DEEP LEARNING

Lecture 3

Linear Algebra & Calculus

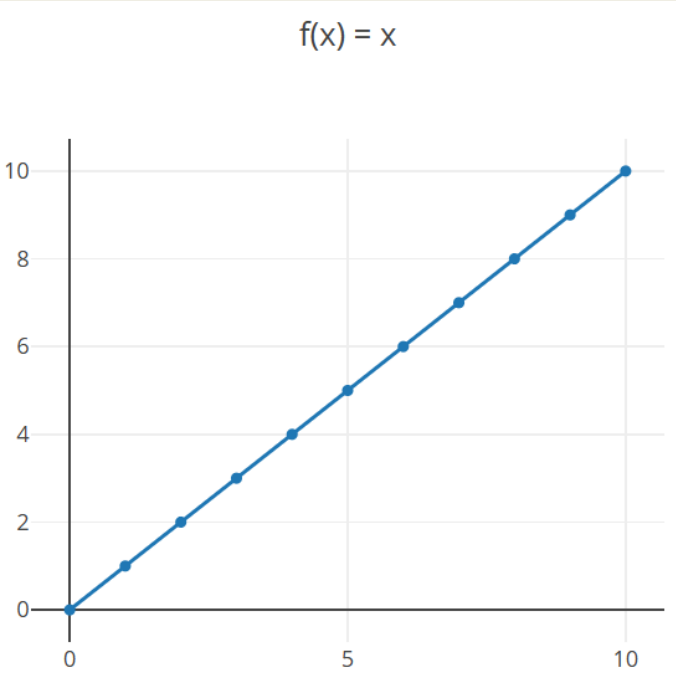
Instructor: Zafar Iqbal

ML & DL Mathematics

- The main branches of **Mathematics** involved in **Machine Learning** are:
  - *Linear Functions*
  - *Linear Graphics*
  - *Linear Algebra*
  - *Probability*
  - *Statistics*

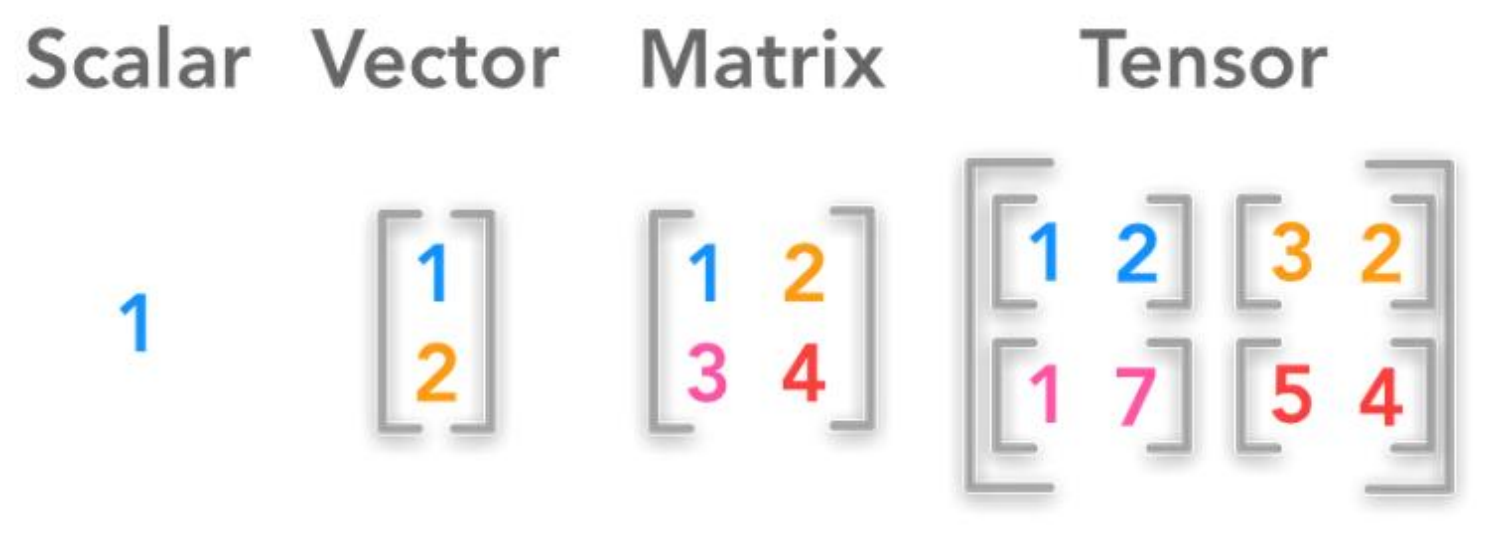
Linear Functions

- Linear means **straight**
- A **linear function** is a **straight line**
- A **linear graph** represents a **linear function**



Linear Algebra

- Linear algebra is the bedrock of data science.
- Knowing linear algebra boosts your ability to understand data science algorithms.

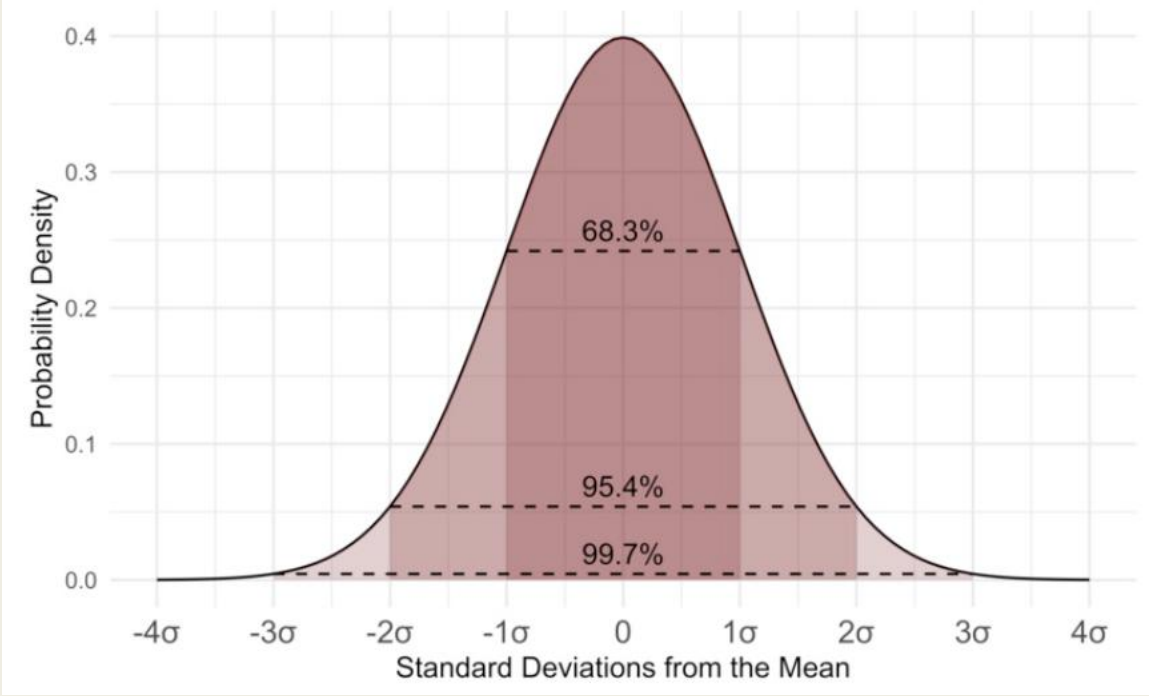


Probability

- **Probability** is how likely something is to occur, or how likely something is true.
- I have 6 balls in a bag: 3 reds, 2 are green, and 1 is blue.
- Blindfolded. What is the probability that I pick a green one?
- Number of **ways** it can happen are 2 (there are 2 greens).
- Number of **outcomes** are 6 (there are 6 balls).
- The probability is 2 out of 6: 2/6 = 0.333333…
- Probability = Ways / Outcomes

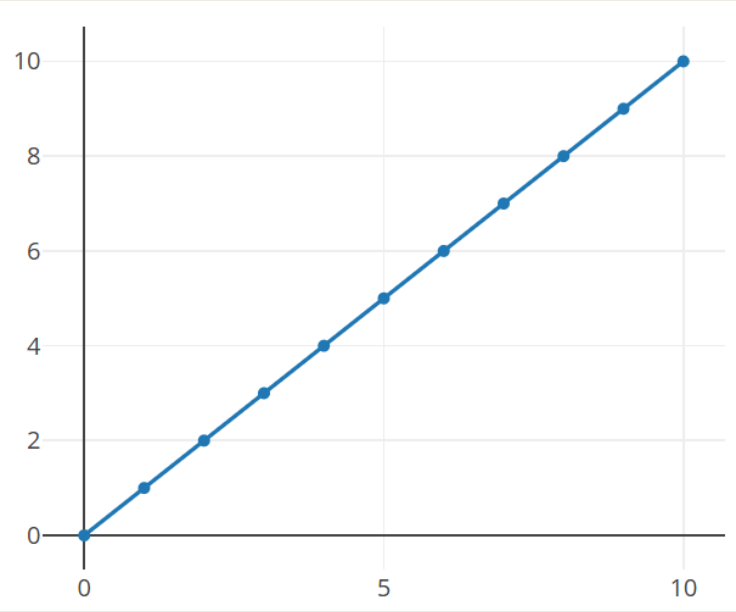
Statistics

- Statistics is about how to collect, analyze, interpret, and present data.
- Statistics works with questions like:
- What is the most **Common?**
- What is the most **Expected?**
- What is the most **Normal?**



Linear Functions

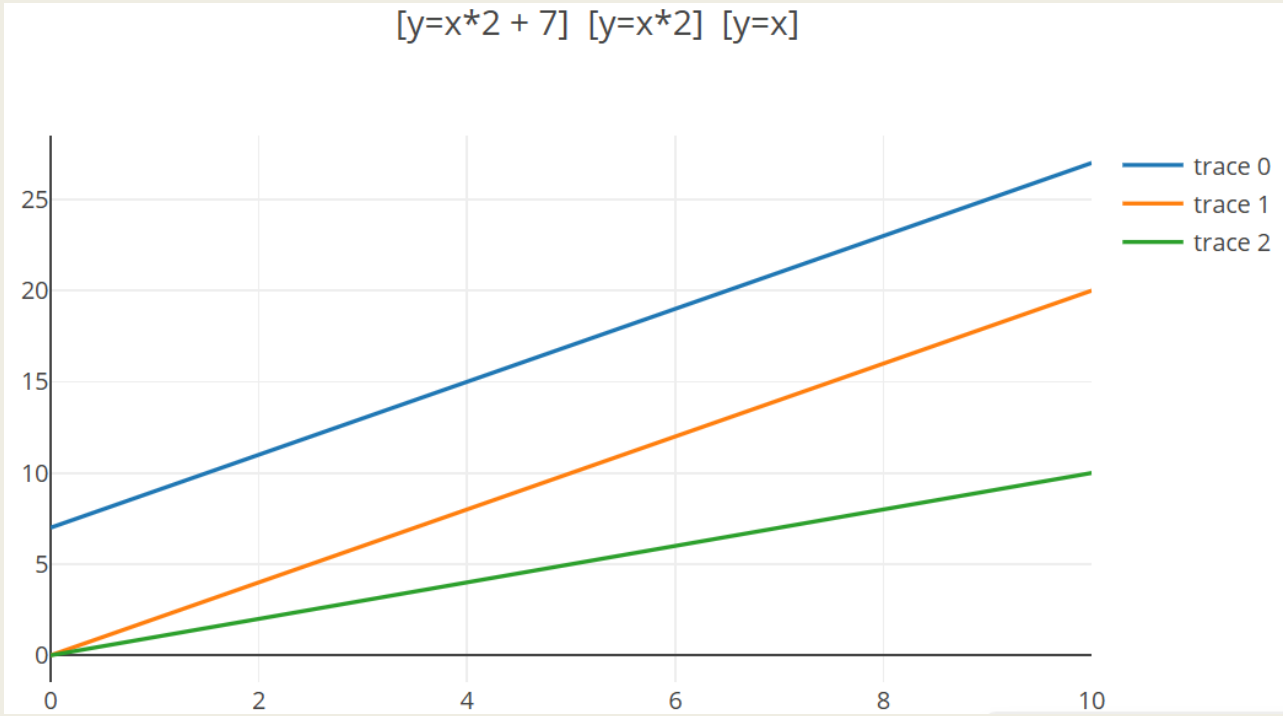
- Linear means **straight**
- A **linear function** is a **straight line**
- A **linear graph** represents a **linear function**
- A **Function** is special relationship where each input has an output.
- A function is often written as **f(x)** where x is the input:



x	y	y = x
1	1	y = x = 1
2	2	y = x = 2
3	3	y = x = 3
4	4	y = x = 4
5	5	y = x = 5

Linear Equations

- A Linear Equation is an equation for a straight line:
- y = x
- y = x*2
- y = x*2 + 7
- y = ax + b
- 5x = 3y
- y/2 = 6



Non-Linear Equations

- A Linear Equation can NOT contain exponents or square roots:
- y = x**2
- y = Math.sqrt(x)
- y = Math.sin(x)

Linear Regression

- A Linear regression tries to model the relationship between two variables by fitting a linear graph to data.
- One variable (x) is considered to be data, and the other (y) is considered to be dependent.
- For example, a Linear Regression can be a model to relate the price of houses to their size.

Linear Least Squares

- Linear algebra is used to solve Linear Equations.
- Linear Least Squares (LLS) is a set of formulations for solving statistical problems involved in Linear Regression.

Linear Algebra

- **Machine Learning** and Deep Learning experts cannot live without **Linear Algebra**:
- ML % DL make heavy use of Scalars
- ML % DL make heavy use of Vectors
- ML % DL make heavy use of Matrices
- ML % DL make heavy use of Tensors

Vectors and Matrices

- **Vectors** and **Matrices** are the languages of data.
- With ML and DL, most things are done with vectors and matrices.
- With vectors and matrices, you can **Discover Secrets**.

Vectors

- Vectors are 1-dimentional **Arrays**
- Vectors have a **Magnitude** and a **Direction**
- Vectors typically describes **Motion** or **Force**
- **Vector Notation**
  - Vectors can be written in many ways. The most common are:

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

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Vectors in Geometry

The image to the left is a **Vector**.

The **Length** shows the **Magnitude**.

