1. Matrix, vector and scalar representation

1.1 Matrix

Example:

$$x = \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix}$$

x_{ij} is the element at the i^{th} row and j^{th} column. Here: $x_{11} = 4.1, x_{32} = -1.8$.

Dimension of matrix x is the number of rows times the number of columns.
Here $dim(x) = 3 \times 2$. x is said to be a 3×2 matrix.

The set of all 3×2 matrices is $\mathbb{R}^{3\times 2}$.

1.2 Vector

Example:

$$y = \begin{bmatrix} 4.1 \\ -3.9 \\ 6.4 \end{bmatrix}$$

$y_i = i^{th}$ element of y. Here: $y_1 = 4.1, y_3 = 6.4$.

Dimension of vector y is the number of rows.
Here $\mathrm{dim(y)} = 3 \times 1$ or $\mathrm{dim(y)} = 3$. y is said to be a 3-dim vector.

The set of all 3-dim vectors is $\mathbb{R}^3$.

1.3 Scalar

Example:

$$z = 5.6$$

A scalar has no dimension.

The set of all scalars is $\mathbb{R}$.

Note: $z = [5.6]$ is a 1-dim vector, not a scalar.

Question 1: Represent matrix, vector and scalar in Python

Hint: You may use numpy library, shape(), type(), dtype.

In [ ]:
```
import numpy as np
```

```
# ++++++++++++++++++++++++++++++++++++++++++++++++++
# YOUR CODE HERE
x           = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
size_x      = x.shape
type_x      = x.dtype

y           = np.array([1, 2, 3], dtype=np.float64)
size_y      = y.shape
type_y      = y.dtype

z           = np.array([1], dtype=np.float64)
size_z      = z.shape
type_z      = z.dtype
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('******************************')
print('x = ')
print(x)
print('******************************')
print('size of x = ')
print(size_x)
print('******************************')
print('type of x = ')
print(type_x)

print('******************************')
print('y = ')
print(y)
print('******************************')
print('size of y = ')
print(size_y)
print('******************************')
print('type of y = ')
print(type_y)

print('******************************')
print('z = ')
print(z)
print('******************************')
print('size of z = ')
print(size_z)
print('******************************')
print('type of z = ')
print(type_z)
print('******************************')
```

```
******************************
x =
[[1. 2. 3.]
 [4. 5. 6.]]
******************************
size of x =
(2, 3)
******************************
type of x =
float64
******************************
y =
[1. 2. 3.]
******************************
size of y =
(3,)
******************************
type of y =
float64
******************************
z =
[1.]
******************************
size of z =
(1,)
******************************
type of z =
float64
******************************
```

2. Matrix addition and scalar-matrix multiplication

2.1 Matrix addition

Example:

$$\begin{array}{ccccc} \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} & + & \begin{bmatrix} 2.7 & 7.3 \\ 3.5 & 2.4 \\ 6.0 & -1.1 \end{bmatrix} & = & \begin{bmatrix} 4.1+2.7 & 5.3+7.3 \\ -3.9+3.5 & 8.4+2.4 \\ 6.4+6.0 & -1.8-1.1 \end{bmatrix} \\ 3\times 2 & + & 3\times 2 & = & 3\times 2 \end{array}$$

All matrix and vector operations must satisfy dimensionality properties. For example, it is not allowed to add two matrices of different dimentionalities, such as

$$\begin{array}{ccccc} \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} & + & \begin{bmatrix} 2.7 & 7.3 & 5.0 \\ 3.5 & 2.4 & 2.8 \end{bmatrix} & = & \text{Not allowed} \\ 3\times 2 & + & 2\times 3 & = & \text{Not allowed} \end{array}$$

2.1 Scalar-matrix multiplication

Example:

$$\begin{array}{ccccccc} 3 & \times & \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} & = & \begin{bmatrix} 3\times 4.1 & 3\times 5.3 \\ 3\times -3.9 & 3\times 8.4 \\ 3\times 6.4 & 3\times -1.8 \end{bmatrix} \\ \text{No dim} & + & 3\times 2 & = & 3\times 2 \end{array}$$

