

# Mathematics for Data Science

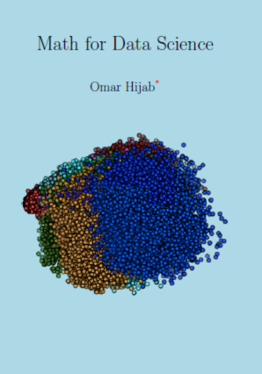
Dr. S. M. Moosavi

*smohsenmoosavi2009@gmail.com*

September 16, 2024



The following slides are arranged (with some modifications) based on the book "*Math for Data Science*" by "**Omar Hijab**".




You can follow me on [Linkedin](). Also, for course materials such as slides and the related python codes, see this [Github]() repository.



# Outline



# Outline





# What is a dataset?

**Definition 1.1**

*Geometrically, a dataset is a sample of $N$ points $x_1, x_2, \cdots, x_N$ in $d$-dimensional space $\mathbb{R}^d$. Algebraically, a dataset is an N $\times$ d matrix.*

Practically speaking, the following are all representations of datasets:

$$\text{matrix} = \text{CSV file} = \text{spreadsheet} = \text{SQL table} = \text{array} = \text{dataframe}$$

**Definition 1.2**

*Each point $x = (t_1, t_2, \cdots, t_d)$ in the dataset is a sample or an example, and the components $t_1, t_2, \cdots, t_d$ of a sample point $x$ are its features or attributes. As such, $d$-dimensional space $\mathbb{R}^d$ is feature space.*

**Definition 1.3**

*Sometimes one of the features is separated out as the label. In this case, the dataset is a labelled dataset.*



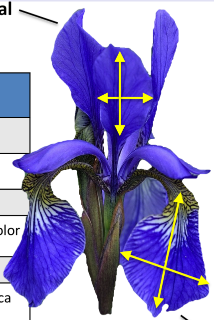


# Iris dataset

The *Iris dataset* contains 150 examples of four features of Iris flowers, and there are three classes of Irises, *Setosa*, *Versicolor* and *Virginica*, with 50 samples from each class.

**Samples**
(instances, observations)

| | Sepal length | Sepal width | Petal length | Petal width | Class label |
|---|---|---|---|---|---|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | Setosa |
| … | | | | | |
| 50 | 6.4 | 3.5 | 4.5 | 1.2 | Versicolor |
| … | | | | | |
| 150 | 5.9 | 3.0 | 5.0 | 1.8 | Virginica |




**Features**
(attributes, measurements, dimensions)

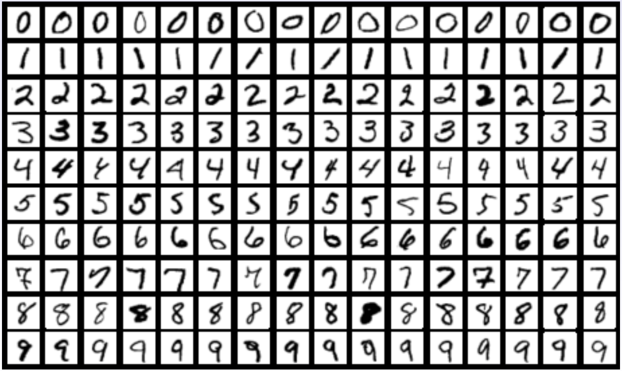
**Class labels**
(targets)



# MNIST dataset

The *MNIST dataset* consists of $60,000$ images of hand-written digits. There are 10 classes of images, corresponding to each digit $0, 1, \cdots, 9$. We seek to compress the images while preserving as much as possible of the images' characteristics.

Each image is a grayscale $28 \times 28$ pixel image. Since $28^2 = 784$, each image is a point in $d = 784$ dimensions. Here there are $N = 60000$ samples and $d = 784$ features.





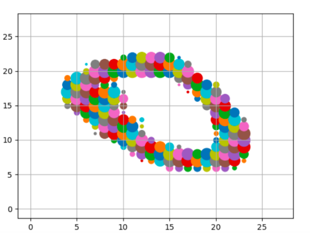
# Exercises

## Exercise 1.1

*Use* `sklearn` *to download Iris dataset.*

## Exercise 1.2

- *From* `keras` *read the MNIST dataset.*
- *Let* `(train_X, train_y), (test_X, test_y) = mnist.load_data()`
- *Let* `pixels = train_X[1]`.
- *Do* `for` *loops over* `i` *and* `j` *in* `range(28)` *and use* `scatter` *to plot points at location* `(i,j)` *with size given by* `pixels[i,j]`, *then show the following image.*





# Introduction

Suppose we have a population of things (people, tables, numbers, vectors, images, etc.) and we have a sample of size $N$ from this population:

```
l = [x_1,x_2,...,x_N]
```

The total population is the *population* or the *sample space*.

## Example 1.1

The sample space consists of all real numbers and we take $N = 5$ samples from

```
l = [3.95, 3.20, 3.10, 5.55, 6.93]
```

## Example 1.2

The sample space consists of all integers and we take $N = 5$ samples from

```
l = [35, -32, -8, 45, -8]
```



# Introduction

## Example 1.3

The sample space consists of all Python strings and we take $N = 5$ samples from

```
l = ['a2e?','#%T','7y5,','kkk>><</','[[)*+']
```

## Example 1.4

The sample space consists of all HTML colors and we take $N = 5$ samples from

```
from random import choice
import matplotlib.pyplot as plt

def hexcolor():
  return "#" + ''.join([choice('0123456789abcdef') for
                                        _ in range(6)])

for i in range(5): plt.scatter(i,0, c=hexcolor())
plt.show()
```





# Mean

Let `l` be a list as above. The goal is to compute the sample *average* or *mean* of the list, which is

$$mean = average = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

In the Example (1.1), the average is

$$\frac{3.95 + 3.20 + 3.10 + 5.55 + 6.93}{5} = 4.546.$$

## Example 1.5

```
import numpy as np

dataset = np.array([3.95, 3.20, 3.10, 5.55, 6.93])
print(np.mean(dataset))

output: 4.546
```

In the Example (1.2), the average is $\frac{32}{5}$. In the Example (1.3), while we can add strings, we can't divide them by 5, so the average is undefined. Similarly for colors: the average is undefined.



# Vector space

A sample space or population $V$ is called a *vector space* if, roughly speaking, one can compute means or averages in $V$ . In this case, we call the members of the population "*vectors*".

## Definition 1.4 (Vector space)

*Let $V$ be a set. $V$ is a vector space (over $\mathbb{R}$) if for every $u, v, w \in V$ and $r, s \in \mathbb{R}$:*

1. *vectors can be added (and the sum $v + w$ is back in $V$);*
2. *vector addition is commutative $v + w = w + v$*
3. *vector addition is associative $u + (v + w) = (u + v) + w$;*
4. *there is a zero vector $\mathbf{0}$ ($\mathbf{0} + v = v$);*
5. *vectors $v$ have negatives (or opposites) $-v$ ($v + (-v) = \mathbf{0}$);*
6. *vectors can be multiplied by real numbers (and the product $rv$ is back in $V$);*
7. *multiplication is distributive over addition $(r + s)v = rv + sv$ and $r(u + v) = ru + rv$;*
8. *$1v = v$ and $0v = \mathbf{0}$;*
9. *$r(sv) = (rs)v$.*



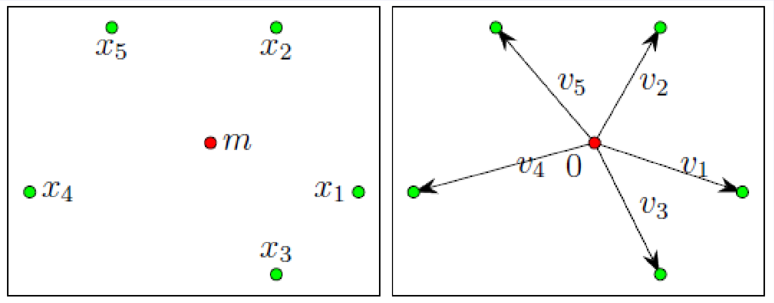
# Centered dataset

## Definition 1.5 (Centered Versus Non-Centered)

*If* $x_1, x_2, \cdots, x_N$ *is a dataset of points with mean* $m$ *and*

$$v_1 = x_1 - m, v_2 = x_2 - m, \cdots, v_N = x_N - m,$$

*then* $v_1, v_2, \cdots, v_N$ *is a centered dataset of vectors where its mean is zero.*

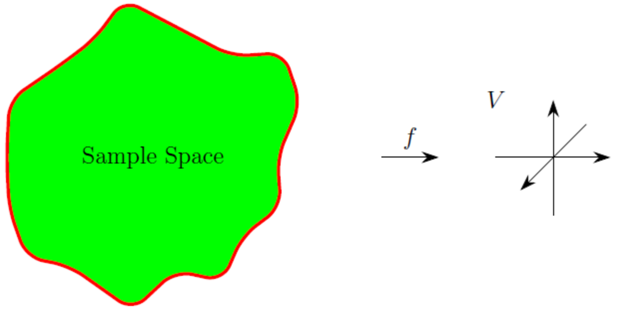




# Some notes

- When we work with vector spaces, numbers are referred to as *scalars*.
- When we multiply a vector $v$ by a scalar $r$ to get the scaled vector $rv$, we call it *scalar multiplication*.
- The set of all real numbers $\mathbb{R}$ is a vector space.
- The set of all integers $\mathbb{Z}$ is not a vector space.
- The set of all rational numbers $\mathbb{Q}$ is a vector space over $\mathbb{Q}$ but not over $\mathbb{R}$.
- The set of all Python strings is not a vector space.
- Usually, we can't take sample means from a population, we instead take the sample mean of a *statistic* associated to the population. A statistic is an assignment of a number $f(\text{item})$ to each item in the population. For example, the human population on Earth is not a vector space (they can't be added), but their heights is a vector space (heights can be added). For the Python strings, a statistic might be the length of the strings. For the HTML colors, a statistic is the HTML code of the color.

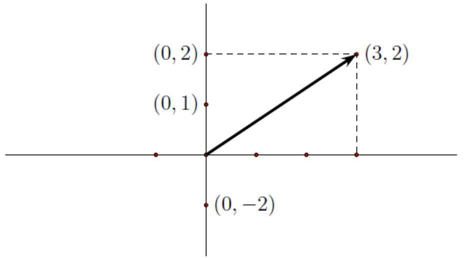
# Statistic




In general, a statistic need not be a number. A statistic can be anything that "behaves like a number". For example, $f(\text{item})$ can be a vector or a matrix. More generally, a statistic's values may be anything that lives in a vector space $V$.

# Cartesian plane

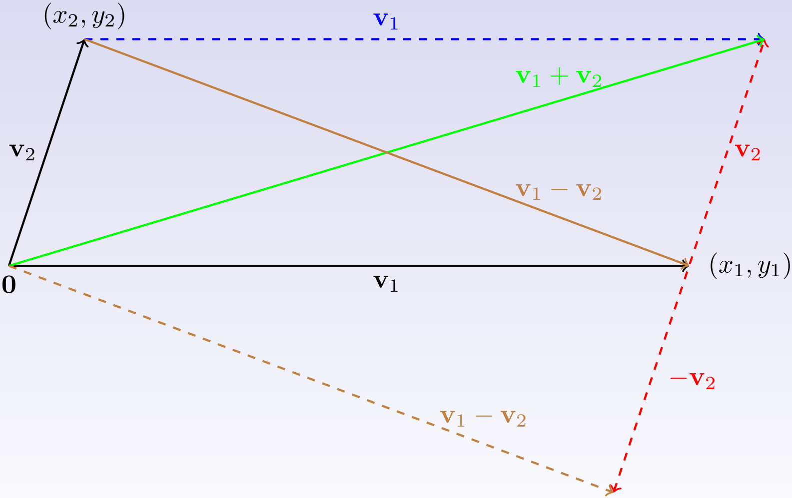
The *cartesian plane* $\mathbb{R}^2$, also called the 2-dimensional real space is a vector space.




For $\mathbf{v}_1 = (x_1, y_1), \mathbf{v}_2 = (x_2, y_2) \in \mathbb{R}^2$ and $t \in \mathbb{R}$ define

- $\mathbf{v}_1 + \mathbf{v}_2 = (x_1 + x_2, y_1 + y_2)$ (Addition).
- $\mathbf{0} = (0, 0)$ (Zero).
- $t\mathbf{v}_1 = (tx_1, ty_1)$ (Scaling).
- $-\mathbf{v}_1 = (-1)\mathbf{v}_1$ (Negative).
- $\mathbf{v}_1 - \mathbf{v}_2 = \mathbf{v}_1 + (-\mathbf{v}_2) = (x_1 - x_2, y_1 - y_2)$ (Subtraction).

# Operations


$(x_2, y_2)$
$\mathbf{v}_1$
$\mathbf{v}_1 + \mathbf{v}_2$
$\mathbf{v}_2$
$\mathbf{v}_2$
$\mathbf{v}_1 - \mathbf{v}_2$
$\mathbf{0}$
$\mathbf{v}_1$
$(x_1, y_1)$
$-\mathbf{v}_2$
$\mathbf{v}_1 - \mathbf{v}_2$



# 2d example

## Example 1.6

```
import numpy as np

v1 = (1,2)
v2 = (3,4)
print(v1 + v2 == (1+3,2+4)) # returns False

v1 = [1,2]
v2 = [3,4]
print(v1 + v2 == [1+3,2+4]) # returns False

v1 = np.array([1,2])
v2 = np.array([3,4])
print(v1 + v2 == np.array([1+3,2+4]))
# returns [ True  True]
print(3*v1 == np.array([3,6]))
# returns [ True  True]
print(-v1 == np.array([-1,-2]))
# returns [ True  True]
print(v1 - v2 == np.array([1-3,2-4]))
# returns [ True  True]
```



# 2d example

For the two-dimensional dataset

$$\mathbf{x}_1 = (1,2), \mathbf{x}_2 = (3,4), \mathbf{x}_3 = (-2,11), \mathbf{x}_4 = (0,66),$$

or, equivalently,

$$\mathbf{x} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ -2 & 11 \\ 0 & 66 \end{pmatrix},$$

the average is

$$\frac{(1,2)+(3,4)+(-2,11)+(0,66)}{4} = (0.5, 20.75).$$

## Example 1.7

```python
import numpy as np

dataset = np.array([[1,2], [3,4], [-2,11], [0,66]])
print(np.mean(dataset, axis=0))
# returns [ 0.5 , 20.75]
```

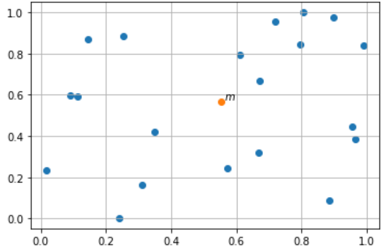


# 2d example

## Example 1.8

Generate a 2 dimensional dataset of random points and their mean

```
import numpy as np
from numpy.random import random as rd
import matplotlib.pyplot as plt
N = 20
dataset = np.array([[rd(), rd()] for _ in range(N)])
mean = np.mean(dataset,axis=0)
plt.grid()
X, Y = dataset[:,0], dataset[:,1]
plt.scatter(X,Y)
plt.scatter(*mean)
plt.annotate('$m$', xy=mean+0.01)
plt.show()
```



Math for Data

Dr. S. M. Moosavi



# Magnitude

## Definition 1.6 (Distance Formula)

*If* $\mathbf{v}_1 = (x_1, y_1)$ *and* $\mathbf{v}_2 = (x_2, y_2)$, *then the distance between* $\mathbf{v}_1$ *and* $\mathbf{v}_2$ *is*

$$|\mathbf{v}_1 - \mathbf{v}_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

*The distance of* $\mathbf{v} = (x, y)$ *to the origin* $\mathbf{0} = (0, 0)$ *is its magnitude or norm or length*

$$r = |\mathbf{v}| = |\mathbf{v} - \mathbf{0}| = \sqrt{x^2 + y^2}.$$

## Example 1.9

For $\mathbf{v}_1 = (1, 2)$ and $\mathbf{v}_2 = (3, 4)$

$$|\mathbf{v}_1| = \sqrt{1^2 + 2^2} = \sqrt{5} \simeq 2.236,$$

$$|\mathbf{v}_1 - \mathbf{v}_2| = \sqrt{(1-3)^2 + (2-4)^2} = \sqrt{4+4} = \sqrt{8} \simeq 2.828.$$

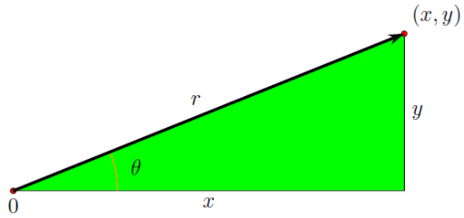
```
import numpy as np

v1 = np.array([1,2])
v2 = np.array([3,4])
print(np.linalg.norm(v1)) #returns 2.23606797749979
print(np.linalg.norm(v1-v2)) #returns 2.
    8284271247461903
```



# Polar representation

In terms of $r$ and $\theta$, the *polar representation* of $(x, y)$ is

$$x = r\cos\theta, \quad y = r\sin\theta.$$




The *unit circle* consists of the vectors which are distance 1 from the origin $\mathbf{0}$. When $\mathbf{v}$ is on the unit circle, the magnitude of $\mathbf{v}$ is 1, and we say $\mathbf{v}$ is a *unit vector*. In this case, the line formed by the scalings of $\mathbf{v}$ intersects the unit circle at $\pm\mathbf{v}$.

When $\mathbf{v}$ is a unit vector, then $r = 1$ and $\mathbf{v} = (x, y) = (\cos\theta, \sin\theta)$.



# Polar representation

By the distance formula, a vector $\mathbf{v} = (x, y)$ is a unit vector when

$$x^2 + y^2 = 1.$$

More generally, any circle with *center* $(a, b)$ and radius $r$ consists of vectors $\mathbf{v} = (x, y)$ satisfying

$$(x - a)^2 + (y - b)^2 = r^2.$$

Let $R$ be a point on the unit circle, and let $t > 0$. The scaled point $tR$ is on the circle with center $(0, 0)$ and radius $t$. Moreover, if $Q$ is any point, $Q + tR$ is on the circle with center $Q$ and radius $t$. It is easy to check that $|t\mathbf{v}| = |t||\mathbf{v}|$ for any real number $t$ and vector $\mathbf{v}$.