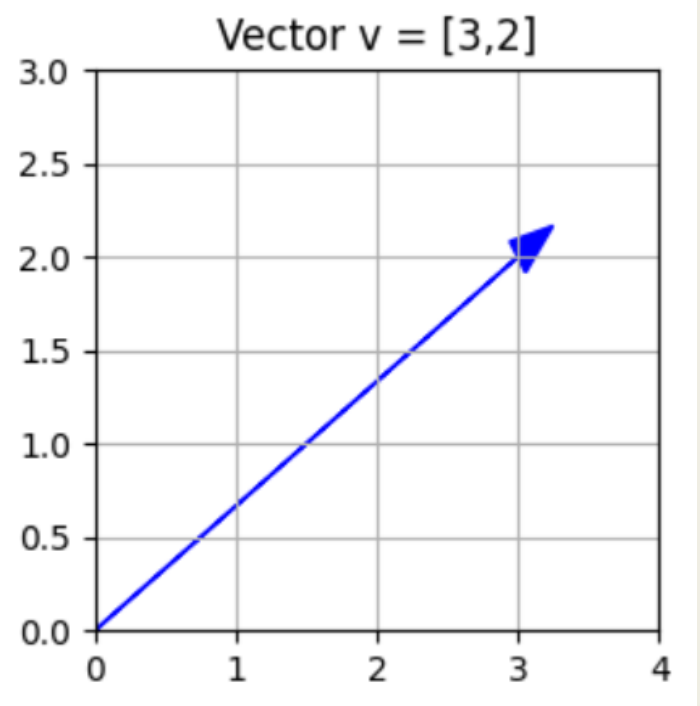
The **Arrow** shows the **Direction**.

Motion

- Vectors are the building blocks of **Motion**
- In geometry, a vector can describe a movement from one point to another.
- The vector [3, 2] says go 3 right and 2 up.

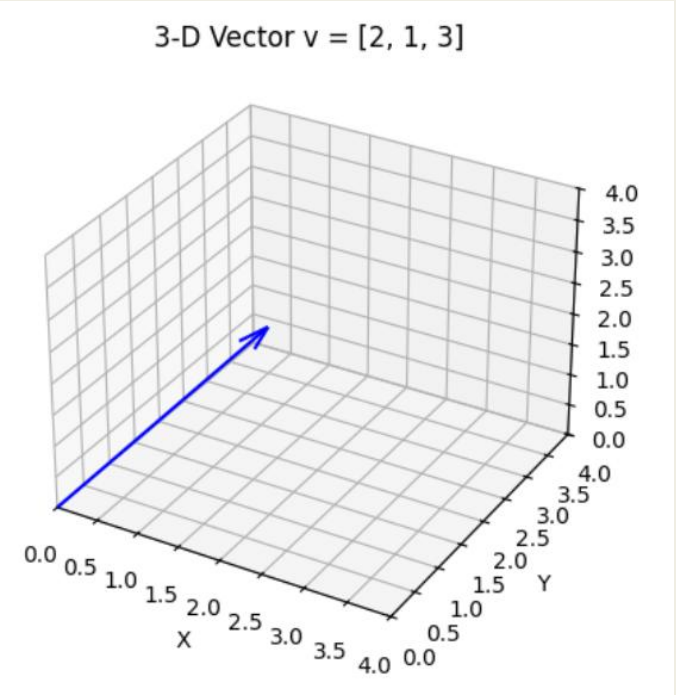
Vectors

```
import numpy as np
import matplotlib.pyplot as plt
v = np.array([3,2]);
plt.figure(figsize=(3,3))
plt.arrow(0,0,*v,head_width=.2,color='b');
plt.grid()
plt.xlim(0,4); plt.ylim(0,3);
plt.title('Vector v = [3,2]')
plt.show()
```



Task 1: Plotting a 3-Dimensional Vector

- Visualize a three-dimensional vector **v = [2, 1, 3]** using Python.
  - Use appropriate libraries (e.g., NumPy and Matplotlib).
  - Label the axes clearly (x, y, z).
  - Display the vector originating from the origin (0, 0, 0).



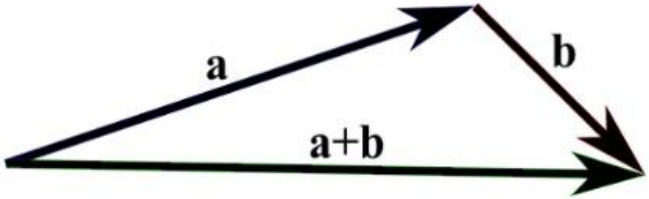
Solution

```
import numpy as np, matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

v = np.array([2, 1, 3])
ax = plt.figure().add_subplot(111, projection='3d')
ax.quiver(0, 0, 0, *v, color='b',
          arrow_length_ratio=0.1)
ax.set(xlabel='X', ylabel='Y', zlabel='Z',
       xlim=(0,4), ylim=(0,4), zlim=(0,4))
plt.title('3-D Vector v = [2, 1, 3]');
plt.grid(); plt.show()
```

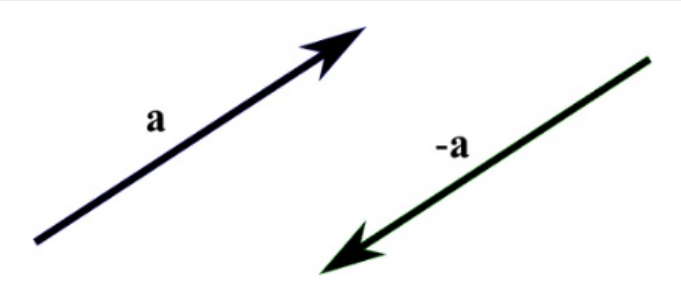
Vector Addition

- The sum of two vectors (**a+b**) is found by moving the vector **b** until the tail meets the head of vector **a**. (This does not change vector b).
- Then, the line from the tail of **a** to the head of **b** is the vector **a+b**:



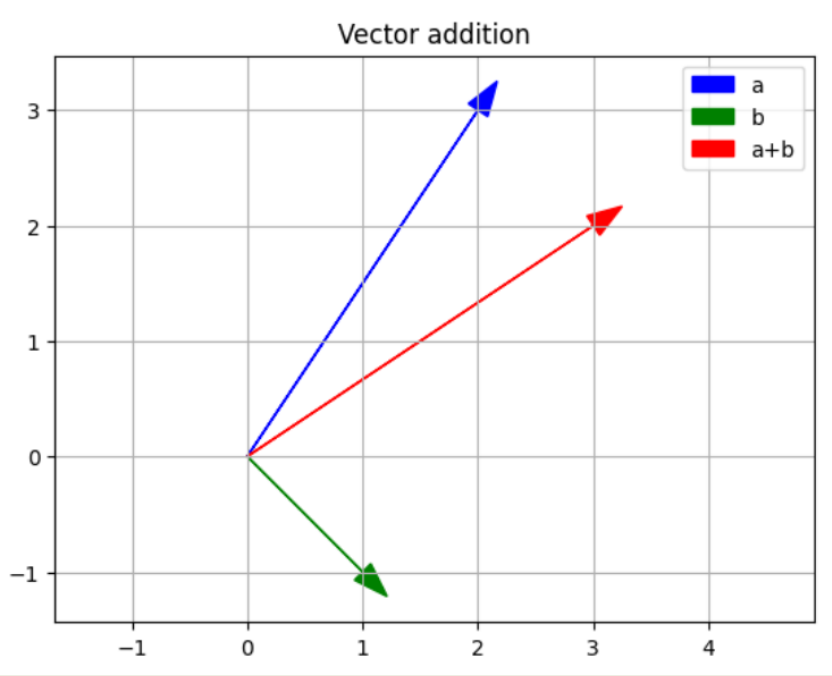
Vector Subtraction

- Vector **-a** is the opposite of **+a**.
- This means that vector a and vector -a has the same magnitude in opposite directions:



Vector Addition & Subtraction

```
a,b = np.array([2,3]), np.array([1,-1])
plt.arrow(0,0,*a,head_width=.2,color='b',label='a')
plt.arrow(0,0,*b,head_width=.2,color='g',label='b')
plt.arrow(0,0,*(a+b),head_width=.2,color='r',label='a+b')
plt.legend(); plt.grid(); plt.axis('equal')
plt.title('Vector addition'); plt.show()
```

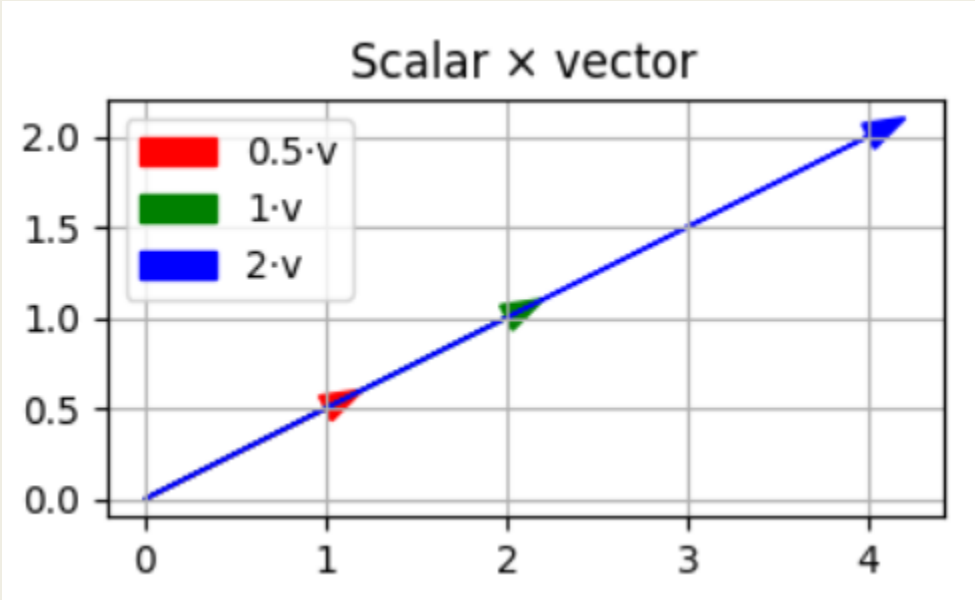


Scalar Operations

- Vectors can be modified by adding, subtracting, or multiplying a scalar (number) from all the vector values:
- a = [1 1 1]
- a + 1 = [2 2 2]
- [1 2 3] + 1 = [2 3 4]
- Vector multiplications has much of the same properties as normal multiplication:
- [2 2 2] * 3 = [6 6 6]
- [6 6 6] / 3 = [2 2 2]

Scalar Multiplication

```
v = np.array([2,1]); s = [0.5,1,2]
plt.figure(figsize=(4,2))
for k,c in zip(s,['r','g','b']):
    plt.arrow(0,0,*(k*v),head_width=.15,color=c,label=f'{k}·v')
plt.legend(); plt.grid(); plt.axis('equal')
plt.title('Scalar × vector'); plt.show()
```



Matrices

- A matrix is set of Numbers.
- A matrix is an Rectangular Array.
- A matrix is arranged in Rows and Columns.

Matrix Dimensions

- This **Matrix** has **1** row and **3** columns:

$$C = \begin{bmatrix} 2 & 5 & 3 \end{bmatrix}$$

- The **Dimension** of the matrix is (**1**x**3**).
- This matrix has **2** rows and **3** columns:

$$C = \begin{bmatrix} 2 & 5 & 3 \\ 4 & 7 & 1 \end{bmatrix}$$

- The dimension of the matrix is (**2**x**3**).

Square Matrices

- A **Square Matrix** is a matrix with the same number of rows and columns.
- An n-by-n matrix is known as a square matrix of order n.
- A **2-by-2** matrix (Square matrix of order 2):

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

- A **4-by-4** matrix (Square matrix of order 4):

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

Diagonal Matrices

- A Diagonal Matrix has values on the diagonal entries, and **zero** on the rest:

$$C = \begin{bmatrix} \mathbf{2} & 0 & 0 \\ 0 & \mathbf{5} & 0 \\ 0 & 0 & \mathbf{3} \end{bmatrix}$$

Scalar Matrices

- A Scalar Matrix has equal diagonal entries and **zero** on the rest:

$$C = \begin{bmatrix} \mathbf{3} & 0 & 0 & 0 \\ 0 & \mathbf{3} & 0 & 0 \\ 0 & 0 & \mathbf{3} & 0 \\ 0 & 0 & 0 & \mathbf{3} \end{bmatrix}$$

The Identity Matrix

- The **Identity Matrix** has **1** on the diagonal and **0** on the rest.
- This is the matrix equivalent of 1. The symbol is **I**.
- If you multiply any matrix with the identity matrix, the result equals the original.

$$I = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \end{bmatrix}$$

The Zero Matrix

- The **Zero Matrix** (Null Matrix) has only zeros.

$$C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Equal Matrices

- Matrices are **Equal** if each element correspond:

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

Negative Matrices

- The **Negative** of a matrix is easy to understand:

$$-\begin{bmatrix} -2 & 5 & 3 \\ -4 & 7 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -5 & -3 \\ 4 & -7 & -1 \end{bmatrix}$$

