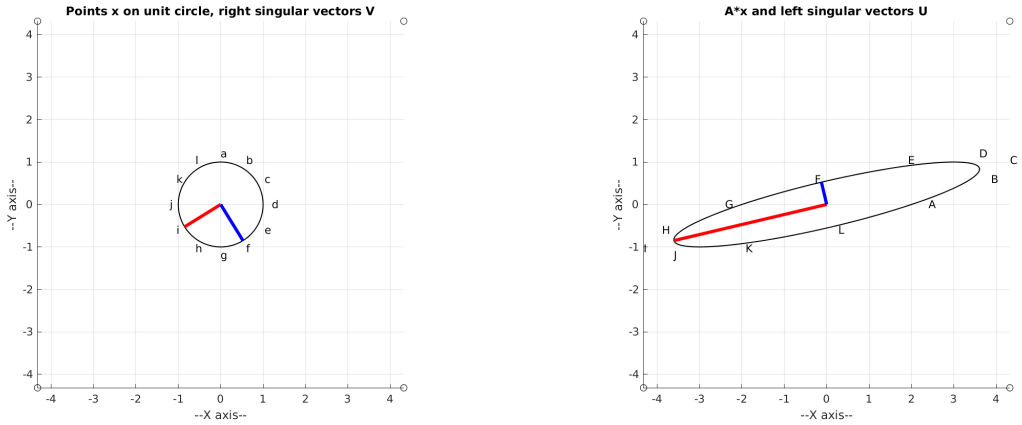
# The Singular Value Decomposition
# ML_2022: Machine Learning

https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/svd_lecture/svd_lecture.pdf




*A*x distorts a unit circle. SVD identifies fundamental directions.*

**The SVD**

*Many machine learning cases can be seen as linear mappings $x \rightarrow A * x$. The singular value decomposition reveals the structure of such transformations.*

- *Every matrix $A$ (square/rectangular, real/complex, singular/nonsingular) has a singular value decomposition $A = U * S * V$;*
- *$U$ and $V$ are orthogonal basis vectors for domain and range;*
- *$S$ is a diagonal matrix whose entries represent growth or shrinkage of components;*

*Note: "everyone" writes the SVD as $A = U*S*V'$, except for Python's numpy library prefers the formulation $A = U * S * V$. We can live with either pattern, but this means that there will be many times when you have to stop and ask yourself which pattern is being used!*

Machine learning extracts information from massive sets of data.

The singular value decomposition (SVD) starts with "data" which is a matrix $A$, and produces "information" which is a factorization $A = U * S * V$ that explains how the matrix transforms vectors to a new space;

In many machine learning problems, the massive sets of data can be regarded as a collection of $m$-vectors, which can be arranged into an $m \times n$ matrix.

SVD application include:

1. least squares line, *data: points in 2D*;
2. curve-fitting, *data: polynomial coefficients*;
3. matrix approximation, *data: entries in a matrix*;
4. image compression, *data: columns of an image*;
5. facial recognition, *data: pictures of a face* .
6. principal component analysis;

# 1 Some linear algebra

Linear algebra has objects called vectors, and operators called matrices which transform vectors to new vectors. We need to know how to set these things up in Python.

## 1.1 Row and column vectors, matrices

In mathematics, we primarily think of vectors as *column vectors*, that is, a vertical stack of numbers. Python's fundamental vector is a row vector, which is essentially just a list of numbers enclosed in square brackets. To make a column vector, we must list several rows, each of length 1. Similarly, an $m \times n$ matrix is described using $m$ row vectors, each of length $n$. The transpose operation swaps the rows and columns of a vector or matrix.

We create vectors and matrices using the numpy function `array()`. We can transpose a vector or matrix by appending the transpose operator `.T` to its name.

```
import numpy as np

t = np.array ( [ 1, 2, 3 ] )                        # a row vector;
u = np.array ( [ [ 4 ], [ 5 ], [ 6 ], [ 7 ] ] )  # a column vector
A = np.array ( [ \
  [ 11, 12, 13, 14 ], \
  [ 21, 22, 23, 24 ], \
  [ 31, 32, 33, 24 ] ] ) # a matrix of 3 rows and 4 columns

v = t.T                  # converts our row vector t to a column vector.
w = u.T                  # converts our column vector u to a row vector.
B = A.T                  # converts our 3x4 matrix A to a 4x3 matrix B.
```

We want to describe the size and shape of our objects. To recover the dimensions of any object, we can simply use the `shape()` function, either as a `numpy` function, or as an attribute of the array.

```
np.shape ( v )
v.shape        # both return (1,3)
A.shape
np.shape ( A ) # both return (3,4)
A.shape[0]     # returns 3
A.shape[1]     # returns 4
```

## 1.2 Norm of a vector

When we describe the "length" of a vector, we usually don't want the number of entries, but rather the geometric length of the object. To avoid confusion, the word *norm* should be preferred over *length*. The norm is represented mathematically by $||v||$ or $||v||_2$, and has the formula:

$$||v|| = \sqrt{\sum_{i=1}^{M} v_i^2}$$

A unit vector has norm 1; unit vectors are often denoted by $u$.

In Python, we can determine a vector norm by:

```
vnorm = np.linalg.norm ( v )
```

It turns out that there are several other norms available, and that matrices can also have norms; however these are advanced topics that may be considered elsewhere.

## 1.3 Angle between vectors

A vector has a direction as well as a norm. If two vectors $v_1$ and $v_2$ aren't identical, we still might consider them close, in the sense that they point in almost the same direction. To make this judgement, we need to be able to measure the angle $\alpha$ between them. To do this, we start by computing the *dot product*, which reveals the cosine of this angle:

$$v_1 \cdot v_2 = \sum_{i=0}^{n-1} v_1(i)\, v_2(i) = ||v_1||\, ||v_2||\, cos(\alpha)$$

We can solve for the angle:

$$\alpha = \arccos\left(\frac{v_1 \cdot v_2}{||v_1||\, ||v_2||}\right)$$

Note that if $v_1$ and $v_2$ are both unit vectors, then the formula for the angle between them is simply:

$$\alpha = \arccos(v_1 \cdot v_2)$$

In Python, we would write:

```
v1dotv2 = np.dot ( v1, v2 )
alpha = np.arccos ( v1dotv2 )
```

For any pair of vectors, we can rewrite the dot product relationship as:

$$cos(\alpha) = \frac{v_1 \cdot v_2}{||v_1||\, ||v_2||}$$

It is natural to use $\cos(\alpha)$ as a measure of how close the vectors are in direction. If we feel that a vector going in the exact opposite direction is essentially using the same direction, the we concentrate on $|\cos(\alpha)|$ instead. We can say:

$$\cos(\alpha) = \begin{cases} 1 & \text{vectors have exactly the same direction} \\ \text{between 0 and 1} & \text{vectors have somewhat the same direction} \\ 0 & \text{vectors are perpendicular, orthogonal} \\ \text{between -1 and 0} & \text{vectors have somewhat the opposite direction} \\ -1 & \text{vectors have exactly opposite direction} \end{cases}$$

In general, the magnitude of $\cos(\alpha)$ is an important indicator of similarity, while the sign is not. If $v_1$ and $v_2$ have opposite directions, then in fact $v_2$ is exactly a linear multiple of $v_1$, with the minor point that the coefficient must be negative:

$$v_2 = -c * v_1$$

But in this case, how would we compute the value $c$? If you think about it, you can see that $c$ is simply the factor that divides $v_1$ by its length, and multiplies it by the length of $v_2$. In other words:

$$v_2 = -\frac{||v_2||}{||v_1||} * v_1$$

Another way to see this is to realize that the vector $\frac{v_1}{||v_1||}$ has the same direction as $v_1$, but has unit length. A similar statement is true for $\frac{v_2}{||v_2||}$. Since the direction of $v_2$ is the negative of the direction of $v_2$, we have

$$\frac{v_2}{||v_2||} = -\frac{v_1}{||v_1||}$$

and rearranging this equation gives us the relationship above.

## 1.4 Projection

Suppose that we have a vector $u$ of unit norm, and another vector $v$, and we want to know how closely $v$ points in the direction $u$. We do this by computing the *projection* of $v$ onto $u$, which has the formula:

$$v_{uproj} = (v \cdot u)u$$

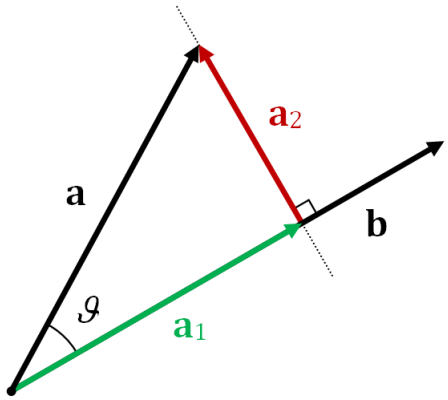
The part of $v$ that is perpendicular to $u$ is found by subtracting the projection:

$$v_{uperp} = v - v_{uproj}$$

The better way to consider this is that

$$v = v_{uproj} + v_{uperp}$$

that is, given a unit vector $u$, any vector $v$ has a component in the $u$ direction, plus the remainder in a perpendicular direction.




*Projection splits vector a into components parallel and perpendicular to b.*