\ -2 & 1 \end{pmatrix} = \begin{pmatrix} 3 \cdot 1 & 3 \cdot 4 \\ 3 \cdot 2 & 3 \cdot 1 \end{pmatrix} + \begin{pmatrix} -2 & 4 \\ -4 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 3 & 12 \\ 6 & 3 \end{pmatrix} + \begin{pmatrix} -2 & 4 \\ -4 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 3-2 & 12+4 \\ 6-4 & 3+2 \end{pmatrix} = \begin{pmatrix} 1 & 16 \\ 2 & 5 \end{pmatrix}$$

## Vectorised functions for matrices.

`sqrt`, `abs`, `exp`, and `log`, are vectorised for matrices as well. The same is the case for trigonometric functions:

$$\sin \begin{pmatrix} 0 & \pi/2 \\ \pi & 3\pi/2 \end{pmatrix} = \begin{pmatrix} \sin 0 & \sin(\pi/2) \\ \sin \pi & \sin(3\pi/2) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix}$$

```
np.sin(np.array([[0,np.pi/2],[np.pi,3*np.pi/2]]))
```

```
array([[ 0.00000000e+00,  1.00000000e+00],  
       [ 1.2246468e-16, -1.00000000e+00]])
```

max and min are also vectorised for matrices.

```
A = np.array([[1,2,3],[4,5,6]])  
B = np.array([[4,5,6],[1,2,3]])  
np.maximum(A,B)
```

```
array([[4, 5, 6],  
       [4, 5, 6]])
```

```
np.minimum(A,B)
```

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

# Multiplication and taking powers:

$A*B$

```
array([[ 4, 10, 18],  
       [ 4, 10, 18]])
```

$A**B$

```
array([[ 1, 32, 729],  
       [ 4, 25, 216]])
```

# Special Python syntax

- ▶ If  $A$  is a matrix and  $b$  is a scalar,  $A + b$  in Python returns

$$\begin{pmatrix} a_{11} + b & a_{12} + b & \cdots & a_{1n} + b \\ a_{21} + b & a_{22} + b & \cdots & a_{2n} + b \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b & a_{m2} + b & \cdots & a_{mn} + b \end{pmatrix}$$

- ▶  $1/A$  returns the matrix with components  $1/a_{ij}$ .
- ▶ If  $A$  is a column vector ( $a$ ),  $B$  a row vector ( $b$ ),  $a + b$  returns

$$\begin{pmatrix} a_1 + b_1 & a_1 + b_2 & \cdots & a_1 + b_n \\ a_2 + b_1 & a_2 + b_2 & \cdots & a_2 + b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_m + b_1 & a_m + b_2 & \cdots & a_m + b_n \end{pmatrix},$$

i.e., the matrix with components  $(i, j)$  equal to  $a_i + b_j$ !

# The Hilbert matrix

The *Hilbert matrix*  $H_n$  is the  $n \times n$ -matrix with components

$$h_{kl} = 1/(k + l - 1),$$

where  $1 \leq k, l \leq n$ . In particular

$$H_4 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix}.$$

To obtain this Hilbert matrix in Python: First write

```
y = np.arange(1,5)  
y
```

```
array([1, 2, 3, 4])
```

to get a vector with 4 components (no interpretation in rows and columns).

To *reshape*  $y$  to a row vector (a  $1 \times 4$ -matrix) or a column vector (a  $4 \times 1$ -matrix):

```
y.reshape((1,-1))
```

```
array([[1, 2, 3, 4]])
```

```
y.reshape((-1,1))
```

```
array([[1],  
       [2],  
       [3],  
       [4]])
```

—1 here means that the number of rows/columns is calculated automatically, from the requirement that there should be only one row/column.

