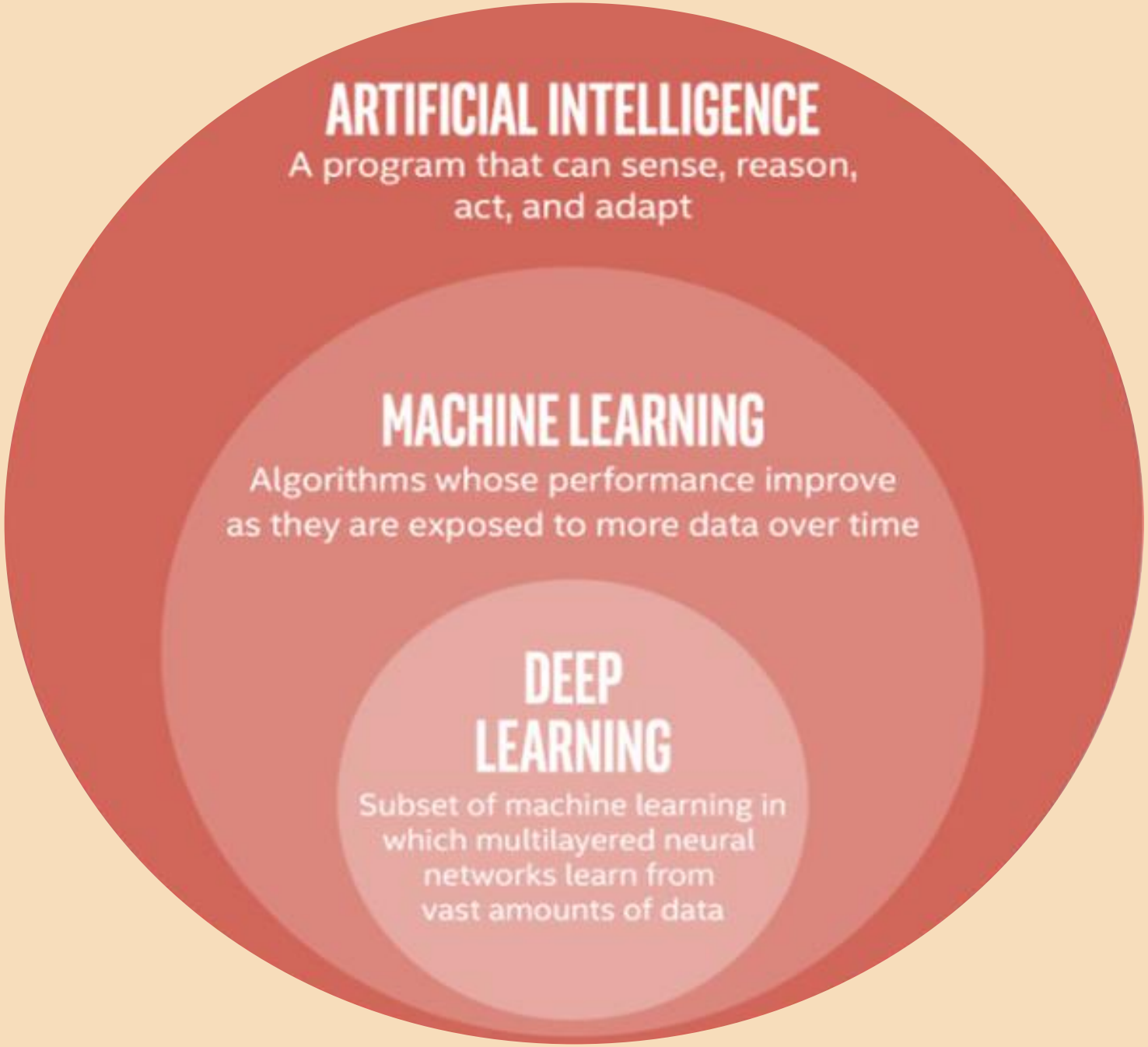
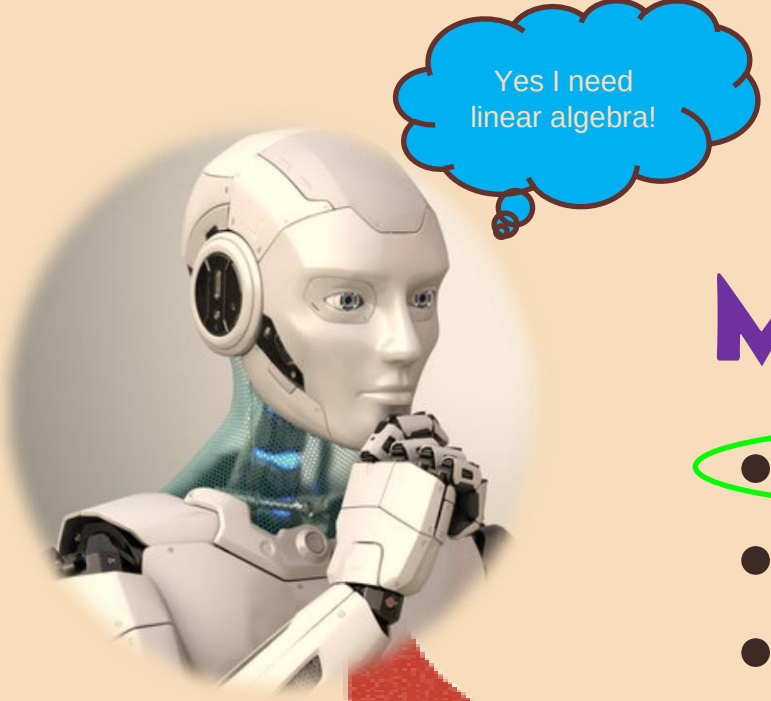
Introduction to linear algebra for AI

instructed by **Mehran Tamjidi**


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data



Mathematic Programming

- Linear algebra
- Probability and Statistics
- multivariate calculus
- Optimization

AI

We have to be a good programmer!

python™

NumPy

SciPy

01

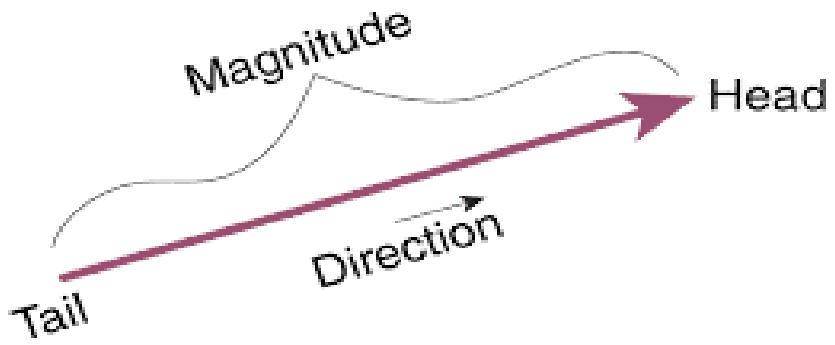
Basics of linear algebra

"Let's Explore the Wonders of Linear Algebra Together"

M.Tamjidi

What is vector

```python
import numpy as np

np.array([x1, x2, x3])

a = np.ones((5,1))
b = np.zeros((5,1))
```



$$\vec{B} = \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix}$$

What is vector

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

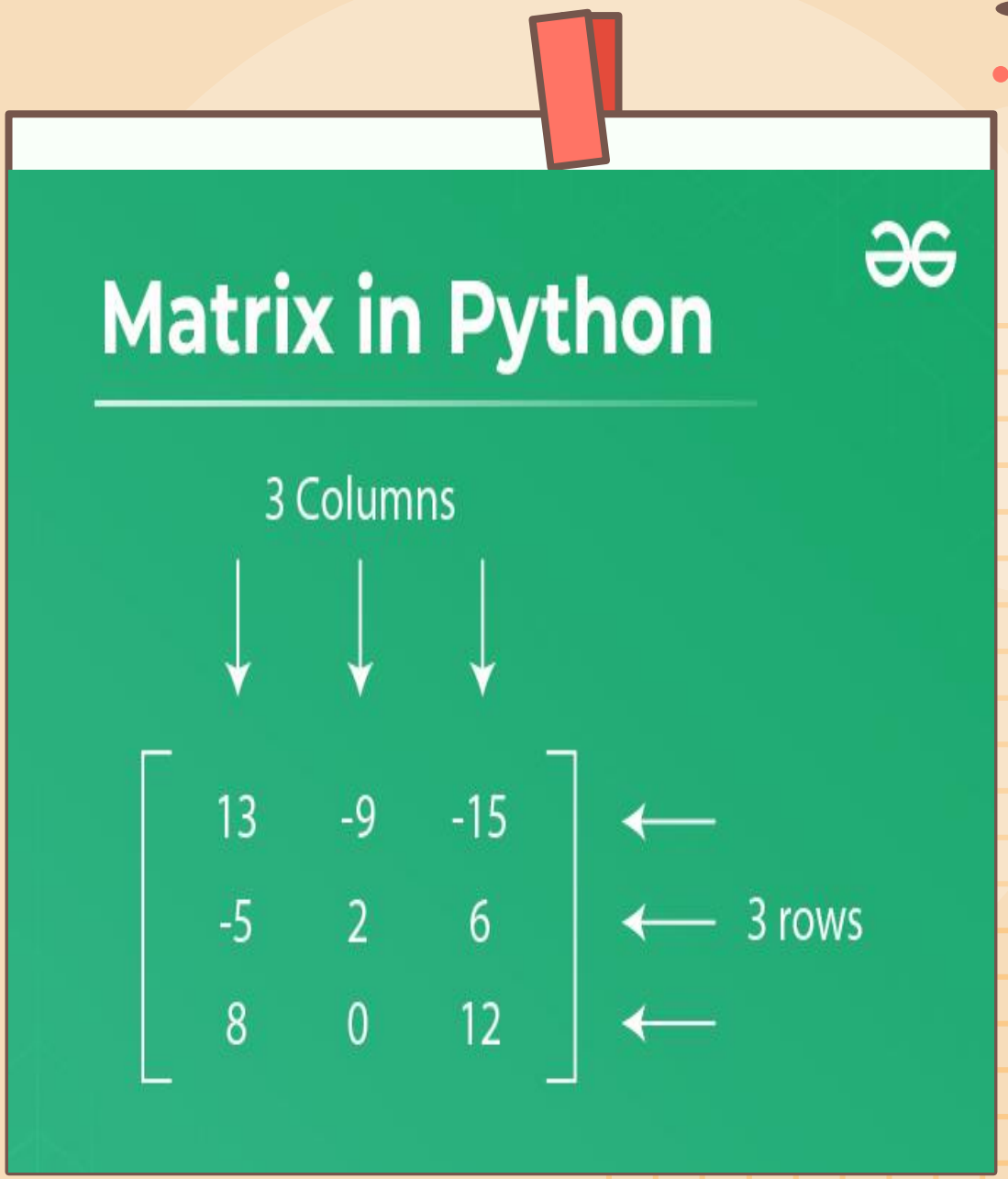
$$X^T = [x_1\ x_2\ x_3]$$

```python
import numpy as np

X = np.array([x1, x2, x3])
X_transposed = X.T
```

M.Tamjidi

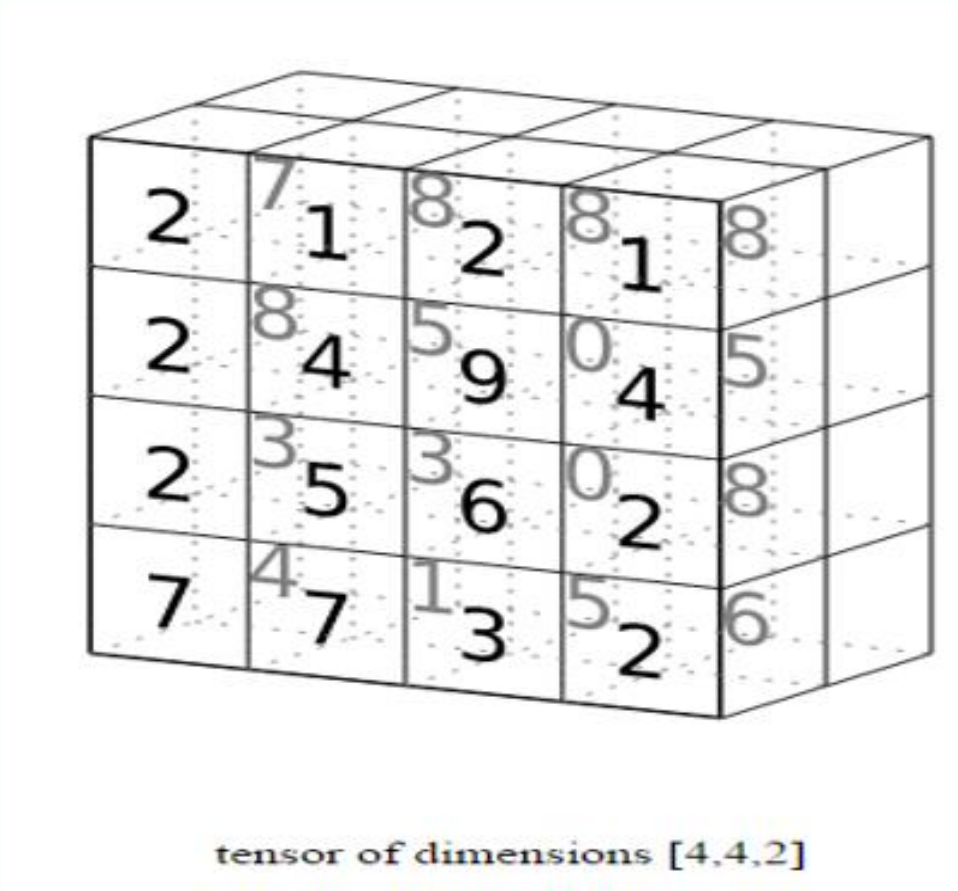
Matrix

```python
import numpy as np
X = np.array([[13, -9, -15],
     [-5, 2, 6],
     [8, 0, 12]])
```



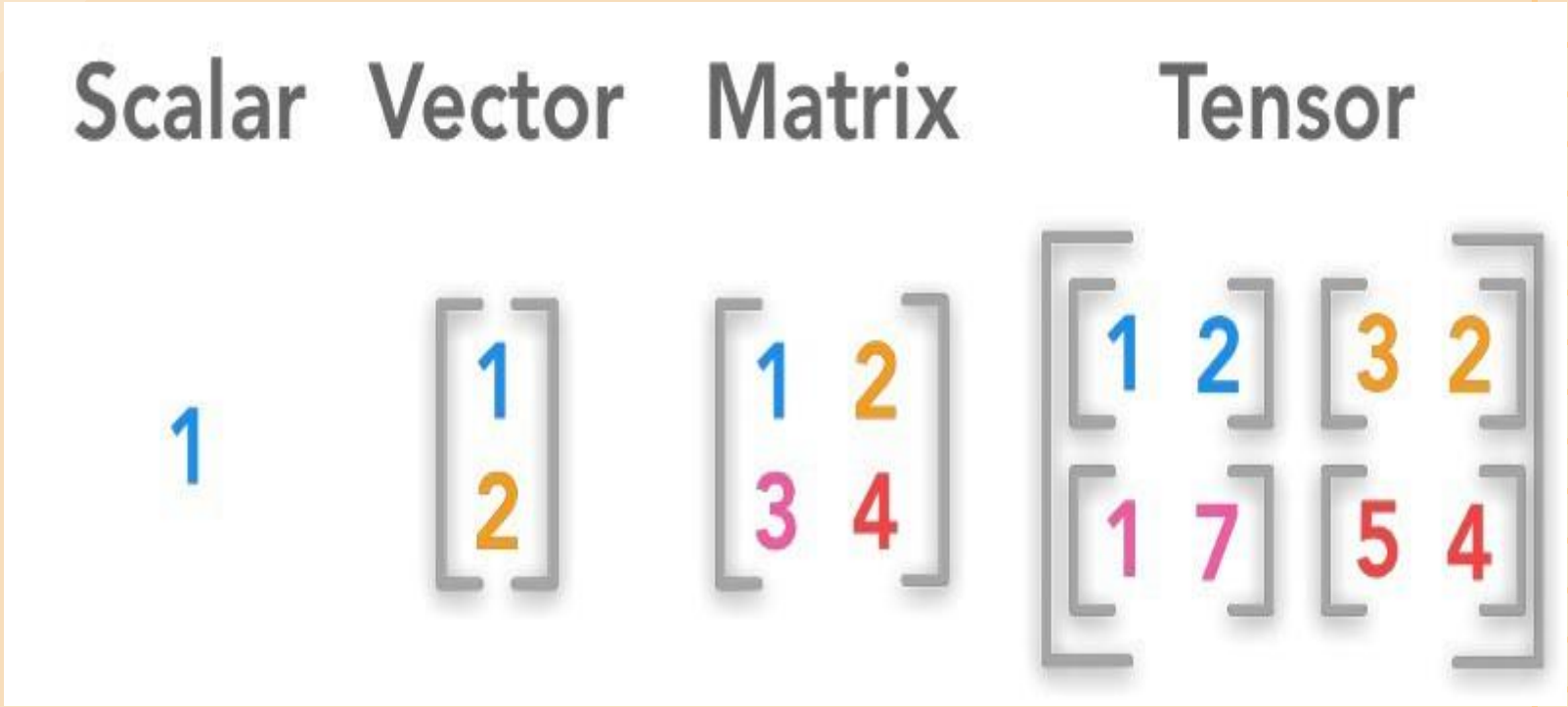
Tensor

In AI we often working with them!



tensor of dimensions [4,4,2]

Comparison between vector, matrix and Tensor



Space

A set with a **structure**

Vector Space

a set V

- scalars $\in$ R (C, or any field)
- Vector addition + ($u + v$ for $u,v \in V$)
- scalar multiplication ($a\,u$ for $a \in R$, $u \in V$)
  - Commutativity: $u + v = v + u$
  - Associativity: $u + (v + w) = (u + v) + w$
  - Identity element: $\exists\, z \in V : v + z = z + v = v$
  - Inverse: for each $v \in V$ there is $v' : v + v' = z$ (z defined above)
  - $(ab)\,v = a\,(b\,v)$
  - $1\,v = v$
  - $a\,(u + v) = a\,u + a\,v$
  - $(a+b)\,v = a\,v + b\,v$

Linear Combination

Let a, b ∈ R. The vector **a** x + **b** y is a linear combination of the vectors x and y.

$$a.\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b.\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = C$$

Let $a_i \in R$. The vector $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n$ is a linear combination of the vectors $x_1, x_2, \ldots, x_n$.

Span

Span(x, y) = { a x + b y | a, b ∈ R}

The space of all linear combinations of x and y.

$\text{Span}(x_1, x_2, \ldots, x_n) = \{a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \mid a_i \in R\}$

We say that x1, x2, …. ,xn span S if S = span(x1, x2, …. ,xn).

Linear dependence

x1, x2, …. ,xn ∈ V are **linearly dependent** if one of them can be written as a linear combination of the others (one of them is in the span of the others).

Equivalently:

a1 x1 + a2 x2 + …. + an xn = 0 ⇒ a1, a2 , …. , an can be non zero

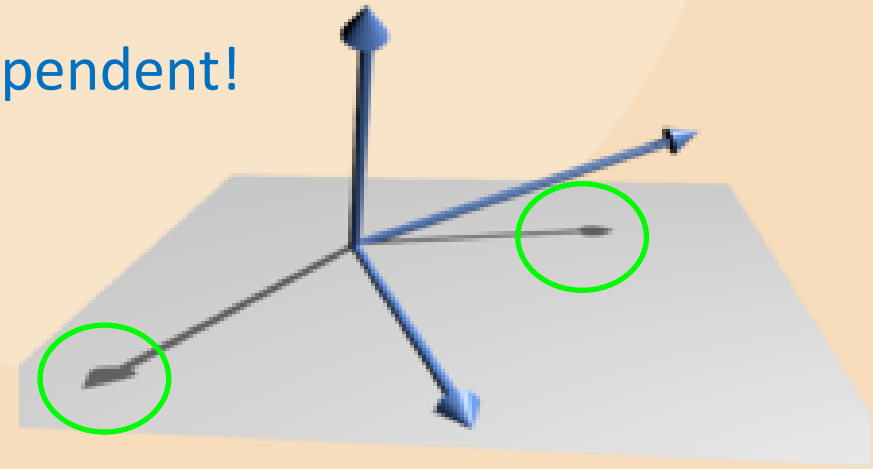
$$c_1\begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} + c_2\begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} + \cdots + c_n\begin{bmatrix} x_{1n} \\ x_{2n} \\ \vdots \\ x_{nn} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Linear dependence

We also can say at least one of them is in the span of others

Question : which vectors are linearly dependent?