From this, if a vector $\mathbf{v}$ is unit and $r > 0$, then $r\mathbf{v}$ has magnitude $r$. If $\mathbf{v}$ is any vector not equal to the zero vector, then $r = |\mathbf{v}|$ is positive, and

$$\left|\frac{1}{r}\mathbf{v}\right| = \frac{1}{r}|\mathbf{v}| = \frac{1}{r}r = 1$$

so $\mathbf{v}/r$ is a unit vector.



# Inner product

## Definition 1.7

*Let* $\mathbf{v}_1 = (x_1, y_1), \mathbf{v}_2 = (x_2, y_2) \in \mathbb{R}^2$. *The inner product or the dot product of* $\mathbf{v}_1$ *and* $\mathbf{v}_2$ *is given algebraically as*

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = x_1 x_2 + y_1 y_2.$$

From the geometric view, we have:

## Theorem 1.1 (Dot Product Identity)

$$x_1 x_2 + y_1 y_2 = \mathbf{v}_1 \cdot \mathbf{v}_2 = |\mathbf{v}_1||\mathbf{v}_2| \cos\theta,$$

*where* $\theta$ *is the angle between* $\mathbf{v}_1$ *and* $\mathbf{v}_1$.

## Exercise 1.3

*Prove the "Dot Product Identity", Theorem (1.1).*
*Hint: Use Pythagoras' theorem for general triangles.*



# The angle between two vectors

In Python, the dot product is given by `numpy.dot` and as a consequence of the dot product identity, we have the code for the angle between two vectors:

$$\theta_{\mathbf{v}_1,\mathbf{v}_2} = \arccos\left(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{|\mathbf{v}_1||\mathbf{v}_2|}\right).$$

## Example 1.10

Find the angle between the vectors $\mathbf{v}_1 = (1, 2)$ and $\mathbf{v}_2 = (3, 4)$.

```
import numpy as np

def angle(u,v):
  a = np.dot(u,v)
  b = np.dot(u,u)
  c = np.dot(v,v)
  theta = np.arccos(a / np.sqrt(b*c))
  return np.degrees(theta)

v1 = np.array([1,2])
v2 = np.array([3,4])
print(angle(v1,v2)) #returns 10.304846468766044 in degree
```



# Cauchy-Schwarz Inequality

Recall that $-1 \leq \cos\theta \leq 1$. Using the dot product identity, we obtain the important inequality:

**Theorem 1.2 (Cauchy-Schwarz Inequality)**

*If $u$ and $v$ are any two vectors, then*

$$-|u||v| \leq u \cdot v \leq |u||v|.$$

**Exercise 1.4**

*Prove the "Cauchy-Schwarz Inequality".*



# 2d linear equations system

Consider the homogeneous system

$$\begin{cases} ax + by = 0 \\ cx + dy = 0 \end{cases} \tag{1.1}$$

and let $A$ be the $2 \times 2$ matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}. \tag{1.2}$$

$(x, y) = (-b, a)$ is a solution of the first equation in (1.1). If we want this to be a solution of the second equation as well, we must have $cx + dy = ad - bc = 0$.

**Definition 1.8 (Determinant)**

*The determinant of A is*

$$\det(A) = \det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc.$$



# 2d linear equations system

## Theorem 1.3 (Homogeneous System)

*When* $\det(A) = 0$, *the homogeneous system (1.1) has a nonzero solution, and all solutions are scalar multiples of* $(x, y) = (-b, a)$. *When* $\det(A) \neq 0$, *the only solution is* $(x, y) = (0, 0)$.

For the inhomogeneous case

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases} \tag{1.3}$$

we have

## Theorem 1.4 (Inhomogeneous System)

*When* $\det(A) \neq 0$, *the inhomogeneous system (1.3) has the unique solution*

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{\det(A)} \begin{pmatrix} de - bf \\ af - ce \end{pmatrix}.$$

*When* $\det(A) = 0$, *(1.3) has a solution iff* $ce = af$ *and* $de = bf$.

# 2d linear equations system

When $a^2 + b^2 \neq 0$, a solution is

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{a^2 + b^2} \begin{pmatrix} ae \\ be \end{pmatrix}.$$

When $c^2 + d^2 \neq 0$, a solution is

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{c^2 + d^2} \begin{pmatrix} cf \\ df \end{pmatrix}.$$

Any other solution differs from these solutions by a scalar multiple of the homogeneous solution $(x, y) = (-b, a)$.

**Exercise 1.5**

*Prove the Theorems (1.3) and (1.4).*



# Complex numbers

Roughly speaking, the set of all *complex numbers* is the set of all points in $\mathbb{R}^2$ with different multiplication rule.

**Definition 1.9 (Complex numbers)**

*The complex numbers,* $\mathbb{C}$*, is the set*

$$\mathbb{C} = \{(x, y) \in \mathbb{R}^2\}$$

*with operations*

- *Addition:* $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$.
- *Scalar Multiplication:* $t(x, y) = (tx, ty)$
- *Multiplication:* $(x_1, y_1)(x_2, y_2) = (x_1x_2 - y_1y_2, x_1y_2 + x_2y_1)$.

Then, in $\mathbb{C}$, we have

- zero: $0 = (0, 0)$.
- opposite or additive inverse: $-(x, y) = (-x, -y)$.
- one: $1 = (1, 0)$.



# Example

## Example 1.11

- $(1,2)+(3,4)=(4,6)$.
- $(0,0)+(1,2)=(1,2)$.
- $3(1,2)=(3,6)$.
- $(1,0)(1,2)=(1-0,2+0)=(1,2)$.
- $(1,2)(3,4)=(3-8,4+6)=(-5,10)$.
- $(x,0)+(y,0)=(x+y,0)$.
- $(x,0)(y,0)=(xy,0)$.

**Note**. By the last two examples, we see that complex numbers with $0$ as their second component act like real numbers in addition and multiplication. So, from now on, we set $x=(x,0)$.

## Example 1.12

- $0=(0,0)$.
- $1=(1,0)$.
- $-1=(-1,0)$.



# Imaginary number

## Definition 1.10 (Imaginary number)

$i = (0, 1).$

**Note**. Python uses the symbol $j$ for imaginary number.

## Theorem 1.5

*For each* $z = (x, y) \in \mathbb{C}$*, we can write*

$$z = x + iy.$$

*We call* $x$ *as the real part of* $z$*, and* $y$ *the imaginary part of* $z$*.*

$$x = Re(z), \quad y = Im(z).$$

**Proof**. $x + iy = (x, 0) + (0, 1)(y, 0) = (x, 0) + (0 - 0, 0 + y) = (x, y).$

## Theorem 1.6

$i^2 = -1.$

**Proof**. $i^2 = (0, 1)(0, 1) = (0 - 1, 0 + 0) = (-1, 0) = -1.$





# Example

## Example 1.13

In complex numbers:

- $\sqrt{-1} = i.$
- $\sqrt{-4} = 2i.$
- $$\begin{aligned}(1,2)(3,4) &= (1+2i)(3+4i)\\ &= 3+4i+6i+8i^2\\ &= 3+10i-8\\ &= -5+10i\\ &= (-5,10).\end{aligned}$$
- $$\begin{aligned}(1,2)^3 &= (1+2i)^3\\ &= (1)^3+3(1)^2(2i)+3(1)(2i)^2+(2i)^3\\ &= 1+6i+12i^2+8i^3\\ &= 1+6i-12-8i\\ &= -11-2i\\ &= -(11,2).\end{aligned}$$



# Conjugate

## Definition 1.11 (Conjugate)

*For* $z = (x, y) \in \mathbb{C}$, *the conjugate is*

$$\bar{z} = (x, -y) = x - iy \in \mathbb{C}.$$

**Some properties**.

- $z + \bar{z} = 2Re(z)$, $z - \bar{z} = 2iIm(z)$.
- $z\bar{z} = Re(z)^2 + Im(z)^2$,

$$\Rightarrow |z| = \sqrt{Re(z)^2 + Im(z)^2} = \sqrt{z\bar{z}}$$
$$\Rightarrow |z|^2 = z\bar{z}.$$

## Example 1.14

For $z = (4, -3) \in \mathbb{C}$:

- $\bar{z} = (4, 3) = 4 + 3i$,
- $z + \bar{z} = 2 \times 4 = 8$, $z - \bar{z} = 2i \times (-3) = -6i$.
- $z\bar{z} = (4)^2 + (-3)^2 = 16 + 9 = 25 \Rightarrow |z| = \sqrt{25} = 5$.
- $z^2 = (4 - 3i)^2 = 7 - 24i$.
- $|z|^2 = 25$.



# Inverse

## Theorem 1.7

*For a non-zero $z \in \mathbb{C}$, the inverse of $z$ is*

$$z^{-1} = \frac{1}{z} = \frac{\bar{z}}{z\bar{z}} = \frac{\bar{z}}{|z|^2}.$$

**Proof**. Firstly, if $z = (x, y)$ then $\frac{1}{z} \in \mathbb{C}$, because,

$$\frac{1}{z} = \frac{x - iy}{x^2 + y^2} = (\frac{x}{x^2 + y^2}, \frac{-y}{x^2 + y^2}) \in \mathbb{C}.$$

Secondly,

$$zz^{-1} = (x + iy)\left(\frac{x - iy}{x^2 + y^2}\right) = \frac{x^2 + y^2}{x^2 + y^2} = 1.$$

## Corollary 1.1 (Division)

*For $z_1 \in \mathbb{C}$ and $0 \neq z_2 \in \mathbb{C}$*

$$\frac{z_1}{z_2} = z_1 z_2^{-1}.$$



# Definitions

## Definition 1.12 (Mean-squared distance)

*Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ be a dataset, say $D$, in $\mathbb{R}^d$, and let $\mathbf{x} \in \mathbb{R}^d$. The mean-squared distance of $\mathbf{x}$ to $D$ is*

$$MSD(\mathbf{x}) = \frac{1}{N} \sum_{k=1}^{N} |\mathbf{x}_k - \mathbf{x}|^2.$$

## Definition 1.13 (Mean)

*Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ be a dataset in $\mathbb{R}^d$. The mean or sample mean is*

$$\mathbf{m} = \bar{\mathbf{x}}_N = \frac{1}{N} \sum_{k=1}^{N} \mathbf{x}_k = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_N}{N}.$$

## Theorem 1.8 (Point of Best-fit)

*The mean is the point of best-fit: The mean minimizes the mean-squared distance to the dataset.*

## Exercise 1.6

*Prove the Theorem (1.8).*



# Point of Best-fit

## Example 1.15

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

np.random.seed(1)
N = 20
rnd = np.random.random
dataset = np.array([ [rnd(), rnd()] for _ in range(N) ])
# Mean
m = np.mean(dataset, axis=0)
#Random point
p = np.array([rnd(), rnd()])

plt.grid()
X, Y = dataset[:,0], dataset[:,1]
plt.scatter(X,Y)
for v in dataset:
    plt.plot([m[0],v[0]],[m[1],v[1]],c='green')
    plt.plot([p[0],v[0]],[p[1],v[1]],c='red')
plt.show()

# Comparison of MSD of the mean and a random point
MSD_m = np.sum(np.abs(dataset-m)**2)/N
MSD_p = np.sum(np.abs(dataset-p)**2)/N
print(MSD_m, MSD_p) # 0.160478187272121 0.5984208474157081
```

# Point of Best-fit

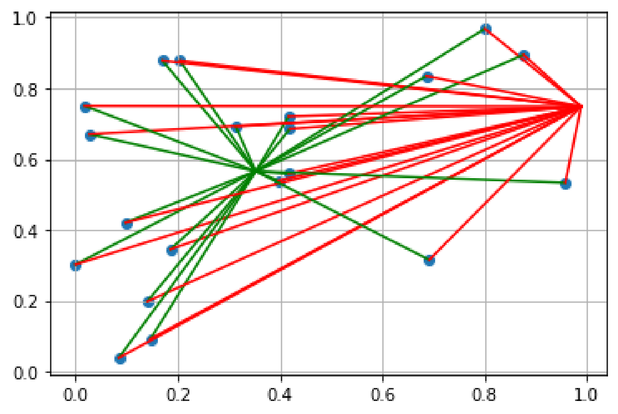



Figure 1.1: MSD for the mean (green) versus MSD for a random point (red).



# Tensor product

For simplicity, let $\mathbf{u} = (a, b)$ and $\mathbf{v} = (c, d, e)$ be two vectors.

## Definition 1.14 (Tensor product)

*The tensor product of* $\mathbf{u}$ *and* **text** *is the matrix*

$$\mathbf{u} \otimes \mathbf{v} = \begin{pmatrix} ac & ad & ae \\ bc & bd & be \end{pmatrix} = \begin{pmatrix} c\mathbf{u} & d\mathbf{u} & e\mathbf{u} \end{pmatrix} = \begin{pmatrix} a\mathbf{v} \\ b\mathbf{v} \end{pmatrix}$$

## Definition 1.15 (Trace of a matrix)

*The trace of a squared matrix* $A$ *is the sum of the diagonal entries.*

**Note**. For any vectors $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$:

- $\mathbf{v} \otimes \mathbf{u} = (\mathbf{u} \otimes \mathbf{v})^t$.

  In square case:
- $\det(\mathbf{u} \otimes \mathbf{v}) = 0$.
- $trace(\mathbf{u} \otimes \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$.
- $trace(\mathbf{u} \otimes \mathbf{u}) = |\mathbf{u}|^2$.
- $(\mathbf{u} \otimes \mathbf{v})\mathbf{w} = (\mathbf{v} \cdot \mathbf{w})\mathbf{u}$.



# Covariance

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be a dataset in $\mathbb{R}^d$ with $\mathbf{m}$ as its mean.

## Definition 1.16 (1d Covariance)

*When $d = 1$, the covariance $q$ is a scalar*

$$q = \frac{1}{N}\sum_{k=1}^{N}(x_k - m)^2 = MSD(m).$$

*In the scalar case, the covariance is called the variance of the scalar dataset.*

In general, the covariance is a symmetric $d \times d$ matrix $Q$. We can center the dataset as

$$\mathbf{v}_1 = \mathbf{x}_1 - \mathbf{m}, \mathbf{v}_2 = \mathbf{x}_2 - \mathbf{m}, \ldots, \mathbf{v}_N = \mathbf{x}_N - \mathbf{m}.$$

Then the *covariance matrix* is the $d \times d$ matrix $Q$ as

$$Q = \frac{\mathbf{v}_1 \otimes \mathbf{v}_1 + \mathbf{v}_2 \otimes \mathbf{v}_2 + \ldots + \mathbf{v}_N \otimes \mathbf{v}_N}{N}. \tag{1.4}$$



# Example

## Example 1.16

Suppose $N = 5$ and

$$\mathbf{x}_1 = (1,2), \quad \mathbf{x}_2 = (3,4), \quad \mathbf{x}_3 = (5,6), \quad \mathbf{x}_4 = (7,8), \quad \mathbf{x}_5 = (9,10).$$

Then $m = (5,6)$ and

$$\mathbf{v}_1 = \mathbf{x}_1 - \mathbf{m} = (-4,-4), \quad \mathbf{v}_2 = \mathbf{x}_2 - \mathbf{m} = (-2,-2),$$
$$\mathbf{v}_3 = \mathbf{x}_3 - \mathbf{m} = (0,0), \quad \mathbf{v}_4 = \mathbf{x}_4 - \mathbf{m} = (2,2), \quad \mathbf{v}_5 = \mathbf{x}_5 - \mathbf{m} = (4,4).$$

Since

$$(\pm 4, \pm 4) \otimes (\pm 4, \pm 4) = \begin{pmatrix} 16 & 16 \\ 16 & 16 \end{pmatrix},$$
$$(\pm 2, \pm 2) \otimes (\pm 2, \pm 2) = \begin{pmatrix} 4 & 4 \\ 4 & 4 \end{pmatrix},$$
$$(0,0) \otimes (0,0) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

then

$$Q = \begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}.$$



# Example

## Example 1.17

```
import numpy as np

def tensor(u,v):
  return np.array([ [ a*b for b in v] for a in u ])

np.random.seed(1)
N = 20
rnd = np.random.random
dataset = np.array([[rnd(), rnd()] for _ in range(N)])
# mean
m = np.mean(dataset,axis=0)
# center dataset
vectors = dataset - m
# covariance
Q = np.mean([ tensor(v,v) for v in vectors ],axis=0)
print(Q)
```



# Standardized

**Note**. The covariance matrix as written in (1.4) is the *biased covariance matrix*. If the denominator is instead $N-1$, the matrix is the *unbiased covariance matrix*.
For datasets with large $N$, it doesn't matter, since $N$ and $N-1$ are almost equal.

In `numpy`, the Python covariance constructor is

## Example 1.18

```python
import numpy as np

np.random.seed(1)
N = 20
rnd = np.random.random
dataset = np.array([[rnd(), rnd()] for _ in range(N)])
# covariance
Q = np.cov(dataset, bias=True, rowvar=False)
print(Q)
```



# Total variance

## Definition 1.17 (Total variance)

*From $trace(\mathbf{u} \otimes \mathbf{u}) = |\mathbf{u}|^2$, if $Q$ is the covariance matrix then*

$$trace(Q) = \frac{1}{N} \sum_{k=1}^{N} |\mathbf{x}_k - \mathbf{m}|^2. \tag{1.5}$$

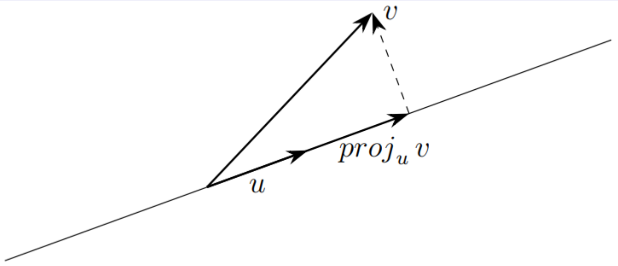
*We call (1.5) the total variance of the dataset. Thus the total variance equals $MSD(\mathbf{m})$.*

## Example 1.19

```python
import numpy as np

np.random.seed(1)
N = 20
rnd = np.random.random
dataset = np.array([[rnd(), rnd()] for _ in range(N)])
# covariance
Q = np.cov(dataset.T,bias=True)
print(Q.trace()) # returns 0.16047818727212101
```

# Projections

We would like to project a $2d$ dataset onto a line. Let $\mathbf{u}$ be a unit vector (a vector of length one, $|\mathbf{u}| = 1$), and let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$ be a $2d$ dataset, assumed for simplicity to be centered. We wish to project this dataset onto the line through $\mathbf{u}$. This will result in a $1d$ dataset.




When a vector $\mathbf{v}$ is projected onto the line through $\mathbf{u}$, the length of the projected vector reads

$$|proj_{\mathbf{u}}\mathbf{v}| = |\mathbf{v}| \cos\theta,$$

where $\theta$ is the angle between the vectors $\mathbf{v}$ and $\mathbf{u}$. Since $|\mathbf{u}| = 1$, this length equals the dot product $\mathbf{v} \cdot \mathbf{u}$. Hence the projected vector is

$$proj_{\mathbf{u}}\mathbf{v} = (\mathbf{v} \cdot \mathbf{u})\mathbf{u}.$$



# Projections

Hence,

## Definition 1.18 (Reduced dataset)

*The projected dataset of* $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N$ *onto the line through* $\mathbf{u}$ *is the dataset*

$$(\mathbf{v}_1 \cdot \mathbf{u})\mathbf{u}, (\mathbf{v}_2 \cdot \mathbf{u})\mathbf{u}, \dots (\mathbf{v}_N \cdot \mathbf{u})\mathbf{u}.$$

*The projected datasetc is in* $\mathbb{R}^2$*. The reduced dataset is*

$$(\mathbf{v}_1 \cdot \mathbf{u}), (\mathbf{v}_2 \cdot \mathbf{u}), \dots (\mathbf{v}_N \cdot \mathbf{u}),$$

*which is in* $\mathbb{R}$*.*

## Exercise 1.7

*Show that when a* $2d$ *dataset is centered then the mean of the reduced dataset is 0.*

## Exercise 1.8

*Prove that if* $Q$ *is the covariance matrix of a* $2d$ *dataset, then the variance of the projected dataset onto the line through the vector* $\mathbf{u}$ *equals the quadratic function* $\mathbf{u} \cdot Q\mathbf{u}$*:*

$$q = \frac{1}{N} \sum_{k=1}^{N} \mathbf{u} \cdot (\mathbf{v}_k \otimes \mathbf{v}_k)\mathbf{u} = \mathbf{u} \cdot Q\mathbf{u}.$$



# Covariance ellipse

Hence,

**Definition 1.19 (Covariance ellipse)**

*The contour of all points $\mathbf{x}$ satisfying $\mathbf{x} \cdot Q\mathbf{x} = 1$ is the covariance ellipsoid. In two dimensions $d = 2$, this is the covariance ellipse. The contour of all points $\mathbf{x}$ satisfying $\mathbf{x} \cdot Q^{-1}\mathbf{x} = 1$ is the inverse covariance ellipsoid. In two dimensions $d = 2$, this is the inverse covariance ellipse.*

In two dimensions $d = 2$, a covariance matrix has the form

$$Q = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

If we write $\mathbf{u} = (x, y)$ for a vector in the plane, the covariance ellipse is

$$\mathbf{u} \cdot Q\mathbf{u} = (x, y) \cdot \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = ax^2 + 2bxy + cy^2 = 1.$$

The covariance ellipse and inverse covariance ellipses described above are centered at the origin $(0, 0)$. When a dataset has mean $\mathbf{m}$ and covariance $Q$, the ellipses are drawn centered at $\mathbf{m}$.

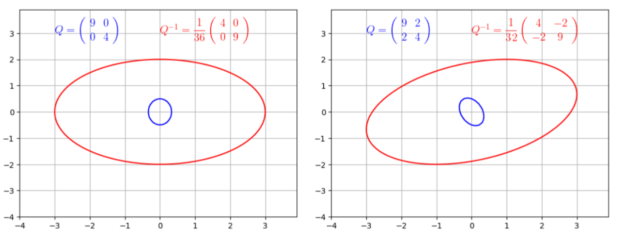
In particular, when $a = c$ and $b = 0$, then $Q = aI$ is a multiple of the identity, the inverse covariance ellipse is the circle of radius $\sqrt{a}$, and the covariance ellipse is the circle of radius $\frac{1}{\sqrt{a}}$.

# Covariance ellipse I

## Example 1.20

Plot the contour ellipses for

$$Q_1 = \begin{pmatrix} 9 & 0 \\ 0 & 4 \end{pmatrix}, \quad Q_2 = \begin{pmatrix} 9 & 2 \\ 2 & 4 \end{pmatrix}.$$





# Covariance ellipse II

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

def ellipse(a, b, c, levels, color):
  L, delta = 4, .1
  x = np.arange(-L,L,delta)
  y = np.arange(-L,L,delta)
  X,Y = np.meshgrid(x, y)
  plt.contour(X, Y, a*X**2 + 2*b*X*Y + c*Y**2, levels,
                                             colors=color)

# Q1 Covariance entities
a, b, c = 9, 0, 4

# Inverse Covariance entities
det = a*c - b**2
A, B, C = c/det, -b/det, a/det

plt.grid()
ellipse(a, b, c, [20], 'blue')
ellipse(A, B, C, [1], 'red')
plt.show()
```



# Covariance ellipse III

```

# Q2 Covariance entities
a, b, c = 9, 2, 4

# Inverse Covariance entities
det = a*c - b**2
A, B, C = c/det, -b/det, a/det

plt.grid()
ellipse(a, b, c, [1], 'blue')
ellipse(A, B, C, [1], 'red')
plt.show()
```





# Standardization

Here, we describe how to standardize datasets in $\mathbb{R}^2$. *Standardizing* the dataset means to center the dataset and to place the x and y features on the same scale.

Consider the dataset $\mathbf{x}_1 = (x_1, y_1), \mathbf{x}_2 = (x_2, y_2), \ldots, \mathbf{x}_N = (x_N, y_N)$ with mean $\mathbf{m} = (m_x, m_y)$. Then the covariance matrix is

$$Q = \begin{pmatrix} a & b \\ b & c \end{pmatrix},$$

where

$$a = \frac{1}{N}\sum_{k=1}^{N}(x_k - m_x)^2, \quad b = \frac{1}{N}\sum_{k=1}^{N}(x_k - m_x)(y_k - m_y),$$

$$c = \frac{1}{N}\sum_{k=1}^{N}(y_k - m_y)^2.$$



# Standardization

If $a$ and $c$ differ, the different scales of x's and y's distorts the relation between them, and $b$ may not accurately reflect the correlation. To correct for this, we center and re-scale

$$x_1, x_2, \ldots, x_N \to x_1' = \frac{x_1 - m_x}{\sqrt{a}}, x_2' = \frac{x_2 - m_x}{\sqrt{a}}, \ldots, x_N' = \frac{x_N - m_x}{\sqrt{a}}$$

and

$$y_1, y_2, \ldots, y_N \to y_1' = \frac{y_1 - m_y}{\sqrt{c}}, y_2' = \frac{y_2 - m_y}{\sqrt{c}}, \ldots, y_N' = \frac{y_N - m_y}{\sqrt{c}}$$

This results in a new dataset

$\mathbf{v}_1 = (x_1', y_1'), \mathbf{v}_2 = (x_2', y_2'), \ldots, \mathbf{v}_N = (x_N', y_N')$ that is centered:

$$\frac{\mathbf{v}_1 + \mathbf{v}_2 + \ldots + \mathbf{v}_N}{N} = 0,$$

with each feature standardized to have unit variance,

$$\frac{1}{N}\sum_{k=1}^{N} x_k' = 1, \quad \frac{1}{N}\sum_{k=1}^{N} y_k' = 1.$$

This is the *standardized dataset*.



# Standardization

The covariance matrix of the standardized dataset has the form

$$Q' = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix},$$

where

$$\rho = \frac{1}{N} \sum_{k=1}^{N} x'_k y'_k = \frac{b}{\sqrt{ac}} = \frac{\sum_{k=1}^{N} (x_k - m_x)(y_k - m_y)}{\sqrt{\left(\sum_{k=1}^{N} (x_k - m_x)^2\right)\left(\sum_{k=1}^{N} (y_k - m_y)^2\right)}}$$

is the *Pearson correlation coefficient* of the dataset. The matrix $Q'$ is the *correlation matrix*, or the *standardized covariance matrix*.

**Example 1.21**

$$Q = \begin{pmatrix} 9 & 2 \\ 2 & 4 \end{pmatrix} \quad \Rightarrow \quad \rho = \frac{b}{\sqrt{ac}} = \frac{1}{3} \quad \Rightarrow \quad Q' = \begin{pmatrix} 1 & 1/3 \\ 1/3 & 1 \end{pmatrix}.$$



# Standardization

From the Cauchy-Schwarz inequality, the correlation coefficient $\rho$ is always between $-1$ and $1$ . When $\rho = \pm 1$, the dataset samples are perfectly correlated and lie on a line passing through the mean. When $\rho = 1$, the line has slope $1$, and when $\rho = -1$, the line has slope $-1$. When $\rho = 0$, the dataset samples are completely uncorrelated and are considered two independent one-dimensional datasets (In standardized case).

In Python `numpy`, the correlation matrix is returned by

```
import numpy as np
np.corrcoef(dataset.T)
```

Here again, we input the transpose of the dataset if our default is vectors as rows.

Notice the $1/N$ cancels in the definition of $\rho$. Because of this, corrcoef is the same whether we deal with biased or unbiased covariance matrices.



# Outline



# Introduction

## Definition 2.1

*A matrix is a listing arranged in a rectangle of rows and columns. Specifically, an $N \times d$ matrix $A$ has $N$ rows and $d$ columns,*

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1d} \\ a_{21} & a_{22} & \dots & a_{2d} \\ \vdots & \vdots & \dots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{Nd} \end{pmatrix}$$

*The transpose of $A$ is*

$$A^t = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{N1} \\ a_{12} & a_{22} & \dots & a_{N2} \\ \vdots & \vdots & \dots & \vdots \\ a_{1d} & a_{2d} & \dots & a_{Nd} \end{pmatrix}$$



# Introduction

## Example 2.1

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix} \Rightarrow A^t = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}.$$

```python
import numpy as np

A = np.array([[1,6,11],[2,7,12],[3,8,13],[4,9,14],[5,10,15]])
print(A)
print(A.shape)
print(len(A))
print(A[1])
print(A[1,2])
print(A[1:3])

# transpose
A_t = np.transpose(A)
print(A_t)
print(A_t.shape)
print(len(A_t))
print(A_t[1])
print(A_t[1,2])
print(A_t[1:3])
```



# Introduction

## Definition 2.2

*A $d$-dimensional vector $\mathbf{v}$ may be written as a $1 \times d$ matrix*

$$\mathbf{v} = \begin{pmatrix} t_1 & t_2 & \cdots & t_d \end{pmatrix}.$$

*In this case, we call $\mathbf{v}$ a row vector.*

## Definition 2.3

*An $N$-dimensional vector $\mathbf{v}$ may be written as an $N \times 1$ matrix*

$$\mathbf{v} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}.$$

*In this case, we call $\mathbf{v}$ a column vector.*



# Introduction

Vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ with the same dimension may be stacked as columns (`np.column_stack` in `Python`) of a matrix,

$$A = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_d \end{pmatrix}.$$

Similarly, vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$ with the same dimension may be stacked as rows (`np.row_stack` in `Python`) of a matrix,

$$A = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_N \end{pmatrix}.$$



# Introduction

## Example 2.2

The row stack of $\mathbf{v}_1 = (1, 6, 11)$, $\mathbf{v}_2 = (2, 7, 12)$, $\mathbf{v}_3 = (3, 8, 13)$, $\mathbf{v}_4 = (4, 9, 14)$ and $\mathbf{v}_5 = (5, 10, 15)$ reads:

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix},$$

and the column stack of them is:

$$A^t = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}.$$

```python
import numpy as np

v1 = [1,6,11]
v2 = [2,7,12]
v3 = [3,8,13]
v4 = [4,9,14]
v5 = [5,10,15]
A = np.row_stack((v1,v2,v3,v4,v5))
print(A)
A_t = np.column_stack((v1,v2,v3,v4,v5))
print(A_t)
```



# Introduction

## Definition 2.4

*A matrix is square if the number of rows equals the number of columns.*

## Definition 2.5

*A matrix is diagonal if the off-diagonal entities are zero.*

## Example 2.3

The matrix

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{pmatrix},$$

is square and diagonal.

The following matrices are not square but they are diagonal:

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \end{pmatrix}, \quad \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \\ 0 & 0 & 0 \end{pmatrix}.$$



# Datasets

## Definition 2.6

*A dataset is a collection of points* $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ *in* $\mathbb{R}^d$. *After centering the mean to the origin, the dataset becomes a collection of vectors* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$. *Usually the vectors are presented as the rows of an* $N \times d$ *matrix* $A$.

Corresponding to this, datasets are often provided as a CSV file. The matrix $A$ is the dataset matrix. In excel, this is called a spreadsheet. In SQL, this is called a table. In `numpy`, it's an array. In `pandas`, it's a dataframe. So, effectively,

matrix = dataset = CSV file = spreadsheet = table = array = dataframe



# Datasets

## Example 2.4

For the Iris dataset:

```python
import numpy as np
import pandas as pd
from sklearn import datasets

iris = datasets.load_iris()

# The dataset
dataset = iris["data"]

# To center the dataset
m = np.mean(dataset,axis=0)
vectors = dataset - m

# To make a data frame
centered_df = pd.DataFrame(data=vectors)
```



# Addition & scalar multiplication

Matrices consisting of numbers are added and multiplied by scalars as follows. With $t$ as an scalar and the matrices

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1d} \\ a_{21} & a_{22} & \dots & a_{2d} \\ \vdots & \vdots & \dots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{Nd} \end{pmatrix} \quad \text{and} \quad A' = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1d} \\ a'_{21} & a'_{22} & \dots & a'_{2d} \\ \vdots & \vdots & \dots & \vdots \\ a'_{N1} & a'_{N2} & \dots & a'_{Nd} \end{pmatrix}$$

we have

$$A + A' = \begin{pmatrix} a_{11} + a'_{11} & a_{12} + a'_{12} & \dots & a_{1d} + a'_{1d} \\ a_{21} + a'_{21} & a_{22} + a'_{22} & \dots & a_{2d} + a'_{2d} \\ \vdots & \vdots & \dots & \vdots \\ a_{N1} + a'_{N1} & a_{N2} + a'_{N2} & \dots & a_{Nd} + a'_{Nd} \end{pmatrix},$$

and

$$tA = \begin{pmatrix} ta_{11} & ta_{12} & \dots & ta_{1d} \\ ta_{21} & ta_{22} & \dots & ta_{2d} \\ \vdots & \vdots & \dots & \vdots \\ ta_{N1} & ta_{N2} & \dots & ta_{Nd} \end{pmatrix}.$$

Matrices may be added only if they have the same shape.



# Addition & scalar multiplication

## Example 2.5

```
import numpy as np

A = np.zeros((4,3))
print(A)
B = np.eye(3)
print(B)
C = np.eye(4,3)
print(C)
D = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
print(D)
E = np.diag([1,2,3,4])
print(E)

print(A+C)
print(C+D)
print(4*D)
print(-D)
print(-2*D)
```



# Introduction

Let $t$ be a scalar, $\mathbf{u}, \mathbf{v}, \mathbf{w}$ be vectors, and let $A, B$ be matrices. We already know how to compute $t\mathbf{u}$, $t\mathbf{v}$, and $tA$, $tB$. In this section, we compute the *dot product* $\mathbf{u} \cdot \mathbf{v}$, the *matrix-vector product* $A\mathbf{v}$, and the *matrix-matrix product* $AB$.



# Dot product

In the first chapter, we defined the dot product in two dimensions. We now generalize it to any dimension $d$. Suppose $\mathbf{u}, \mathbf{v}$ are vectors in $\mathbb{R}^d$. Then their dot product $\mathbf{u} \cdot \mathbf{v}$ is the scalar obtained by multiplying corresponding features and then summing the products. This only works if the dimensions of $\mathbf{u}$ and $\mathbf{v}$ agree.

In other words, if $\mathbf{u} = (u_1, u_2, \ldots, u_d)$ and $\mathbf{v} = (v_1, v_2, \ldots, v_d)$, then

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \ldots + u_d v_d.$$

It's best to think of this as "row-times-column" multiplication,

$$\mathbf{u} \cdot \mathbf{v} = \begin{pmatrix} u_1 & u_2 & \cdots & u_d \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix} = u_1 v_1 + u_2 v_2 + \ldots + u_d v_d.$$



# Dot product

## Example 2.6

In Python, calculate the dot product of $\mathbf{u} = (1, 2, 3)$ and $\mathbf{v} = (4, 5, 6)$.

```python
import numpy as np

u = np.array([1,2,3])
v = np.array([4, 5, 6])

u_dot_v = np.dot(u,v)
print(u_dot_v)

u_dot_v_ = u[0]*v[0] + u[1]*v[1] + u[2]*v[2]
print(u_dot_v_)

print(u_dot_v == u_dot_v_)
```



# Dot product

As we mentioned in 2 dimensions, we have the following generalizations in $d$ dimension:

## Definition 2.7

*The length or norm or magnitude of a vector* $\mathbf{v}$ *is the square root of the dot product* $\mathbf{v}\cdot\mathbf{v}$,

$$|\mathbf{v}| = \sqrt{\mathbf{v}\cdot\mathbf{v}}$$

## Theorem 2.1 (Dot Product)

*The dot product* $\mathbf{u}\cdot\mathbf{v}$ *satisfies*

$$\mathbf{u}\cdot\mathbf{v} = |\mathbf{u}||\mathbf{v}|\cos\theta,$$

*where* $\theta$ *is the angle between* $\mathbf{u}$ *and* $\mathbf{v}$.

## Corollary 2.1

*To calculate the angle* $\theta$ *between* $\mathbf{u}$ *and* $\mathbf{v}$ *we have:*

$$\cos\theta = \frac{\mathbf{u}\cdot\mathbf{v}}{\sqrt{|\mathbf{u}||\mathbf{v}|}} = \frac{\mathbf{u}\cdot\mathbf{v}}{\sqrt{(\mathbf{u}\cdot\mathbf{u})(\mathbf{v}\cdot\mathbf{v})}}.$$



# Dot product

## Corollary 2.2 (Cauchy-Schwarz Inequality)

*The dot product of two vectors is absolutely less or equal to the product of their lengths,*

$$|\mathbf{u} \cdot \mathbf{v}| \leq |\mathbf{u}||\mathbf{v}| \quad \textit{or} \quad |\mathbf{u} \cdot \mathbf{v}| \leq (\mathbf{u} \cdot \mathbf{u})(\mathbf{v} \cdot \mathbf{v}).$$

## Definition 2.8

*Vectors $\mathbf{u}$ and $\mathbf{v}$ are said to be perpendicular or orthogonal if $|\mathbf{u} \cdot \mathbf{v}| = 0$.*
*A collection of vectors is orthogonal if any pair of vectors in the collection are orthogonal.*
*Vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$ are said to be orthonormal if they are both unit vectors and orthogonal.*

## Exercise 2.1

*The zero vector is orthogonal to every vector. The converse is true as well: if a vector is orthogonal to every vector then it is the zero vector.*



# Matrix-vector product

## Definition 2.9

*Suppose $\mathbf{v}$ is a vector and $A$ is a matrix. If the rows of $A$ have the same dimension as that of $\mathbf{v}$, we can take the dot product of each row of $A$ with $\mathbf{v}$, obtaining the matrix-vector product $A\mathbf{v}$: $A\mathbf{v}$ is the vector whose features are the dot products of the rows of $A$ with $\mathbf{v}$.*

**Note:**

- In Python we use again `np.dot(A,v)` for matrix-vector product.
- If $\mathbf{u}$ and $\mathbf{v}$ are vectors, we can think of $\mathbf{u}$ as a row vector, or a matrix consisting of a single row. With this interpretation, the matrix-vector product $\mathbf{u}\mathbf{v}$ equals the dot product $\mathbf{u} \cdot \mathbf{v}$.
- If $\mathbf{u}$ and $\mathbf{v}$ are vectors, we can think of $\mathbf{u}$ as a column vector, or a matrix consisting of a single column. With this interpretation, $\mathbf{u}^t$ is a single row, and the matrix-vector product $\mathbf{u}^t\mathbf{v}$ equals the dot product $\mathbf{u} \cdot \mathbf{v}$.
- $(A\mathbf{v})^t = \mathbf{v}^t A^t$.
- $(A\mathbf{u}) \cdot \mathbf{v} = \mathbf{u} \cdot (A^t\mathbf{v})$.



# Matrix-vector product

## Example 2.7

Calculate $A\mathbf{v}$, when

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \quad \text{and} \quad \mathbf{v} = (1, 2, 3, 4).$$

**Answer**:

$$A\mathbf{v} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

$$= \begin{pmatrix} (1 \times 1) + (2 \times 2) + (3 \times 3) + (4 \times 4) \\ (5 \times 1) + (6 \times 2) + (7 \times 3) + (8 \times 4) \\ (9 \times 1) + (10 \times 2) + (11 \times 3) + (12 \times 4) \end{pmatrix} = \begin{pmatrix} 30 \\ 70 \\ 110 \end{pmatrix}$$

```python
import numpy as np

A = np.arange(1,13).reshape(3,4)
v = np.array([1,2,3,4])

Av = np.dot(A, v)
print(Av)
```



# Matrix-matrix product

**Definition 2.10**

*Let $A$ and $B$ be two matrices. If the row dimension of $A$ equals the column dimension of $B$, the matrix-matrix product $AB$ is defined. When this condition holds, the entries in the matrix $AB$ are the dot products of the rows of $A$ with the columns of $B$.*

**Note:**

- In Python we use again `np.dot(A,B)` for matrix-vector product.
- $(AB)^t = B^t A^t$.



# Matrix-vector product

## Example 2.8

Calculate $AB$, when

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 13 & 14 \\ 15 & 16 \\ 17 & 18 \\ 19 & 20 \end{pmatrix}.$$

**Answer**:

$$AB = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \begin{pmatrix} 13 & 14 \\ 15 & 16 \\ 17 & 18 \\ 19 & 20 \end{pmatrix}$$

$$= \begin{pmatrix} (1 \times 13) + (2 \times 15) + (3 \times 17) + (4 \times 19) & (1 \times 14) + (2 \times 16) + (3 \times 18) + (4 \times 20) \\ (5 \times 13) + (6 \times 15) + (7 \times 17) + (8 \times 19) & (5 \times 14) + (6 \times 16) + (7 \times 18) + (8 \times 20) \\ (9 \times 13) + (10 \times 15) + (11 \times 17) + (12 \times 19) & (9 \times 14) + (10 \times 16) + (11 \times 18) + (12 \times 20) \end{pmatrix}$$

$$= \begin{pmatrix} 170 & 180 \\ 426 & 452 \\ 682 & 724 \end{pmatrix}$$

```python
import numpy as np

A = np.arange(1,13).reshape(3,4)
B = np.arange(13,21).reshape(4,2)

AB = np.dot(A, B)
print(AB)
```



# Orthonormal Rows and Columns

Assume the rows of a matrix $A$ are $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$. Since matrix-matrix multiplication is $row \times column$, we have

$$AA^t = \begin{pmatrix} \mathbf{v}_1 \cdot \mathbf{v}_1 & \mathbf{v}_1 \cdot \mathbf{v}_2 & \cdots & \mathbf{v}_1 \cdot \mathbf{v}_N \\ \mathbf{v}_2 \cdot \mathbf{v}_1 & \mathbf{v}_2 \cdot \mathbf{v}_2 & \cdots & \mathbf{v}_2 \cdot \mathbf{v}_N \\ \vdots & \vdots & & \vdots \\ \mathbf{v}_N \cdot \mathbf{v}_1 & \mathbf{v}_N \cdot \mathbf{v}_2 & \cdots & \mathbf{v}_N \cdot \mathbf{v}_N \end{pmatrix}.$$

**Corollary 2.3**

*Let $U$ be a matrix.*

- *$U$ has orthonormal rows iff $UU^t = I$.*
- *$U$ has orthonormal columns iff $U^tU = I$.*



# Tensor product

## Definition 2.11

*If $\mathbf{u}$ and $\mathbf{v}$ are vectors, the tensor product $\mathbf{u} \otimes \mathbf{v}$ is the matrix-matrix product $\mathbf{u}^t\mathbf{v}$, with $\mathbf{u}$ and $\mathbf{v}$ row vectors. If $\mathbf{u}$ is $N$-dimensional and $\mathbf{v}$ is $d$-dimensional, then $\mathbf{u} \otimes \mathbf{v}$ is an $N \times d$ matrix.*

## Example 2.9

if $\mathbf{u} = (a, b, c)$ and $\mathbf{v} = (\alpha, \beta)$, then

$$\mathbf{u} \otimes \mathbf{v} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \begin{pmatrix} \alpha & \beta \end{pmatrix} = \begin{pmatrix} a\alpha & a\beta \\ b\alpha & b\beta \\ c\alpha & c\beta \end{pmatrix}.$$

Using the tensor product, we have

## Theorem 2.2 (Tensor Identity)

*Let $A$ be a matrix with rows $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$. Then*

$$A^tA = \mathbf{v}_1 \otimes \mathbf{v}_1 + \mathbf{v}_2 \otimes \mathbf{v}_2 + \cdots + \mathbf{v}_N \otimes \mathbf{v}_N.$$

## Exercise 2.2

*Prove the tensor identity.*



# Some definitions

## Definition 2.12

*A matrix $Q$ is symmetric if $Q = Q^t$.*
*For any matrix $A$, $Q = AA^t$ and $Q = A^tA$ are symmetric.*
*A symmetric matrix $Q$ satisfying $\mathbf{v} \cdot Q\mathbf{v} \geq 0$ for every vector $\mathbf{v}$ is nonnegative.*
*A symmetric matrix $Q$ satisfying $\mathbf{v} \cdot Q\mathbf{v} > 0$ for every nonzero vector $\mathbf{v}$ is positive.*

## Definition 2.13

*The trace of a square matrix is the sum of its diagonal elements.*

Even though in general $AB \neq BA$, it is always true that

## Exercise 2.3

$trace(AB) = trace(BA)$.

## Exercise 2.4

$\mathbf{u} \cdot Q\mathbf{v} = trace(Q(\mathbf{v} \otimes \mathbf{u}))$.