The Hilbert matrix is now

```
x = y.reshape((1,-1))  
1/(x + x.T - 1)
```

```
array([[1.          , 0.5          , 0.33333333, 0.25          ],  
       [0.5          , 0.33333333, 0.25          , 0.2          ],  
       [0.33333333, 0.25          , 0.2          , 0.16666667],  
       [0.25          , 0.2          , 0.16666667, 0.14285714]])
```

To reshape a row/column vector back to a vector (without interpretation in rows/columns):

```
x.reshape(-1)
```

```
array([1, 2, 3, 4])
```

# The Vandermonde matrix

The matrix

$$V = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{pmatrix}$$

is called a *Vandermonde matrix*,  $x_1, \dots, x_m$  are called its *generators*.

If we collect the generators of the vandermonde matrix in the column vector  $x$ , and the powers  $0, \dots, n$  in the row vector  $p$ , we obtain the Vandermonde matrix from

```
p = np.arange(0,n+1).reshape((1,-1));  
V = x**p
```

# The discrete Fourier transform

Matrices can also have complex components: The *DFT matrix*,  $F_N$ , is the  $N \times N$ -vandermonde matrix with generators  $\{e^{-2\pi i k/N}\}_{k=0}^{N-1}$ .

$\text{DFT}_N$  thus has components  $e^{-2\pi i k n/N}$ , where  $0 \leq k, n < N$ .

It can be obtained with the code

```
p = np.arange(0,N).reshape((1,-1))  
F = np.exp( -2*np.pi*1j*p.T*p )
```

Here there is vectorised multiplication of a column vector with a row vector in the exponent, giving us all possible values for  $kn$ . The exponential function is also vectorised, i.e.,  $e^A$  gives the matrix with components  $(k, n)$  equal to  $e^{a_{kn}}$ .

# Definition of matrix-vector multiplication

Hvis  $A$  er  $m \times n$  og  $\vec{x}$  en søylevektor med  $n$  komponenter.

Vi definerer

$$A \vec{x} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \end{pmatrix}$$

Ved definisjonen av vektoraddisjon kan dette skrives

$$x_1 \vec{a}_1 + x_2 \vec{a}_2 + \dots + x_n \vec{a}_n, \text{ der } \vec{a}_1 \text{ er første søyle i } A, \\ \vec{a}_2 \text{ er andre søyle i } A.$$

$$\text{kan også skrives } (A\vec{x})_i = a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n$$

This means that the system

$$\begin{array}{cccccc} a_{11}x_1 & +a_{12}x_2 & + \cdots & +a_{1n}x_n & = & b_1 \\ a_{21}x_1 & +a_{22}x_2 & + \cdots & +a_{2n}x_n & = & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ a_{m1}x_1 & +a_{m2}x_2 & + \cdots & +a_{mn}x_n & = & b_m \end{array} \quad (1)$$

more compactly can be written as  $A\mathbf{x} = \mathbf{b}$ .  $A$  is called the *coefficient matrix* of the system.

- ▶  $A\mathbf{x}$  is defined if and only if the number of columns in  $A$  equals the number of components in  $\mathbf{x}$ ,
- ▶  $A\mathbf{x}$  then can be interpreted as  $m$  scalar products (between the rows in  $A$ , and  $\mathbf{x}$ ).

## Example

Let us compute  $A\mathbf{x}$  when  $A = \begin{pmatrix} 2 & 1 & -2 \\ 3 & 0 & -1 \\ 7 & -4 & 2 \end{pmatrix}$  and  $\mathbf{x} = \begin{pmatrix} 3 \\ -1 \\ -2 \end{pmatrix}$

(well-defined since the number of columns in  $A$  equals the number of components in  $\mathbf{x}$ ).

$$\begin{aligned} A\vec{x} &= \begin{pmatrix} 2 & 1 & -2 \\ 3 & 0 & -1 \\ 7 & -4 & 2 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ -2 \end{pmatrix} = \begin{pmatrix} 2 \cdot 3 + 1 \cdot (-1) + (-2) \cdot (-2) \\ 3 \cdot 3 + 0 \cdot (-1) + (-1) \cdot (-2) \\ 7 \cdot 3 + (-4) \cdot (-1) + 2 \cdot (-2) \end{pmatrix} \\ &= \begin{pmatrix} 6 - 1 + 4 \\ 9 + 0 + 2 \\ 21 + 4 - 4 \end{pmatrix} = \begin{pmatrix} 9 \\ 11 \\ 21 \end{pmatrix} \end{aligned}$$

# Definition of matrix-matrix multiplication

We want to solve Equation 1 for different right hand sides  $\mathbf{b}_1, \dots, \mathbf{b}_k$ :

$$A\mathbf{x}_1 = \mathbf{b}_1 \quad A\mathbf{x}_2 = \mathbf{b}_2 \quad \dots \quad A\mathbf{x}_k = \mathbf{b}_k$$

We seek a compact notation also for the collection of these systems.