Ans: These vectors are linearly dependent!



Linear independence

x1, x2, …. ,xn ∈ V are **linearly independent** if none of them can be written as a linear combination of the others.

Equivalently:

a1 x1 + a2 x2 + …. + an xn = 0 ⇒ a1 = a2 = …. = an = 0

If a1 = a2 = …. = an ≠ 0 =>

$$c_1 \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{bmatrix} + c_2 \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{n2} \end{bmatrix} + \cdots + c_n \begin{bmatrix} x_{1n} \\ x_{2n} \\ \vdots \\ x_{nn} \end{bmatrix} \not\equiv \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

In the other words

Linear independence

x,y,z are independent if

- $x \notin span(y,z)$, AND
- $y \notin span(z,x)$, AND
- $z \notin span(x,y)$

Why independency is important?

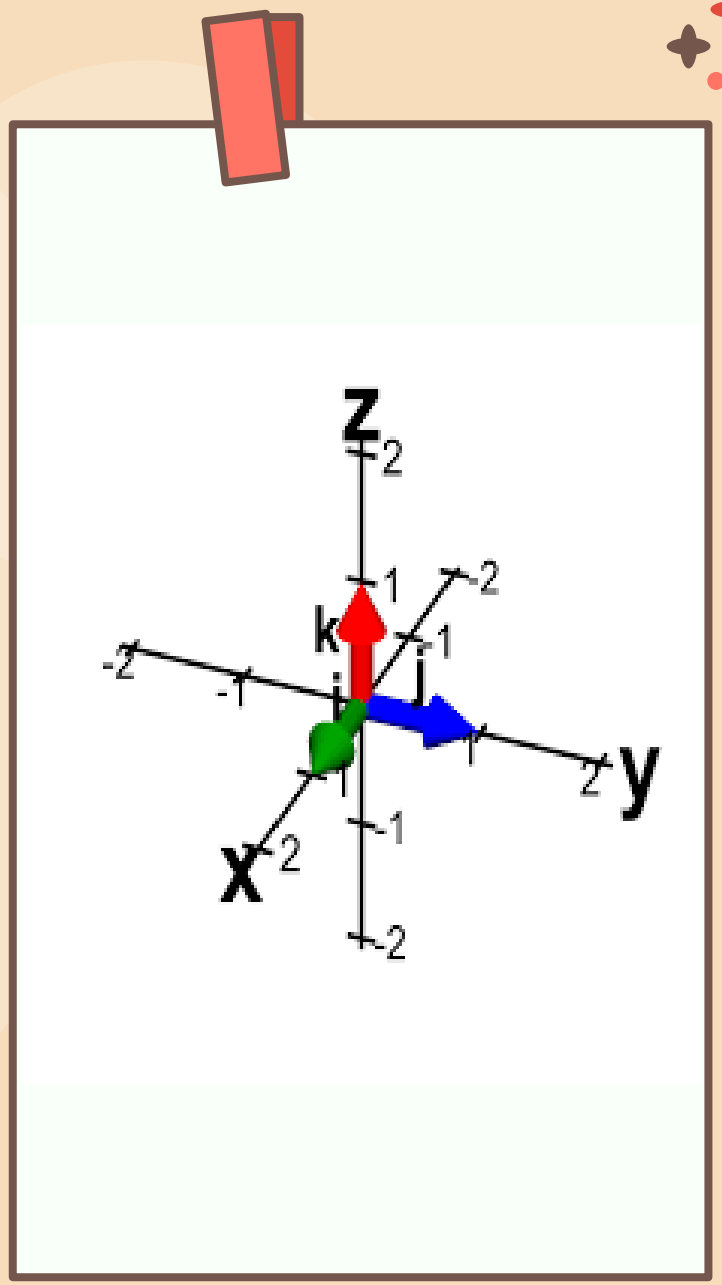
Independence = Uniqueness

Basis

$v_1, v_2, \ldots, v_n \in V$ such that

- $v_1, v_2, \ldots, v_n$ are linearly independent
- $v_1, v_2, \ldots, v_n$ span V

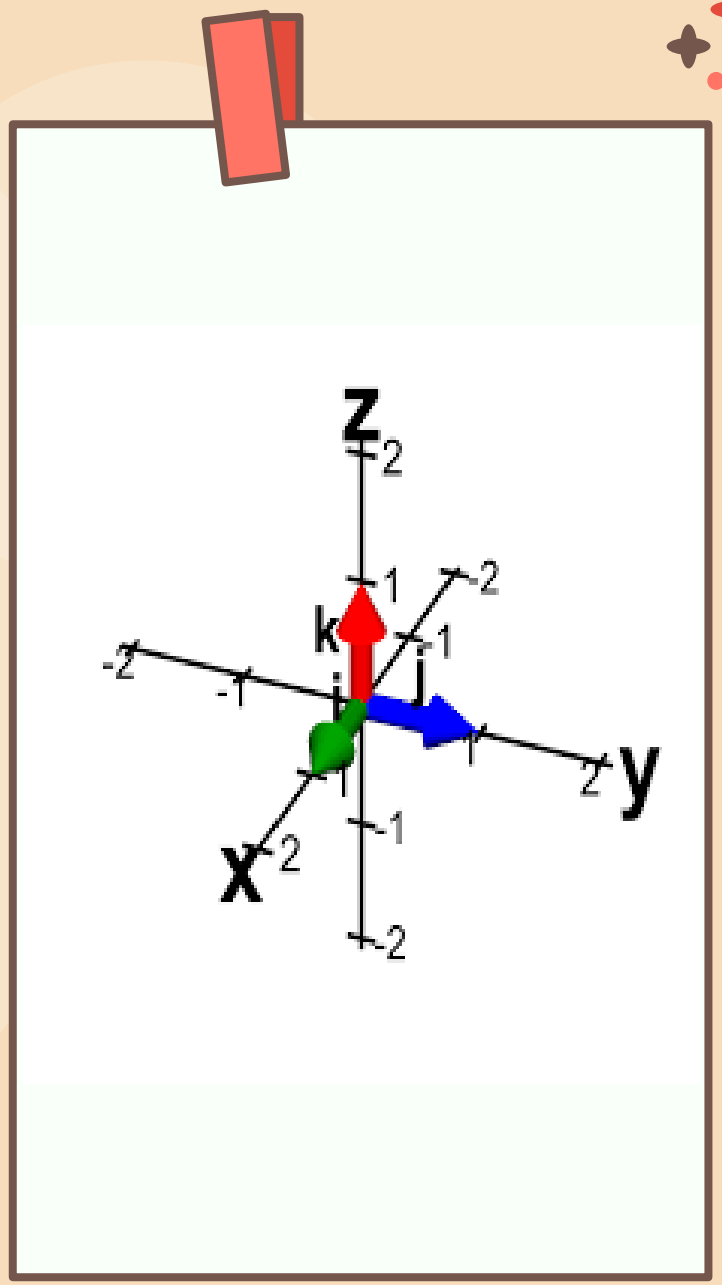
Note: Any space has a basis



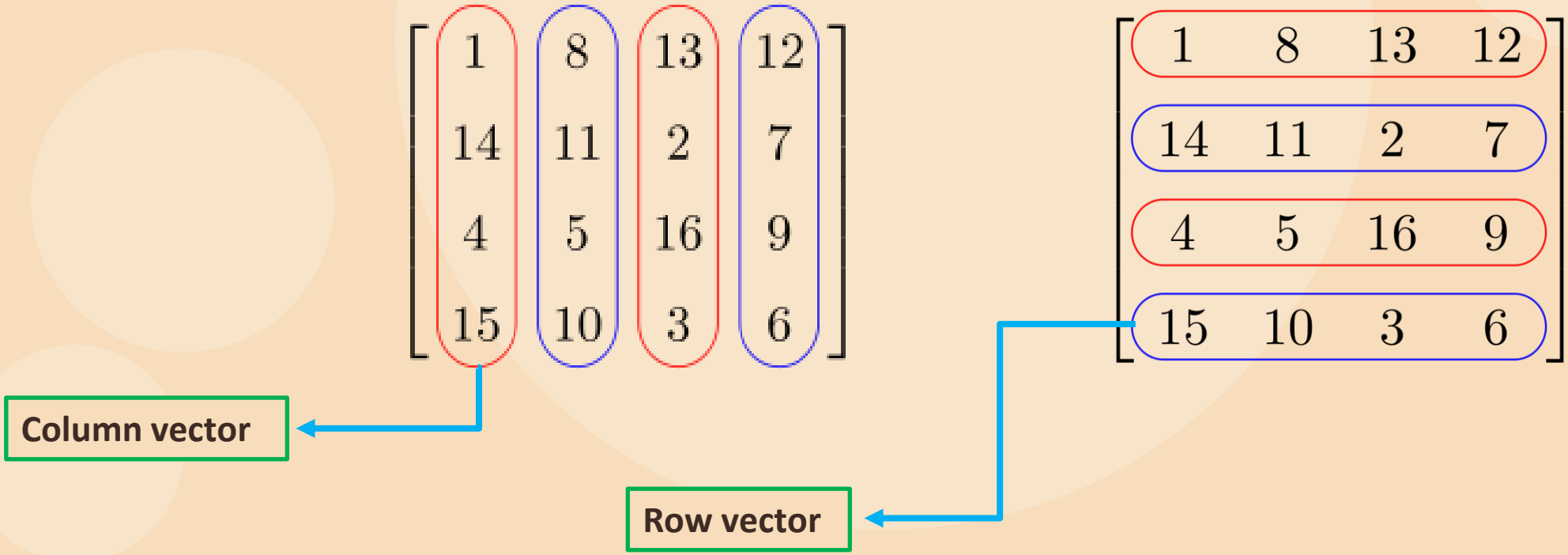
Dimension

Cardinality of $\{v_i\}_{i\in I}$ is called the dimension of V

Which v_i is one of the basis of V



Column and Row vectors

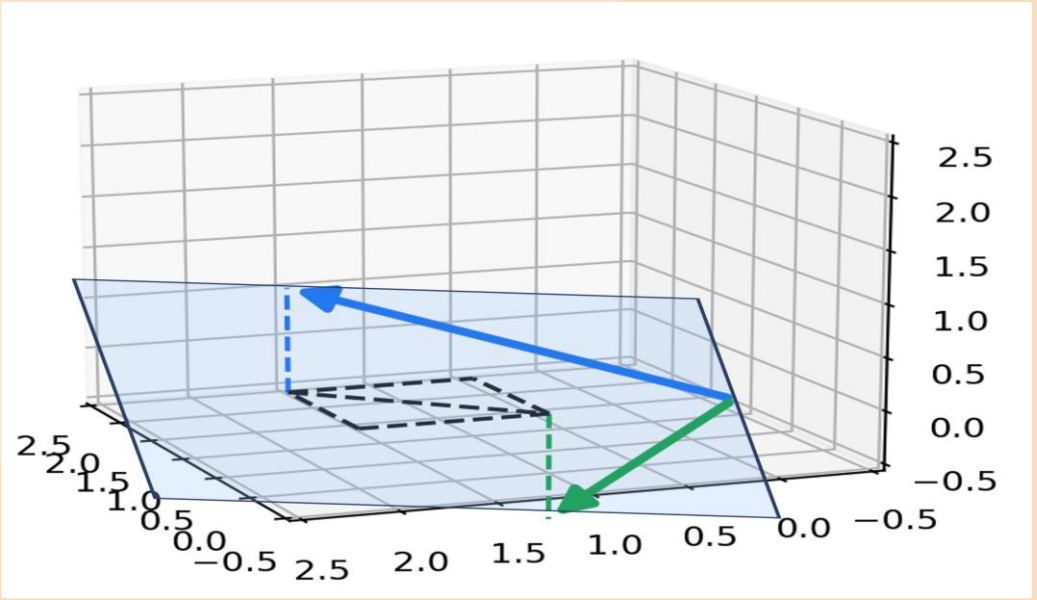


Span in matrix form!

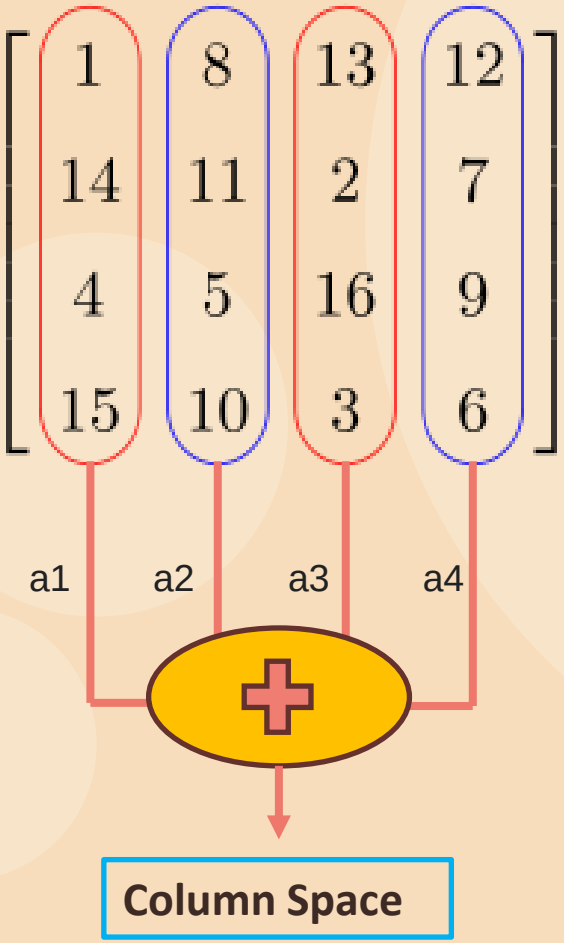
Column space = Span(column vectors)

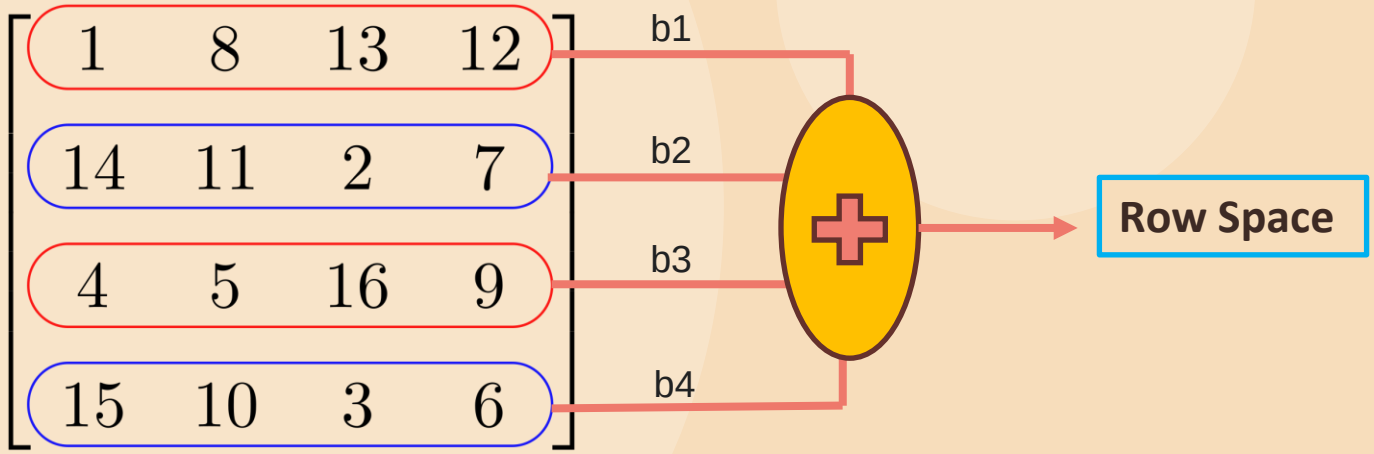
Row space = Span(Row vectors)

In the two column/row basis scenario it can be like that:



Column and Row Space





Example

Row space

Suppose we have matrix A:

$$A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \end{bmatrix}$$

Then the row vectors are $\mathbf{r}_1$ = [1, 0, 2] and $\mathbf{r}_2$ = [0, 1, 0]. A linear combination of $\mathbf{r}_1$ and $\mathbf{r}_2$ is any vector of the form:

$$c_1 \begin{bmatrix} 1 & 0 & 2 \end{bmatrix} + c_2 \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & 2c_1 \end{bmatrix}$$

Question

A is 3 by 3 matrix, does this equation means that A=0 ?

$$A \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

False

$$if\ we\ get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

looking for matrix X

$$get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0_{(3\times1)}$$

$$if\ we\ get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

looking for matrix X

$$get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 0_{(3\times1)}$$

$$if\ we\ get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

looking for matrix X

$$get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} 0 \\ 2 \\ 10 \end{bmatrix} = 0_{(3\times1)}$$

$$if\ we\ get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

looking for matrix X

$$get\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} 0 \\ \alpha \\ \beta \end{bmatrix} = 0_{(3\times1)}$$

These vectors are called **null vector**

Null Vector, Space

Null vector : *The vector Solving this hemogenious equation*

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$Ax = 0$$

Null Space : *Span of null vectors*

Guess what is the left null space! $x^T A = 0^T$

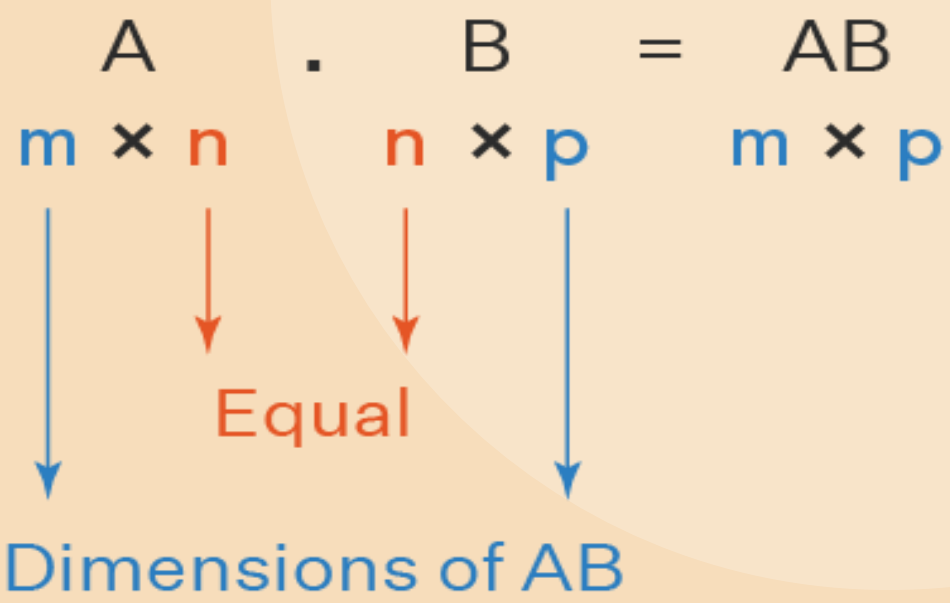
Lets talk more about matrices

"Matrices are the gateway to higher dimensions.”

Michio Kaku

Matrix Multiplication

Looking for the compatible shapes!



Making matrices compatible with reshape

```
import numpy as np

A = np.array([[0, 1],[2, 3],[4,5]])

A.shape # (3, 2)
B = np.reshape(A, (2, 3))

A.reshape((2, 3))

# Matrix B is
>>> array([[0, 1, 2],[3, 4, 5]])
```

Vectorization is all you need !

What is explicit for loop ?

You define a loop and access to current value by indexing into the iterator.

```
# Using explicit for loop for matrix multiplication
def matrix_multiply_for_loop(A, B):
    assert A.shape[1] == B.shape[0], "Not matched"
    n, m = A.shape[0], B.shape[1]
    p = A.shape[1]  # Number of columns in A,and rows in B
    C = np.zeros((n, m))  # Creating an empty array
    for i in range(n):
        for j in range(m):
            for k in range(p):
                C[i, j] += A[i, k] * B[k, j]
    return C
```

Problem:

That’s so slow

Matrix Multiplication

There are several methods for matrix multiplication in NumPy (using vectorization techniques)

```
import numpy as np

C = np.matmul(A,B)
C = np.dot(A,B)
C = A.dot(B)

C = A@B
```

"We typically avoid explicit for loops whenever possible"

Hadamard product

Element wise multiplication

$$A \odot B$$

For vectors we also have

$$X.Diag(Y)$$

$$\overset{G}{\begin{bmatrix} 3 & 5 & 7 \\ 4 & 9 & 8 \end{bmatrix}} \odot \overset{H}{\begin{bmatrix} 1 & 6 & 3 \\ 0 & 2 & 9 \end{bmatrix}} = \overset{N}{\begin{bmatrix} 3\times1 & 5\times6 & 7\times3 \\ 4\times0 & 9\times2 & 8\times9 \end{bmatrix}}$$

```
import numpy as np

C = np.multiply(A, B)

C = A * B
```

x, y must have the same dimension

M.Tamjidi

Dot product

For vectors

$$R^n \rightarrow R$$

$$dot\,(X,Y) = X^T.Y$$

```python
import numpy as np

z = np.dot(X, Y)
```

$$X^T Y = [x_1 \;\; x_2 \;\; x_3].\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1y_1 + x_2y_2 + x_3y_3$$

X ,Y must have the same dimension

Inner product

For matrix and vector

$$R^{m \times n} \rightarrow R$$

$$\text{Inner product}(A, B) = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} \times B_{ij}$$

$$Inner\ product(A, B) = Trace(A^T, B)$$

```python
import numpy as np

C=np.sum(np.multiply(B, A))

C=np.trace(np.matmul((A.T), B))
C=np.trace(np.dot((A.T), B))
```

A ,B must have the same dimension

M.Tamjidi

```python
import numpy as np

C = np.outer(u,v)
```

Outer Product

$$u \otimes v = u\, v^T$$

it somehow has a concept of corrolation

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathsf{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}$$

M.Tamjidi

Column rank

Just count number of independant columns

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 \\ 1.5 & 3 & 4.5 & 6 \end{bmatrix}$$

Column rank = 1

Row rank

Just count number of independant rows

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 \\ 1.5 & 3 & 4.5 & 6 \end{bmatrix}$$

Column rank = 1

Rank

Rank-Nullity Theorem:

$$column\ rank = row\ rank = rank$$

```
import numpy as np
rank = np.linalg.matrix_rank(matrix)
```

If all of the columns (or rows) are independent this matrix is named full rank

Full-rank

When a matrix is $n \times n$ (square), if its columns (or rows) are independent, we call this matrix **full-rank** or **non-singular**.

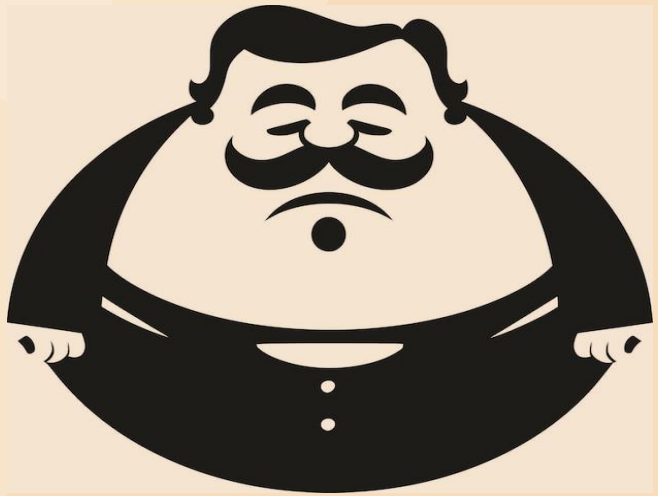
Non-singular matrices are invertible, and vice versa

In linear algebra, a matrix that is not invertible is often referred to as **singular** or **rank-deficient**. Specifically, a matrix is singular if and only if it is not full rank, meaning that at least one of its rows or columns can be expressed as a linear combination of the others.

```python
import numpy as np
rank = np.linalg.matrix_rank(matrix)
print("Rank of the matrix:", rank)

>>> Rank of the matrix: 3
```

Fat matrix

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



If all rows of fat matrix are independent this matrix named **Full-row rank**

Thin matrix

```python
import numpy as np
rank = np.linalg.matrix_rank(matrix)
print("Rank of the matrix:", rank)

>>> Rank of the matrix: 2
```

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

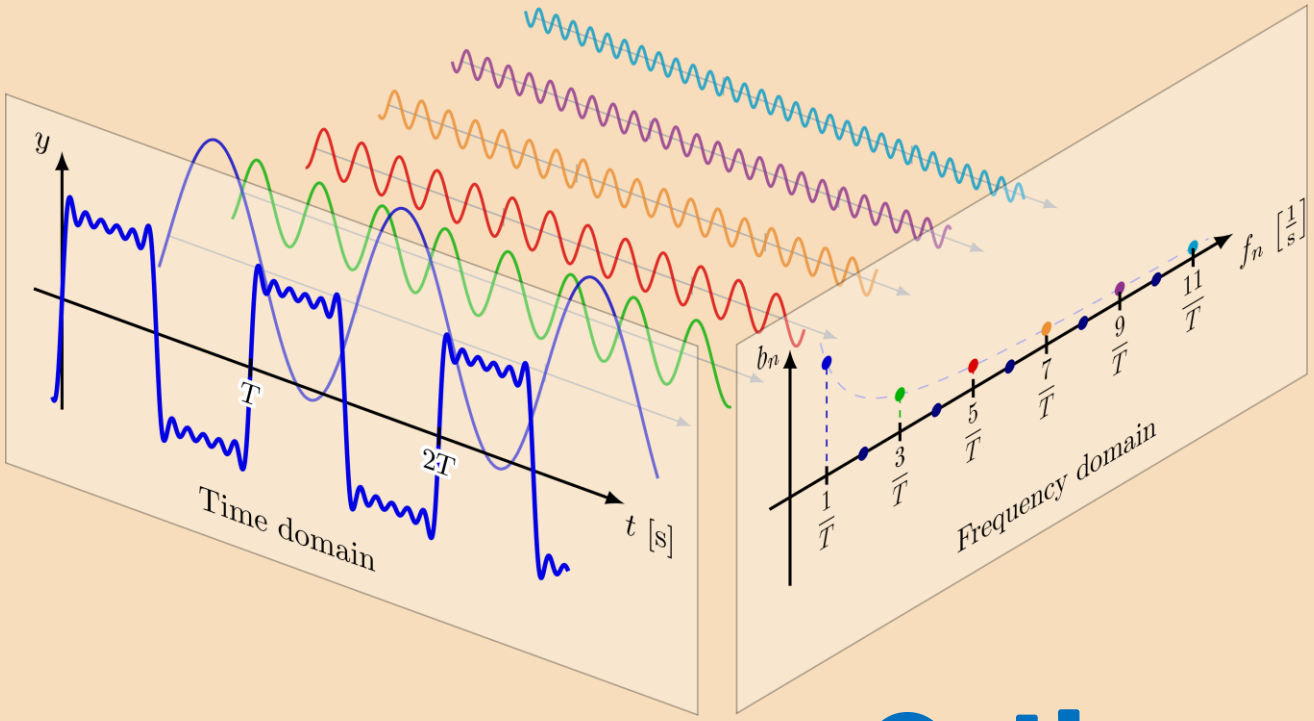
If all rows of fat matrix are independent this matrix named **Full-column rank**

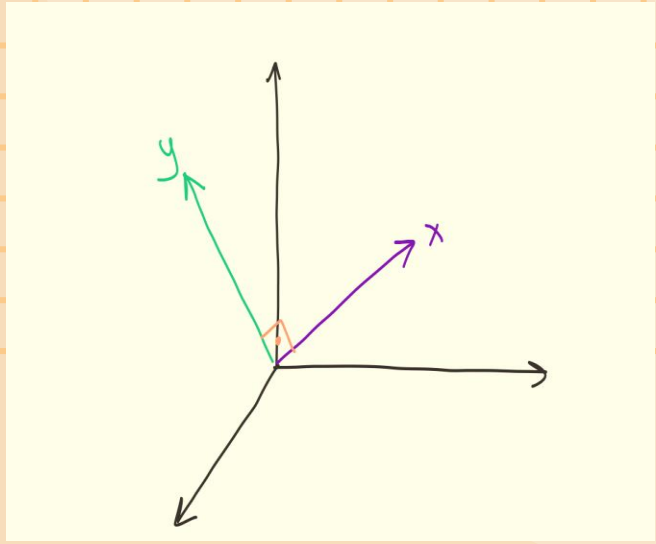
Inverse of matrix

If matrix is singular it doesn't have inverse

A square matrix A of size of n×n is considered singular if its determinant is zero. Conversely, if det(A)≠0 , then A is non-singular.

"Always remember: Determinant and non-singularity are defined only for square matrices."





Orthogonality

”Orthogonality is the key to effective separation.”

M.Tamjidi

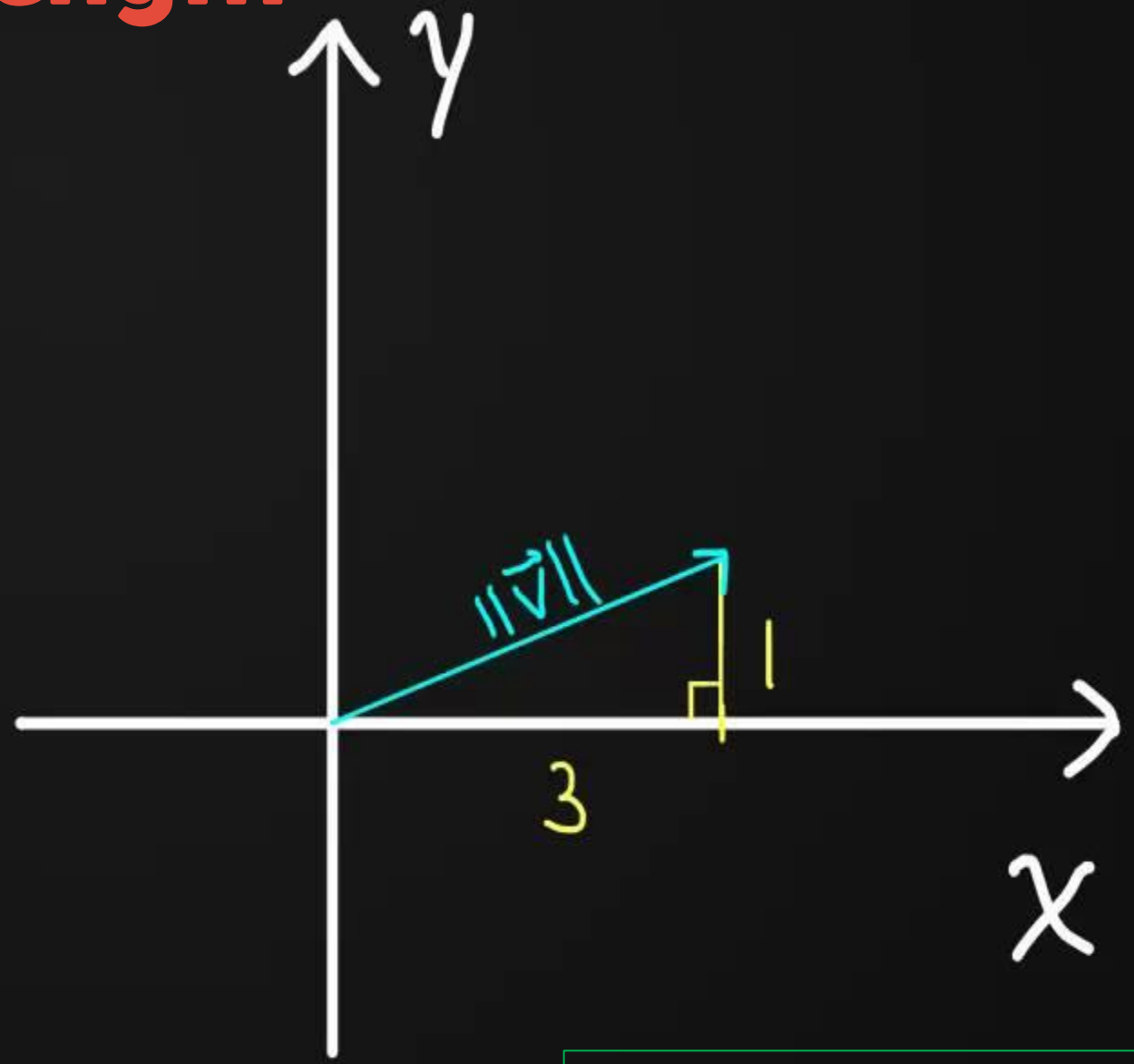
Vector length

$$\vec{v} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \begin{matrix} x \\ y \end{matrix}$$

$$\|\vec{v}\|^2 = 3^2 + 1^2$$

$$\Rightarrow \|\vec{v}\| = \sqrt{3^2 + 1^2}$$

$$= \sqrt{10}$$



Just for recalling ...

Vector length

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

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

$$||V|| = \sqrt{V^T V}$$

$$= \sqrt{< V.V >}$$

Projceting to itself

M.Tamjidi

Normalized vector vectors

making magnitude of vector $V = 1$

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

M.Tamjidi

Orthogonal and orthonormal vectors

Orthogonal vectors $\longleftrightarrow$ $< V.V > = 0$

Orthonormal vectors $\longleftrightarrow$ $< V.V > = 0 \quad \& \quad ||V||=1$

1- Orthogonal vectors

$$\mathbf{v}_1 = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

2. **Orthonormal Vector:**

$$\mathbf{u} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad \text{and} \quad \mathbf{v} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Orthogonal matrix

I know that's a confusing little

The matrix is orthogonal if and only if its columns (or rows) are composed of **orthonormal** vectors.

Main property of orthogonal matrices

$$U^T = U^{-1}$$
$$UU^T = U^T U = I$$

As you see it's columns are it's rows are orthonormal

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

M.Tamjidi

Orthogonal matrix

That's a little confusing

The matrix is orthogonal if and only if its columns (or rows) are composed of **orthonormal** vectors.

Main property of orthogonal matrices

$$U^T = U^{-1}$$
$$UU^T = U^T U = I$$

As you see it's columns are it's rows are orthonormal

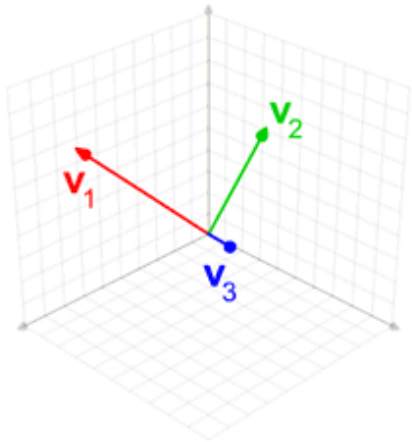
$$\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

M.Tamjidi

Gram–Schmidt process

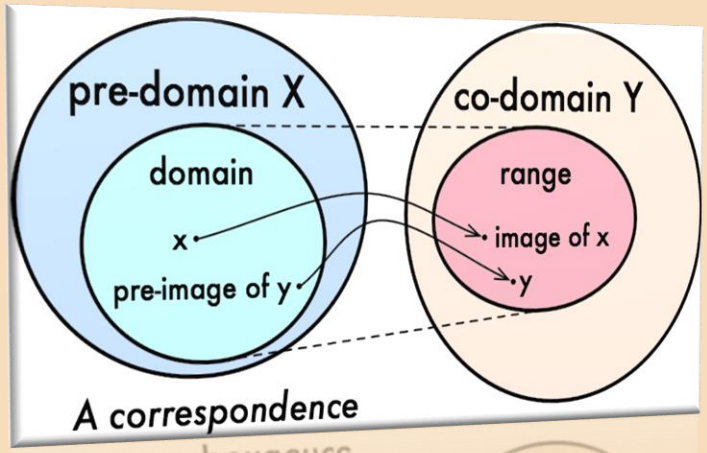
$$\operatorname{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u},$$

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_2), & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \operatorname{proj}_{\mathbf{u}_2}(\mathbf{v}_3), & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
\mathbf{u}_4 &= \mathbf{v}_4 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_4) - \operatorname{proj}_{\mathbf{u}_2}(\mathbf{v}_4) - \operatorname{proj}_{\mathbf{u}_3}(\mathbf{v}_4), & \mathbf{e}_4 &= \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|} \\
&\vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{j=1}^{k-1} \operatorname{proj}_{\mathbf{u}_j}(\mathbf{v}_k), & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}.
\end{aligned}$$



https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

M.Tamjidi



Some frequently used functions

" Functions are bridges connecting theory to reality."

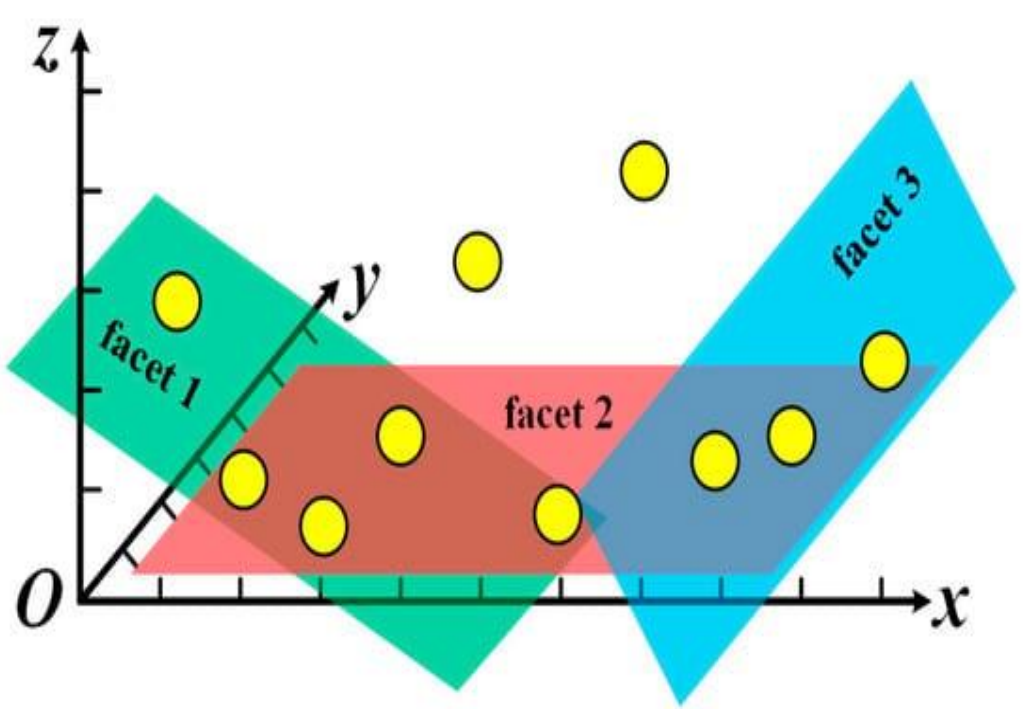
Affine function

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

Example

$$f(\mathbf{x}) = \begin{bmatrix} 2 & 1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Illustration of some affine functions



Quadratic function

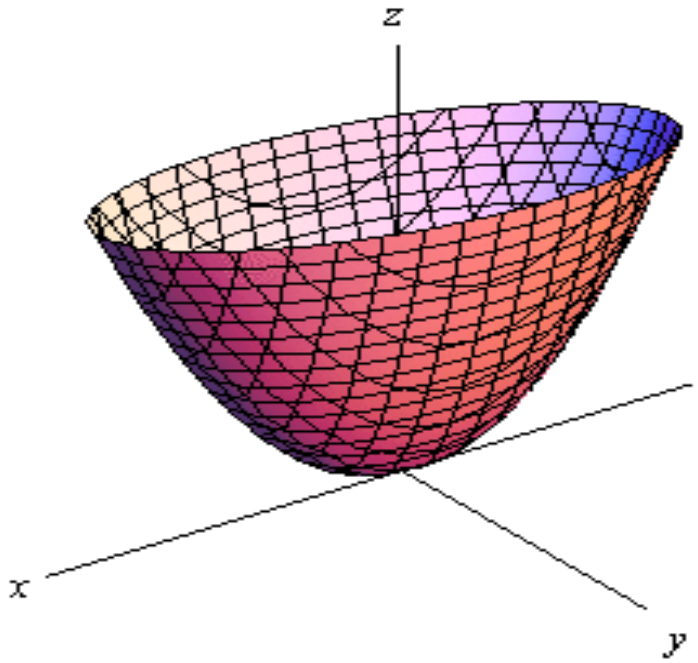
$$f(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$$

Example

$$f(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 1.5 \\ 1.5 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 7$$

You will need this function for SVM

Illustration of quadratic function



M.Tamjidi

02

Norms & Normalizations

"Normalization is the key to simplicity."

Vector norm

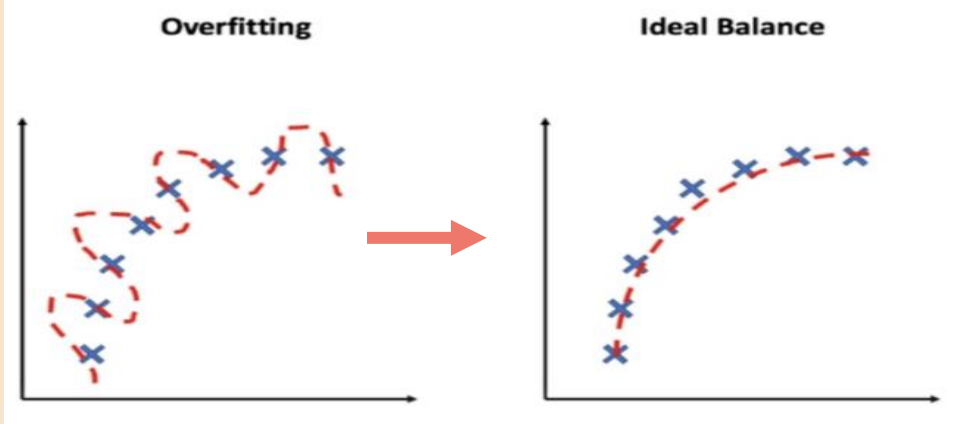
1. **Non-negativity**: For any vector $\mathbf{x}$, the norm is non-negative:
   $\|\mathbf{x}\| \geq 0$
2. **Zero vector**: The norm of the zero vector is zero:
   $\|\mathbf{0}\| = 0$
3. **Scalar multiplication**: For any scalar α, the norm of the scalar multiple of a vector is the absolute value of the scalar multiplied by the norm of the vector:
   $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$
4. **Triangle inequality**: The norm satisfies the triangle inequality, which states that the norm of the sum of two vectors is less than or equal to the sum of the norms of the individual vectors:
   $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

LP-norms are a family of norms, often denoted as $\|\cdot\|_p$, where p is a positive real number. The LP-norm of a vector $\mathbf{x}$ is defined as:

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

Vector norm

We can use these norms and add them to the objective function to prevent overfitting



- L_1 norm (Manhattan norm): $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$
- L_2 norm (Euclidean norm): $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
- L_∞ norm (Maximum absolute value norm): $\|\mathbf{x}\|_\infty = \max_i |x_i|$

LP norms in NumPy

```python
import numpy as np

x = np.array([1, 2, -3])
l1_norm = np.linalg.norm(x, ord=1)
l2_norm = np.linalg.norm(x, ord=2)
linf_norm = np.linalg.norm(x, ord=np.inf)
```

M.Tamjidi

Matrix norms

1- Frobenius norm

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

```python
import numpy as np

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

frobenius_norm = np.linalg.norm(A, 'fro')
```

Matrix norms

2- Spectral norm

The default mode!

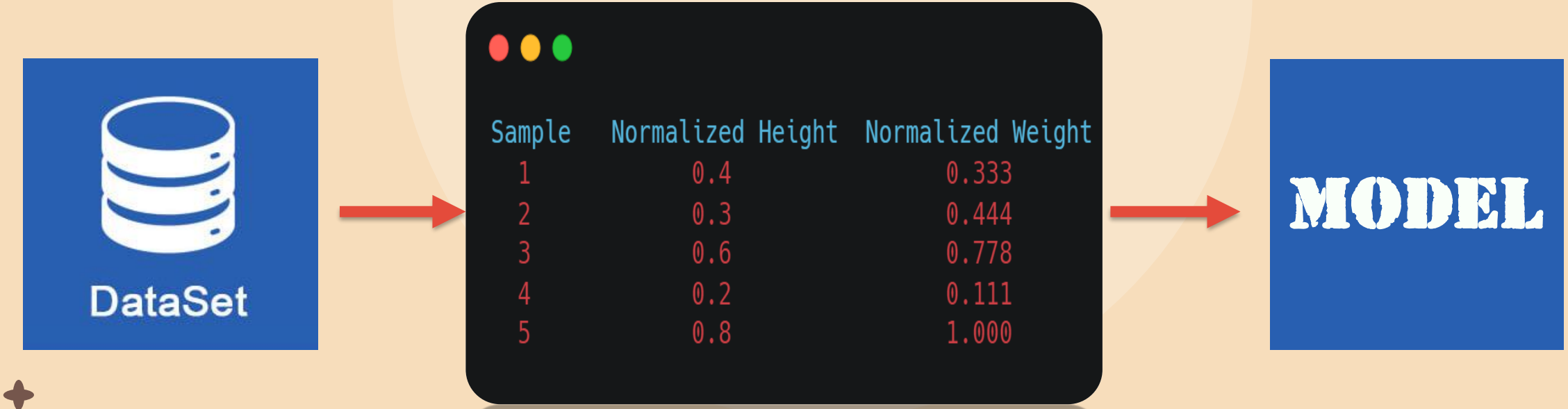
$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

norm

```python
import numpy as np
import scipy

spectral_norm = scipy.linalg.norm(A, 2)  # Compute spectral norm using SciPy
print("Spectral norm of A:", spectral_norm)
```

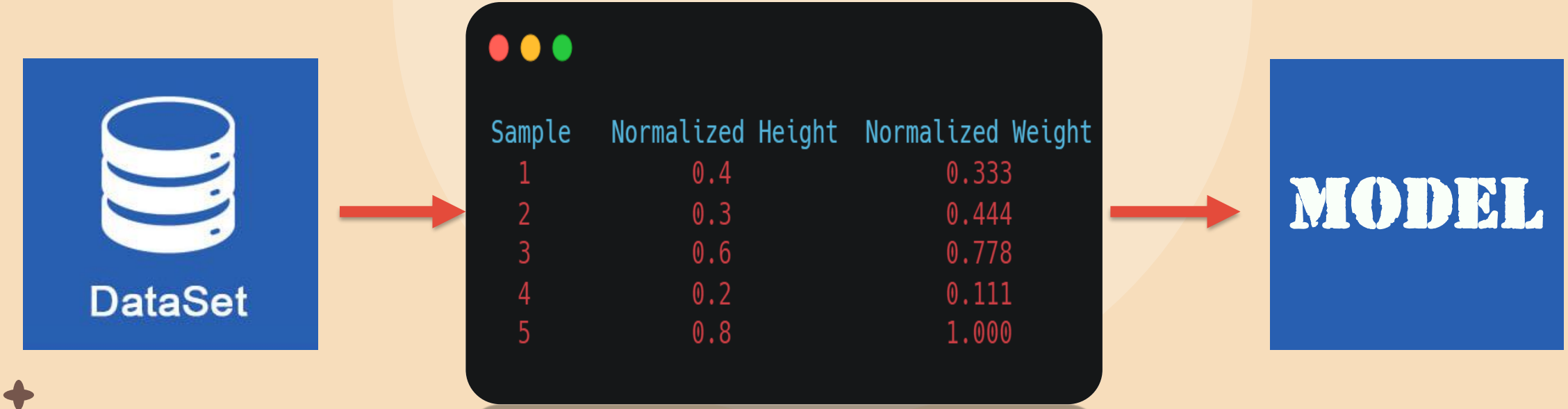
Importance of feature scaling

Biased Model : Features with larger scales may dominate the model training process.



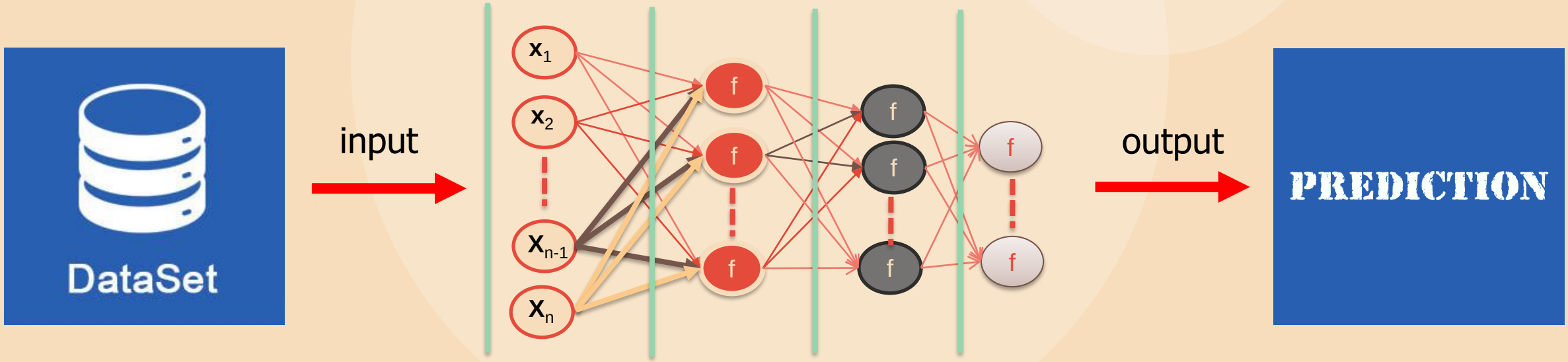
Importance of feature scaling

Biased Model : Feature **Lets normalize it** te the model training process.



Importance of feature scaling

Neural network

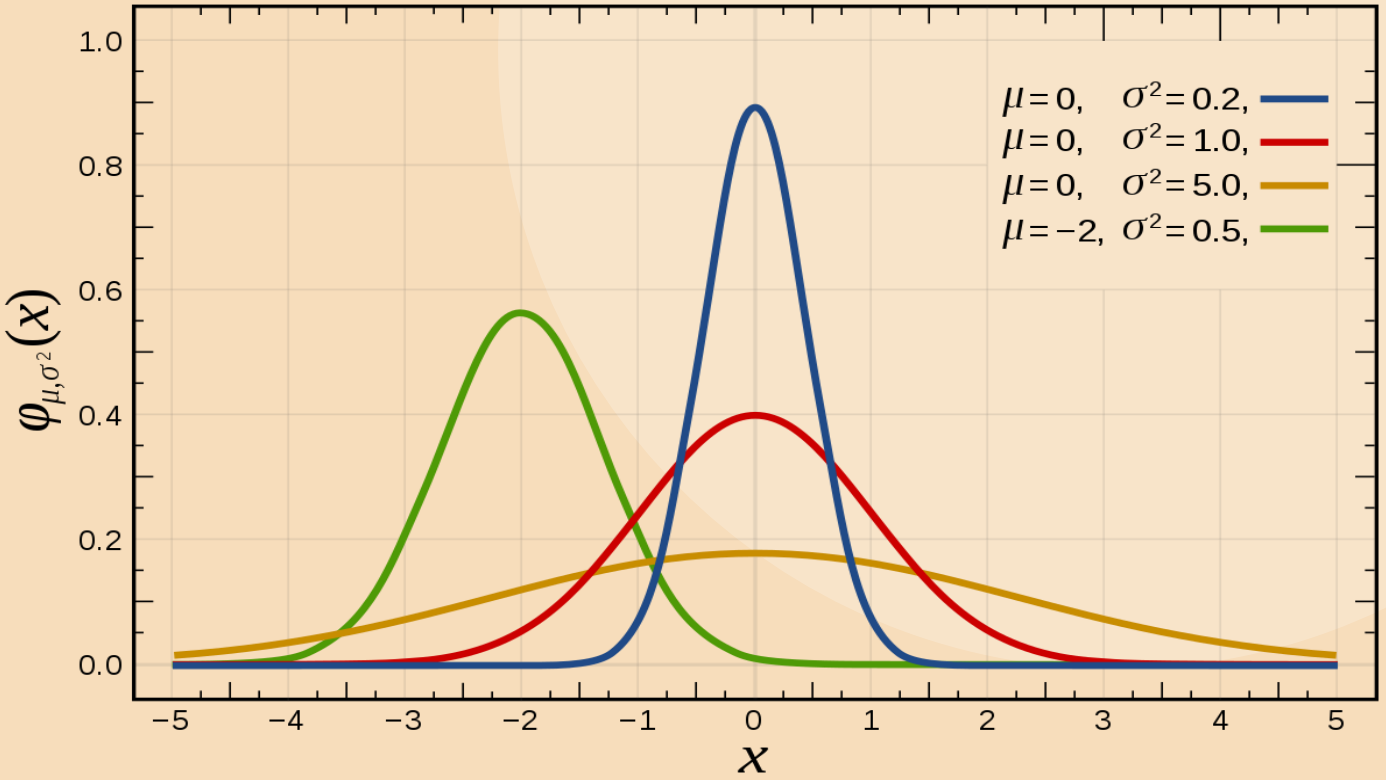


Now our network is ready for learning

Slow convergence and Numerical Stability in Gradient-based optimization algorithms

Normalization

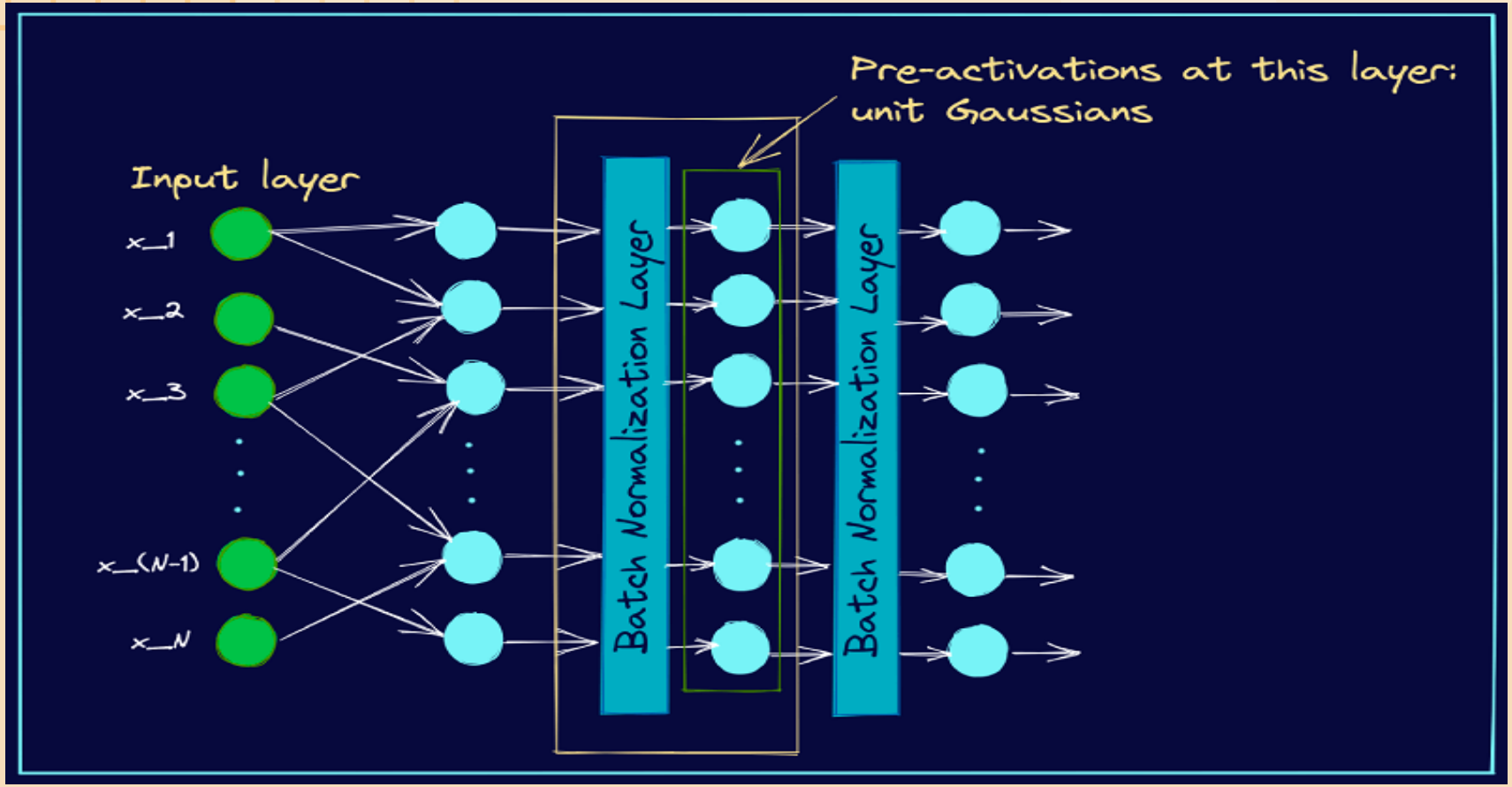
First Idea is using gaussian distribution



$$Z = \frac{x - \mu}{\sigma}$$

$feature$ $mean$ SD

Normalization



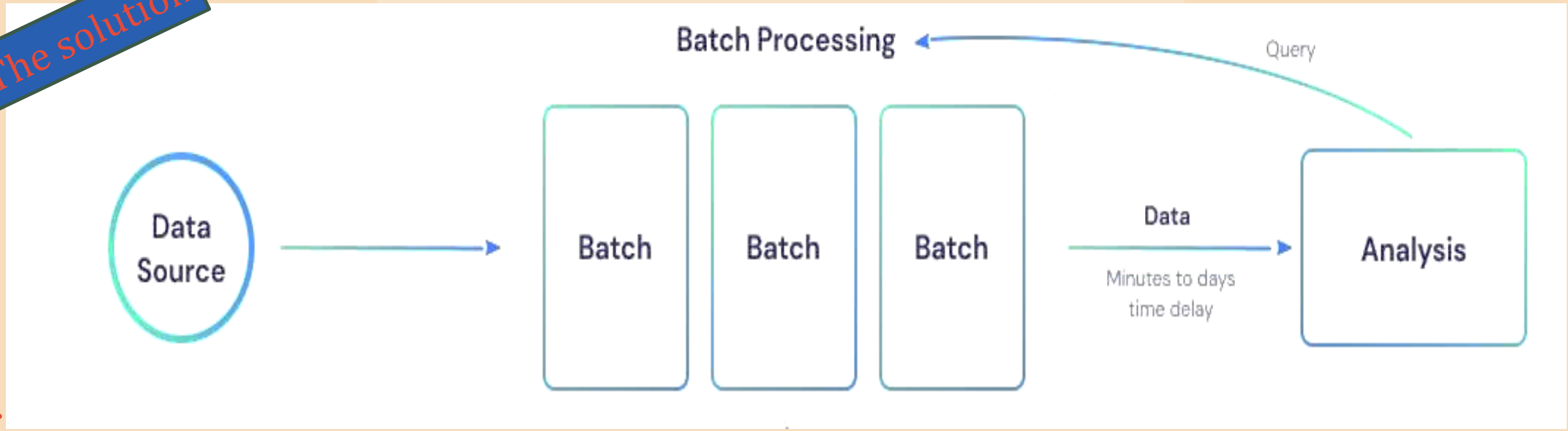
https://www.pinecone.io/learn/batch-layer-normalization/

Why we often batch our data?

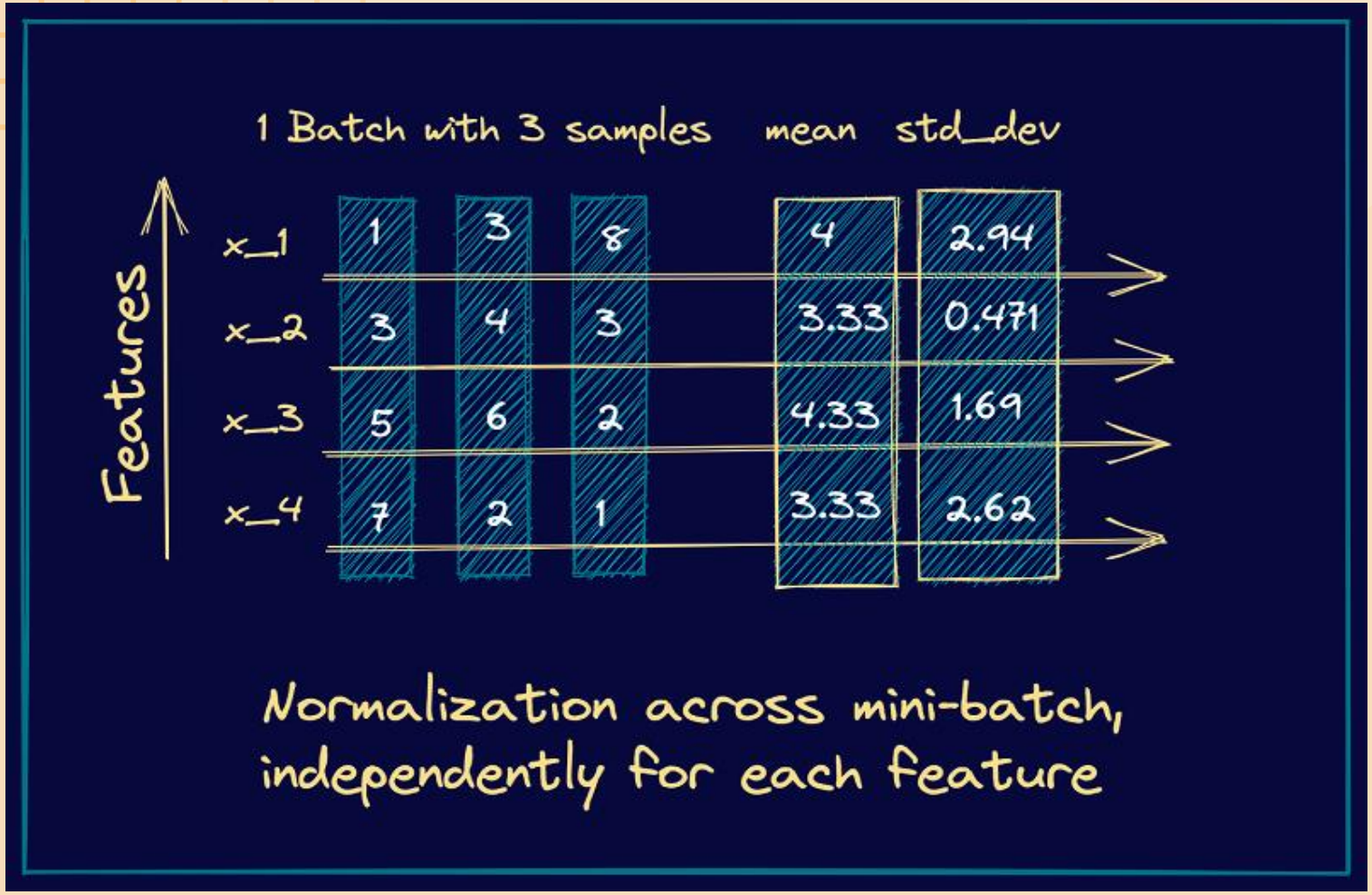
$1 - Having\ large\ dataset!$

We need more RAM

The solution

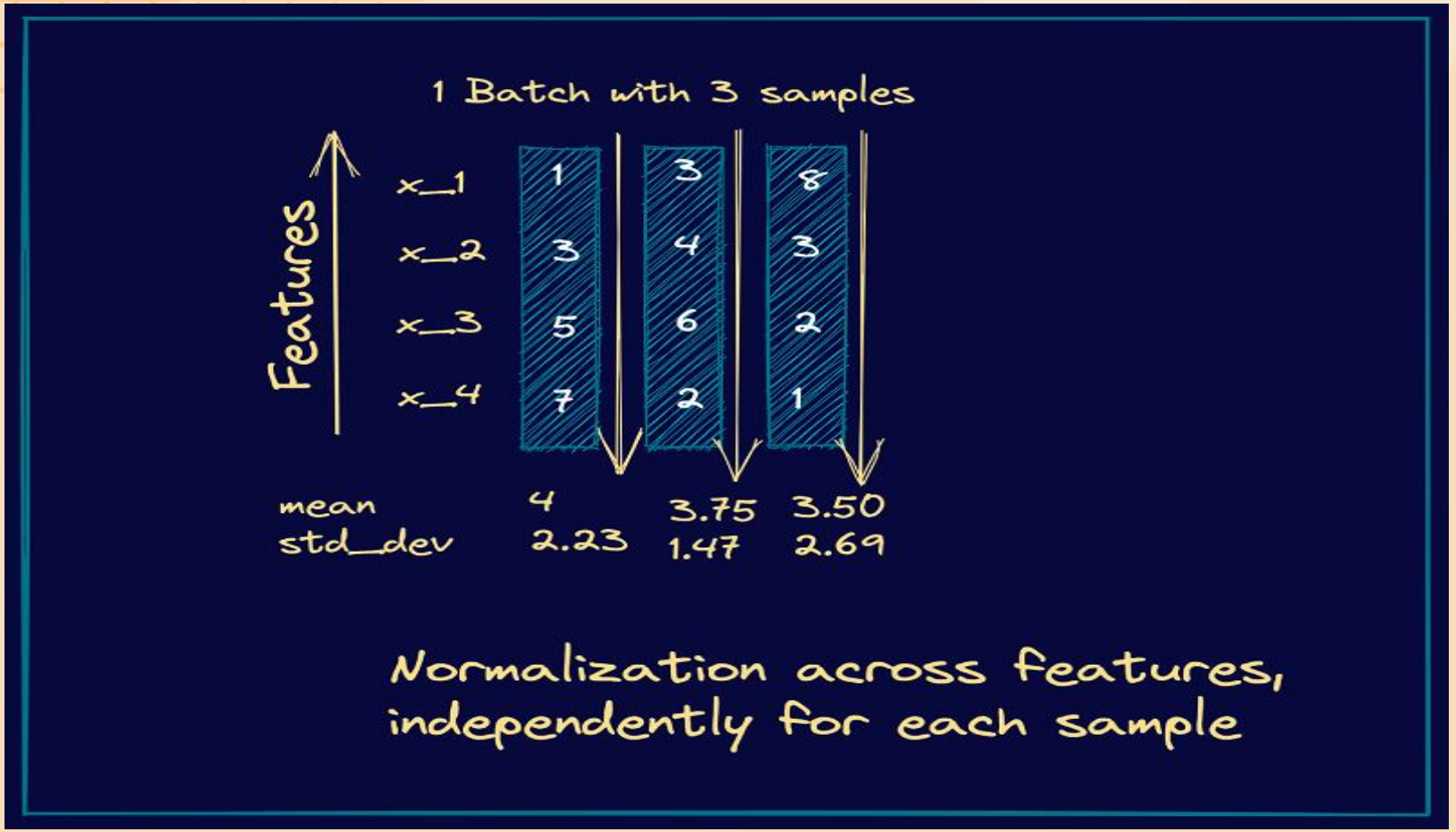


Batch Normalization



https://www.pinecone.io/learn/batch-layer-normalization/

Layer Normalization



https://www.pinecone.io/learn/batch-layer-normalization/

Standardization

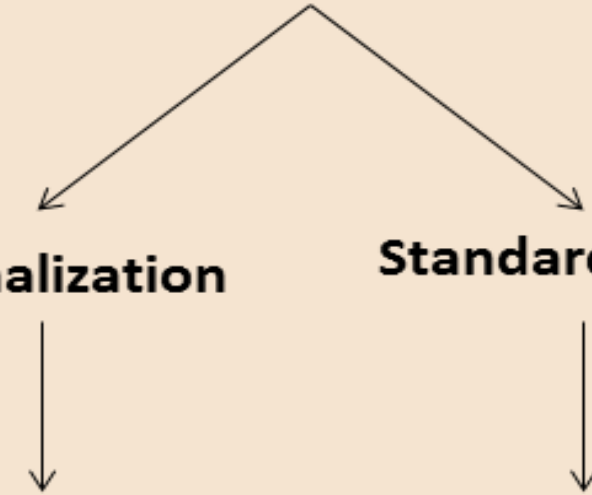
Second Idea is using range of data

$$X_{normalized} = \frac{(X - X_{minimum})}{(X_{maximum} - X_{minimum})}$$

In preprocessing stage

Conclusion

Feature scaling



$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad X' = \frac{\text{X} - \text{Mean}}{\text{Standard deviation}}$$

normalizing data is a crucial step in the preprocessing stage of machine learning pipelines

03

Equations

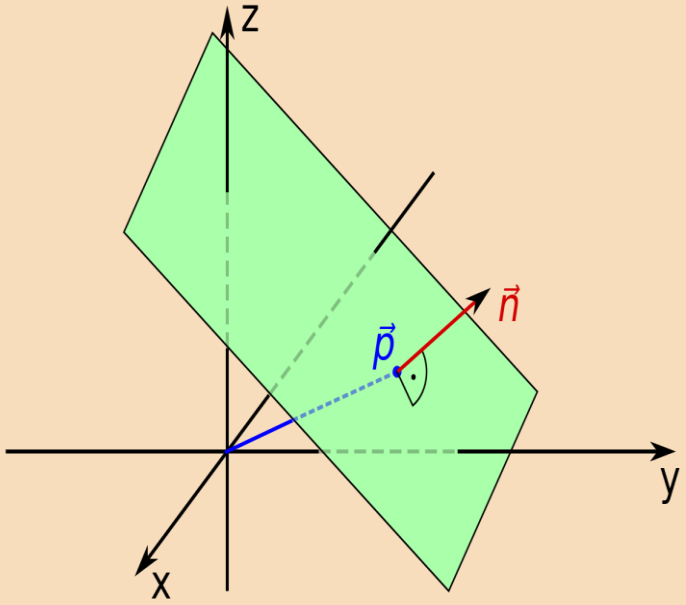
"Equations are the silent orchestrators of the mathematical symphony."

Homogenous equation

$$AX = 0$$

Answers are null vectors of A

Linear (affine) equation



$$AX = b$$

Question

When this equation has answer?

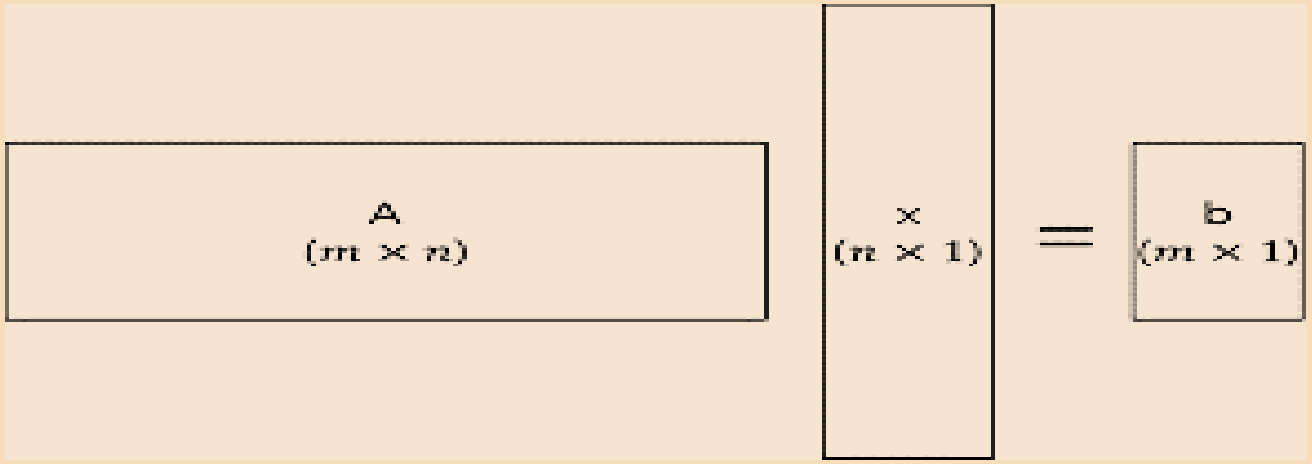
Answer : $if\ b \in C(A)$

M.Tamjidi

Fat matrix

Under determined system

if there are fewer equations than unknowns



$b \in C(A) = R^m$ == > *always has answer*

Thin matrix

Over determined system

If there are more equations than unknowns

$$\begin{bmatrix} 2 & 1 \\ -3 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 1 \end{bmatrix}$$

$b \notin C(A) = R^2$ ==> *often has no answer in reality*

M.Tamjidi

Thin matrix

Noisy measurement system

If there are more equations(and noisy) than unknowns

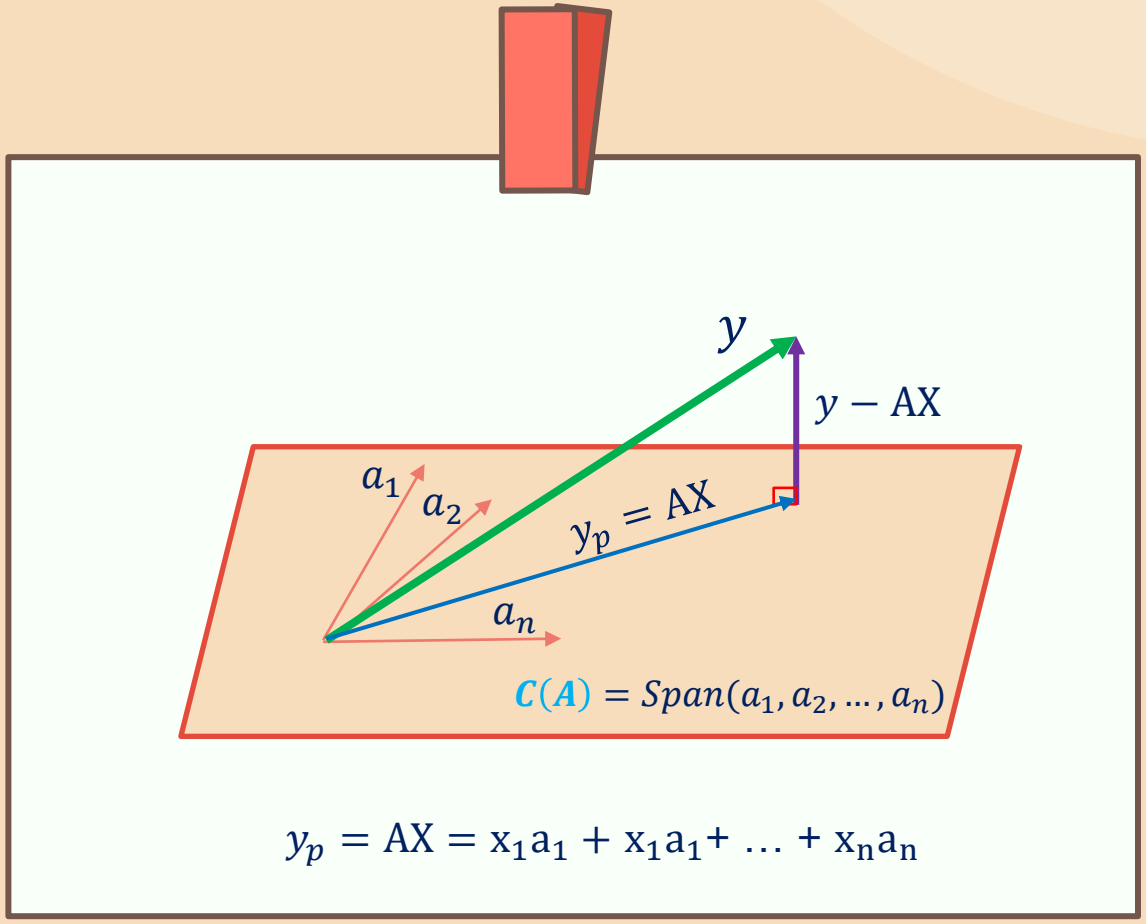
$$AX = y$$

Converting this problem to minimization problem

$$X = argmin\ ||AX - y||$$

Least squares solution

$$X = \left(A^T A\right)^{-1} A^T y$$



For thin matrices

Least squares solution

Least squares solution

$$X = (A^T A)^{-1} A^T y$$

What does the term "Squares" mean?

$$X = argmin\ ||AX - y||$$

$$X = argmin\ ||AX - y||^2$$

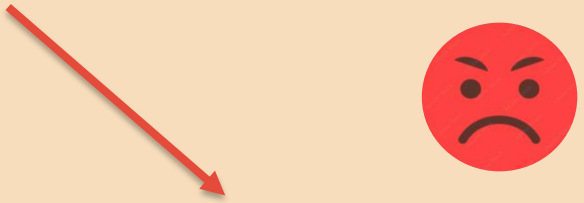
$$X = argmin\ (a_1^T X - y_1)^2 + (a_2^T X - y_2)^2 + \ \dots\ + (a_1^T X - y_1)^2$$

$$X = argmin\left(\left\|\begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{bmatrix}\right\|^2 [X] - [Y]\right)$$

Least squares , a good way but only for small data sets!!

Inverse of matrix must exsist(only for non-singular matrices)
We also have this problem for even near-singular (ill-conditions matrices)

we have to get inverse and hold it in memory



We need more RAM

```python
from scipy.linalg import lstsq

# Define the coefficients matrix (A) and the right-
hand side vector (b)
A = np.array([[1, 2], [3, 4], [5, 6]])
b = np.array([7, 8, 9])

# Solve the system using scipy.linalg.lstsq
x, _, rank, _ = lstsq(A, b)
```

04

Eigens and Positive Definity

"Essential tools for unveiling the underlying structures dictating matrix and transformation behaviors."

Eigne values and vectors

For a matrix A there is a vector V ≠ 0 such that AV = λV

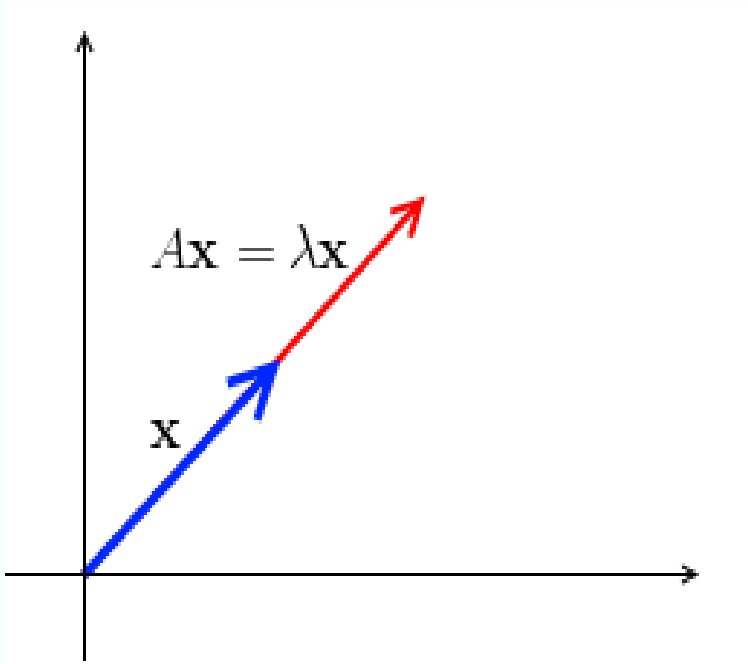
Matrix we are finding the eigenvector/eigenvalue of

eigenvalue

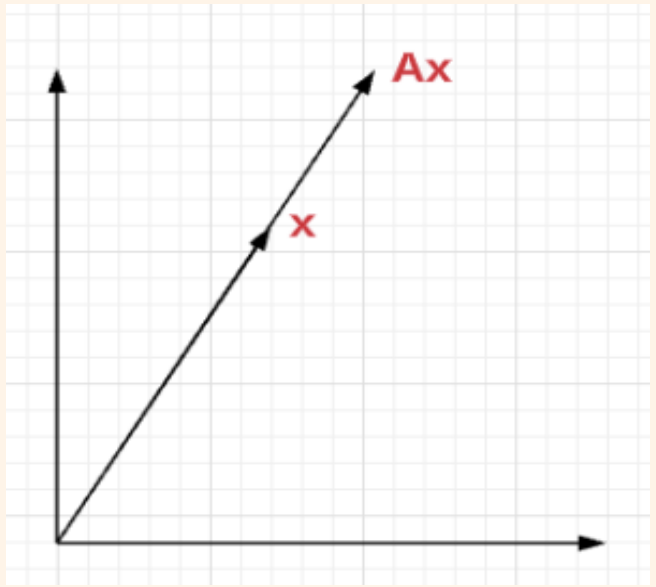
$$Ax = \lambda x$$

$$(A - \lambda I)x = 0$$

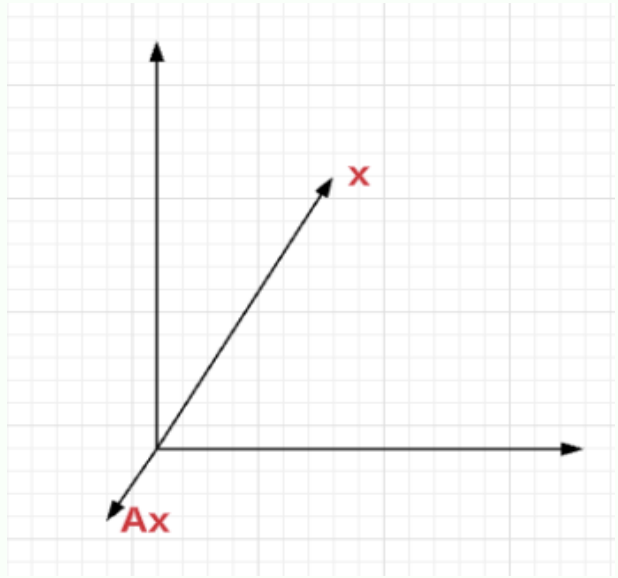
identity matrix



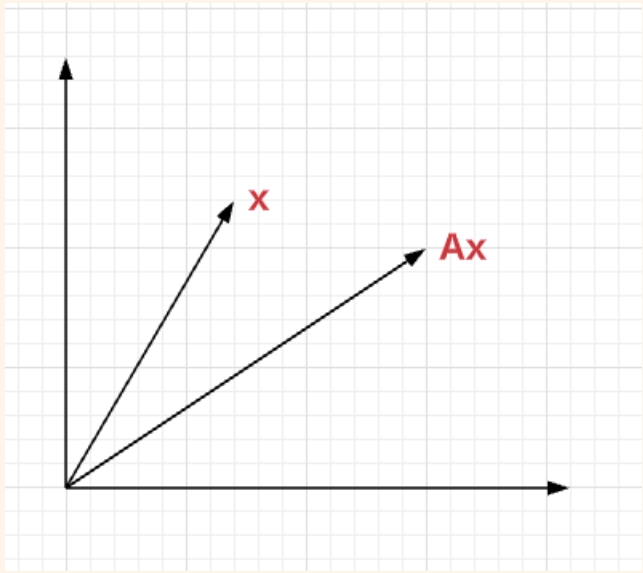
Which one is eigen vector?



X is eigen vector



X is eigen vector



X is not eigen vector

How to find eigen pairs?

Setting its representative equal to zero.

$$Ax - \lambda x = 0$$
$$(A - \lambda I)x = 0$$

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}$$

$$\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

$$\det(A - \lambda I) = 0$$

$$\det\left(\begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0$$

$$\det\left(\begin{bmatrix} 1-\lambda & 4 \\ 3 & 2-\lambda \end{bmatrix}\right) = 0$$

$$(1-\lambda)(2-\lambda) - 12 = 0$$

$$\lambda^2 - 3\lambda - 10 = 0$$

$$(\lambda - 5)(\lambda + 2) = 0$$

$$\lambda = 5, -2$$

How to find eigen pairs?

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix}$$

$(A - \lambda I)X = 0$

$\rightarrow \lambda = 5 \rightarrow \begin{bmatrix} -4 & 4 \\ 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \begin{matrix} -4x_1 + 4x_2 = 0 \\ 3x_1 - 2x_2 = 0 \end{matrix} \} x_1 = x_2$

$X = \begin{bmatrix} \alpha \\ \alpha \end{bmatrix}$

$\alpha \in \mathbb{R}^2 - \{0\}$

$\lambda = -2 \rightarrow \begin{bmatrix} 3 & 4 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow 3x_1 + 4x_2 = 0$

$X = \begin{bmatrix} 4\beta \\ -3\beta \end{bmatrix}$

$\beta \in \mathbb{R}^2 - \{0\}$

$eigenPairs = \left(5, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right), \left(-2, \begin{bmatrix} 4 \\ -3 \end{bmatrix} \right)$

λ, v

Eigens of symmetirc matrices

If a matrix is real and symmetric, its eigenvalues are all real

If a matrix is real and symmetric, its eigenvectors can be chosen to be orthogonal.

We can select the orthonormal eigenvectors of a symmetric matrix and use each of them as columns of a matrix. This matrix is named the modal matrix.

Modal matrix is orthogonal and non-singular matrix

$$M = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ v_1 & v_2 & & v_N \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

Eigen Decomposition

Eigen decomposition allows us to decompose a symmetric matrix into the modal matrix (composed of the eigenvectors as columns), a diagonal matrix of eigenvalues, and the transpose of the modal matrix

$$AV = V\Lambda \rightarrow A = V\Lambda V^{-1} \rightarrow A = V\Lambda V^{T}$$

Inverse is equal to Transpose

$$A = \begin{bmatrix} | & | & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \begin{bmatrix} | & | & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \\ | & | & | \end{bmatrix}^{-1}$$

square matrix

the inverse exists only if eigenvectors are linearly independent

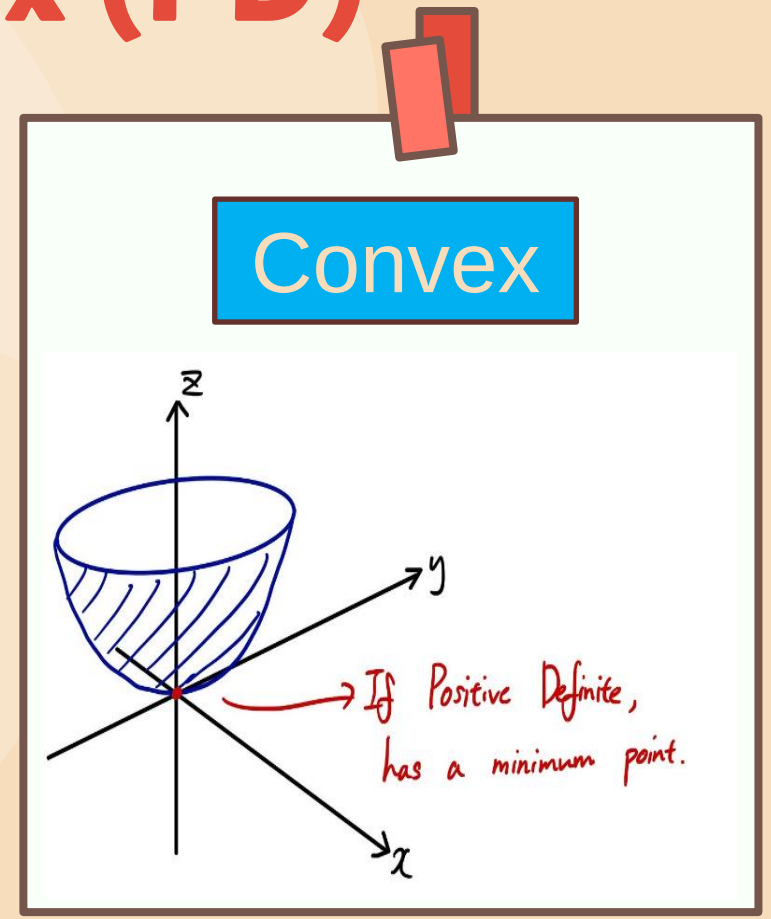
Eigne values and vectors

```python
import numpy as np

# Define a matrix
a = np.array([[1, 2],
              [3, 4]])

# Compute the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(a)
```

Positive definite matrix (PD)

1. It is symmetric

2. $x^T Ax > 0$ for all $x \neq 0$



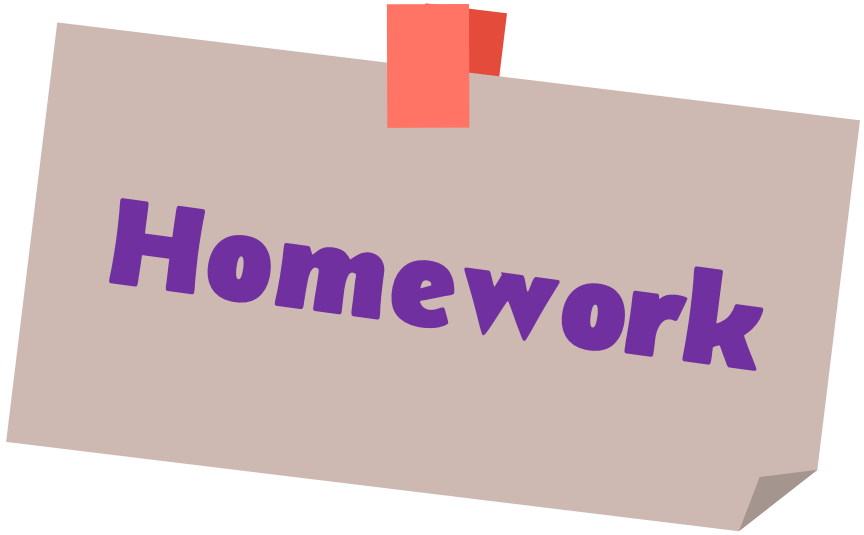
For a positive definite matrix all of its eigenvalues are positive

Positive definite matrix (PSD)

1. It is symmetric

2. $x^T Ax \geq 0$ for all $x \neq 0$

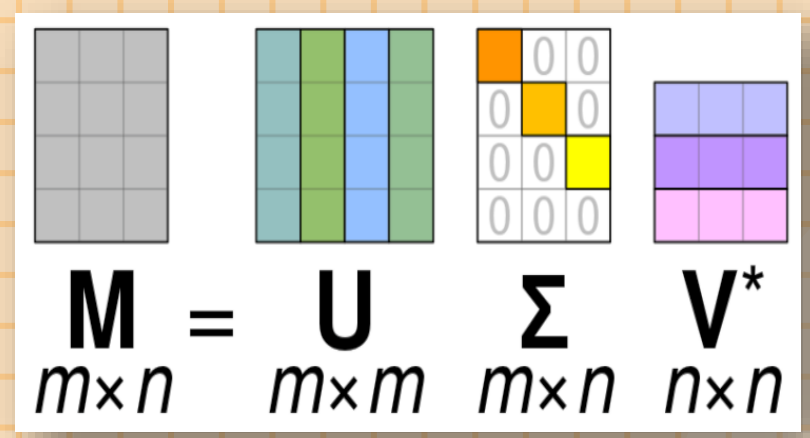
For a positive definite matrix all of its eigenvalues are non-negative



Prove that a matrix is positive definite if and only if its eigenvalues are positive.

Hint: you may need to use eigen decomposition

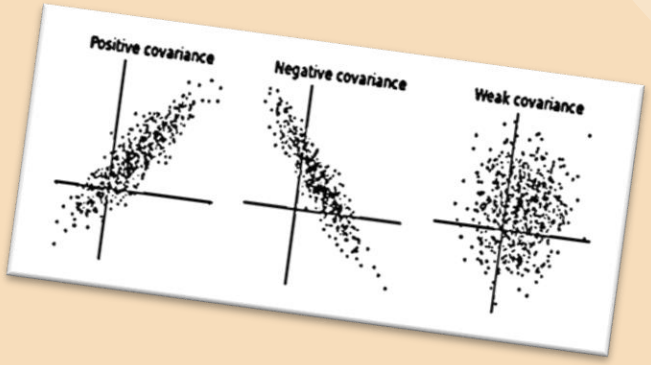
After this assignment you will be mastered at positive definite and eigenvalue concepts



05

SVD and Statistics

“If you want everything from a matrix, utilize Singular Value Decomposition"



M.Tamjidi

SVD

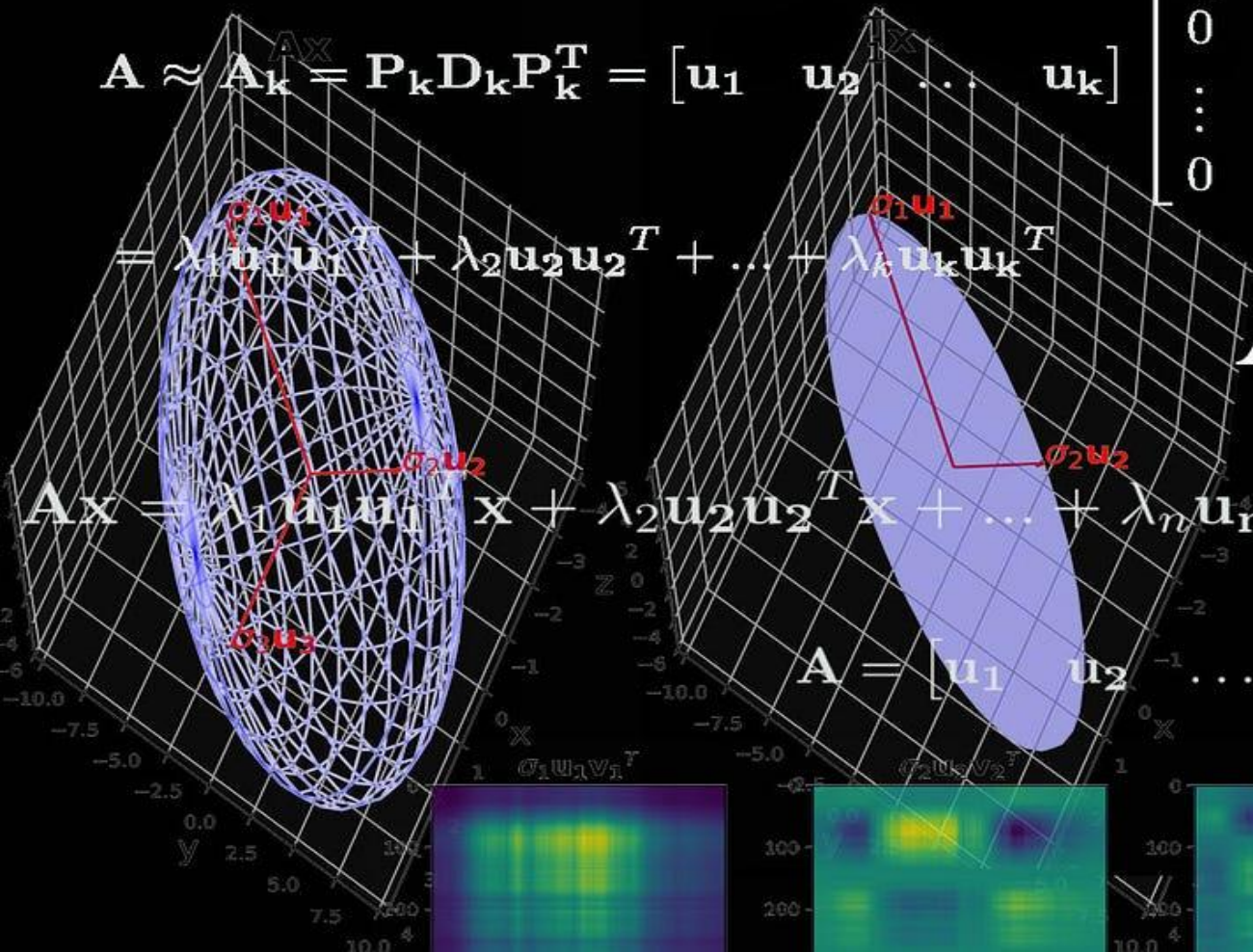
$$A \approx A_k = P_k D_k P_k^T = \begin{bmatrix} u_1 & u_2 & \dots & u_k \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \lambda_k \end{bmatrix} \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_k^T \end{bmatrix}$$

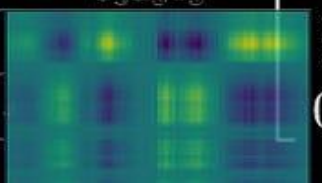
$$= \lambda_1 u_1 u_1^T + \lambda_2 u_2 u_2^T + \dots + \lambda_k u_k u_k^T$$

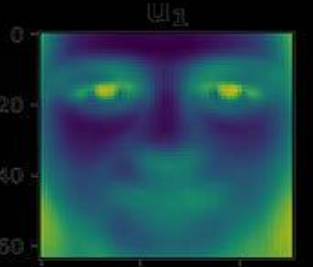
$$Ax = \lambda_1 u_1 u_1^T x + \lambda_2 u_2 u_2^T x + \dots + \lambda_n u_n u_n^T x$$

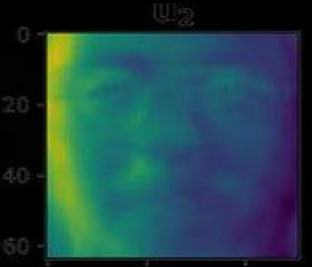
$$A = \begin{bmatrix} u_1 & u_2 & \dots & u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix}$$

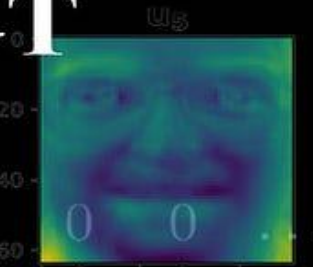
$$A = U \Sigma V^T$$

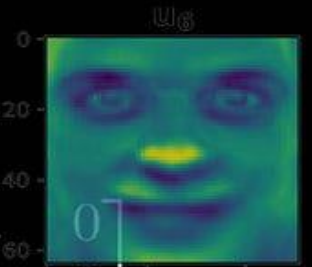






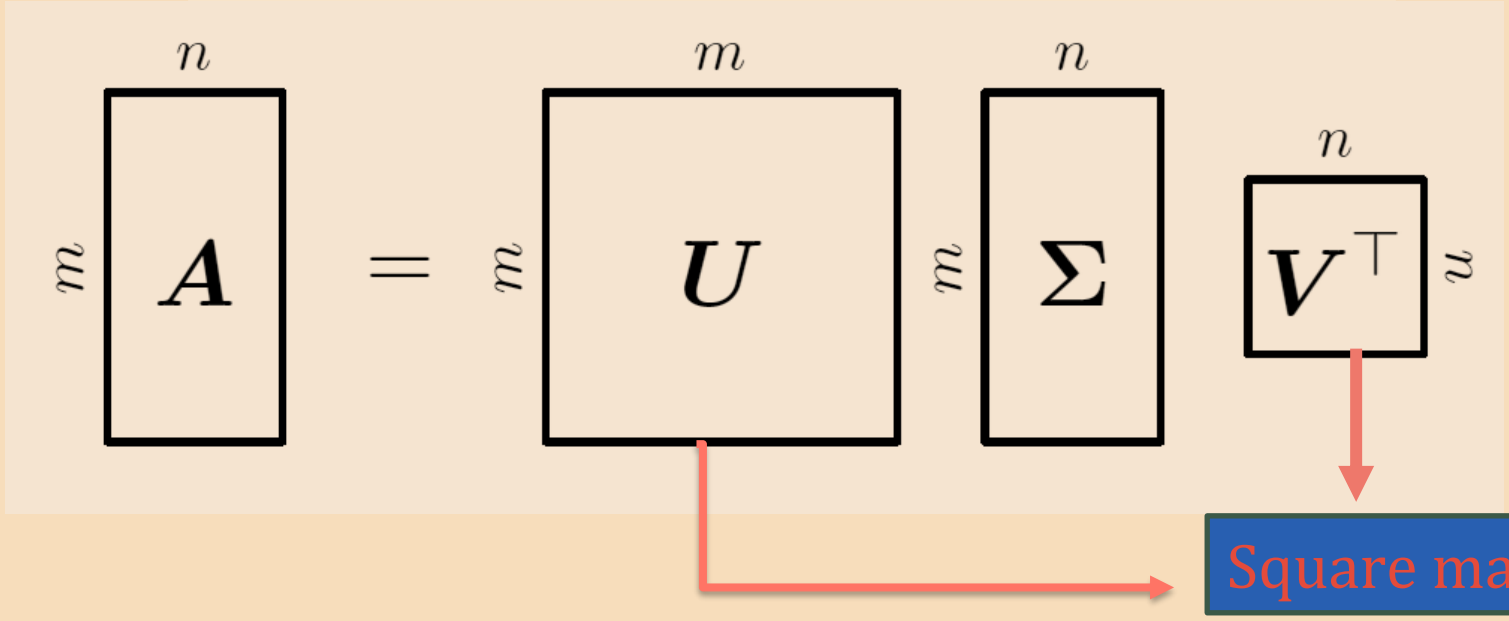






Singular Value Decomposition(SVD)

Any matrix with dimensions *n*×*m* can be decomposed into two orthogonal matrices and one diagonal matrix.



more detail ...

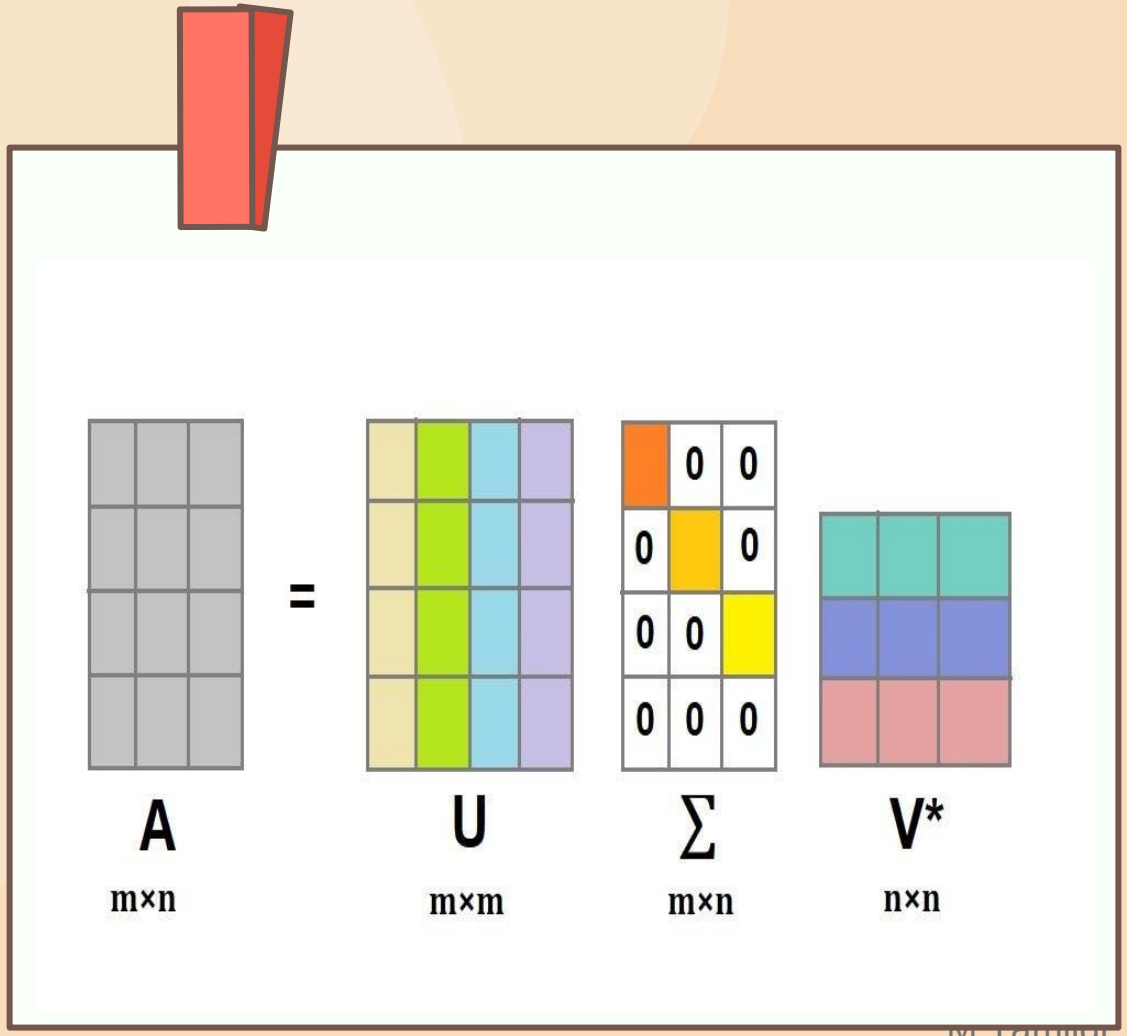
Singular Value Decomposition(SVD)

$$A = U D V^T$$

Left singular vectors

Singular values

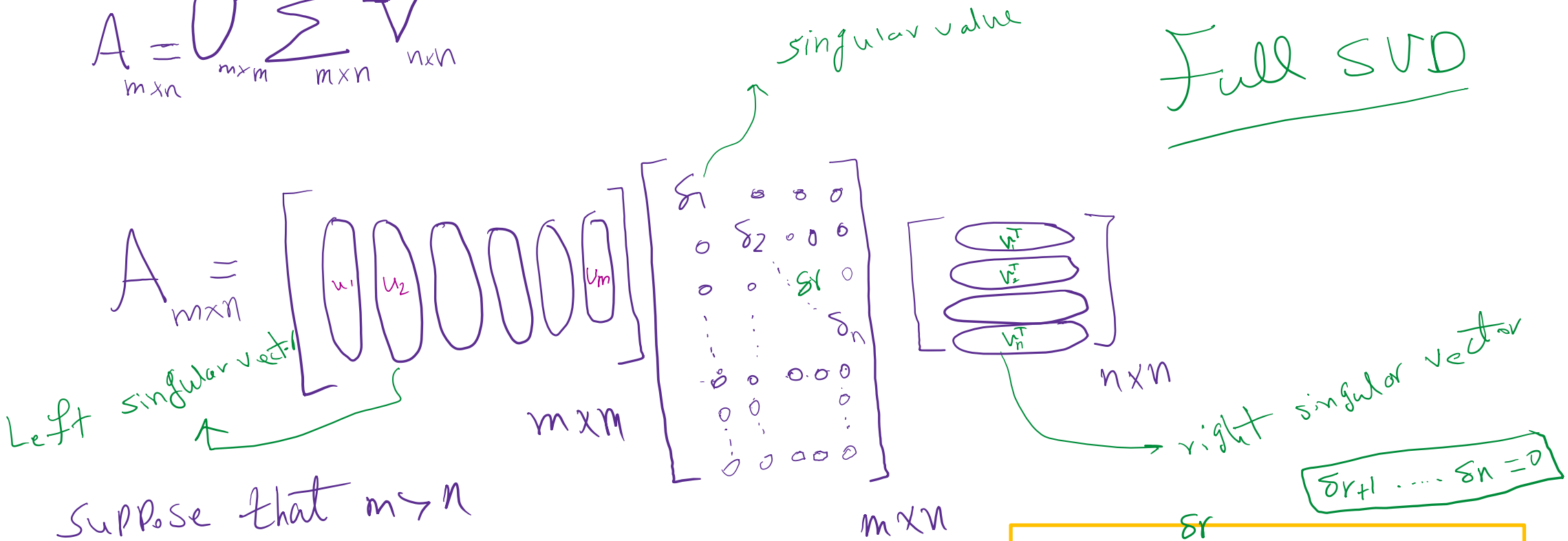
Right singular vectors

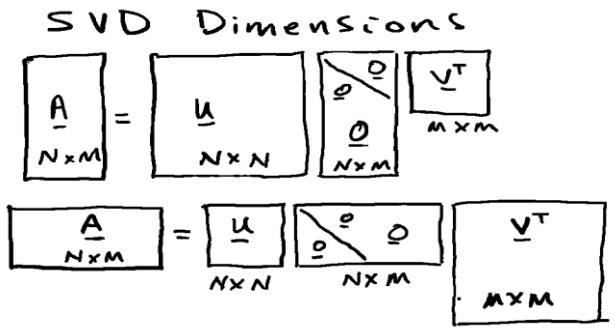


more detail ...

FULL SVD

$$A_{m\times n} = U_{m\times m}\,\Sigma_{m\times n}\,V^{T}_{n\times n}$$





$\delta_{r+1} \dots \delta_n = 0$

$\delta_1, \delta_2 \dots \delta_r \geq 0$

$\delta_1 \geq \delta_2 \geq \dots \geq \delta_r \geq 0$

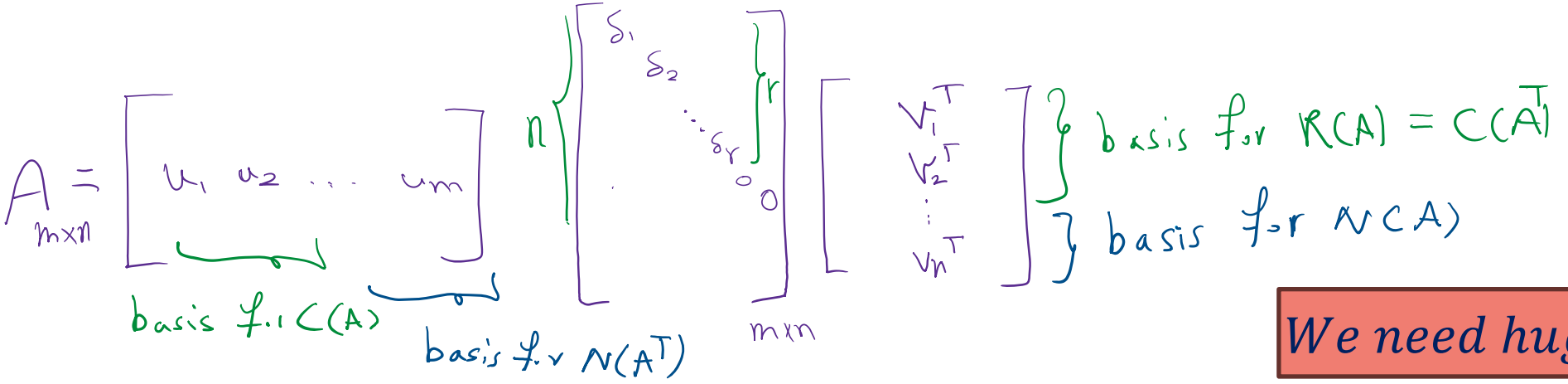
more detail ...

FULL SVD

with SVD you can see everything from the matrix

Such as :

Rank – column space – row space –null space –left null space



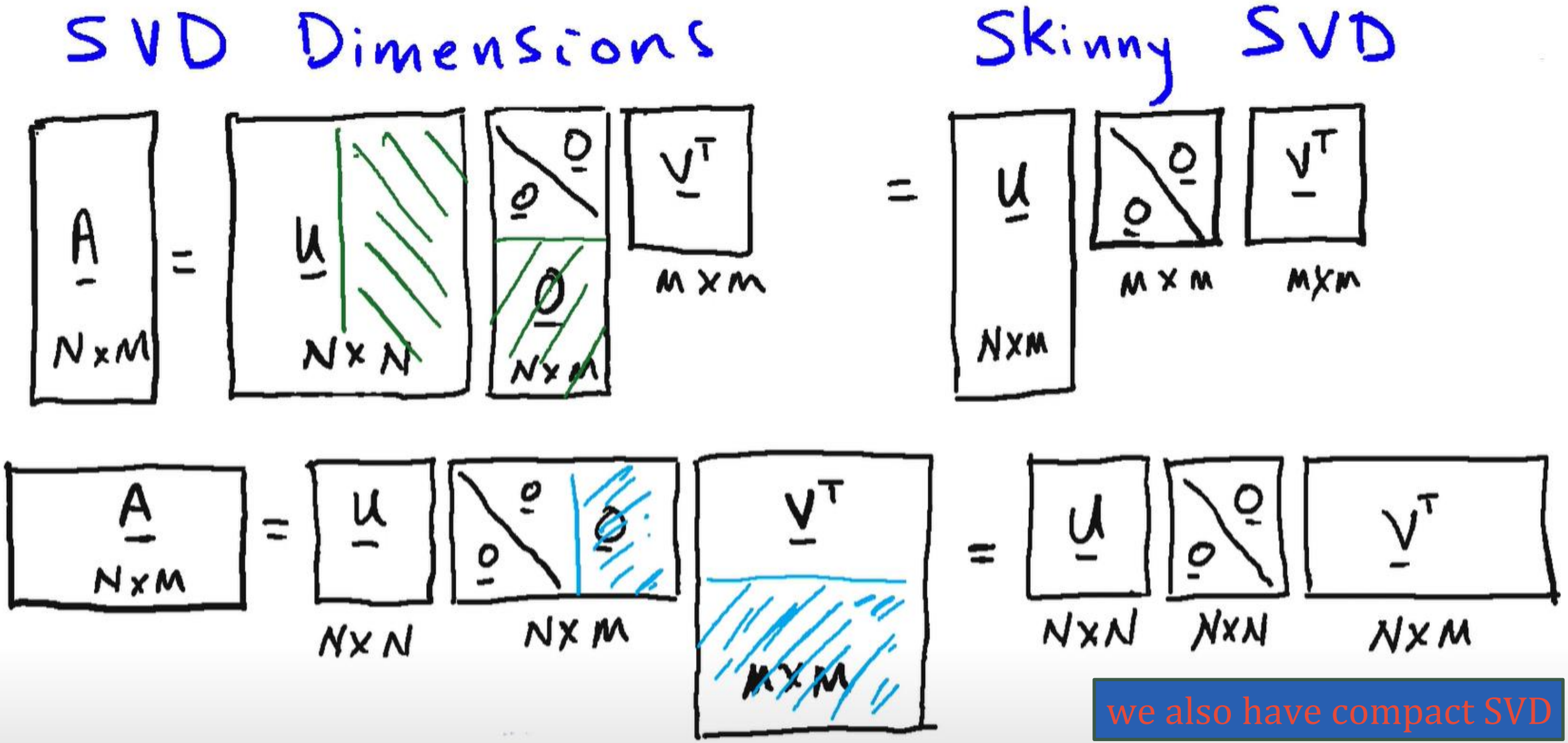
We need huge memory

$u_1\ u_2 \cdots u_r \rightarrow$ basis for $C(A)$

$u_{r+1}\ u_{r+2} \cdots u_m \rightarrow$ basis for $N(A^T)$

more detail ...

SKINNY SVD



we also have compact SVD

Example

Singular Value Decomposition(SVD)

$$A = \begin{bmatrix} 1 & -0.8 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = U\Sigma V^\top$$

$$A = \begin{bmatrix} -0.79 & 0 & -0.62 \\ 0.38 & -0.78 & -0.49 \\ -0.48 & -0.62 & 0.62 \end{bmatrix} \begin{bmatrix} 1.62 & 0 \\ 0 & 1.0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -0.78 & 0.62 \\ -0.62 & -0.78 \end{bmatrix}$$

Singular Value Decomposition(SVD)

You can check the previous example in python

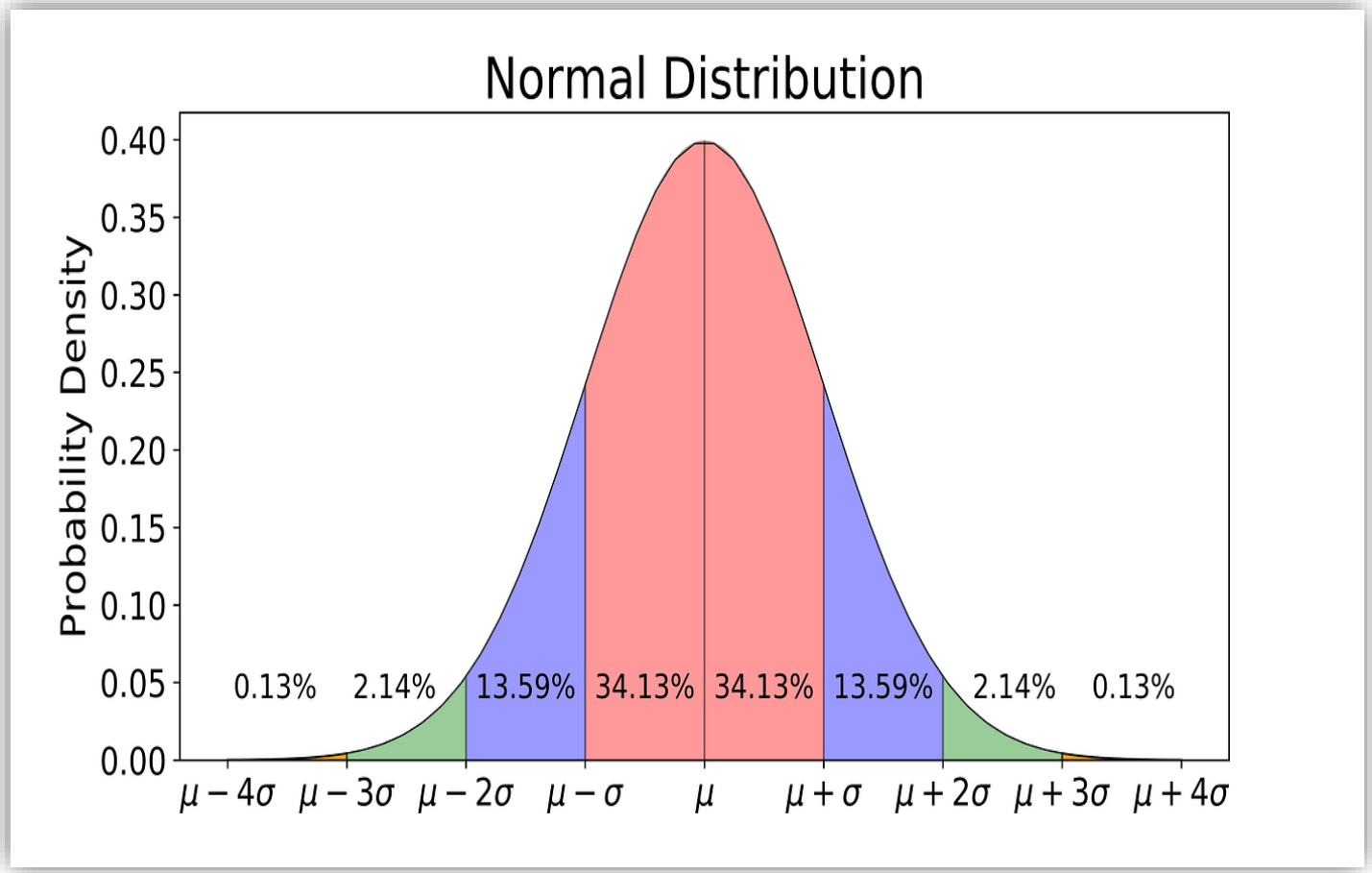
```python
import numpy as np

A = np.array([[1, -0.8], [0,1], [1,0]])

U, s, Vt = np.linalg.svd(A, full_matrices=True)

print("U:")
print(U)
print("\nSingular values:")
print(s)
print("\nV transpose:")
print(Vt)
```

M.Tamjidi

In univariate case !