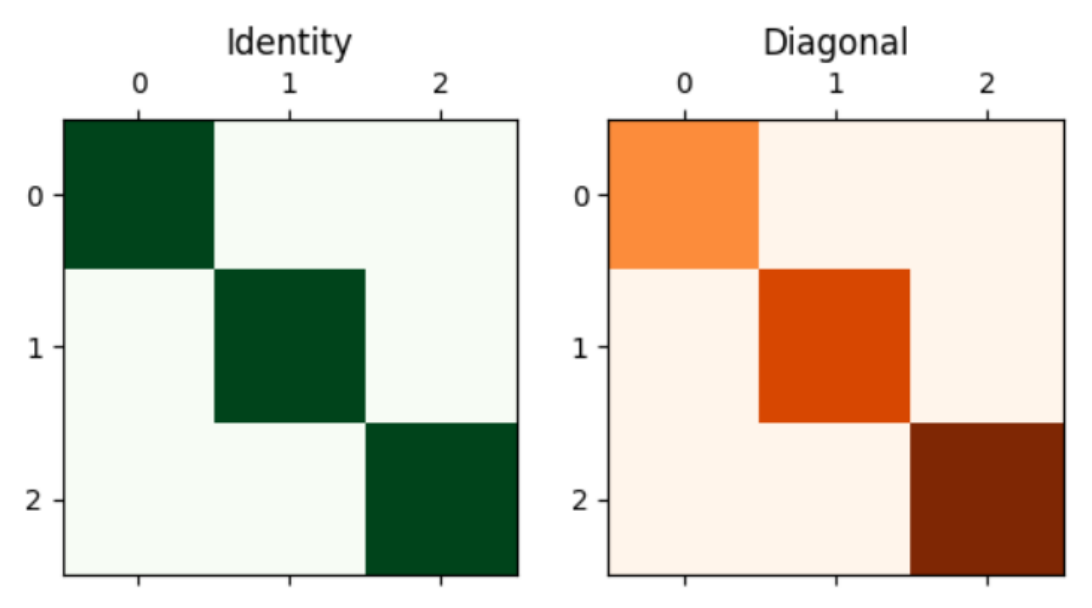
Identity and Diagonal Matrices

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

I = np.eye(3)
D = np.diag([2, 3, 4])
plt.subplot(1, 2, 1); plt.title('Identity');
plt.matshow(I, cmap='Greens', fignum=False)
plt.subplot(1, 2, 2); plt.title('Diagonal');
plt.matshow(D, cmap='Oranges', fignum=False)
plt.show()
```

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$



Task 2 - Create and Display

- Create a 4×4 identity matrix and a 4×4 diagonal matrix with elements [1, 2, 3, 4].
- Display both using plt.matshow() and use different color maps.

Adding Matrices

- If two matrices have the same dimension, we can add them:

$$\begin{bmatrix} 2 & 5 & 3 \\ 4 & 7 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 7 & 1 \\ 2 & 5 & 3 \end{bmatrix} = \begin{bmatrix} 6 & 12 & 4 \\ 6 & 12 & 4 \end{bmatrix}$$

Subtracting Matrices

- If two matrices have the same dimension, we can subtract them:

$$\begin{bmatrix} 2 & 5 & 3 \\ 4 & 7 & 1 \end{bmatrix} - \begin{bmatrix} 4 & 7 & 1 \\ 2 & 5 & 3 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 2 \\ 2 & 2 & -2 \end{bmatrix}$$

Scalar Multiplication

- While numbers in rows and columns are called **Matrices**, single numbers are called **Scalars**.
- It is easy to multiply a matrix with a scalar. Just multiply each number in the matrix with the scalar:

$$\begin{bmatrix} 2 & 5 & 3 \\ 4 & 7 & 1 \end{bmatrix} \times 2 = \begin{bmatrix} 4 & 10 & 6 \\ 8 & 14 & 2 \end{bmatrix}$$

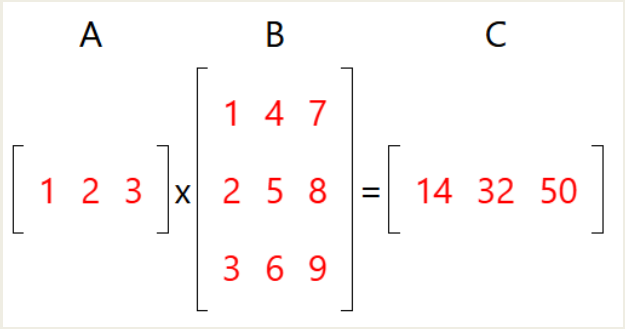
Transpose a Matrix

- To transpose a matrix, means to replace rows with columns.
- When you swap rows and columns, you rotate the matrix around it's diagonal.

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

Multiplying Matrices

- Multiplying matrices is more difficult.
- We can only multiply two matrices if the number of **columns** in matrix A is the same as the number of **rows** in matrix B.
- Then, we need to compile a "dot product":
- We need to multiply the numbers in each **column of A** with the numbers in each **row of B**, and then add the products:



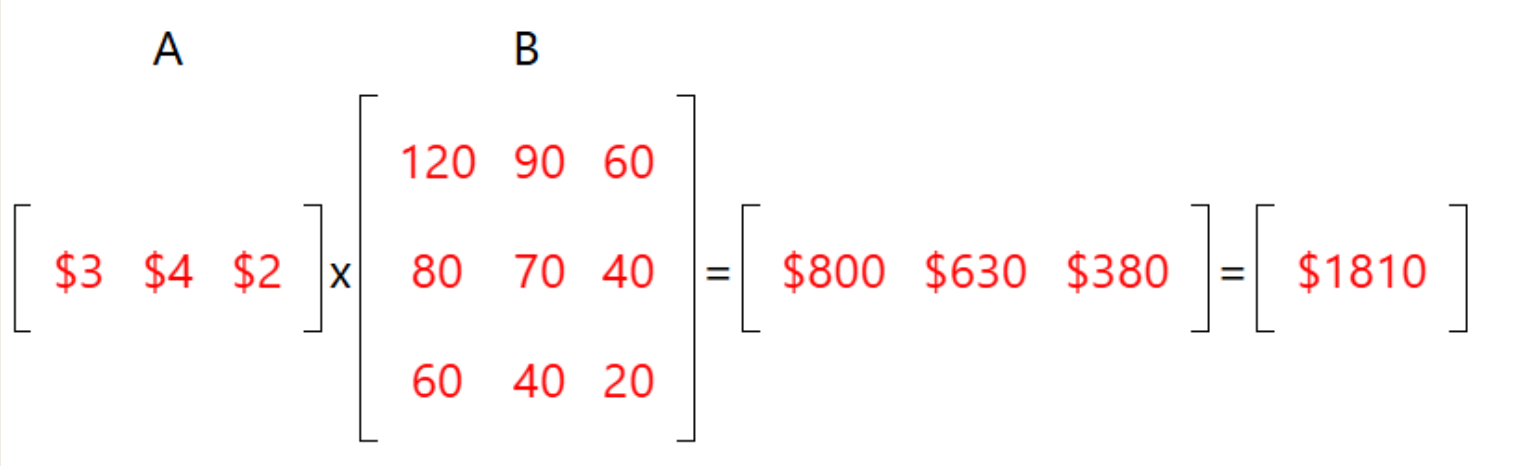
- (1,2,3) * (1,2,3) = 1x1 + 2x2 + 3x3 = 14
- (1,2,3) * (4,5,6) = 1x4 + 2x5 + 3x6 = 32
- (1,2,3) * (7,8,9) = 1x7 + 2x8 + 3x9 = 50

Example

- You sell roses.
- Red roses are $3 each
- White roses are $4 each
- Yellow roses are $2 each
- Monday you sold 260 roses
- Tuesday you sold 200 roses
- Wednesday you sold 120 roses
- What was the value of all the sales?

	 $3	 $4	 $2
Mon	120	80	60
Tue	90	70	40
Wed	60	40	20

Example



- (3,4,2) * (120,80,60) = 3x120 + 4x80 + 2x60 = 800
- (3,4,2) * (90,70,40) = 3x90 + 4x70 + 2x40 = 630
- (3,4,2) * (60,40,20) = 3x60 + 4x40 + 2x20 = 380

Tensors

- A Tensor is a **N-dimensional Matrix:**
- A Scalar is a 0-dimensional tensor
- A Vector is a 1-dimensional tensor
- A Matrix is a 2-dimensional tensor
- A **Tensor** is a generalization of **Vectors** and **Matrices** to higher dimensions.

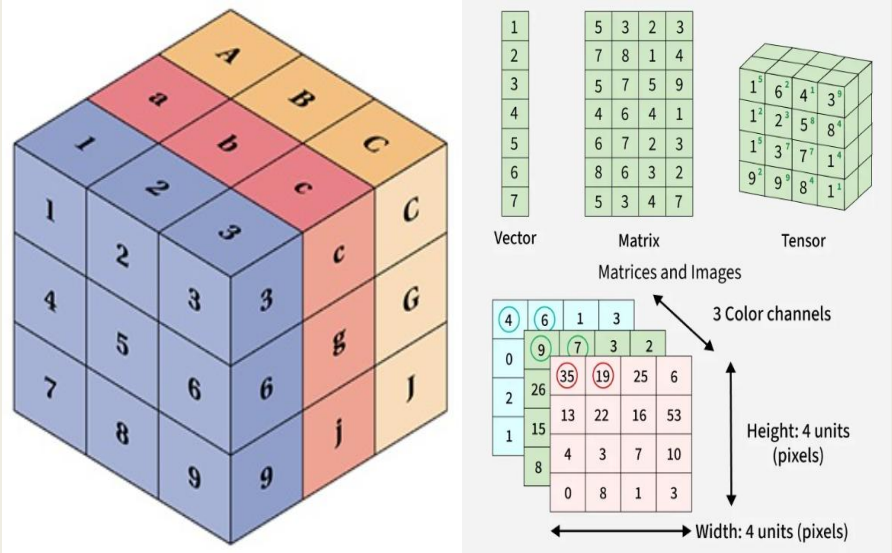
Tensor Ranks

- The number of directions a tensor can have in a **N**-dimensional space, is called the **Rank** of the tensor.
  - The rank is denoted **R**.
  - A **Scalar** is a single number.
  - It has 0 Axes
  - It has a **Rank of 0**
  - It is a 0-dimensional Tensor
  - A **Vector** is an array of numbers.
  - It has 1 Axis
  - It has a **Rank of 1**
  - It is a 1-dimensional Tensor
  - A **Matrix** is a 2-dimensional array.
  - It has 2 Axis
  - It has a **Rank of 2**
  - It is a 2-dimensional Tensor

Real Tensors

- **Properties:**
  - Entries are **real-valued (R)**, not complex.
  - Represent data in **multiple dimensions** (e.g., images ⟶ 3D tensors).
  - Used in **deep learning** for storing and transforming feature maps.

```
import numpy as np
T = np.random.rand(2,2,2)
print("Real Tensor:\n", T)
print("Shape:", T.shape)
```



Random images to understand Tensors

Thank You