Question 2: Add the two matrices, and perform the multiplication scalar-matrix in Python

In [ ]:

```
import numpy as np

# ++++++++++++++++++++++++++++++++++++++++++++++++++
# YOUR CODE HERE
x           = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
size_x      = x.shape

y           = np.array([[10, 20, 30], [40, 50, 60]], dtype=np.float64)
size_y      = y.shape

z           = np.array([2], dtype=np.float64)
size_z      = z.shape

sum_x_y     = x+y
mul_x_z     = x*z
div_x_z     = x/z

size_sum_x_y    = sum_x_y.shape
size_mul_x_z    = mul_x_z.shape
size_div_x_z    = div_x_z.shape
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('******************************')
print('x = ')
print(x)
print('******************************')
print('size of x = ')
print(size_x)

print('******************************')
print('y = ')
print(y)
print('******************************')
print('size of y = ')
print(size_y)

print('******************************')
print('z = ')
print(z)
print('******************************')
print('size of z = ')
print(size_z)

print('******************************')
print('x + y = ')
print(sum_x_y)
print('******************************')
print('size of x + y = ')
print(size_sum_x_y)

print('******************************')
print('x * z = ')
```

```
print(mul_x_z)
print('******************************')
print('size of x * z = ')
print(size_mul_x_z)

print('******************************')
print('x / z = ')
print(div_x_z)
print('******************************')
print('size of x / z = ')
print(size_div_x_z)
print('******************************')
```

```
******************************
x = 
[[1. 2. 3.]
 [4. 5. 6.]]
******************************
size of x = 
(2, 3)
******************************
y = 
[[10. 20. 30.]
 [40. 50. 60.]]
******************************
size of y = 
(2, 3)
******************************
z = 
[2.]
******************************
size of z = 
(1,)
******************************
x + y = 
[[11. 22. 33.]
 [44. 55. 66.]]
******************************
size of x + y = 
(2, 3)
******************************
x * z = 
[[ 2.  4.  6.]
 [ 8. 10. 12.]]
******************************
size of x * z = 
(2, 3)
******************************
x / z = 
[[0.5 1.  1.5]
 [2.  2.5 3. ]]
******************************
size of x / z = 
(2, 3)
******************************
```

3. Matric-vector multiplication

3.1 Example

Example:

$$\begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} \times \begin{bmatrix} 2.7 \\ 3.5 \end{bmatrix} = \begin{bmatrix} 4.1 \times 2.7 + 5.3 \times 3.5 \\ -3.9 \times 2.7 + 8.4 \times 3.5 \\ 6.4 \times 2.7 - 1.8 \times 3.5 \end{bmatrix}$$
$$3 \times 2 \qquad 2 \times 1 = 3 \times 1$$

Dimension of the matric-vector multiplication operation is given by contraction of 3×2 with 2×1 = 3×1.

3.2 Formalization

$$\begin{array}{ccccc} [A] & \times & [x] & = & [y] \\ m \times n & & n \times 1 & = & m \times 1 \end{array}$$

Element y_i is given by multiplying the i^{th} row of A with vector x:

$$\begin{array}{ccccc} y_i & = & A_i & & x \\ 1 \times 1 & = & 1 \times n & \times & n \times 1 \end{array}$$

It is not allowed to multiply a matrix A and a vector x with different n dimensions such as

$$\begin{array}{ccccc} \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} & \times & \begin{bmatrix} 2.7 \\ 3.5 \\ -7.2 \end{bmatrix} & = & ? \\ 3 \times 2 & \times & 3 \times 1 & = & \text{not allowed} \end{array}$$