If  $A$  is an  $m \times n$  matrix, and  $X$  an  $n \times k$ -matrix, we define  $AX$  as the  $m \times k$ -matrix

$$AX = (AX_{:,1} \quad AX_{:,2} \quad \dots \quad AX_{:,k})$$

Column  $i$  in the matrix equation  $AX = B$  is thus exactly the system  $A\mathbf{x}_i = \mathbf{b}_i$ .

# Inner form of a matrix product

From the definition of the matrix-vector product:

$$(AX)_{ij} = (AX_{:,j})_i = \sum_{r=1}^n a_{ir} x_{rj}. \quad (2)$$

which is how we normally compute the product of two matrices.

# Example

Compute the matrix product

$$\begin{pmatrix} 2 & 4 & 3 \\ 1 & -1 & 3 \end{pmatrix} \begin{pmatrix} 2 & -1 \\ 4 & 1 \\ 2 & -1 \end{pmatrix} = \begin{pmatrix} 2 \cdot 2 + 4 \cdot 4 + 3 \cdot 2 & 2 \cdot (-1) + 4 \cdot 1 + 3 \cdot (-1) \\ 1 \cdot 2 + (-1) \cdot 4 + 3 \cdot 2 & 1 \cdot (-1) + (-1) \cdot 1 + 3 \cdot (-1) \end{pmatrix}$$
$$= \begin{pmatrix} 4 + 16 + 6 & -2 + 4 - 3 \\ 2 - 4 + 6 & -1 - 1 - 3 \end{pmatrix} = \begin{pmatrix} 26 & -1 \\ 4 & -5 \end{pmatrix}$$

# Verification with Python

```
A = np.array([[2,4,3],[1,-1,3]])  
B = np.array([[2,-1],[4,1],[2,-1]])  
A @ B
```

```
array([[26, -1],  
       [ 4, -5]])
```

## Interpretation in terms of the scalar product:

- ▶ Let  $A$  have rows  $\mathbf{a}_1$  and  $\mathbf{a}_2$ ,
- ▶  $B$  have columns  $\mathbf{b}_1$  and  $\mathbf{b}_2$ .

We compute

$$\mathbf{a}_1 \cdot \mathbf{b}_1: (2, 4, 3) \cdot (2, 4, 2) = 2 \cdot 2 + 4 \cdot 4 + 3 \cdot 2 = 4 + 16 + 6 = 26$$

$$\mathbf{a}_1 \cdot \mathbf{b}_2: (2, 4, 3) \cdot (-1, 1, -1) = 2 \cdot (-1) + 4 \cdot 1 + 3 \cdot (-1) = -2 + 4 - 3 = -1$$

$$\mathbf{a}_2 \cdot \mathbf{b}_1: (1, -1, 3) \cdot (2, 4, 2) = 1 \cdot 2 + (-1) \cdot 4 + 3 \cdot 2 = 2 - 4 + 6 = 4$$

$$\mathbf{a}_2 \cdot \mathbf{b}_2: (1, -1, 3) \cdot (-1, 1, -1) = 1 \cdot (-1) + (-1) \cdot 1 + 3 \cdot (-1) = -1 - 1 - 3 = -5.$$

# The outer product

Let  $\mathbf{a} \in \mathbb{R}^m$  and  $\mathbf{x} \in \mathbb{R}^n$  be column vectors. We have that

$$\mathbf{a}\mathbf{x}^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} (x_1 \quad x_2 \quad \cdots \quad x_n) = \begin{pmatrix} a_1 x_1 & a_1 x_2 & \cdots & a_1 x_n \\ a_2 x_1 & a_2 x_2 & \cdots & a_2 x_n \\ \vdots & \vdots & \ddots & \vdots \\ a_m x_1 & a_m x_2 & \cdots & a_m x_n \end{pmatrix}$$

$\mathbf{a}\mathbf{x}^T$  is called the *outer product* of  $\mathbf{a}$  and  $\mathbf{x}$ .

# Properties of matrix multiplication

Let  $A$ ,  $B$ , and  $C$  be matrices. The following hold

1.  $(AB)C = A(BC)$
2.  $A(B + C) = AB + AC$
3.  $(B + C)A = BA + CA$
4. For all scalars  $s$  we have that  $s(AB) = (sA)B = A(sB)$ .
5. If  $A$  is  $m \times n$  then  $AI_n = A$  and  $I_m A = A$

## Some observations

- ▶ If we multiply many matrices, 1. here implies that it does not matter which ‘neighbouring matrices’ we multiply first.
- ▶ It therefore does not matter where we set the parenthesis in matrix multiplication.
- ▶ Therefore, the parenthesis are usually omitted, i.e., one writes  $ABC$  instead of  $(AB)C = A(BC)$ .
- ▶ We can, however, not change the order of the matrices:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$
$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}.$$

## Proofs:

1. La  $A$  være  $m \times k$ ,  $B$   $k \times l$ ,  $C$   $l \times n$

$$\begin{aligned} ((AB)C)_{ij} &= \sum_{s=1}^l (AB)_{is} C_{sj} = \sum_{s=1}^l \left( \sum_{r=1}^k a_{ir} b_{rs} \right) C_{sj} \\ &= \sum_{s=1}^l \sum_{r=1}^k a_{ir} b_{rs} C_{sj} \end{aligned}$$

$$(A(BC))_{ij} = \sum_{r=1}^k a_{ir} (BC)_{rj} = \sum_{r=1}^k a_{ir} \left( \sum_{s=1}^l b_{rs} C_{sj} \right) = \sum_{r=1}^k \sum_{s=1}^l a_{ir} b_{rs} C_{sj}$$

Derfor har  $(AB)C$  og  $A(BC)$  like komponenter, slik at

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

Betis for 5: Skal vise at  $AI_n = A$  ( $A$   $m \times n$ )

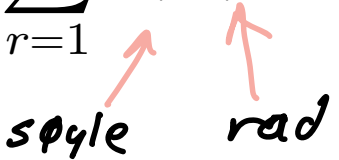
$$(AI_n)_{ij} = \sum_{r=1}^n a_{ir} (I_n)_{rj} \stackrel{\text{ikke kun når } r=j}{=} a_{ij} (I_n)_{jj} = a_{ij} \cdot 1 = a_{ij}$$

Derfor har  $AI_n$  og  $A$  like komponenter, slik at  $AI_n = A$ .

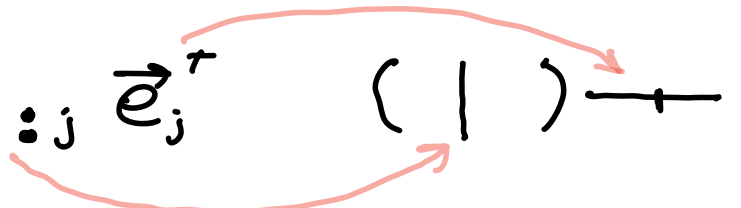
# Outer form of matrix multiplication

We use the notation  $\mathbf{a}_{:,r}$  for column  $r$  in  $A$ , and  $\mathbf{x}_{r,:}$  for row  $r$  in  $X$ :  
We have that

$$AX = \sum_{r=1}^k \mathbf{a}_{:,r} \mathbf{x}_{r,:}$$

 | — = ( )

Bevis: La  $\vec{e}_j$  være søylevektoren med en ener i komponent  $j$ , og nuller ellers.

Sett  $A = \sum_{j=1}^k \mathbf{a}_{:,j} \vec{e}_j^T$    $( \mid ) \rightarrow$  trekker ut matrise  $A$  i søyle  $j$

$$X = \sum_{i=1}^k \vec{e}_i x_{i:} \quad \Bigg| \quad \left( \text{---} \right)$$

trekker ut matrisen  $X$   
i rad  $i$ .

$$AX = \left( \sum_{j=1}^k a_{:j} \vec{e}_j^T \right) \left( \sum_{i=1}^k \vec{e}_i x_{i:} \right)$$

$$= \sum_{i=1}^k \sum_{j=1}^k a_{:j} \underbrace{\vec{e}_j^T \vec{e}_i}_{\delta_{ij}} x_{i:}$$

$$= \sum_{i=1}^k a_{:i} x_{i:}$$

■

$$\vec{e}_j \cdot (0, \dots, 0, 1, 0, \dots, 0) = \begin{pmatrix} 0 \\ \vdots \\ \delta_{ij} \\ \vdots \\ 0 \end{pmatrix}$$

$$= 0 \text{ hvis } i \neq j \\ = 1 \text{ hvis } i = j$$

Proof:

# Linear transformations and matrix multiplication

$\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is called *linear* if

1.  $\mathbf{F}(\mathbf{a}_1 + \mathbf{a}_2) = \mathbf{F}(\mathbf{a}_1) + \mathbf{F}(\mathbf{a}_2)$  for all  $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{R}^n$ ,
2.  $\mathbf{F}(c\mathbf{a}) = c\mathbf{F}(\mathbf{a})$  for all  $\mathbf{a} \in \mathbb{R}^n$ , scalars  $c$

## Proposition

$\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is linear if and only if  $\mathbf{F}(\mathbf{x}) = A\mathbf{x}$  for all  $\mathbf{x} \in \mathbb{R}^n$  for some  $m \times n$  matrix  $A$ . More precisely,

$$A = (\mathbf{F}(\mathbf{e}_1) \quad \mathbf{F}(\mathbf{e}_2) \quad \cdots \quad \mathbf{F}(\mathbf{e}_n)),$$

where  $\mathbf{e}_i$  was the column vector with 1 in component  $i$ , and 0 otherwise.

*A kalles for matrisen til  $\vec{F}$*

Proof: Vi har at  $\vec{x} = (x_1, \dots, x_n) = x_1 \vec{e}_1 + x_2 \vec{e}_2 + \dots + x_n \vec{e}_n$

$$\begin{aligned}\vec{F}(\vec{x}) &= \vec{F}(x_1, \dots, x_n) = \vec{F}(x_1 \vec{e}_1 + x_2 \vec{e}_2 + \dots + x_n \vec{e}_n) \\ &= \vec{F}(x_1 \vec{e}_1) + \vec{F}(x_2 \vec{e}_2) + \dots + \vec{F}(x_n \vec{e}_n) \\ &= x_1 \vec{F}(\vec{e}_1) + x_2 \vec{F}(\vec{e}_2) + \dots + x_n \vec{F}(\vec{e}_n)\end{aligned}$$

det. er matrise-vektor produktet

$$= \underbrace{(\vec{F}(\vec{e}_1) \quad \vec{F}(\vec{e}_2) \quad \dots \quad \vec{F}(\vec{e}_n))}_{\text{matrisen til } \vec{F}} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$$\vec{F}(\vec{x}) = A\vec{x}, \text{ med } A = (\vec{F}(\vec{e}_1) \quad \vec{F}(\vec{e}_2) \quad \dots \quad \vec{F}(\vec{e}_n))$$

Må også vise at  $\vec{F}(\vec{x}) = A\vec{x}$  er lineær:

$$\vec{F}(\vec{x}_1 + \vec{x}_2) = A(\vec{x}_1 + \vec{x}_2) = A\vec{x}_1 + A\vec{x}_2 = \vec{F}(\vec{x}_1) + \vec{F}(\vec{x}_2)$$

$$\vec{F}(c\vec{x}) = A(c\vec{x}) = c(A\vec{x}) = c\vec{F}(\vec{x})$$

# The matrix of the projection operator

The projection operator  $\text{proj}_{\vec{b}}(\vec{a}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{b}|^2} \vec{b}$  is linear in  $\vec{a}$  (follows from properties of the scalar product). We compute the matrix of  $\text{proj}_{\vec{b}}$ :

$$\text{proj}_{\vec{b}} \vec{e}_k = \frac{\vec{e}_k \cdot \vec{b}}{|\vec{b}|^2} = \frac{b_k}{|\vec{b}|^2} \vec{b} = b_k \vec{b} \quad \text{falls } |\vec{b}|=1$$

$$\begin{aligned} A &= (\text{proj}_{\vec{b}} \vec{e}_1 \quad \text{proj}_{\vec{b}} \vec{e}_2 \quad \dots \quad \text{proj}_{\vec{b}} \vec{e}_n) \\ &= (b_1 \vec{b} \quad b_2 \vec{b} \quad \dots \quad b_n \vec{b}) = \vec{b} \underbrace{(b_1 \ b_2 \ \dots \ b_n)}_{\text{radvektor}} = \underbrace{\vec{b} \vec{b}^T}_{\text{kalles projeksjonsmatrise}} \end{aligned}$$

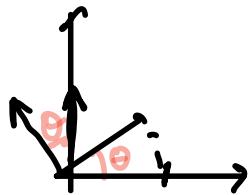
$$\text{Mer generell: } \text{proj}_{\vec{b}} \vec{a} = \text{proj}_{\frac{\vec{b}}{|\vec{b}|}} \vec{a}$$

# The matrix of a rotation

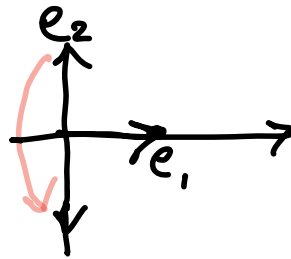
Consider the mapping  $\mathbf{F}$  which rotates a vector in  $\mathbb{R}^2$  with angle  $\theta$ .

We compute its matrix:

$$\vec{e}_1 \rightarrow \begin{pmatrix} \cos \theta \\ \sin \theta \end{pmatrix}$$



Speiling om x-aksen :



matrisen blir:  $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

$$\text{Dermed: } A = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$$\vec{e}_1 \rightarrow \vec{e}_1, \vec{e}_2 \rightarrow -\vec{e}_2$$
$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

$$\vec{e}_2 \rightarrow \begin{pmatrix} \cos(\theta + \frac{\pi}{2}) \\ \sin(\theta + \frac{\pi}{2}) \end{pmatrix} = \begin{pmatrix} -\sin \theta \\ \cos \theta \end{pmatrix}$$

# Plotting polynomials

- ▶ Matrix multiplication can be used naturally to plot polynomials.
- ▶ The function `plt.plot` plots curves in the plane.
- ▶ Common syntax for `plt.plot` takes arguments `x` and `y`, containing the  $x$ - and  $y$ -coordinates of the plot points.

Let us plot the polynomials

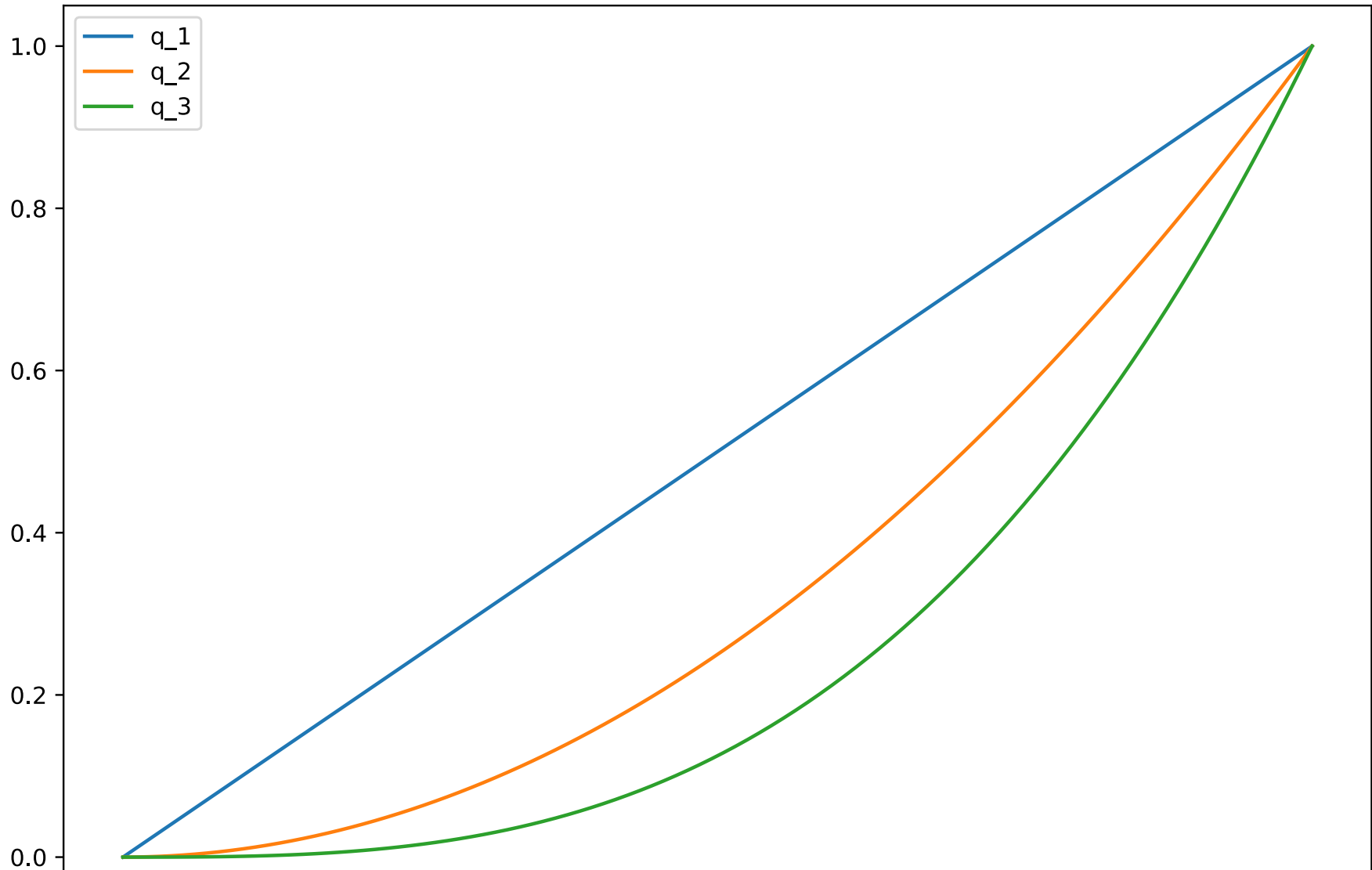
$$q_1(x) = x$$

$$q_2(x) = x^2$$

$$q_3(x) = x^3$$

together, using  $m$  points in the interval  $[0, 1]$ :

```
x = np.linspace(0,1,m).reshape((-1,1))  
plt.plot(x, x)  
plt.plot(x, x**2)  
plt.plot(x, x**3)  
plt.legend(['q_1', 'q_2', 'q_3'])
```



The `plt.plot` commands above can also be assembled into one call:

```
plt.plot(x, x, x, x**2, x, x**3)
```

# Stacking the polynomial coefficients in a vector:

Let the first polynomial be

$$q_1(x) = c_0 + c_1x + \cdots + c_nx^n,$$

and plot points  $x_1, \dots, x_m$ . We write

$$\begin{aligned} q_1(x_1) &= c_0 + x_1c_1 + \cdots + x_1^n c_n \\ q_1(x_2) &= c_0 + x_2c_1 + \cdots + x_2^n c_n \\ &\vdots \\ q_1(x_m) &= c_0 + x_mc_1 + \cdots + x_m^n c_n. \end{aligned}$$

By the definition of matrix-vector-multiplication:

$$\begin{pmatrix} q_1(x_1) \\ q_1(x_2) \\ \vdots \\ q_1(x_m) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}, \quad (3)$$

where we recognise a Vandermonde matrix.

Add another polynomial,  $q_2(x) = d_0 + d_1x + \cdots + d_nx^n$ :

$$\begin{pmatrix} q_2(x_1) \\ q_2(x_2) \\ \vdots \\ q_2(x_m) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \end{pmatrix}. \quad (4)$$

By the definition of matrix multiplication this can be assembled into

$$\begin{pmatrix} q_1(x_1) & q_2(x_1) \\ q_1(x_2) & q_2(x_2) \\ \vdots & \vdots \\ q_1(x_m) & q_2(x_m) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{pmatrix} \begin{pmatrix} c_0 & d_0 \\ c_1 & d_1 \\ \vdots & \vdots \\ c_n & d_n \end{pmatrix}, \quad (5)$$

and in the same way if we have even more polynomials.

Computing the Vandermonde matrix:

```
p = np.arange(0,n+1).reshape((1,-1))  
V = x**p
```

We collect the coefficients of the three polynomials  $q_1$ ,  $q_2$ , and  $q_3$ , in column vectors q1, q2, and q3, and compute Equation 5:

```
q1 = np.array([[0],[1],[0],[0]])  
q2 = np.array([[0],[0],[1],[0]])  
q3 = np.array([[0],[0],[0],[1]])  
Y = V @ np.block([q1,q2,q3])
```

This gives a matrix with three columns, where

- ▶ column 1 contains values for  $q_1$ ,
- ▶ column 2 for  $q_2$ ,
- ▶ column 3 for  $q_3$ .

# Plotting the polynomials

```
plt.plot(x, Y)  
plt.legend(['q_1', 'q_2', 'q_3'])
```

This is a new syntax for the `plt.plot`-function.

# Block matrices

If we have four matrices  $A$ ,  $B$ ,  $C$ ,  $D$ , we write

$$\left( \begin{array}{c|c} A & B \\ \hline C & D \end{array} \right)$$

for the matrix where the components in these four matrices are placed over, under, to the left, or to the right of each other as indicated. Lines are used to separate the blocks. We call this a *block matrix*.

What requirements are there on the matrices used to assemble a block matrix?

Creating a block matrix:

```
np.block([[A,B],[C,D]])
```

## Example

Let

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \quad C = \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix} \quad D = \begin{pmatrix} 4 & 4 \\ 4 & 4 \\ 4 & 4 \end{pmatrix}$$

What is then  $\left( \begin{array}{c|c} A & B \\ \hline C & D \end{array} \right)$ ?

# Verification

```
A = np.array([[1,1,1],[1,1,1]])
B = np.array([[2,2],[2,2]])
C = np.array([[3,3,3],[3,3,3],[3,3,3]])
D = np.array([[4,4],[4,4],[4,4]])
np.block([[A,B],[C,D]])
```

```
array([[1, 1, 1, 2, 2],
       [1, 1, 1, 2, 2],
       [3, 3, 3, 4, 4],
       [3, 3, 3, 4, 4],
       [3, 3, 3, 4, 4]])
```

What are the requirements for the product

$$\left( \begin{array}{c|c} A & B \\ \hline C & D \end{array} \right) \left( \begin{array}{c|c} E & F \\ \hline G & H \end{array} \right) \quad (6)$$

of two block matrices to give meaning?

Can be computed as

```
np.block([[A,B],[C,D]]) @ np.block([[E,F],[G,H]])
```

Can also be computed with ordinary matrix multiplication:

$$\left( \begin{array}{c|c} AE + BG & AF + BH \\ \hline CE + DG & CF + DH \end{array} \right)$$

```
np.block([[A@E + B@G, A@F + B@H], [C@E + D@G, C@F + D@H]])
```