Normal Distribution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

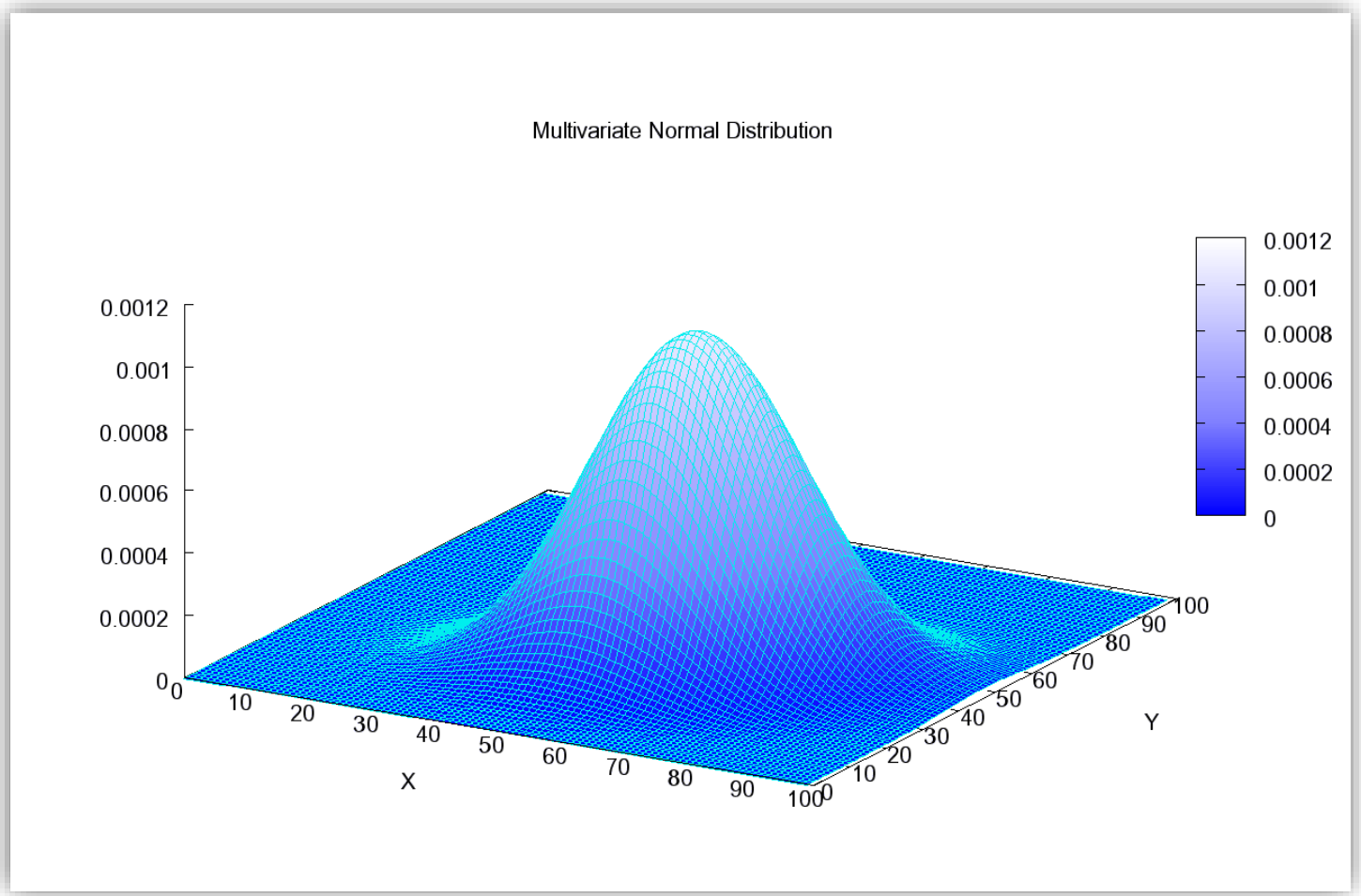


https://en.wikipedia.org/wiki/Multivariate_normal_distribution

In multivariate case !

Normal Distribution

$$f_{\mathbf{x}}(x_1, \ldots, x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}$$



https://en.wikipedia.org/wiki/Multivariate_normal_distribution

Common Formula

Covariance matrix

Points

(x_1, y_1)
(x_2, y_2)
$\vdots$
(x_n, y_n)

Mean

$$(\mu_x, \mu_y) = (\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{n}\sum_{i=1}^{n} y_i)$$

x-Variance

$$var(x) = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_x)^2$$

y-Variance

$$var(y) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \mu_y)^2$$

Covariance

$$cov(x, y) = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y)$$

Covariance matrix

$$\Sigma = \begin{pmatrix} var(x) & cov(x, y) \\ cov(x, y) & var(y) \end{pmatrix}$$

In univariate case !

Covariance matrix

$$E[X] = \sum_i x_i f(x_i)$$

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx$$

$$\begin{aligned}
\operatorname{Var}(X) &= \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] \\
&= \mathrm{E}\left[X^2 - 2X\,\mathrm{E}[X] + \mathrm{E}[X]^2\right] \\
&= \mathrm{E}\left[X^2\right] - 2\,\mathrm{E}[X]\,\mathrm{E}[X] + \mathrm{E}[X]^2 \\
&= \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2 \longrightarrow Var(X) = E(X^2) - \mu^2
\end{aligned}$$

$$\operatorname{Cov}_{X,Y}[x, y] := \mathbb{E}_{X,Y}\left[(x - \mathbb{E}_X[x])(y - \mathbb{E}_Y[y])\right]$$

$$\operatorname{Cov}[x, y] = \mathbb{E}[xy] - \mathbb{E}[x]\mathbb{E}[y]$$

M.Tamjidi

In multivariate case !

Covariance matrix

In many case we use $\frac{1}{n}$ before covariance matrix

symmetric & PSD

often PD

$$\operatorname{Cov}[\boldsymbol{x}, \boldsymbol{y}] = \mathbb{E}[\boldsymbol{x}\boldsymbol{y}^\top] - \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{y}]^\top = \operatorname{Cov}[\boldsymbol{y}, \boldsymbol{x}]^\top \in \mathbb{R}^{D \times E}.$$

$$\begin{aligned}
\mathbb{V}_X[\boldsymbol{x}] &= \operatorname{Cov}_X[\boldsymbol{x}, \boldsymbol{x}] \\
&= \mathbb{E}_X[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top] = \mathbb{E}_X[\boldsymbol{x}\boldsymbol{x}^\top] - \mathbb{E}_X[\boldsymbol{x}]\mathbb{E}_X[\boldsymbol{x}]^\top \\
&= \begin{bmatrix}
\operatorname{Cov}[x_1, x_1] & \operatorname{Cov}[x_1, x_2] & \dots & \operatorname{Cov}[x_1, x_D] \\
\operatorname{Cov}[x_2, x_1] & \operatorname{Cov}[x_2, x_2] & \dots & \operatorname{Cov}[x_2, x_D] \\
\vdots & \vdots & \ddots & \vdots \\
\operatorname{Cov}[x_D, x_1] & \dots & \dots & \operatorname{Cov}[x_D, x_D]
\end{bmatrix}.
\end{aligned}$$

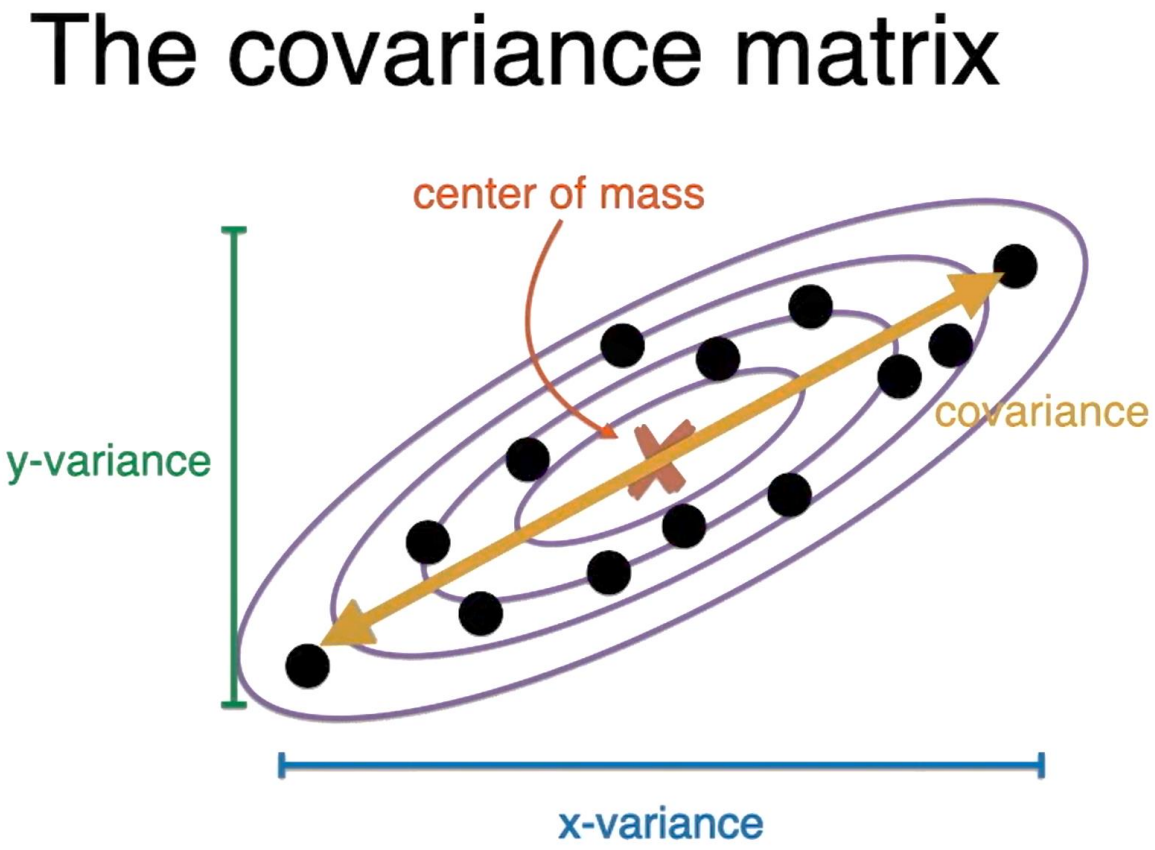
```python
import numpy as np

variance_matrix = np.diag(np.cov(A))
variance_matrix = np.var(A, axis=1)
covariance_matrix = np.cov(A)
```

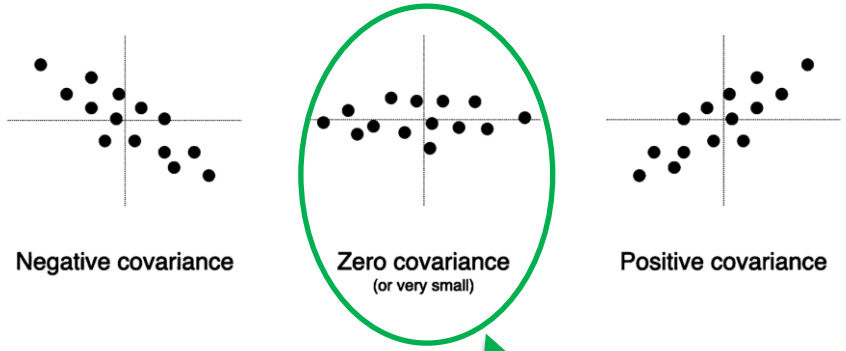
M.Tamjidi

In multivariate case !

Covariance matrix

We could find the relation between features in dataset





We want independent features

$$\mu = Average$$

$$\Sigma = \begin{pmatrix} Var(x) & Cov(x,y) \\ Cov(x,y) & Var(y) \end{pmatrix}$$

Correlation matrix

They have some differences such as:

It uses standardize random variables

Correlation Matrix is not influenced by the change in scale

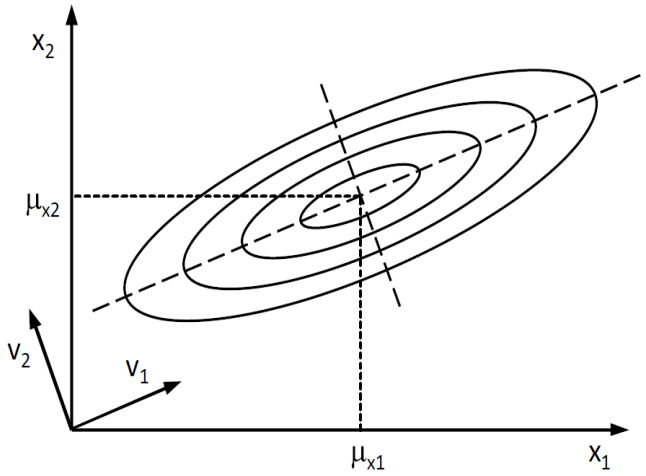
$$\operatorname{corr}[x, y] = \frac{\operatorname{Cov}[x, y]}{\sqrt{\mathbb{V}[x]\mathbb{V}[y]}} \in [-1, 1]$$

In other words, each random variable is divided by its standard deviation (the square root of the variance)

Note: For symmetric functions eigen decomposition and SVD are the same

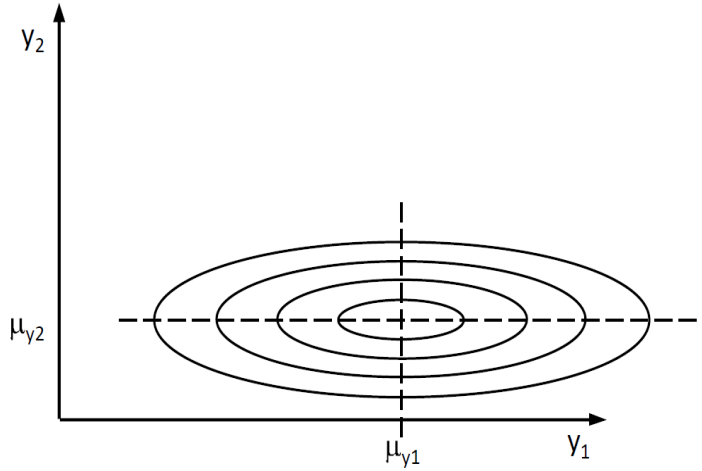
Decorrelating dataset

Correlated dataset



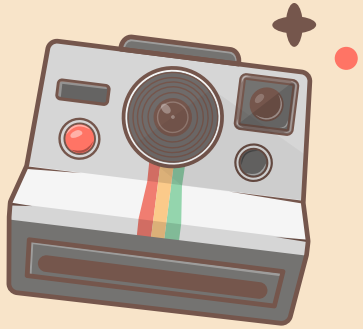
$$Y = M^T X$$

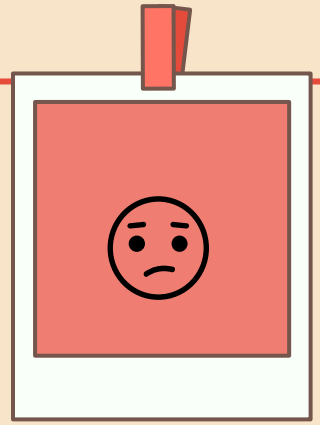
Decorrelated dataset



Which M is modal matrix of Covariance matrix of X

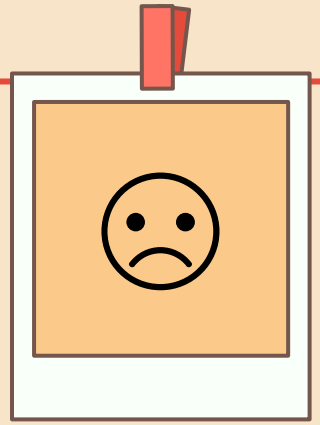
What was not covered?





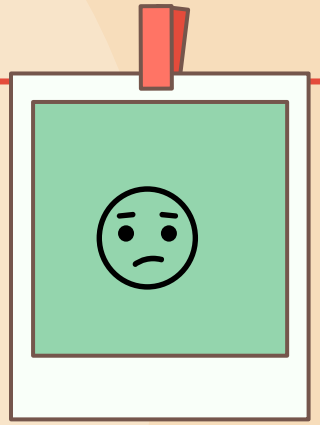
functions

relationships between input and output values



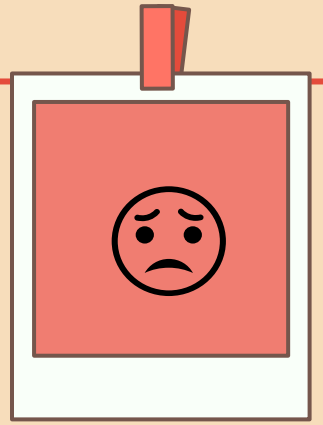
Transformations

Transformations refer to operations that change the shape, orientation, or position of objects in a space.



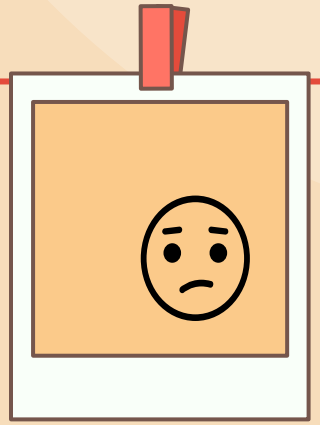
Decompositions

LU – LDR – QR –LD-cholskey decomposition and …



Projections

Projections involve representing higher-dimensional objects or data onto lower-dimensional spaces



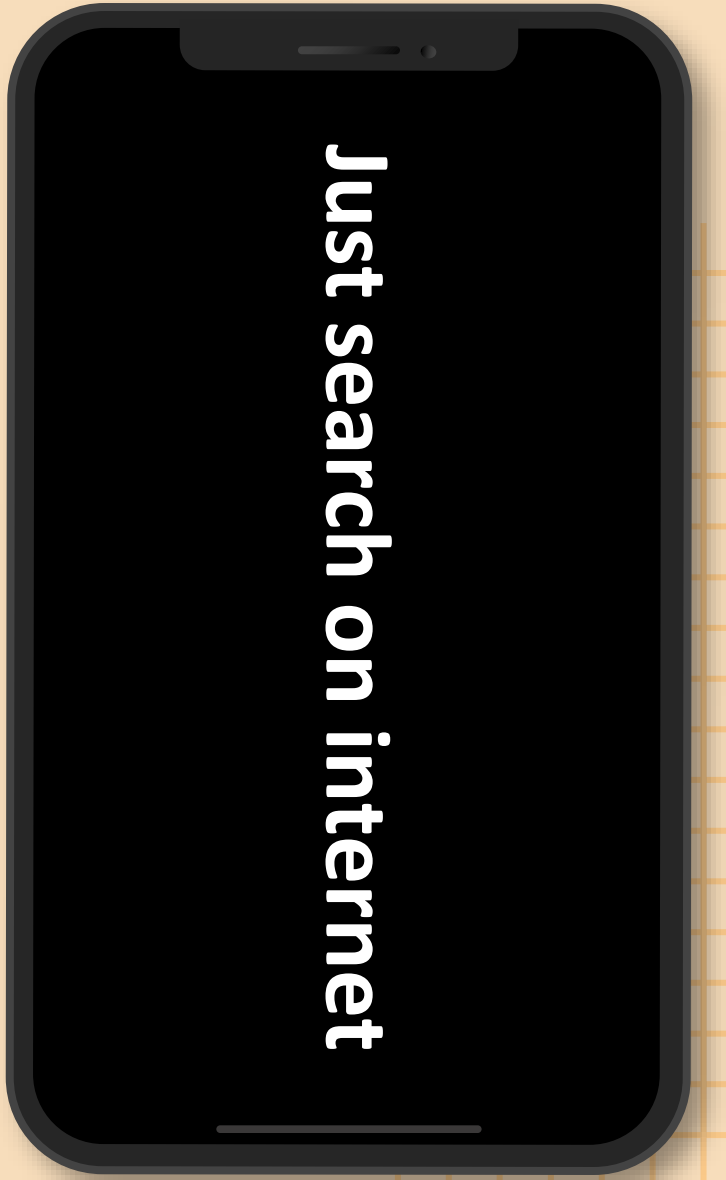
And so on …

Echleon forms

Do you need more ?

YouTube

YouTube is the greatest teacher of all



Mathematics for machine learning

https://mml-book.github.io/book/mml-book.pdf

M.Tamjidi

Other references

1- https://www.deeplearningbook.org/

2- https://www.youtube.com/@behrooznasihatkon7341

3- Bishop, Christopher. "Pattern recognition and machine learning." Springer google schola 2 (2006): 5-43.

Thank you!

Feel free to contact me

Mehrant.0611@gmail.com
Telegram: @Mttnt