Question 3: Multiply the matrix and vector in Python

In [ ]:
```
import numpy as np


# ++++++++++++++++++++++++++++++++++++++++++++++++++
# YOUR CODE HERE
A       = np.array([[1, 2], [3, 4], [5, 6]], dtype=np.float64)
size_A  = A.shape

x       = np.array([[10], [20]], dtype=np.float64)
size_x  = x.shape

y       = np.dot(A, x)
size_y  = y.shape
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('******************************')
print('A = ')
print(A)
print('******************************')
print('size of A = ')
print(size_A)

print('******************************')
print('x = ')
print(x)
print('******************************')
print('size of x = ')
print(size_x)
```

```
print('******************************')
print('y = A x')
print(y)
print('******************************')
print('size of y = ')
print(size_y)
print('******************************')
```

```
******************************
A =
[[1. 2.]
 [3. 4.]
 [5. 6.]]
******************************
size of A =
(3, 2)
******************************
x =
[[10.]
 [20.]]
******************************
size of x =
(2, 1)
******************************
y = A x
[[ 50.]
 [110.]
 [170.]]
******************************
size of y =
(3, 1)
******************************
```

4. Matrix-matrix multiplication

4.1 Example

$$\begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} \times \begin{bmatrix} 2.7 & 3.2 \\ 3.5 & -8.2 \end{bmatrix} = \begin{bmatrix} 4.1\times 2.7 + 5.3\times 3.5 & 4.1\times 3.2 + 5.3 \times \\ -3.9\times 2.7 + 8.4\times 3.5 & -3.9\times 3.2 + 8.4 \\ 6.4\times 2.7 - 1.8\times 3.5 & 6.4\times 3.2 - 1.8 \times \end{bmatrix}$$
$$3\times 2 \quad \times \quad 2\times 2 \quad = \quad 3\times 2$$

Dimension of the matrix-matrix multiplication operation is given by contraction of 3×2 with 2×2 = 3×2.

4.2 Formalization

$$\begin{matrix} [A] & \times & [X] & = & [Y] \\ m\times n & & n\times p & = & m\times p \end{matrix}$$

Like for matrix-vector multiplication, matrix-matrix multiplication can be carried out only if A and X have the same n dimension.

4.3 Linear algebra operations can be parallelized/distributed

Column Y_i is given by multiplying matrix A with the i^{th} column of X:

$$\begin{matrix} Y_i & = & A & \times & X_i \\ 1 \times 1 & = & 1 \times n & \times & n \times 1 \end{matrix}$$

Observe that all columns X_i are independent. Consequently, all columns Y_i are also independent. This allows to vectorize/parallelize linear algebra operations on (multi-core) CPUs, GPUs, clouds, and consequently to solve all linear problems (including linear regression) very efficiently, basically with one single line of code ($Y = AX$ for millions/billions of data). With Moore's law (computers speed increases by 100x every decade), it has introduced a computational revolution in data analysis.

Question 4: Multiply the two matrices in Python

In [ ]:
```
import numpy as np

# ++++++++++++++++++++++++++++++++++++++++++++++++++
# YOUR CODE HERE
A        = np.array([[1, 2], [3, 4], [5, 6]], dtype=np.float64)
size_A   = A.shape

X        = np.array([[10, 20], [30, 40]], dtype=np.float64)
size_X   = X.shape

Y        = np.matmul(A, X)
size_Y   = Y.shape
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('******************************')
print('A = ')
print(A)
print('******************************')
print('size of A = ')
print(size_A)

print('******************************')
print('X = ')
print(X)
print('******************************')
print('size of X = ')
print(size_X)

print('******************************')
print('Y = A X')
print(Y)
print('******************************')
print('size of Y = ')
print(size_Y)
```

```
******************************
A =
[[1. 2.]
 [3. 4.]
 [5. 6.]]
******************************
size of A =
(3, 2)
******************************
X =
[[10. 20.]
 [30. 40.]]
******************************
size of X =
(2, 2)
******************************
Y = A X
[[ 70. 100.]
 [150. 220.]
 [230. 340.]]
******************************
size of Y =
(3, 2)
```

5. Some linear algebra properties

5.1 Matrix multiplication is *not* commutative

$$\begin{array}{ccccccc} A & \times & B & \neq & B & \times & A \\ \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} & \times & \begin{bmatrix} 2.7 & 3.2 \\ 3.5 & -8.2 \end{bmatrix} & \neq & \begin{bmatrix} 2.7 & 3.2 \\ 3.5 & -8.2 \end{bmatrix} & \times & \begin{bmatrix} 4.1 & 5.3 \\ -3.9 & 8.4 \\ 6.4 & -1.8 \end{bmatrix} \end{array}$$

