Introduction to Data Science and Systems

Lecture 2: Introduction of Vectors and Matrices

Dr Zaiqiao Meng

University of Glasgow

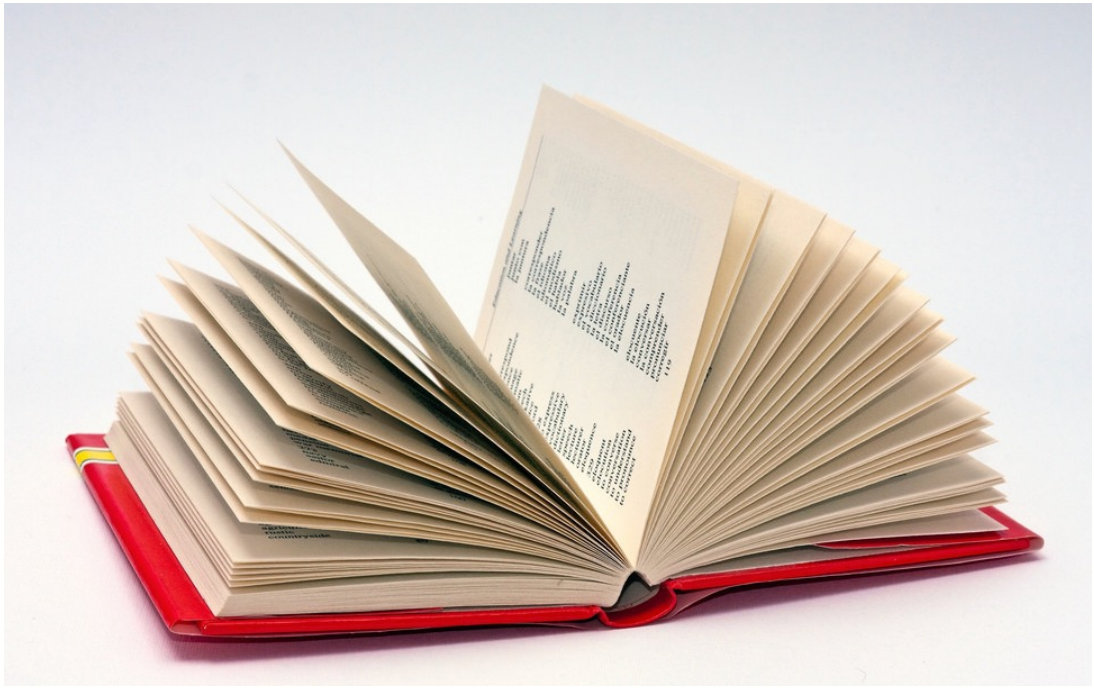
Intended Learning Outcomes

- what a vector is and what a vector space is
- the standard operations on vectors: addition and multiplication
- what a norm is and how it can be used to measure vectors
- what an inner product is and how it gives rise to geometry of vectors
- how mathematical vectors map onto numerical arrays
- the different p-norms and their uses
- important computational uses of vector representations
- how to characterise vector data with a mean vector and a covariance matrix
- the properties of high-dimensional vector spaces
- the basic notation for matrices
- the view of matrices as linear maps
- how basic geometric transforms are implemented using matrices
- how matrix multiplication is defined and its algebraic properties
- the basic anatomy of matrices

University of Glasgow

Example: Text and translation

- Text, as represented by strings in memory, has *weak structure*.
- There are **comparison functions** for strings (e.g. edit distance, Hamming distance), but only character-level semantics
- **String operations** are character-level operations like concatenation or reversal, but not useful for machine translation system



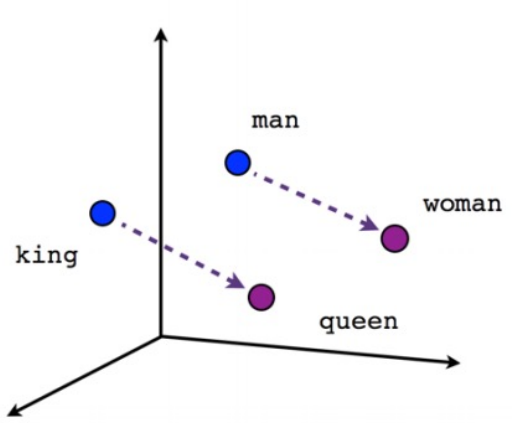
University of Glasgow

Words aren't enough

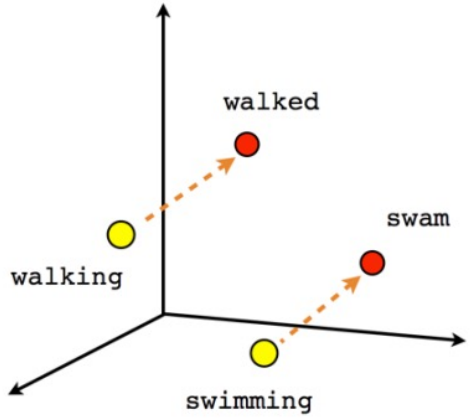
- *Original*
  - "The craft held fast to the bank of the burn."
  - *(the vessel stayed moored to the edge of the stream)*
- *Dictionary lookup (naiive)*
  - French: "L'artisanat tenu rapide à la Banque de la brûlure."
  - *(the artisanal skill held quickly to the financial institution of the burn wounds)*
  - Danish: "Håndværket holdt fast til banken af brænden"
  - *(The craftmanship held fast to the bank [financial institution] of fire)*
- *Correct(ish)*
  - French: "Le bateau se tenait fermement à la rive du ruisseau."
  - *(the boat was firmly attached to the riverbank)*
  - Danish: "Farttøjet holdt fast i bredden af åen"
  - *(The vessel held fast at the bank of the river)*

Solution -- to place them in a vector space

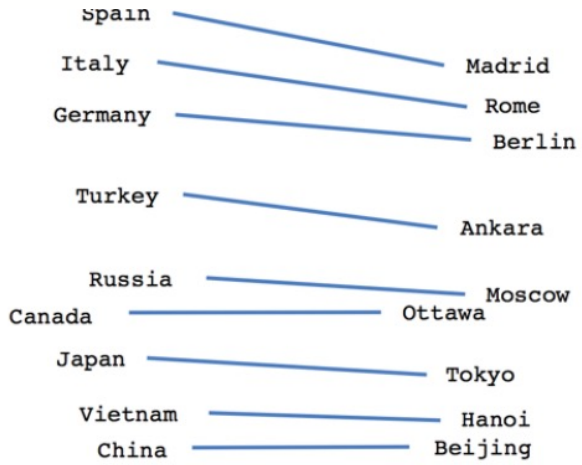
- Imbue text fragments with additional mathematical structure -- **to place them in a vector space**
- Fragments might be words, partial words or whole sentences



Male-Female



Verb tense



Country-Capital

[Image: word2vec example, from https://www.tensorflow.org/tutorials/word2vec, licensed Apache 2.0]

University of Glasgow

The structure of a (topological) vector space

- What words are like salamander?
  - **Distance/metric functions**: norm
  - E.g. the neighbourhood of the vector corresponding to salamander, which might include words like axolotl or waterdog

- What is the equivalent of a king, but with a woman instead of a man?
  - **Operation functions:** subtraction, mean
  - Famously, the original word2vec paper should that on their test data, the equation

$$\text{King} - \text{Man} + \text{Woman} = \text{Queen}$$

University of Glasgow

Vector spaces

- **Vectors** to be ordered tuples of real numbers

$$[x_1, x_2, \ldots x_n], x_i \in \mathbb{R}$$

  - A vector has a fixed dimension n
  - Each element of the vector as representing a distance in a **direction orthogonal** to all the other elements.
  - Orthogonal just means "independent", or, geometrically speaking "at 90 degrees".
  - We write vectors with a bold lower case letter:

$$\mathbf{x} = [x_1, x_2, \ldots, x_d],$$
$$\mathbf{y} = [y_1, y_2, \ldots, y_d],$$

University of Glasgow

Vector spaces

- A **vector space** is a set whose **vectors**.
- For example, a length-3 vector might be used to represent a spatial position in Cartesian coordinates, with three orthogonal measurements for each vector.

- Consider the 3D vector [5, 7, 3]

```
5 * [1,0,0] +
7 * [0,1,0] +
3 * [0,0,1]
```

- Each of these vectors [1,0,0], [0,1,0], [0,0,1] is pointing in a independent direction (orthogonal direction) and has length one.

University of Glasgow

Points in space

Notation:

- $\mathbb{R}$ means the set of real numbers.
- $\mathbb{R}_{\geq 0}$ means the set of non-negative reals.
- $\mathbb{R}^n$ means the set of tuples of exactly n real numbers (vector).
- $\mathbb{R}^{n \times m}$ means the set of 2D arrays (matrix) of real numbers with exactly n rows of m elements.
- The notation $(\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R}$ says that the operation defines a map from a pair of n dimensional vectors to a real number.

University of Glasgow

Points in space

Vector spaces

Any vector of given dimension n lies in a **vector space**, called $\mathbb{R}^n$, along with the operations of:

- **scalar multiplication** so that $a\mathbf{x}$ is defined for any scalar a. For real vectors, $a\mathbf{x} = [ax_1, ax_2, \ldots ax_n]$, elementwise scaling.

$$(\mathbb{R}, \mathbb{R}^n) \rightarrow \mathbb{R}^n$$

- **vector addition** so that $\mathbf{x} + \mathbf{y}$ vectors $\mathbf{x}, \mathbf{y}$ of equal dimension. For real vectors, $\mathbf{x} + \mathbf{y} = [x_1 + y_1, x_2 + y_2, \ldots x_d + y_d]$ the elementwise sum

$$(\mathbb{R}^n, \mathbb{R}^n) \rightarrow \mathbb{R}^n$$

Points in space

Two additional operations

- A **norm** $\| \mathbf{x} \|$ which allows the length of vectors to be measured.

$$\mathbb{R}_n \rightarrow \mathbb{R}_{\geq 0}$$

- An **inner product** $\langle \mathbf{x} \mid \mathbf{y} \rangle$, $\langle \mathbf{x}, \mathbf{y} \rangle$ or $\mathbf{x} \cdot \mathbf{y}$ which allows the angles of two vectors to be compared. The inner product of two orthogonal vectors is 0 . For real vectors

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + x_3 y_3 \ldots x_d y_d$$

$$(\mathbb{R}^n, \mathbb{R}^n) \rightarrow \mathbb{R}$$

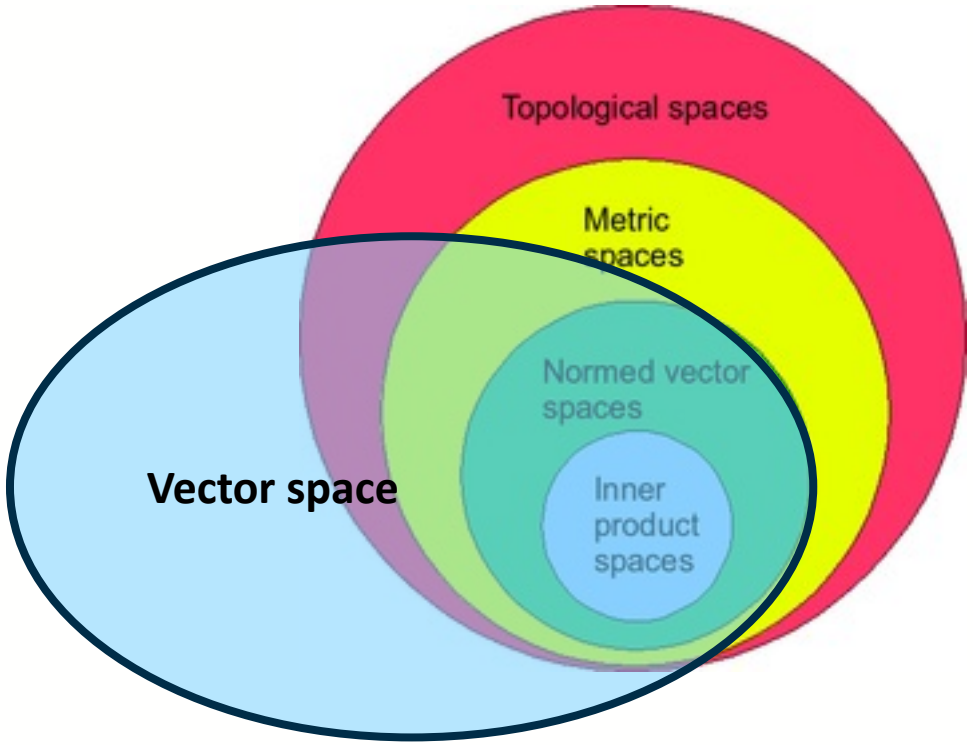
Topological and inner product spaces

- With a norm a vector space is a **normed vector space/topological vector space**.
- With an inner product, a vector space is an **inner product space**, and we can talk about the angle between two vectors.

Topological/geometrical space:
- **a set whose elements are called points**

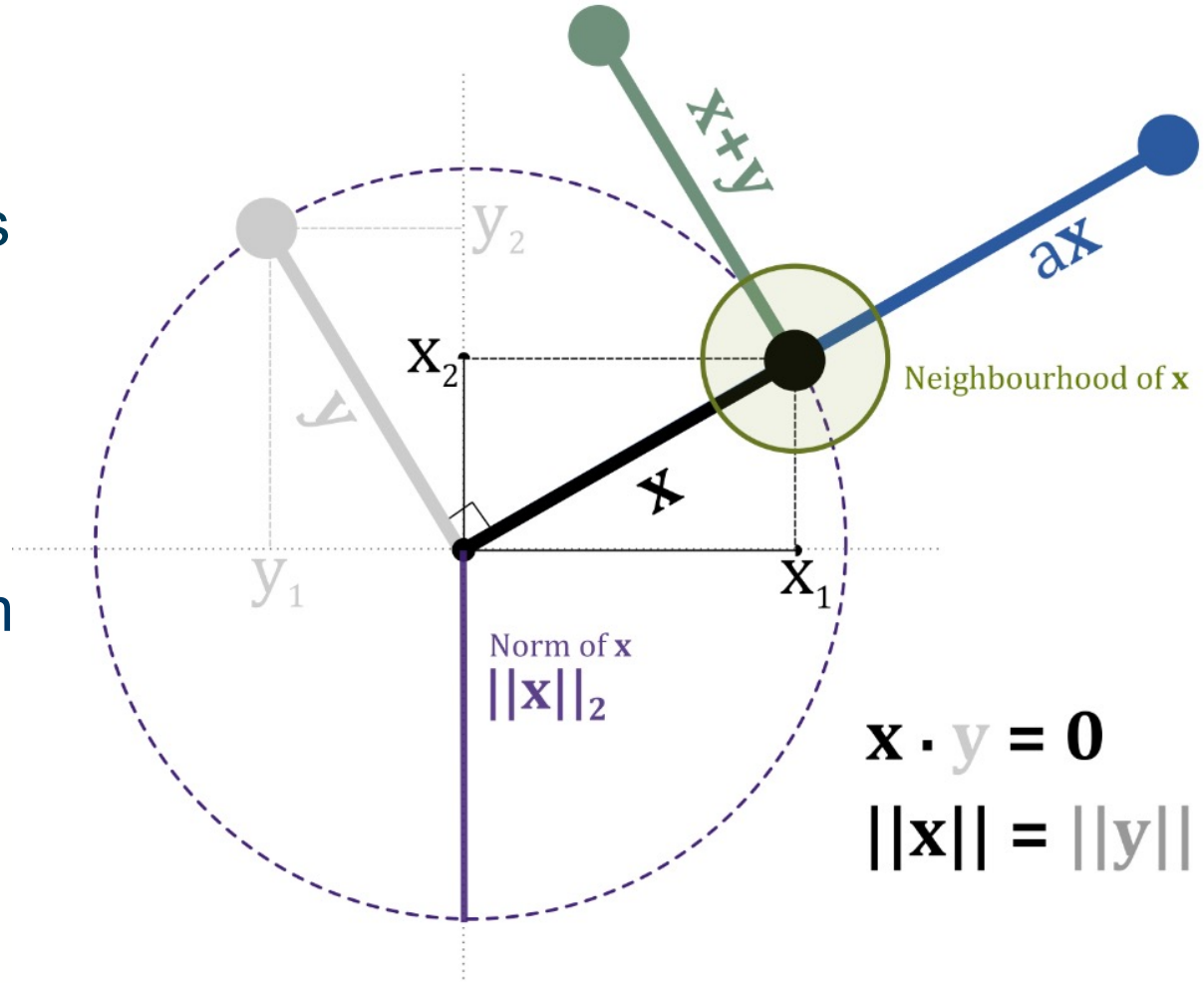
Metric space:
- **define distance between points, e.g. norm**

Normed vector space:
- **a vector space with norm defined**



Topological and inner product spaces

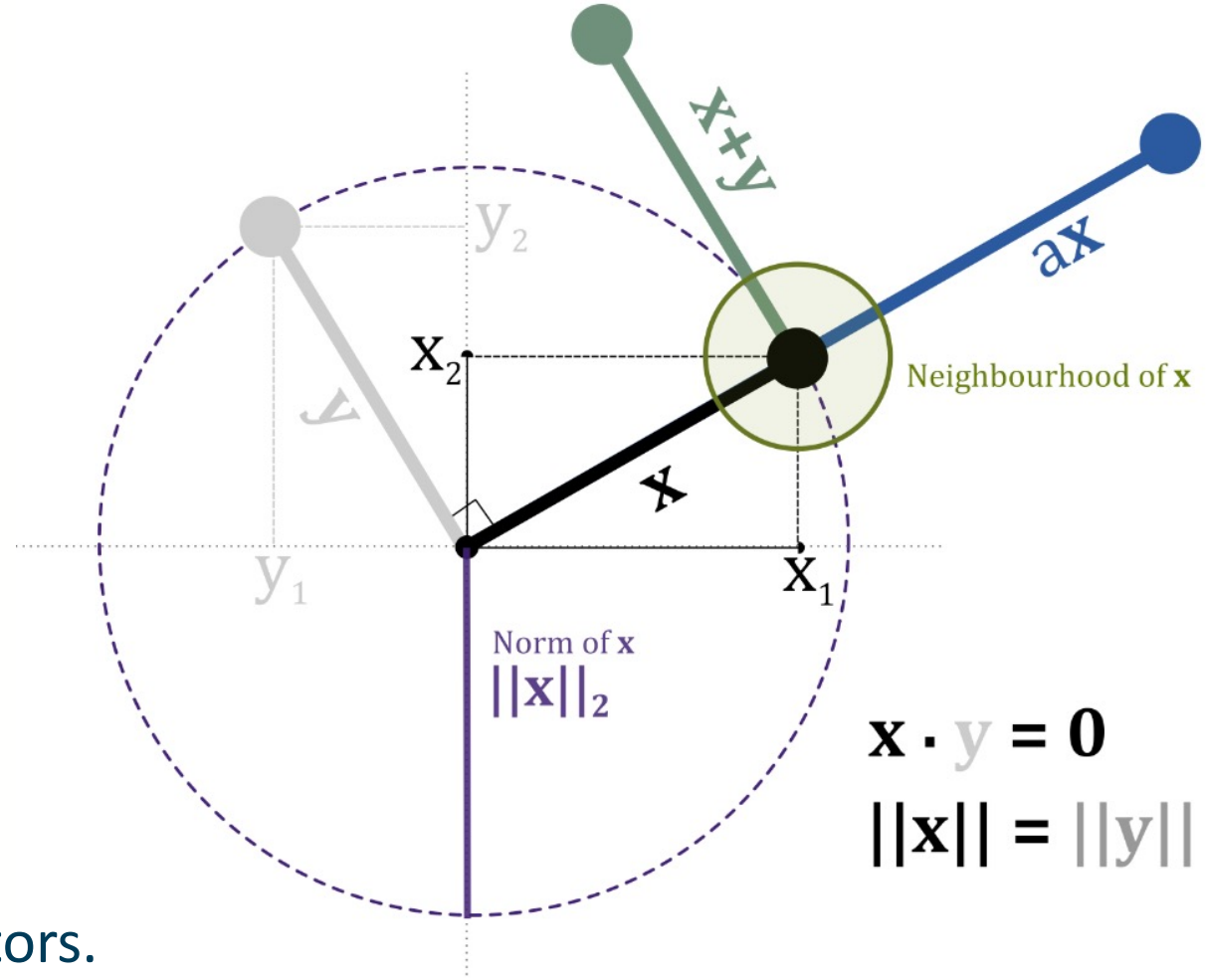
- With a norm a vector space is a **normed vector space**.
- With an inner product, a vector space is an **inner product space**
- Relationship between **Inner Product** and **Norm**:
  - If you have an inner product on a vector space, you can derive a norm from it: $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}|\mathbf{x}\rangle}$
  - However, not all norms come from an inner product.



University of Glasgow

Vectors

- **Points in space**
- **Arrows pointing from the origin**
- **Tuples of numbers**



These are all valid ways of thinking about vectors.

The "**points in space**" mental model is probably the most useful

- vectors to represent *data*; data lies in space
- matrices to represent *operations* on data; matrices warp space.

University of Glasgow

Relation to arrays

- 1D floating point arrays are we called "vectors"
- floating point numbers are NOT real numbers

```
# two 3D vectors (3 element ordered tuples)
x = np.array([0,1,1])
y = np.array([4,5,6])
a = 2
print_matrix("a", a)
print_matrix("x", x)
print_matrix("y", y)
```

$$a = 2$$

x

```
 [[0 1 1]]
```

y

```
 [[4 5 6]]
```

University of Glasgow

Uses of vectors

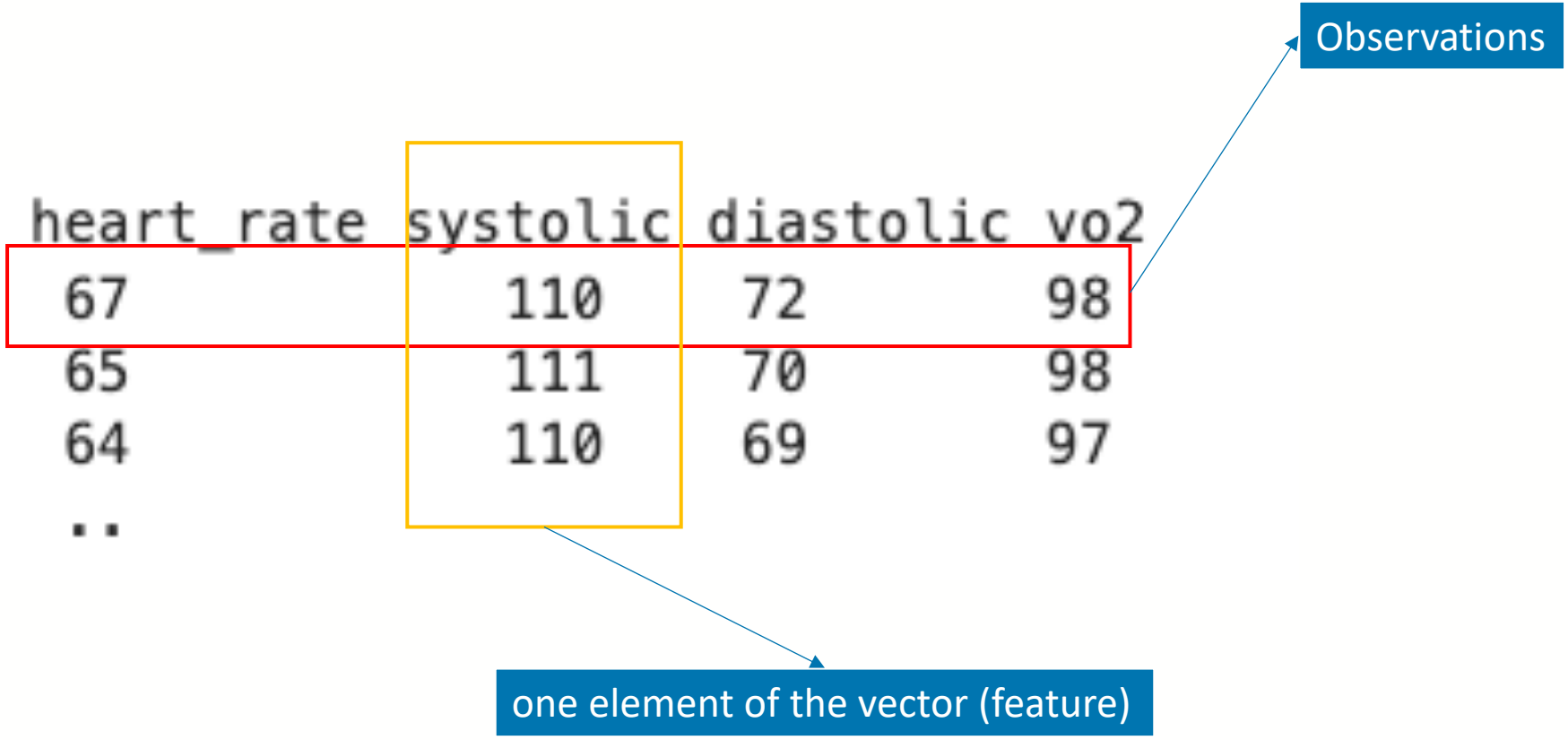
They are a *lingua franca* for data. Because vectors can be

- **composed** (via addition),
- **compared** (via norms/inner products)
- and **weighted** (by scaling),

In Numpy, they map onto the efficient **ndarray** data structure, so we can operate on them efficiently and concisely.

Vector data

- Datasets are commonly stored as 2D **tables**.



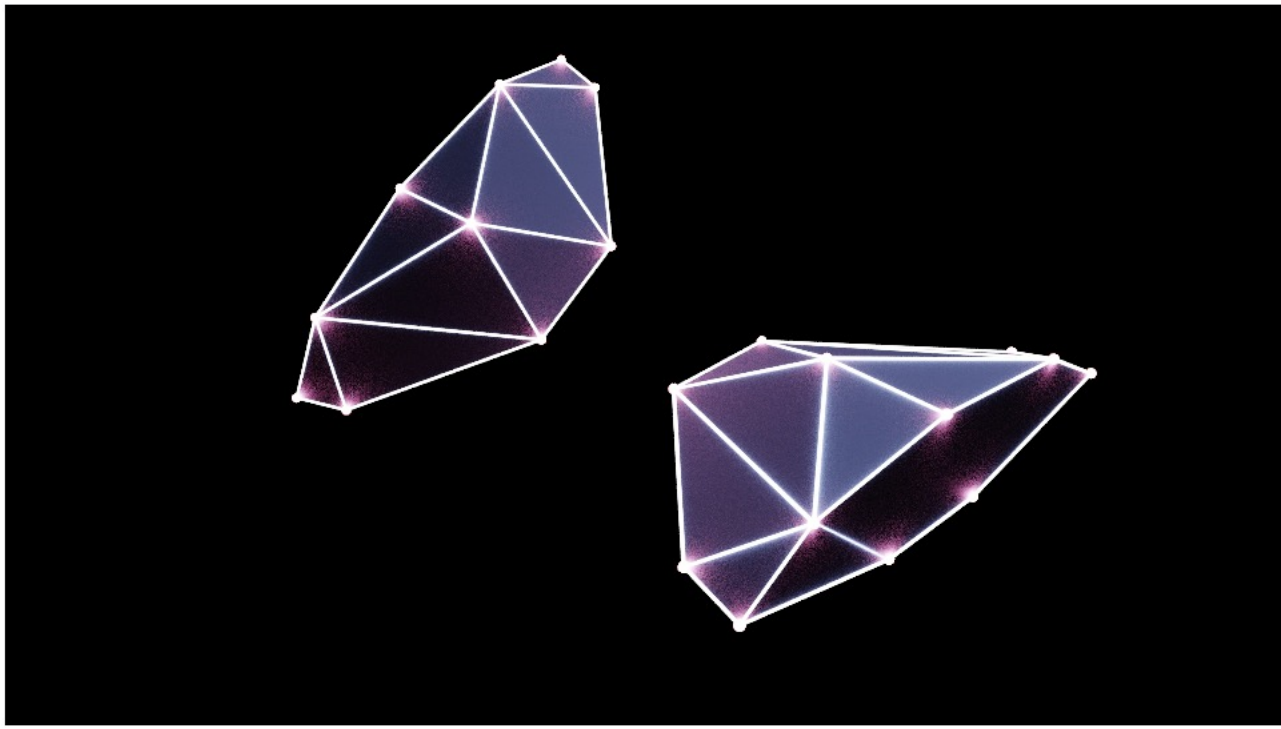
Each **row** can be seen as a vector in $\mathbb{R}^n$

University of Glasgow

Geometric operations

- Use vectors in 3D space
- Transformations in 3D space include:
  - scaling
  - rotation
  - flipping (mirroring)
  - translation (shifting)

```
[[ -0.   15.    0. ]
 [ 16.   -0.5  32.5]
 [-16.   -0.5  32.5]
 [ 16.  -15.  -32.5]
 [-16.  -15.  -32.5]
 [-44.   10.  -32.5]
 [-60.   -3.  -13. ]
 [-65.   -3.  -32.5]
 [ 44.   10.  -32.5]
 [ 60.   -3.  -13. ]
 [ 65.   -3.  -32.5]
 [ -0.   15.  -32.5]]
```



The *Cobra Mk. III* spaceship model above is defined by these vectors specifying the vertices in 3D space

University of Glasgow

Machine learning applications

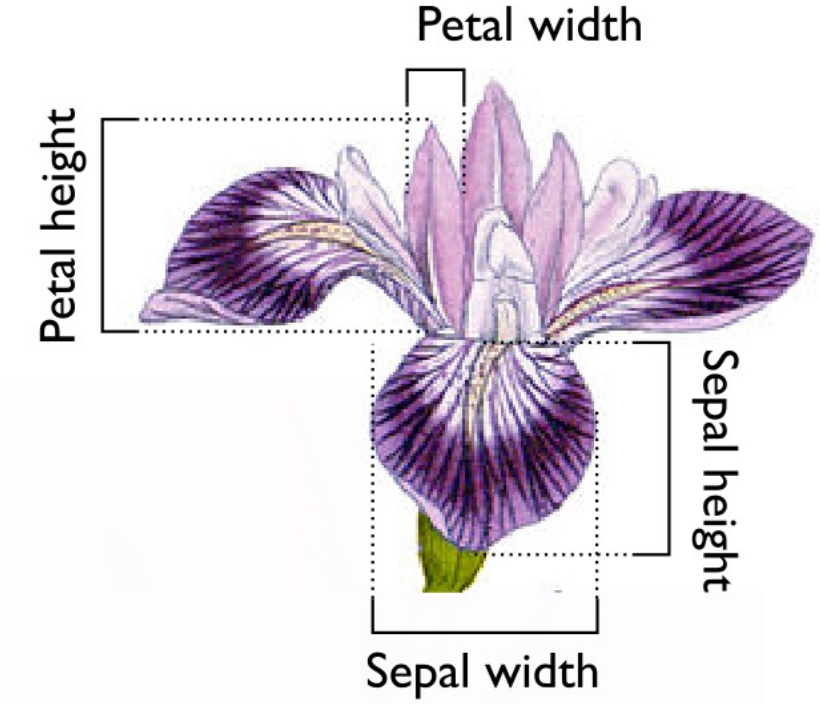
- Machine learning relies heavily on vector representation. A typical machine learning process involves:
  - transforming some data onto **feature vectors**
  - creating a function that transforms **feature vectors** to a prediction (e.g. a class label)

- *Most machine learning algorithms can be seen as doing geometric operations: comparing distances, warping space, computing angles, and so on.*
  - E.g. **k nearest neighbours**

Example: irises classification

The irises classification task:

- *The measurements of the dimensions of the sepals and petals of irises allows classification of species*
- ***Training data**: [sepal, petal]*
- ***Labels**: species of* irises

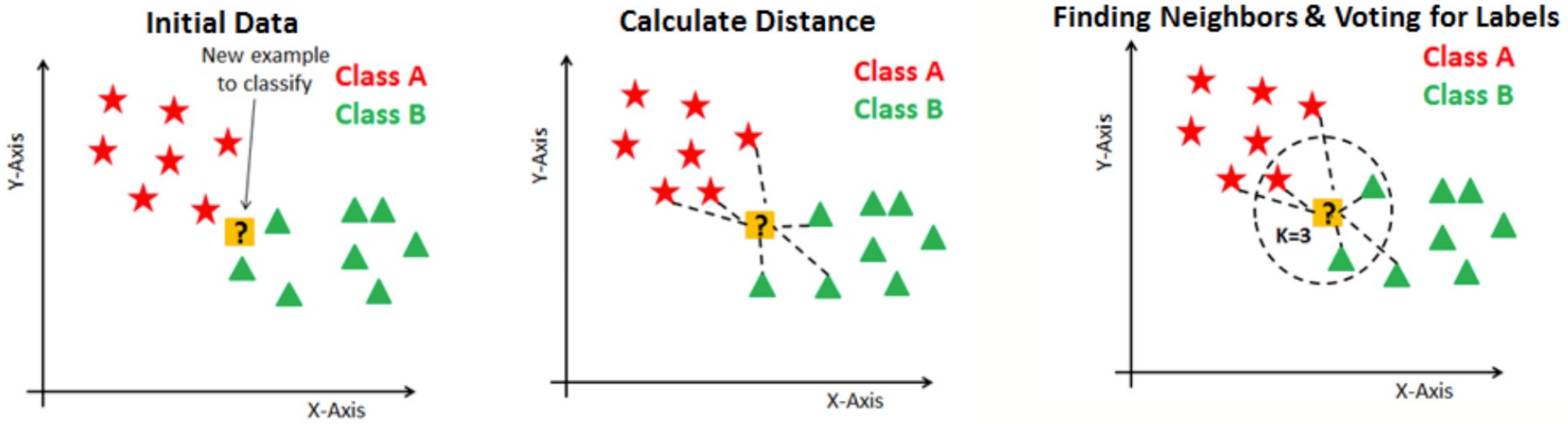


- ***k* nearest neighbours**
  - Using a **norm** to compute distances
  - The output prediction is the class label that occurs most times among these *k* neighbours

University of Glasgow

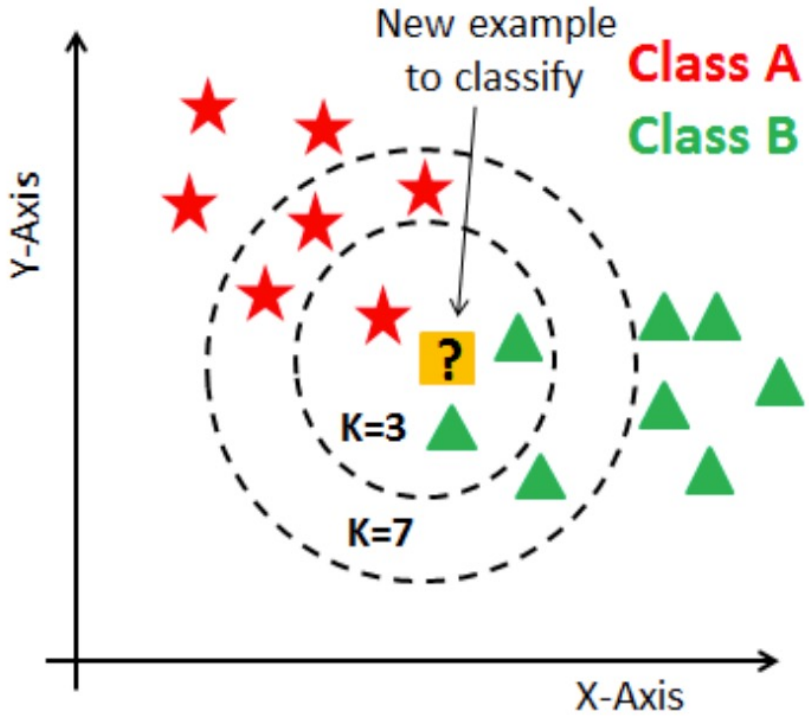
Example: irises classification

- ***k* nearest neighbours**
  - Calculate distance between **training examples** and the target using **norm**
  - Finding the closest *k* neighbors *(e.g. 3, 5, 10), voting for the* ***most majority***



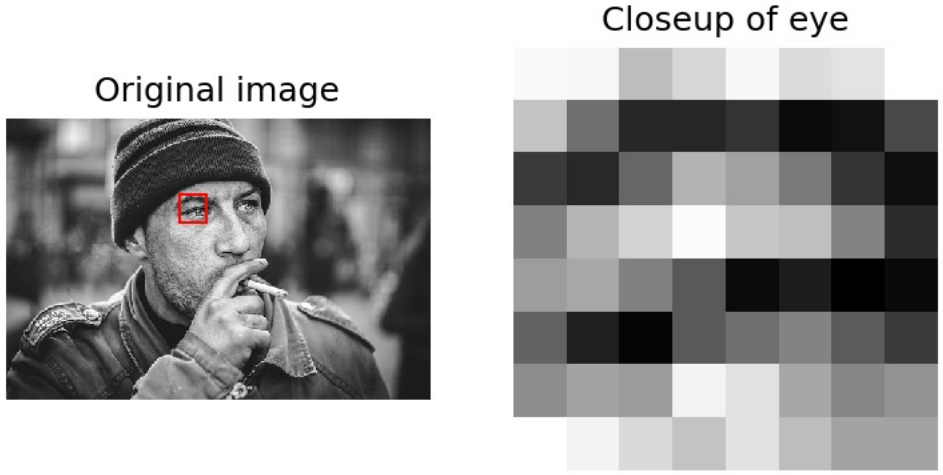
Example: irises classification

- *The choice of k might significantly affect the prediction result*
- *The distance function (norm or cosine similarity) is something need to be considered*

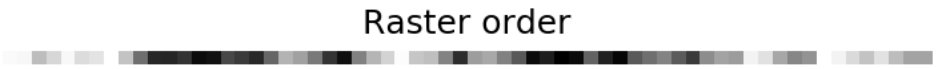


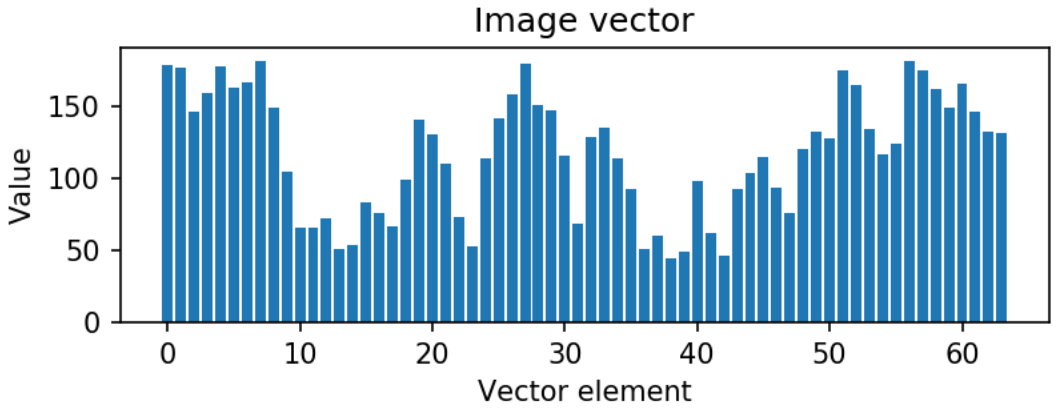
Example: Image compression

- Images can be represented as 2D arrays of brightness
- Groups of pixels -- for example, rectangular patches -- can be unraveled into a vector.
  - E.g. An 8x8 image patch would be unraveled to a 64-dimensional vector.



- Splitting images into patches, and treating each patch as a vector x_1,…,x_n
- The vectors are **clustered** to find a small number of vectors y_1,…,y_m, m<<n that are a reasonable approximation of nearby vectors.
- the vectors for the small number of representative vectors y_i are stored (the **codebook**)



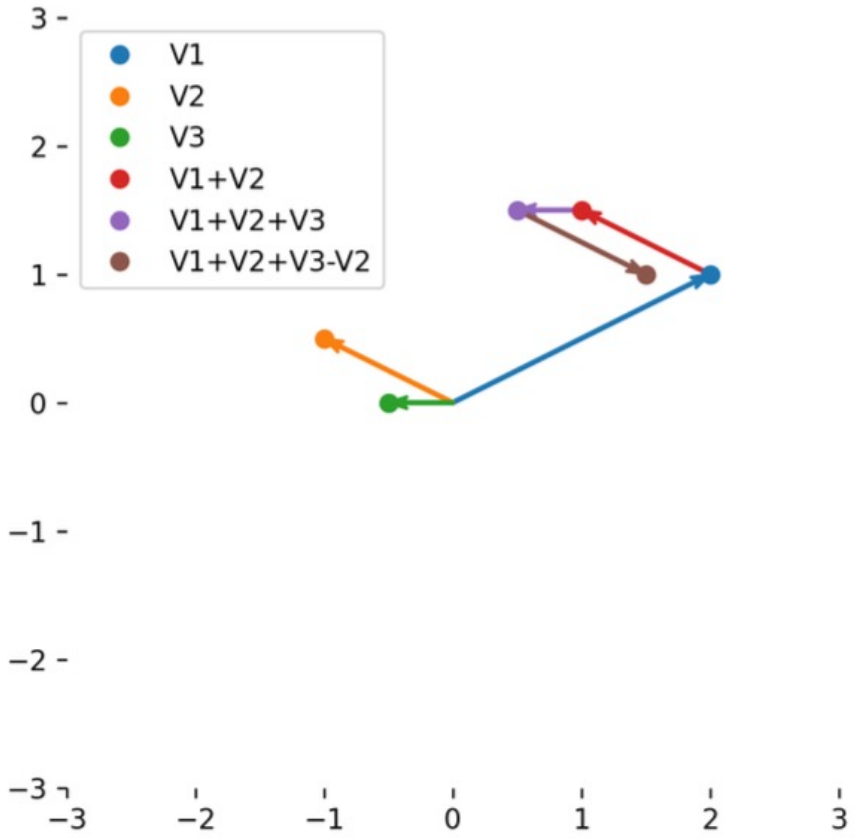


Basic vector operations

- Standard operations:
  - getting the length of vectors (**norm**)
  - computing dot (inner), outer and cross products.

- **Addition and multiplication --** form **weighted sums** of vectors

$$\lambda_1 \mathbf{x_1} + \lambda_2 \mathbf{x_2} + \cdots + \lambda_n \mathbf{x_n}$$



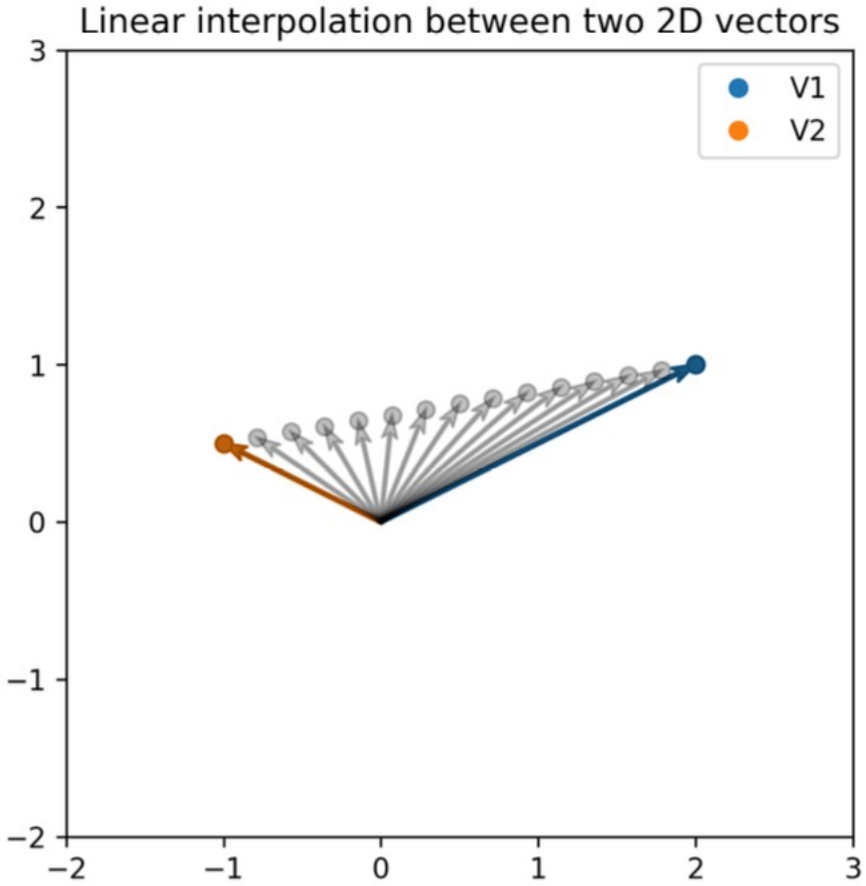
See the lecture note for codes

Basic vector operations

- Many standard statistics and operations can be directly applied.

- **Linear interpolation:** to construct **new** data points within the range of **known** data points.

$$\text{lerp}(\mathbf{x}, \mathbf{y}, \alpha) = (1 - \alpha)\mathbf{x} + (\alpha)\mathbf{y}$$



See the lecture note for codes

University of Glasgow

How big is that vector?

- The Euclidean length of a vector x (written as ||x||) can be computed directly using *np.linalg.norm()*.
- This is equal to:

$$\|\mathbf{x}\|_2 = \sqrt{x_0^2 + x_1^2 + x_2^2 + \cdots + x_n^2}$$

```
x = np.array([1.0, 10.0, -5.0])
y = np.array([1.0, -4.0, 8.0])
print_matrix("x", x)
print_matrix("y", y)

print_matrix("\|x\|", np.linalg.norm(x))
print_matrix("\|y\|", np.linalg.norm(y))
```

```
x
 [[ 1. 10. -5.]]
y
 [[ 1. -4.  8.]]
```

$$\|x\| = 11.224972160321824$$

$$\|y\| = 9.0$$

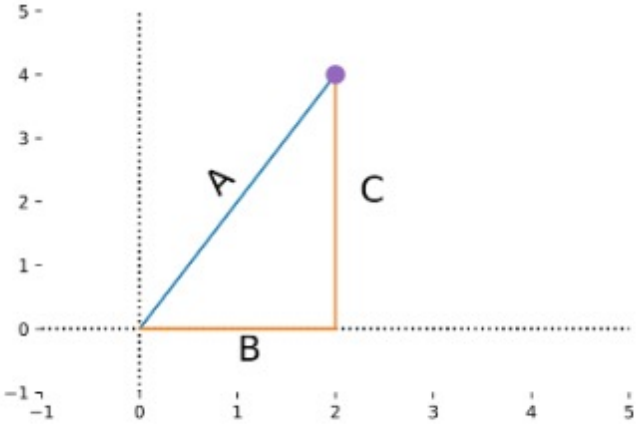
Different norms

- **Euclidean norm** or **Euclidean distance measure**

$$||\mathbf{x} - \mathbf{y}||_2$$

- L_p **-norms** or **Minkowski** norms, which is defined by:

$$\|\mathbf{x}\|_p = \left(\sum_i x_i^p \right)^{\frac{1}{p}}$$



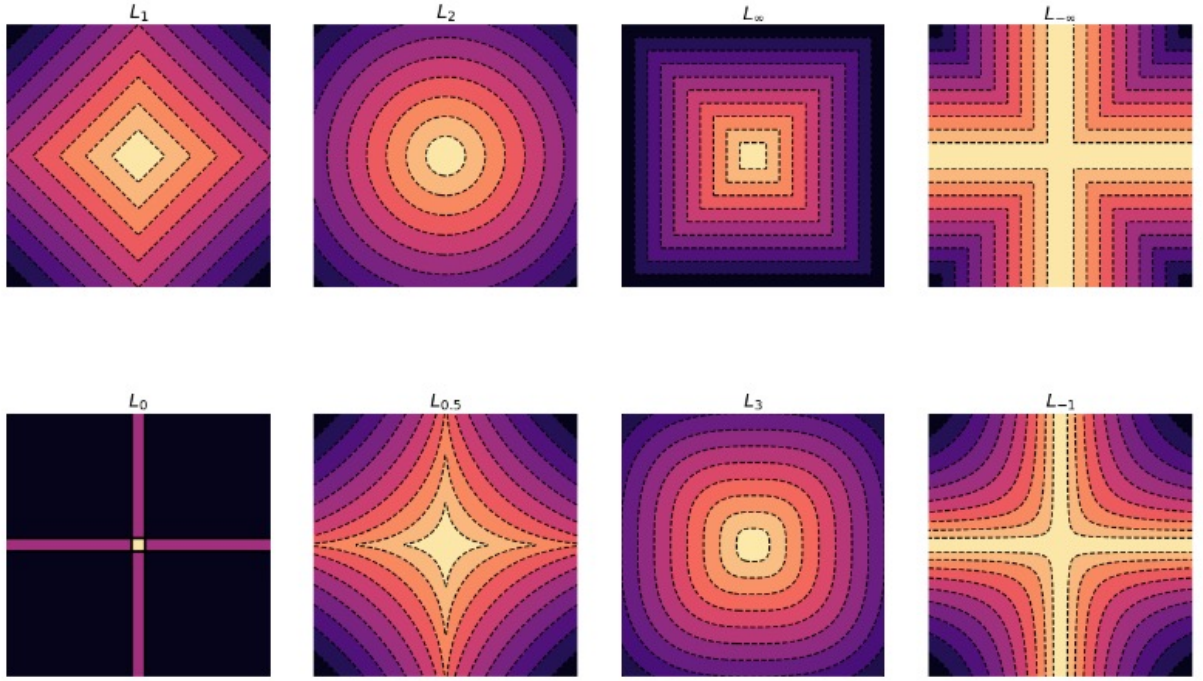
$L_2 = |A|$

$L_1 = |B| + |C|$

$L_\infty = \max(|B|, |C|) = |C|$

Different norms

p	Notation	Common name	Effect	Uses	Geometric view
2	$\|x\|$ or $\|x\|_2$	Euclidean norm	Ordinary distance	Spatial distance measurement	Sphere just touching point
1	$\|x\|_1$	Taxicab norm; Manhattan norm	Sum of absolute values	Distances in high dimensions, or on grids	Axis-aligned steps to get to point
0	$\|x\|_0$	Zero pseudo-norm; non-zero sum	Count of non-zero values	Counting the number of "active elements"	Numbers of dimensions not touching axes
∞	$\|x\|_{\inf}$	Infinity norm; max norm	Maximum element	Capturing maximum "activation" or "excursion"	Smallest cube enclosing point
$-\infty$	$\|x\|_{-\inf}$	Min norm	Minimum element	Capturing minimum excursion	Distance of point to closest axis

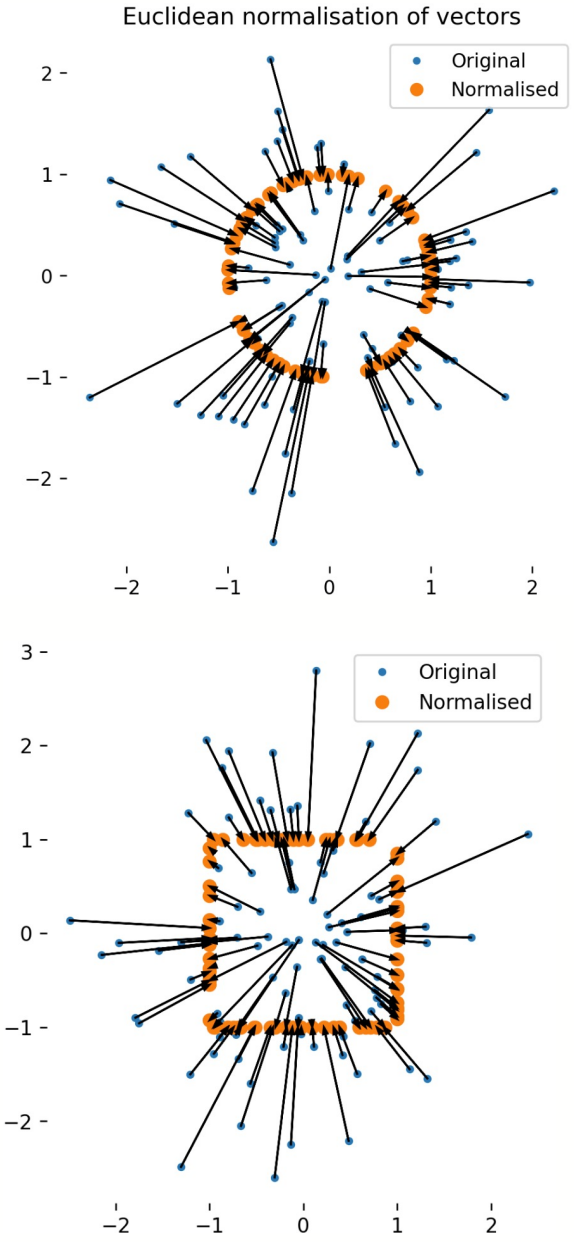


Every dashed line has the same distance to the origin as measured in that norm. The points of equal distance in that norm appear as a connected line.

Unit vectors and normalisation

- A unit vector has norm 1 (the definition of a unit vector depends on the norm used)
  - Normalising for the Euclidean norm can be done by scaling the vector x by $\frac{1}{\|x\|_2}$

- If we think of vectors in the physics sense of having a **direction** and **length**, a unit vector is "pure direction".



University of Glasgow

Inner products of vectors

- An **inner product** $(\mathbb{R}^N \times \mathbb{R}^N) \rightarrow \mathbb{R}$ measures the **angle** between two real vectors.
  - It is related to the cosine distance:

$$\cos\theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\| \mathbf{x} \| \| \mathbf{y} \|}.$$

  - For unit vectors, we can forget about the denominator, since $\| \mathbf{x} \| = 1, \| \mathbf{y} \| = 1$, so $\cos\theta = \mathbf{x} \cdot \mathbf{y}$.
- The computation of the **inner product**, for real-valued vectors in $\mathbb{R}^N$, is simply the sum of the elementwise products:

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$$

Inner products of vectors

- **Inner product** in Numpy

```
x = np.array([1, 2, 3, 4])
y = np.array([4, 0, 1, 4])
print_matrix("x", x)
print_matrix("y", y)

print_matrix("x\cdot y", np.inner(x, y))
```

```
x
 [[1 2 3 4]]
y
 [[4 0 1 4]]
```

$x \cdot y = 23$

- The inner product is only defined between vectors of the same dimension, and only in inner product spaces.
  - ValueError

- **Don't mix it with the dot product**

numpy.dot(*a*, *b*, *out=None*)

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using `matmul` or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to `multiply` and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where `M>=2`), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

For 2-D arrays it is the matrix product:

```
>>> a = [[1, 0], [0, 1]]
>>> b = [[4, 1], [2, 2]]
>>> np.dot(a, b)
array([[4, 1],
       [2, 2]])
```

University of Glasgow

Basic vector statistics

Some **statistics** that generalise the statistics of ordinary real numbers

- **mean vector** of a collection of N vectors

$$\mathrm{mean}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = \frac{1}{N} \sum_i \mathbf{x_i}$$

- The mean vector is the **geometric centroid** of a set of vectors and can be thought of as capturing "centre of mass" of those vectors.

numpy.mean #

```
numpy.mean(a, axis=None, dtype=None, out=None,
keepdims=<no value>, *, where=<no value>)
```
[source]

axis=0 for the purpose of mean vector

Examples

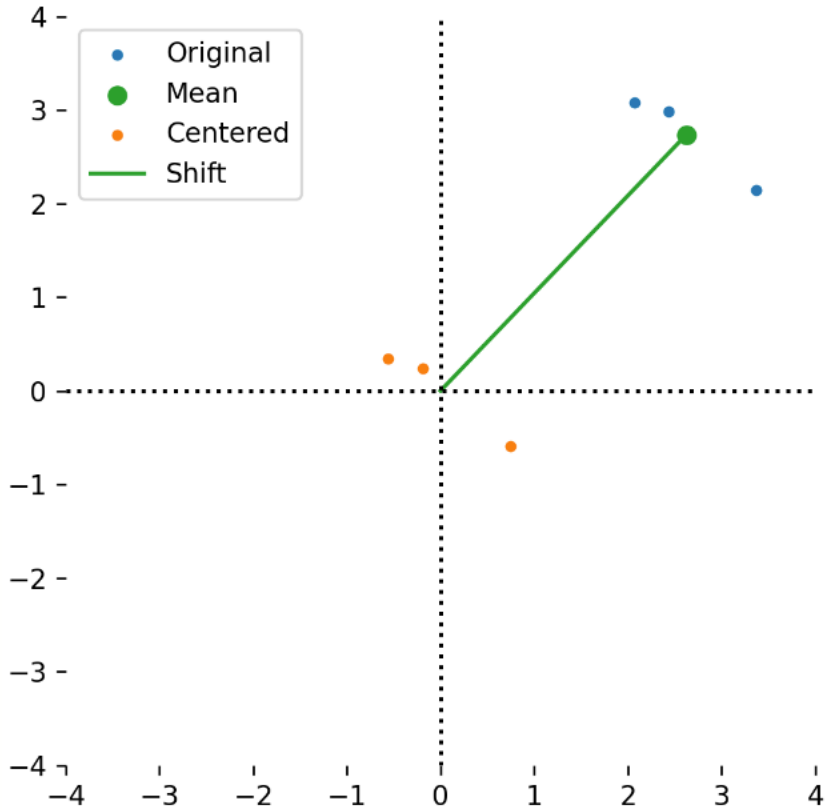
```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([2., 3.])
>>> np.mean(a, axis=1)
array([1.5, 3.5])
```

Center a dataset

- We can **center** a dataset stored as an array of vectors to **zero mean** by just subtracting the mean vector from every row.

```
x_center = x - mu
print_matrix("x_center", x_center)
mu_c = np.mean(x_center, axis=0) #
print_matrix("\mu_c", mu_c)
```

```
x_center
 [[-0.56  0.35 -1.3  -0.36]
 [-0.19  0.24  1.56  0.4 ]
 [ 0.75 -0.59 -0.27 -0.04]]
\mu c
[[ 0.  0.  0. -0.]]
```



University of Glasgow

High-dimensional vector spaces

- Data science often involves **high dimensional vector spaces**
  - High-dimensional can mean any d>3;
  - a 20-dimensional feature set might be called medium-dimensional;
  - a 1000-dimensional might be called high-dimensional;
  - a 1M-dimensional dataset might be called extremely high-dimensional

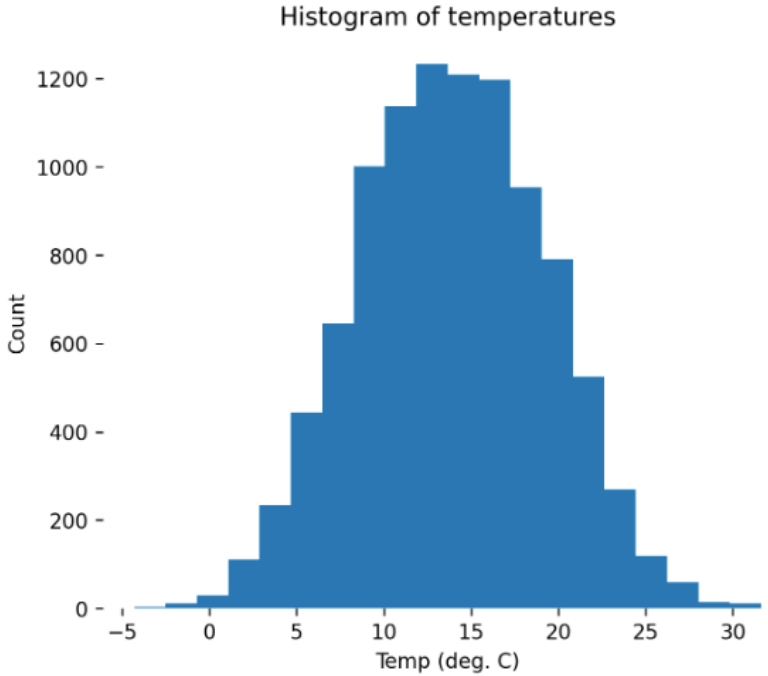
- **Curse of dimensionality:** Many algorithms that work really well in low dimensions break down in higher dimensions.

Example: sailing weather station

- **Task:** to measure local atmospheric conditions during voyages
- **Input**: wind speed, temperature, humidity, sunshine hours, etc., over 10,000 measurements
- **Output:** *is it likely to be above 30C tomorrow?*

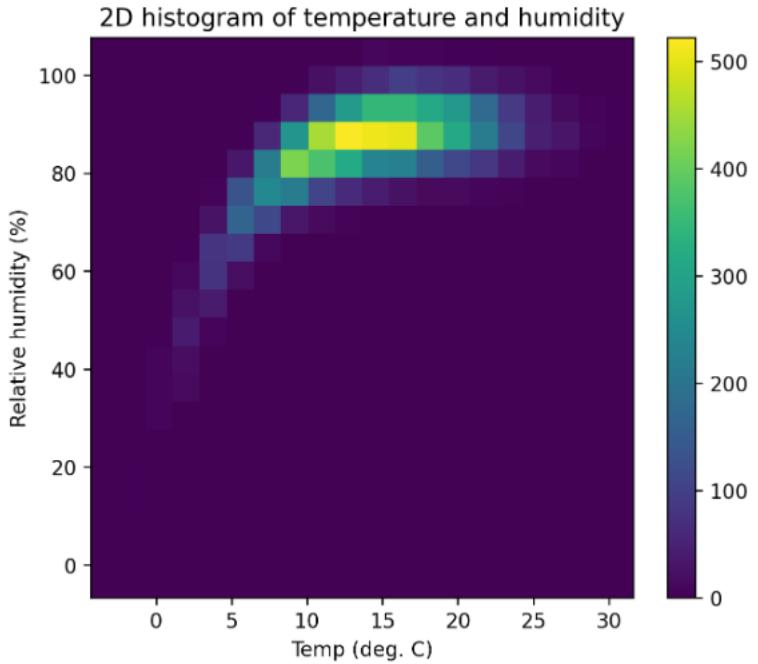
Example: sailing weather station

Temperature (1D)



Now there are 20 bins in total for this dimension

Temperature & Humidity (2D)



Now there are 20 bins in each dimension, for 400 bins total.

Example: sailing weather station

- If we had 10 different measurements (air temperature, air humidity, latitude, longitude, wind speed, wind direction, precipitation, time of day, solar power, sea temperature)
  - We wanted to subdivide them into 20 bins each
  - How many bins in total?

University of Glasgow

Example: sailing weather station

- If we had 10 different measurements (air temperature, air humidity, latitude, longitude, wind speed, wind direction, precipitation, time of day, solar power, sea temperature)
  - We wanted to subdivide them into 20 bins each
  - How many bins in total?

    We would need a histogram with 20^10 bins --
    over ***10 trillion*** bins.

  - even using 8 bit unsigned integers this would be 10TB of memory
- But we only have 10,000 measurements

Example: sailing weather station

- If we had 10 different measurements (air temperature, air humidity, latitude, longitude, wind speed, wind direction, precipitation, time of day, solar power, sea temperature)
  - We wanted to subdivide them into 20 bins each
  - How many bins in total?

    We would need a histogram with 20^10 bins --
    over ***10 trillion*** bins.

  - even using 8 bit unsigned integers this would be 10TB of memory
- But we only have 10,000 measurements

Curse of dimensionality: as dimension increases generalisation gets harder *exponentially*

University of Glasgow

Matrices

Matrices are 2D arrays of reals that define **linear maps**;

- Vectors represent “points in space"
- Matrices represent *operations* that do things to those points in space.

The operations represented by matrices are a particular class of functions on vectors

University of Glasgow

Operations with matrices

There are many things we can do with matrices:

- They can be **added** and **subtracted** $C = A + B$

$$(\mathbb{R}^{n\times m}, \mathbb{R}^{n\times m}) \to \mathbb{R}^{n\times m}$$

- They can be **scaled** with a scalar $C = sA$

$$(\mathbb{R}^{n\times m}, \mathbb{R}) \to \mathbb{R}^{n\times m}$$

- They can be **transposed** $B = A^T$; this exchanges rows and columns

$$\mathbb{R}^{n\times m} \to \mathbb{R}^{m\times n}$$

- They can be **applied** to vectors $\mathbf{y} = A\mathbf{x}$; this applies a matrix to a vector.

$$(\mathbb{R}^{n\times m}, \mathbb{R}^{m}) \to \mathbb{R}^{n}$$

- They can be **multiplied** together $C = AB$; this composes the effect of two matrices

Intro to matrix notation

- We write matrices as a capital letter, e.g. ***A***:

$$A \in \mathbb{R}^{n \times m} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & \dots & a_{2,m} \\ \dots & & & \\ a_{n,1}+ & a_{n,2} & \dots & a_{n,m} \end{bmatrix}, \; a_{i,j} \in \mathbb{R}$$

where each element of the matrix ***A*** is written as $\mathbf{A}_{i,j}$, for the *i*th row and *j*th column.

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

print_matrix("A", a)
# note that code indexes from 0

print_matrix("A_{1,3}", a[0,2])
```

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

$A_{1,3} = 3$

Matrices as maps

- We saw vectors as **points in space**, and matrices as **vector transform in space**.

- Matrices represent **linear maps** -- these are functions applied to vectors which outputs vectors. (applying some function f(x) to the vectors)

$$A\mathbf{x} = f(\mathbf{x})$$

- A n×m matrix ***A*** represents a function f(x) taking m dimensional vectors to n dimensional vectors ($\mathbb{R}^m \rightarrow \mathbb{R}^n$)
- Matrices capture a special set of functions that preserve important properties of the vectors they act on.

University of Glasgow

Linear maps

Linearity

- the transform of the sum of two vectors is the same as the sum of the transform of two vectors
- the transform of a scalar multiple of a vector is the same as the scalar multiple of the transform of a vector

$$\begin{aligned} f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \quad &= A(\mathbf{x}+\mathbf{y}) = A\mathbf{x} + A\mathbf{y}, \\ f(c\mathbf{x}) = cf(\mathbf{x}) \quad &= A(c\mathbf{x}) = cA\mathbf{x}, \end{aligned}$$

Anything which is linear is easy. Anything which isn't linear is hard.

University of Glasgow

Transforms and projections

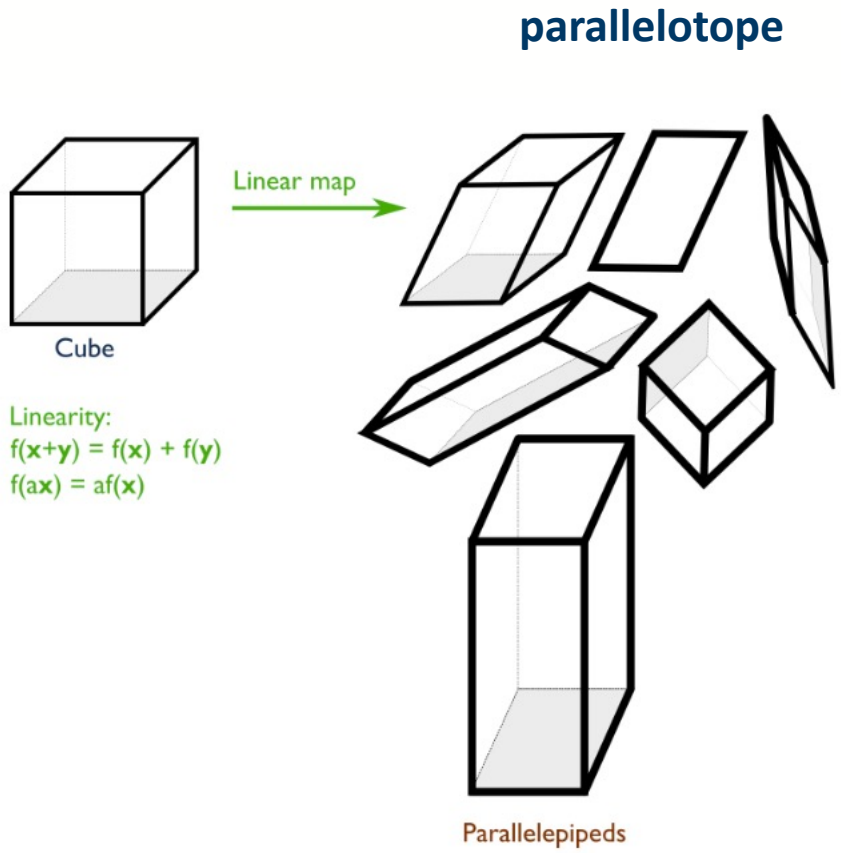
- A **linear map** is any function $f: R^m \rightarrow R^n$ which satisfies the linearity requirements.
- If the map represented by the matrix is $n \times n$ then it maps from a vector space onto the same vector space (e.g. from $\mathbb{R}^n \rightarrow \mathbb{R}^n$), and it is called a **linear transform.**
- If the map has the property $Ax = AAx$ or equivalently $f(x) = f(f(x))$ then the operation is called a **linear projection**;
  - for example, projecting 3D points onto a plane; applying this transform to a set of vectors twice is the same as applying it once.

Matrices represent **linear maps** or **linear functions**.

Geometric intuition (cube -> parallelepiped)

A matrix to transform a **cube** of vector space centered on the origin in one space to a **parallelotope** in another space, with the origin staying fixed.

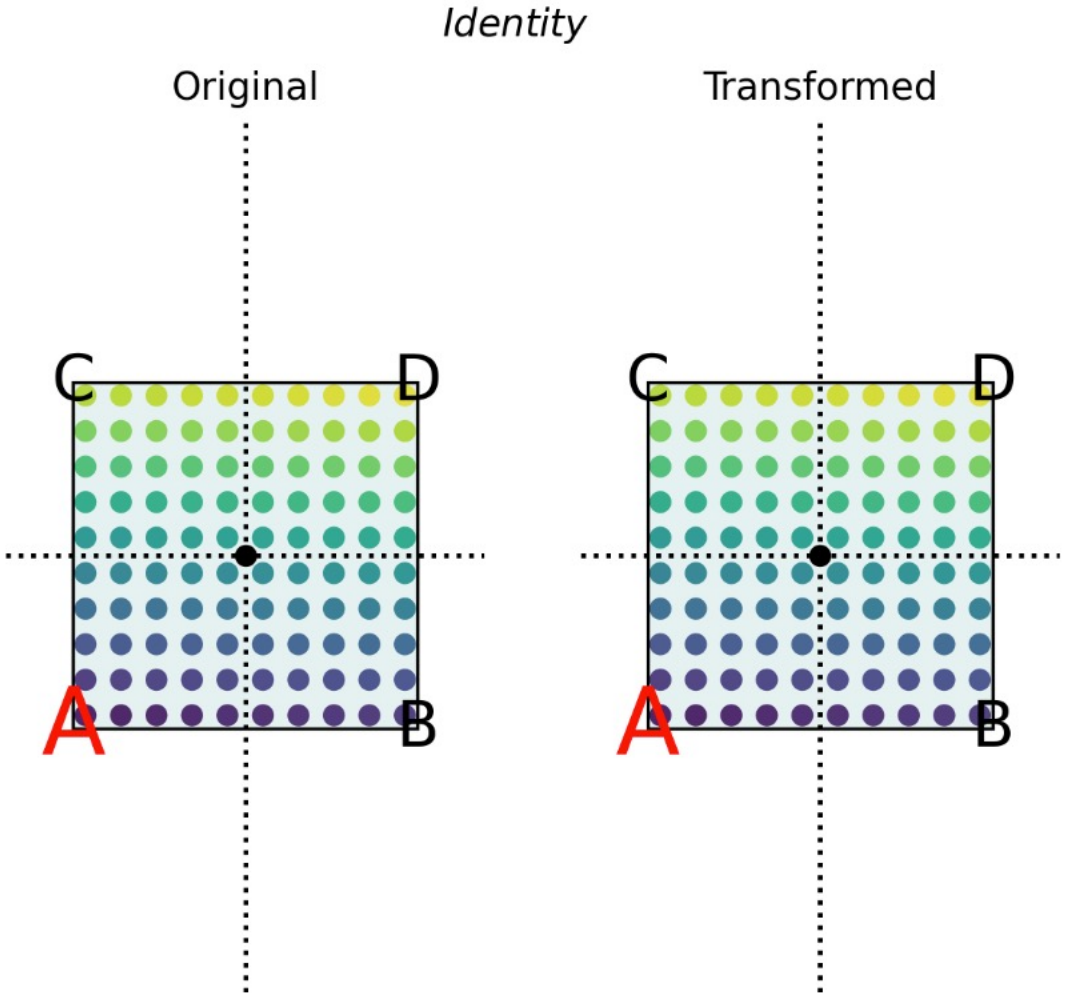
- A **parallelotope** is the generalisation of a **parallelogram**
- A **parallelogram** is a **2-parallelotope**
- A **parallelepiped** is a **3-parallelotope**
- Matrix could possibly transform a **cube (3D)** into a **2-parallelotope**



Examples - effect of linear transforms

- linear transforms (linear maps $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$: $\mathbb{R}^2 \to \mathbb{R}^2$), on the 2D plane.
- We forms the product $\boldsymbol{A}$x, which "applies" the matrix to the vector x.

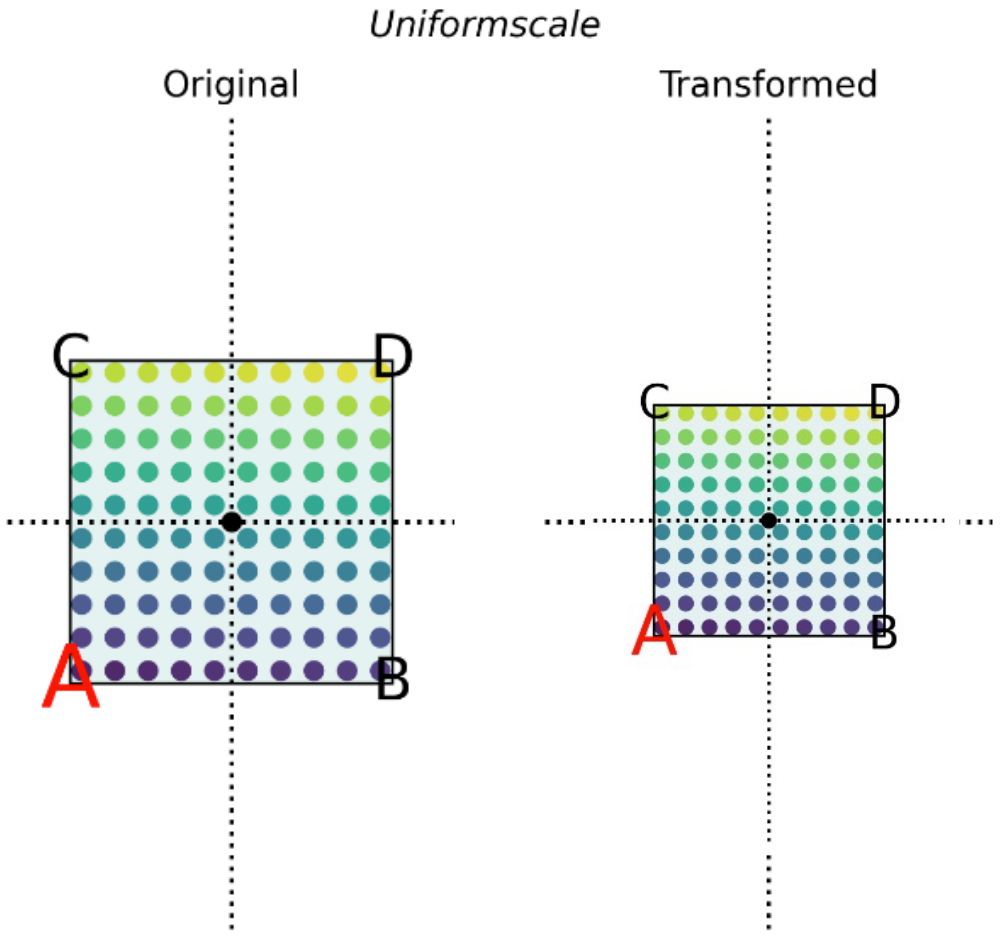
```
Identity
 [[1 0]
 [0 1]]
```



Examples - effect of linear transforms

- linear transforms (linear maps $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$: $\mathbb{R}^2 \to \mathbb{R}^2$), on the 2D plane.
- We forms the product $\boldsymbol{A}$x, which "applies" the matrix to the vector x.

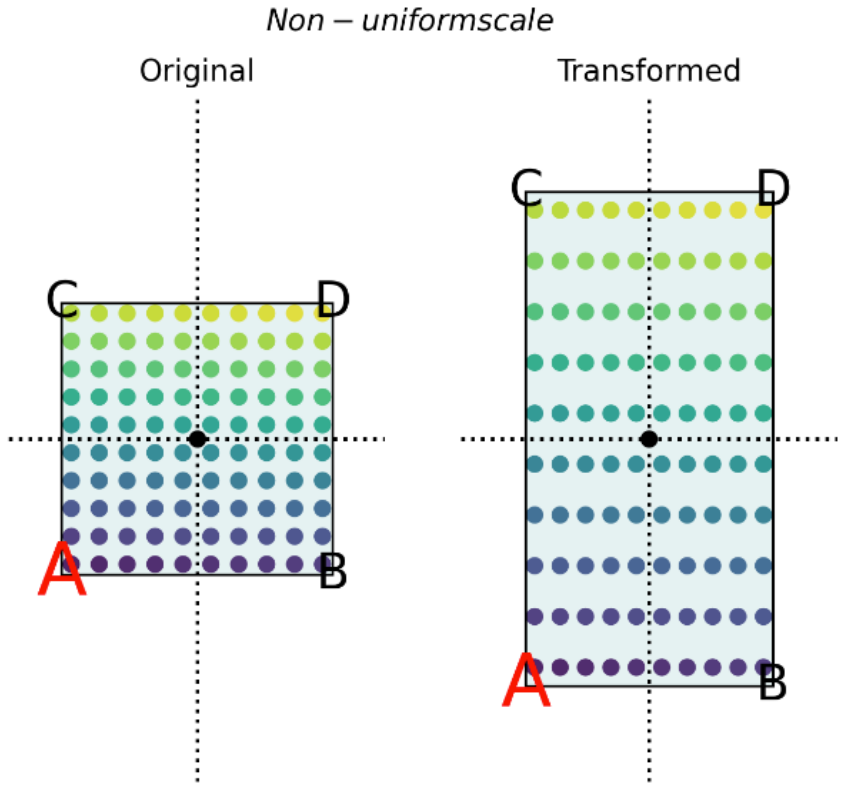
```
Uniform scale
 [[0.5 0. ]
 [0.  0.5]]
```



Examples - effect of linear transforms

- linear transforms (linear maps $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$: $\mathbb{R}^2 \to \mathbb{R}^2$), on the 2D plane.
- We forms the product $\boldsymbol{A}$x, which "applies" the matrix to the vector x.

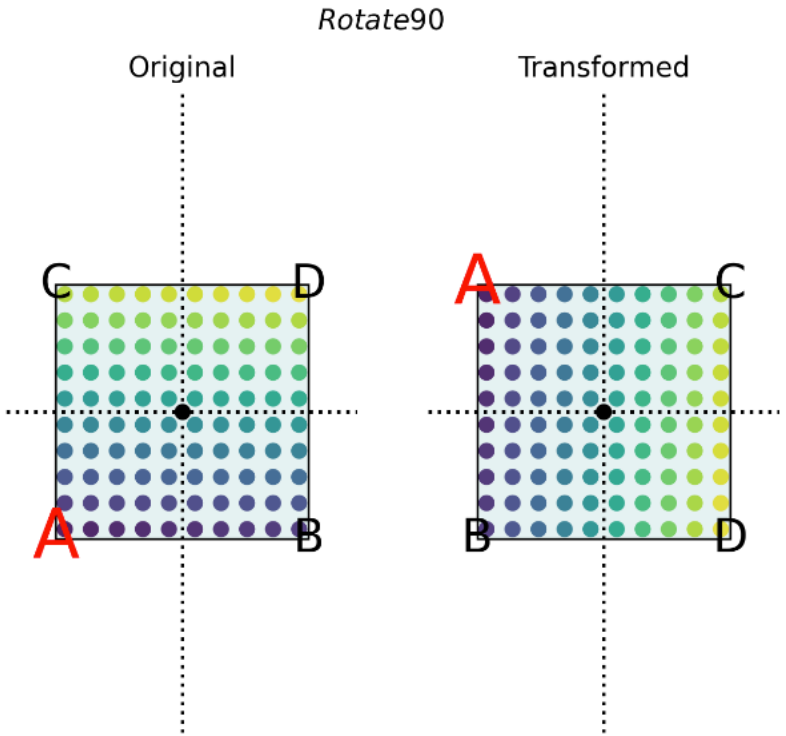
```
Non-uniform scale
 [[0.5 0. ]
 [0.  1. ]]
```



Examples - effect of linear transforms

- linear transforms (linear maps $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$: $\mathbb{R}^2 \to \mathbb{R}^2$), on the 2D plane.
- We forms the product $\boldsymbol{A}$x, which "applies" the matrix to the vector x.

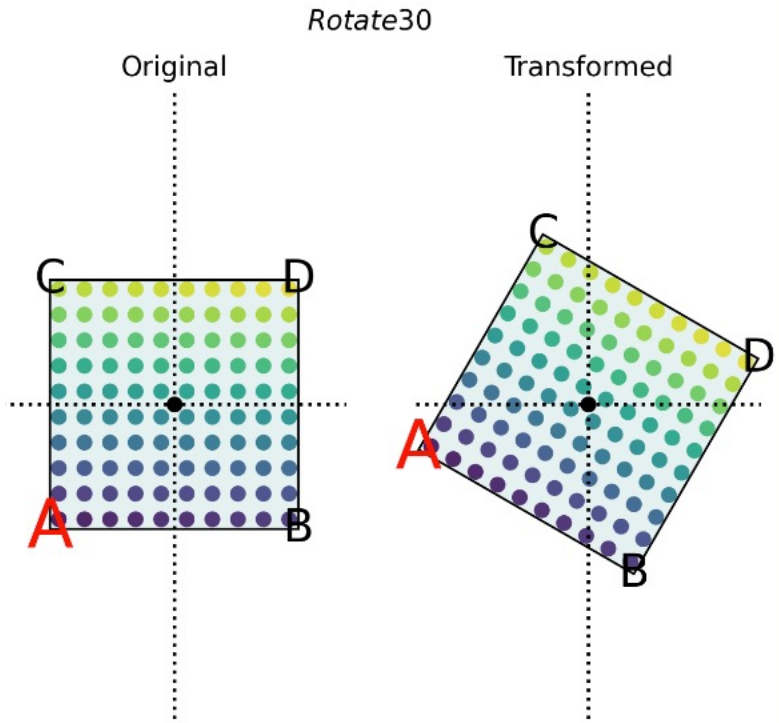
```
Rotate 90
 [[ 0  1]
 [-1  0]]
```



Examples - effect of linear transforms

- linear transforms (linear maps $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$: $\mathbb{R}^2 \to \mathbb{R}^2$), on the 2D plane.
- We forms the product $\boldsymbol{A}$x, which "applies" the matrix to the vector x.

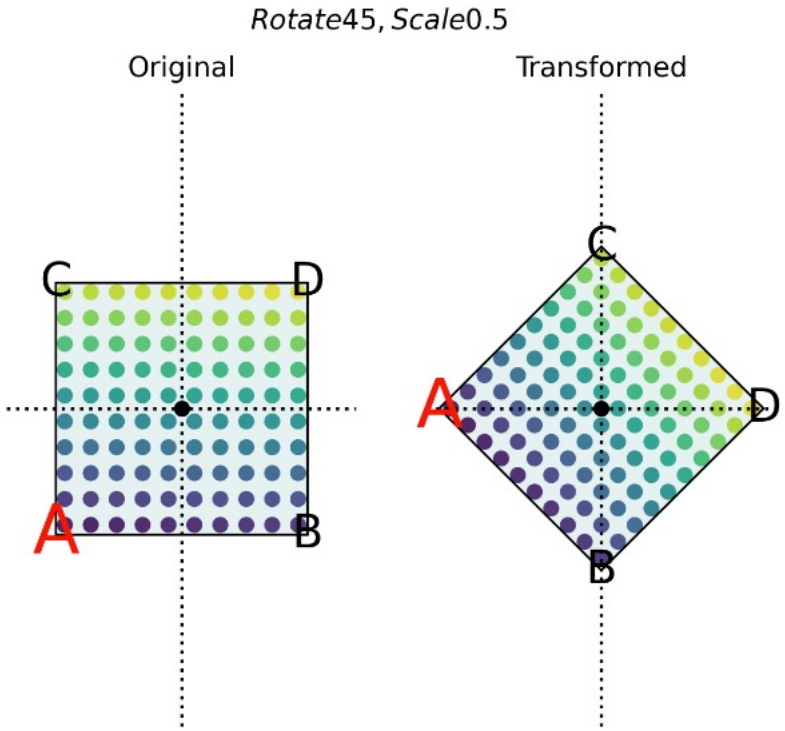
```
Rotate 30
 [[ 0.87   0.5 ]
 [-0.5    0.87]]
```



Examples - effect of linear transforms

- linear transforms (linear maps $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$: $\mathbb{R}^2 \to \mathbb{R}^2$), on the 2D plane.
- We forms the product $\boldsymbol{A}$x, which "applies" the matrix to the vector x.

```
Rotate 45, Scale 0.5
 [[ 0.35  0.35]
 [-0.35  0.35]]
```



University of Glasgow

Matrix operations

The addition of matrices of *equal size* is simple elementwise addition.

$$A + B = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & \dots & a_{1,m} + b_{1,m} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & \dots & a_{2,m} + b_{2,m} \\ \dots & & & \\ a_{n,1} + b_{n,1} & a_{n,2} + b_{n,2} & \dots & a_{n,m} + b_{n,m} \end{bmatrix}$$

University of Glasgow

Matrix operations

The scalar multiplication cA is to multiply each element by c.

$$cA = \begin{bmatrix} ca_{1,1} & ca_{1,2} & \dots & ca_{1,m} \\ ca_{2,1} & ca_{2,2} & \dots & ca_{2,m} \\ \dots & & & \\ ca_{n,1} & ca_{n,2} & \dots & ca_{n,m} \end{bmatrix}$$

University of Glasgow

Application to vectors

- We can apply a matrix to a vector, equivalent to applying the function f(x) to x.

$$A\mathbf{x} = f(\mathbf{x})$$

- If A is $\mathbb{R}^{n\times m}$, and $\mathbf{x}$ is $\mathbb{R}^{m}$, then this will map from an m dimensional vector space to an n dimensional vector space: $(\mathbb{R}^{n\times m}, \mathbb{R}^{m}) \rightarrow \mathbb{R}^{n}$.

- **All application of a matrix to a vector does is form a weighted sum of the elements of the vector.** This is a linear combination (equivalent to a "weighted sum") of the components.

University of Glasgow

Application to vectors

In particular, we take each element of $\mathbf{x}, x_1, x_2, \ldots, x_m$, multiply it with the corresponding column of A, and sum these columns together.

- Set $\mathbf{y} = [0,0,0, \ldots] = 0^n$ (the n-dimensional zero vector)
- For each column $1 \leq i \leq m$ in A
- $\mathbf{y} = \mathbf{y} + x_i A_i$. Note that $x_i A_i$ is scalar times vector, and has n elements. A_i here means the i th column of A.

University of Glasgow

Let’s have a try

f(x)=Ax

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \text{ and } x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \; Ax = ?$$

University of Glasgow

Let's have a try

f(x)=Ax

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \text{ and } x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \; Ax = ?$$

The traditional matrix multiplication approach (row-by-column)

The matrix-vector product is:

$$Ax = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1(1) + 2(2) \\ 3(1) + 4(2) \\ 5(1) + 6(2) \\ 7(1) + 8(2) \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \\ 17 \\ 23 \end{bmatrix} \text{ So, } Ax = \begin{bmatrix} 5 \\ 11 \\ 17 \\ 23 \end{bmatrix}.$$

Let's have a try

f(x)=Ax

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \text{ and } x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad Ax = ?$$

An alterative way :

1. Multiply the first element of (x) with the first column of (A):
   $1 \times [1, 3, 5, 7] = [1, 3, 5, 7]$
2. Multiply the second element of (x) with the second column of (A):
   $2 \times [2, 4, 6, 8] = [4, 8, 12, 16]$
3. Sum the results from steps 1 and 2 element-wise:
   $[1, 3, 5, 7] + [4, 8, 12, 16] = [5, 11, 17, 23]$

So, the result is the same as before: ([5, 11, 17, 23]).

University of Glasgow

Application to vectors

- We can use @ to form products of vectors and matrices in Numpy

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \text{ and } x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad Ax = ?$$

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

print_matrix("A", A)
print_matrix("\\bf x", x)
print_matrix("A\\bf x", A @ x)
```

```
A
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
\bf x
 [[1 2]]
A\bf x
 [[ 5 11 17 23]]
```

Matrix multiplication

- **Matrix multiplication** defines the product $\boldsymbol{C}=\boldsymbol{AB}$, where $\boldsymbol{A,B,C}$ are all matrices.
- Matrix multiplication is defined such that if $\boldsymbol{A}$ represents linear transform $f(\mathbf{x})$ and $\mathbf{B}$ represents linear transform $g(\mathbf{x})$, then $\boldsymbol{BA}\mathbf{x}=g(f(\mathbf{x}))$
- **Multiplying two matrices is equivalent to composing the linear functions they represent, and it results in a matrix which has that affect.**

- *Note that the composition of linear maps is read right to left. To apply the transformation **A, then B**, we form the product **BA**, and so on.*

University of Glasgow

Multiplication algorithm

- *If* ***C=AB*** *then*

$$C_{ij} = \sum_k a_{ik} b_{kj}$$

- **Multiplication** is *only* defined for two matrices **A,B** if:
  - **A** is p×*q* and
  - **B** is *q*×r.

- This gives rise to many important uses of matrices, for example, the product of a scaling matrix and a rotation matrix is a scale-and-rotate matrix.

University of Glasgow

Multiplication algorithm

- Matrix multiplication is of course built in to NumPy

- Matrix multiplication is applied by np.dot(a,b) or by the syntax a @ b

```
# verify that this is the same as the buil
c_numpy = np.dot(a, b)
print_matrix("C_{\\text numpy}", c_numpy)
print(np.allclose(c, c_numpy))

c_at = a @ b
print_matrix("C_{\\text a @ b}", a @ b)
print(np.allclose(c_at, c))
```

```
C_{\text numpy}
 [[-4  4 -4]]
True
C_{\text a @ b}
 [[-4  4 -4]]
True
```

University of Glasgow

Time complexity of multiplication

- Matrix multiplication has, in the general case, of time complexity O(pqr), or for multiplying two square matrices $O(n^3)$.
- However, there are many special forms of matrices for which this complexity can be reduced, such as diagonal, triangular, sparse and banded matrices. (later)
- There are some accelerated algorithms for general multiplication. The time complexity of all of them is $>O(N^2)$ but $<O(N^3)$.

University of Glasgow

Transposition

- The **transpose** of a matrix A is written A^T and has the same elements, but with the rows and columns exchanged.
- Two ways of using numpy
  - A.T
  - np.transpose(A)

```
### Transpose
A = np.array([[2, -5], [1, 0], [3, 3]])
print_matrix("A", A)
print_matrix("A^T", A.T)
```

```
A
 [[ 2 -5]
 [ 1  0]
 [ 3  3]]
A^T
 [[ 2  1  3]
 [-5  0  3]]
```

```
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> np.transpose(a)
array([[1, 3],
       [2, 4]])
```

Special matrix multiplication

- **outer product:** the product of a Mx1 with a 1xN vector, which is an M×N matrix

$$\mathbf{x} \otimes \mathbf{y} = \mathbf{x}^T \mathbf{y}$$

- **inner product**: the product of a 1xN with an Nx1 vector is a 1x1 matrix, which is a scalar

$$\mathbf{x} \bullet \mathbf{y} = \mathbf{x}\mathbf{y}^{\mathbf{T}}$$

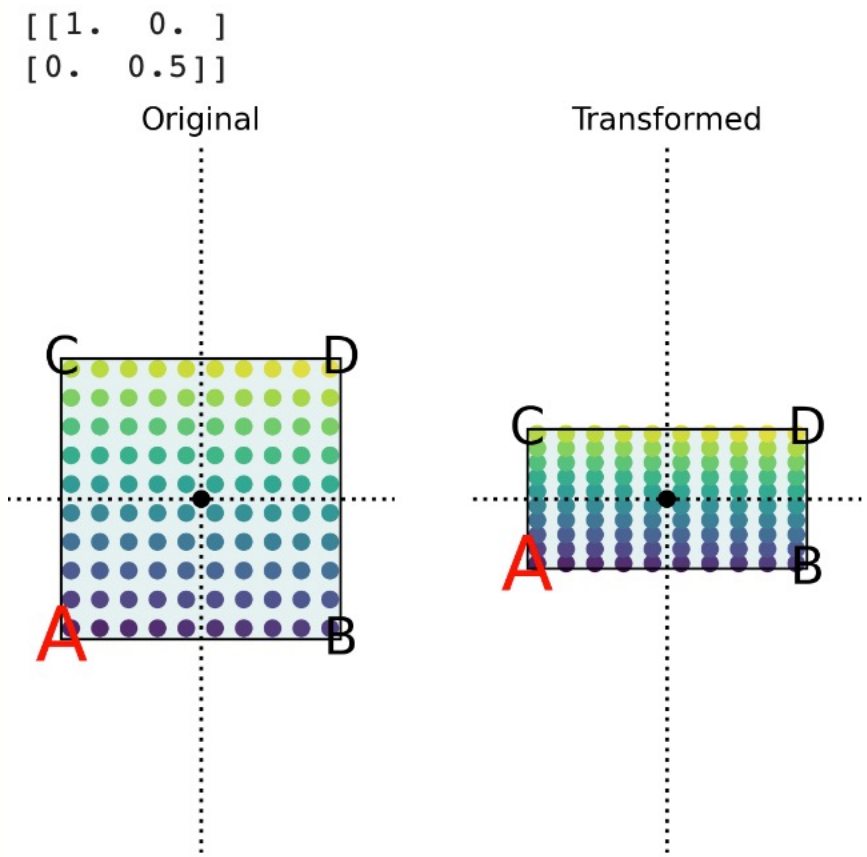
University of Glasgow

Matrix multiplication as composed maps

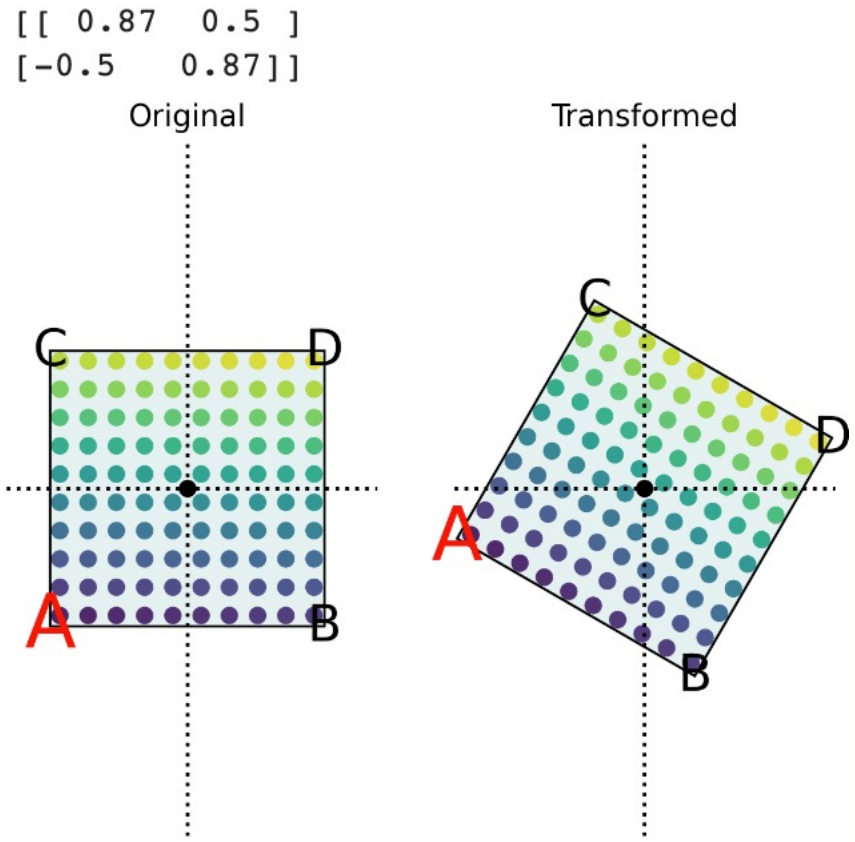
- We saw vectors as **points in space**, and matrices as **vector transform in space**.
- **Multiplication is composition:** If A represents f(x) and B represents g(x), then the product BA represents g(f(x)).
- BAx=B(Ax) means do A to x, then do B to the result

Composed maps

- nonuniform scaling



- rotation



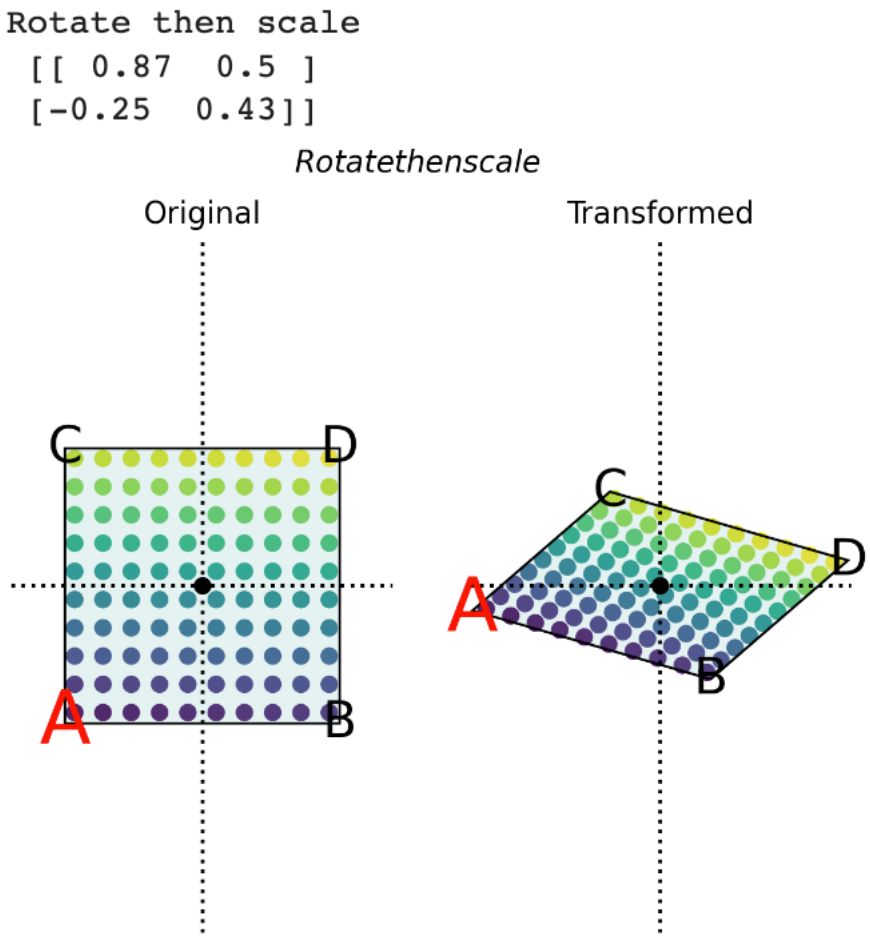
Composed maps

- Nonuniform scaling matrix: scale_x
- Rotation matrix: rot30

$$\text{Rotate\_then\_scale} = \text{scale\_x} \times \text{rot30}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \times \begin{bmatrix} 0.87 & 0.5 \\ -0.5 & 0.87 \end{bmatrix}$$

$$= \begin{bmatrix} 0.87 & 0.5 \\ -0.25 & 0.435 \end{bmatrix}$$



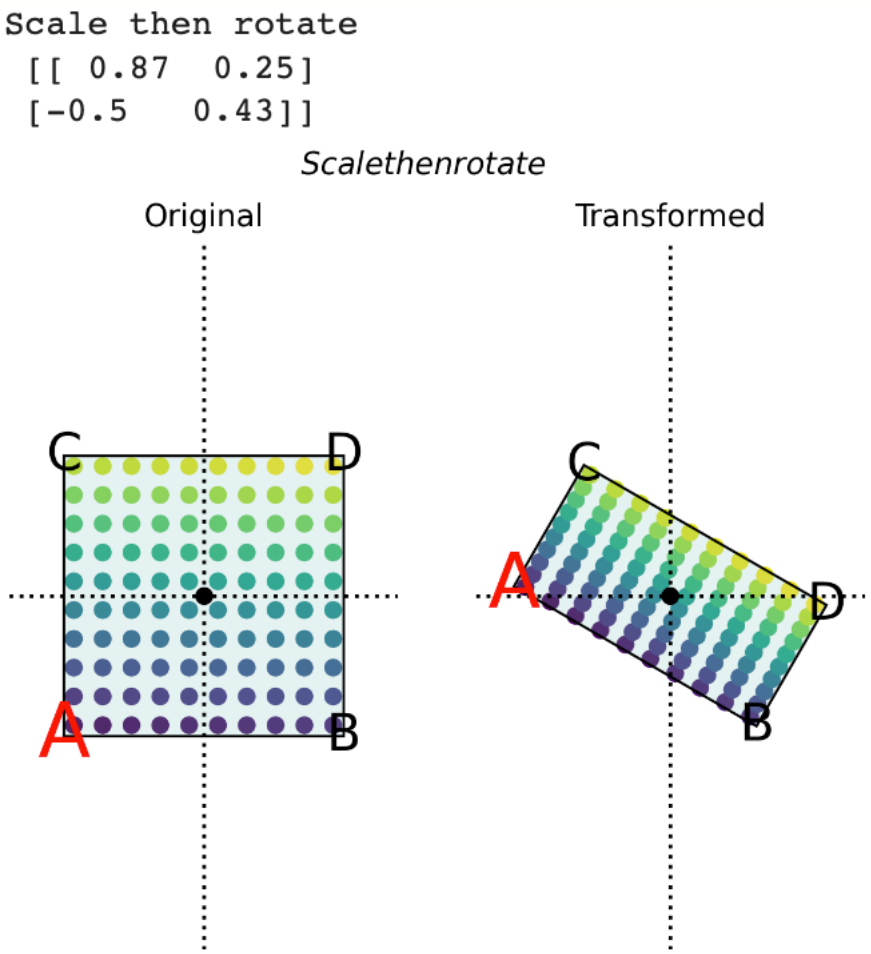
Composed maps

- Nonuniform scaling matrix: scale_x
- Rotation matrix: rot30

$$\text{Scale\_then\_rotate} = \text{rot30} \times \text{scale\_x}$$

$$= \begin{bmatrix} 0.87 & 0.5 \\ -0.5 & 0.87 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0.87 & 0.25 \\ -0.5 & 0.435 \end{bmatrix}$$



University of Glasgow

Concatenation of transforms

- Many software operations take advantage of the definition of matrix multiplication as the composition of linear maps.
- In a graphics processing pipeline, for example, all of the operations to position, scale and orient visible objects are represented as matrix transforms.
- Multiple operations can be combined into *one single matrix operation*.

University of Glasgow

Commutativity and Transpose

- The order of multiplication is important.
- Matrix multiplication does **not** commute

$$AB \neq BA$$

- Transpose order switching

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

- It is also true that

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

An example matrix for measuring spread: covariance matrices

- **mean vector**: the **geometric centroid** of a set of vectors

- **variance:** measures the spread of a dataset

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \mu_i)^2$$

- In the multidimensional case: ***covariance***

$$\Sigma_{ij} = \frac{1}{N-1}\sum_{k=1}^{N}(X_{ki} - \mu_i)(X_{kj} - \mu_j)$$

An example matrix for measuring spread: covariance matrices

- Generate data examples

```
x = np.random.normal(0,1,(500, 5))

mu = np.mean(x, axis=0)
sigma_cross = ((x - mu).T @ (x - mu)) / (x.shape[0]-1)
np.set_printoptions(suppress=True, precision=2)
print_matrix("\Sigma_{\\text{cross}}", sigma_cross)
```

```
\Sigma_{\text{cross}}
 [[ 1.04  0.04  0.02  0.04 -0.02]
 [ 0.04  1.03 -0.03  0.07 -0.03]
 [ 0.02 -0.03  1.04 -0.08  0.01]
 [ 0.04  0.07 -0.08  1.09 -0.02]
 [-0.02 -0.03  0.01 -0.02  0.98]]
```

- 500*5 maxtix

- Compute covariance using *np.cov()*

```
# verify it is close to the function provided by NumPy
sigma_np = np.cov(x, rowvar=False)
print_matrix("\Sigma_{\\text{numpy}}",
             sigma_np)
```

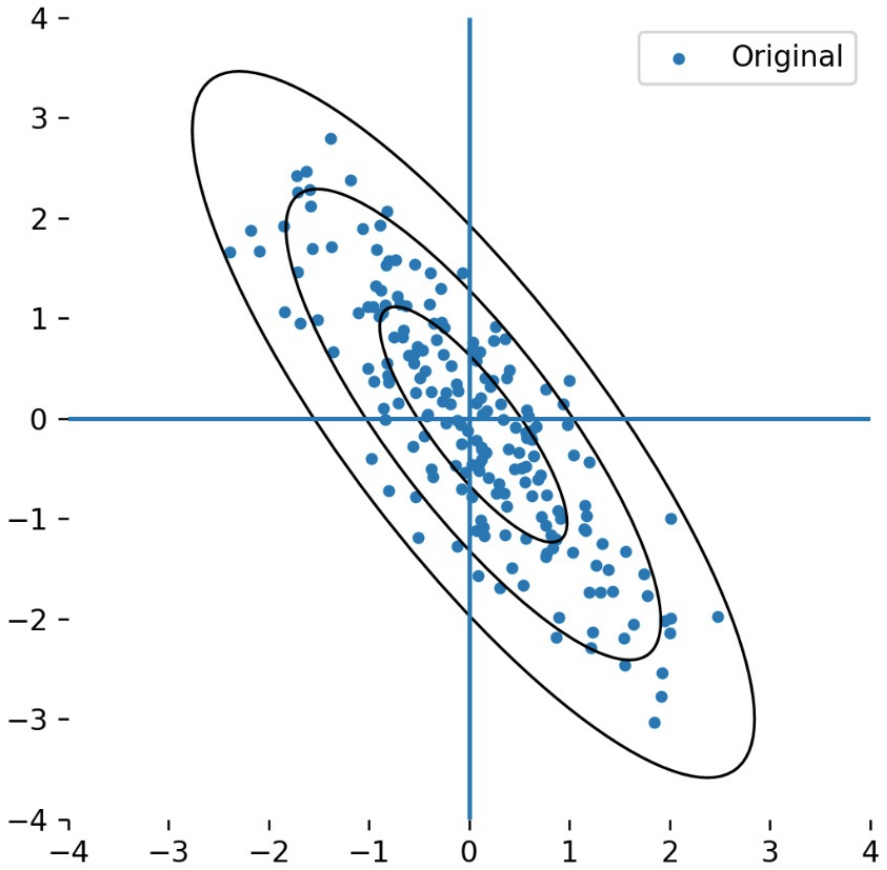
```
\Sigma_{\text{numpy}}
 [[ 1.04  0.04  0.02  0.04 -0.02]
 [ 0.04  1.03 -0.03  0.07 -0.03]
 [ 0.02 -0.03  1.04 -0.08  0.01]
 [ 0.04  0.07 -0.08  1.09 -0.02]
 [-0.02 -0.03  0.01 -0.02  0.98]]
```

- *5*5 matrix*

Covariance ellipses

- For a 2-D dataset
  - **mean vector: 1*2**
  - **Covariance matrix: 2*2**

  - The mean vector captures the idea of "centre"
  - The covariance matrix captures the "spread" of a collection of points in a vector space.



University of Glasgow

Special matrix forms

- **Diagonal** matrix
  - `np.diag(x)`

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

- **Identity** matrix
  - `np.eye(n)`

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

- **Zero** matrix
  - np.zeros()

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

- **Upper triangular**

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

- **Lower triangular**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{bmatrix}$$

University of Glasgow

Beyond this course

- 3blue1brown Linear Algebra series (**strongly recommended**)
- Introduction to applied linear algebra *by S. Boyd and L. Vandenberghe*

- **Linear Algebra Done Right** *by Sheldon Axler* (excellent introduction to the "pure math" side of linear algebra) ISBN-13: 978-0387982588
- **Coding the Matrix: Linear Algebra through Applications to Computer Science** *by Philip N Klein* (top quality textbook on how linear algebra is implemented, all in Python) ISBN-13: 978-0615880990
- **Linear Algebra and Learning from Data** *Gilbert Strang*, ISBN-13: 978-069219638-0, explains many detailed aspects of linear algebra and how they relate to data science.

- The Matrix Cookbook *by Kaare Brandt Petersen and Michael Syskind Pedersen*. If you need to do a tricky calculation with matrices, this book will probably tell you how to do it.


University of Glasgow
Thank you
Contact:
Zaiqiao.Meng@Glasgow.ac.uk