# Norm squared

## Definition 2.14

*If $A = (a_{ij})$ is any matrix, then the norm squared of $A$ is*

$$\parallel A \parallel^2 = \sum_{i,j} a_{ij}^2.$$

## Theorem 2.3 (Norm Squared of Matrix)

*Let $A$ be a matrix with rows $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$. Then*

$$\parallel A \parallel^2 = |\mathbf{v}_1|^2 + |\mathbf{v}_2|^2 + \ldots + |\mathbf{v}_N|^2,$$

*and*

$$\parallel A \parallel^2 = trace(A^t A).$$

## Exercise 2.5

*Prove Theorem (2.3).*



# Iris

If $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ is a dataset of points in $\mathbb{R}^d$ with mean $\mathbf{m}$, and $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$ is the corresponding centered dataset, then we saw that the covariance matrix $Q$ is the average of tensor products

$$Q = \frac{\mathbf{v}_1 \otimes \mathbf{v}_1 + \mathbf{v}_2 \otimes \mathbf{v}_2 + \cdots + \mathbf{v}_N \otimes \mathbf{v}_N}{N}.$$

Let $A$ be the matrix with rows $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$. By Theorem (2.2), the last equation is the same as

$$Q = \frac{1}{N} A^t A.$$



# Iris

## Example 2.10

Calculate the mean, covariance and total variance of the Iris dataset.

```
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()

# The dataset
dataset = iris["data"]

# Mean
m = np.mean(dataset, axis=0)

# Centered dataset
vectors = dataset - m

# Covariance
N = len(vectors)
#   Biased
Q = np.dot(vectors.T, vectors)/N
Q = np.cov(dataset, rowvar=False, ddof=0) # ddof = delta degrees of freedom
Q = np.cov(dataset.T, ddof=0)

#   Unbiased
Q = np.dot(vectors.T, vectors)/(N-1)
Q = np.cov(dataset, rowvar=False)
Q = np.cov(dataset.T)

# Total Variance
TV = np.trace(Q)
```

# Standardized dataset

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ is a dataset of points in $\mathbb{R}^d$. Each sample point $\mathbf{x}$ has $d$ features $(t_1, t_2, \ldots, t_d)$. We compute the variance of each feature separately.

Let $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d$ be the standard basis in $\mathbb{R}^d$, and, for each $j = 1, 2, \ldots, d$, project the dataset onto $\mathbf{e}_j$ , obtaining the scalar dataset $\mathbf{x}_1 \cdot \mathbf{e}_j, \mathbf{x}_2 \cdot \mathbf{e}_j, \ldots, \mathbf{x}_N \cdot \mathbf{e}_j$, consisting of the $j$-th feature of the samples. If $q_{jj}$ is the variance of this scalar dataset, then $q_{11}, q_{22}, \ldots, q_{dd}$ are the diagonal entries of the covariance matrix.

To standardize the dataset, we center it, and rescale the features to have variance one, as follows. Let $\mathbf{m} = (m_1, m_2, \ldots, m_d)$ be the dataset mean. For each sample point $\mathbf{x} = (t_1, t_2, \ldots, t_d)$, the standardized vector is

$$\mathbf{v} = \left( \frac{t_1 - m_1}{\sqrt{q_{11}}}, \frac{t_2 - m_2}{\sqrt{q_{22}}}, \ldots, \frac{t_d - m_d}{\sqrt{q_{dd}}} \right).$$

Then the standardized dataset is $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$.



# Standardized dataset

**Definition 2.15**

*If $Q = (q_{ij})$ is the covariance matrix, then the correlation matrix is the $d \times d$ matrix $Q' = (q'_{ij})$ with entries*

$$q'_{ij} = \frac{q_{ij}}{\sqrt{q_{ii}q_{jj}}}, \quad i, j = 1, 2, \ldots, d.$$

**Theorem 2.4 (Standardized Covariance Equals Correlation)**

*The covariance matrix of the standardized dataset equals the correlation matrix of the original dataset.*

**Exercise 2.6**

*Prove Theorem (2.4).*



# Iris

## Example 2.11

For the Iris dataset check Theorem (2.4).

```python
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()

# The dataset
dataset = iris["data"]

# standardize dataset
vectors = StandardScaler().fit_transform(dataset)
Qcorr = np.corrcoef(dataset.T)
Qcov = np.cov(vectors.T,bias=True)
np.allclose(Qcov,Qcorr)
```



# Definition

## Definition 2.16

*Given a square matrix $A$, the inverse matrix is the matrix $B$ satisfying*

$$AB = I = BA.$$

*When $A$ has an inverse, we say $A$ is invertible. If a matrix is $d \times d$, then the inverse is also $d \times d$. We write $B = A^{-1}$ for the inverse matrix of $A$.*

Here $I$ is the identity matrix. Not every square matrix has an inverse. For example, the zero matrix does not have an inverse.

## Example 2.12

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \Rightarrow A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Since we can't divide by zero, a $2 \times 2$ matrix is invertible only if $ad - bc \neq 0$.



# Notes

## Exercise 2.7

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

## Exercise 2.8

*Prove that for a linear system* $A\mathbf{x} = \mathbf{b}$, *if* $A$ *is invertible then* $\mathbf{x} = A^{-1}\mathbf{b}$.

## Example 2.13

Solve the following linear system

$$\begin{cases} x + 2y + 3z = 1 \\ -3x + 6y = 2 \\ 10x - 5y + 23z = 3 \end{cases}$$

```python
import numpy as np

A = np.array([[1,2,3],[-3,6,0], [10,-5,23]])
b = np.array([1,2,3])
# Determinant of A
np.linalg.det(A)
# Inverse of A
np.linalg.inv(A)
# Solution of Ax=b
x = np.dot(np.linalg.inv(A),b)
```



# Definition

## Definition 2.17 (Linear combination)

*A linear combination of vectors* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ *is*

$$t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + \ldots + t_d\mathbf{v}_d,$$

*where* $t_1, t_2, \ldots, t_d$ *are scalars.*

## Example 2.14

Let $\mathbf{u}, \mathbf{v}, \mathbf{w}$ be three vectors. Then

$$3\mathbf{u}\frac{1}{6}\mathbf{v} + 9\mathbf{w} \quad , 5\mathbf{u} + 0\mathbf{v} - \mathbf{w}, \quad 0\mathbf{u} + 0\mathbf{v} + 0\mathbf{w},$$

are linear combinations of $\mathbf{u}, \mathbf{v}, \mathbf{w}$.

## Example 2.15

Let $A$ be a matrix with columns $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$, and let $\mathbf{x} = (t_1, t_2, \ldots, t_d)$. Then $A\mathbf{x}$ is a linear combination of the columns of $A$ as:

$$A\mathbf{x} = t_1\mathbf{v}_1 + t_2\mathbf{v}_2 + \ldots + t_d\mathbf{v}_d.$$



# Definition

## Definition 2.18 (Span)

*The span of* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ *is the set* $S$ *of all linear combinations of* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$, *and we write*

$$S = span(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d).$$

## Exercise 2.9

*Let* $A$ *be the matrix with columns* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$. *Then* $S = span(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d)$ *is the set* $S$ *of all vectors of the form* $Ax$.

## Exercise 2.10

*If each vector* $\mathbf{v}_k$ *of* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ *is a linear combination of vectors* $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N$, *then*

$$span(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d) \subseteq span(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N).$$



# Definition

## Definition 2.19

*Let $A$ be a matrix. The column space of $A$ is the span of its columns.*

## Example 2.16

```
import sympy as sp
import scipy as sc
import numpy as np

A = sp.Matrix([[1, 6, 11], [2, 7, 12], [3, 8, 13], [4, 9, 14], [5, 10, 15]])
A
# column vectors
u = sp.Matrix([1,2,3,4,5])
v = sp.Matrix([6,7,8,9,10])
w = sp.Matrix([11,12,13,14,15])
A = sp.Matrix.hstack(u,v,w)
A
# returns minimal spanning set for column space of A
A.columnspace()
# returns minimal spanning orthonormal set for column space of A
A = np.array([[1, 6, 11], [2, 7, 12], [3, 8, 13], [4, 9, 14], [5, 10, 15]])
sc.linalg.orth(A)
```

`A.columnspace()` returns a minimal set of vectors spanning the column space of $A$. The *column rank* of $A$ is the number of vectors returned: for $A$ in the above example, the column rank is 2. `sc.linalg.orth()` returns a minimal orthonormal set of vectors spanning the column space of $A$.



# Notes

**Exercise 2.11**

*As in example 2.16, show that if*

$$\mathbf{v}_1 = (1,2,3,4,5), \quad \mathbf{v}_2 = (6,7,8,9,10), \quad \mathbf{v}_3 = (11,12,13,14,15)$$

*then* $span(\mathbf{v}_1, \mathbf{v}_2) = span(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$.

**Exercise 2.12**

*Show that: the column space of a matrix $A$ consists of all vectors of the form $A\mathbf{x}$. A vector $\mathbf{b}$ is in the column space of $A$ when $A\mathbf{x} = \mathbf{b}$ has a solution.*

The augmented matrix $\bar{A} = (A, \mathbf{b})$ is obtained by adding $\mathbf{b}$ as an extra column next to the columns of $A$.

**Exercise 2.13**

*Let $\bar{A}$ be the matrix $A$ augmented by a vector $\mathbf{b}$. Then $\mathbf{b}$ is in the column space of $A$ iff*

$$\textit{column rank}(A) = \textit{column rank}(\bar{A}).$$