5.2 Scalar multiplication is associative

$$\begin{array}{ccccccc} \alpha & \times & B & = & B & \times & \alpha \\ 4.1 & \times & \begin{bmatrix} 2.7 & 3.2 \\ 3.5 & -8.2 \end{bmatrix} & = & \begin{bmatrix} 2.7 & 3.2 \\ 3.5 & -8.2 \end{bmatrix} & \times & 4.1 \end{array}$$

5.3 Transpose matrix

$$\begin{array}{ccc} X_{ij}^T & = & X_{ji} \\ \begin{bmatrix} 2.7 & 3.2 & 5.4 \\ 3.5 & -8.2 & -1.7 \end{bmatrix}^T & = & \begin{bmatrix} 2.7 & 3.5 \\ 3.2 & -8.2 \\ 5.4 & -1.7 \end{bmatrix} \end{array}$$

5.4 Identity matrix

$$I = I_n = Diag([1, 1, \dots, 1])$$

such that

$$I \times A = A \times I$$

Examples:

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.5 Matrix inverse

For any square $n \times n$ matrix A, the matrix inverse A^{-1} is defined as

$$AA^{-1} = A^{-1}A = I$$

Example:

$$\begin{array}{ccccc} \begin{bmatrix} 2.7 & 3.5 \\ 3.2 & -8.2 \end{bmatrix} & \times & \begin{bmatrix} 0.245 & 0.104 \\ 0.095 & -0.080 \end{bmatrix} & = & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ A & \times & A^{-1} & = & I \end{array}$$

Some matrices do not hold an inverse such as zero matrices. They are called degenerate or singular.

Question 5: Compute the matrix transpose in Python. Determine also the matrix inverse in Python.

In [ ]:
```
import numpy as np

# ++++++++++++++++++++++++++++++++++++++++++++++++++
# YOUR CODE HERE
A                   = np.array([[1, 2], [3, 4]], dtype=np.float64)
size_A              = A.shape

AT                  = A.transpose()
size_AT             = A.shape

mul_AT_A            = np.matmul(AT, A)
size_mul_AT_A       = mul_AT_A.shape

invA                = np.linalg.inv(A)
size_invA           = invA.shape

inv_mul_AT_A        = np.linalg.inv(mul_AT_A)
size_inv_mul_AT_A   = inv_mul_AT_A.shape
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('******************************')
print('A = ')
print(A)
print('******************************')
print('size of A = ')
print(size_A)

print('******************************')
print('AT = transpose of A ')
print(AT)
print('******************************')
print('size of AT = ')
```

```
print(size_AT)

print('*******************************')
print('AT A = multiplication of AT and A')
print(mul_AT_A)
print('*******************************')
print('size of multiplication of AT and A = ')
print(size_mul_AT_A)

print('*******************************')
print('inverse of A = ')
print(invA)
print('*******************************')
print('size of inverse of A = ')
print(size_invA)

print('*******************************')
print('inverse of multiplication of A transpose and A = ')
print(inv_mul_AT_A)
print('*******************************')
print('size of inverse of multiplication of A transpose and A = ')
print(size_inv_mul_AT_A)
print('*******************************')
```

```
*******************************
A =
[[1. 2.]
 [3. 4.]]
*******************************
size of A =
(2, 2)
*******************************
AT = transpose of A
[[1. 3.]
 [2. 4.]]
*******************************
size of AT =
(2, 2)
*******************************
AT A = multiplication of AT and A
[[10. 14.]
 [14. 20.]]
*******************************
size of multiplication of AT and A =
(2, 2)
*******************************
inverse of A =
[[-2.   1. ]
 [ 1.5 -0.5]]
*******************************
size of inverse of A =
(2, 2)
*******************************
inverse of multiplication of A transpose and A =
[[ 5.  -3.5]
 [-3.5  2.5]]
*******************************
size of inverse of multiplication of A transpose and A =
(2, 2)
*******************************
```

In [ ]: