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:

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

$x_i = i^{th}$ element of x. Here: $x_1 = 4.1, x_3 = 6.4$.

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

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

1.3 Scalar

Example:

$$x = 5.6$$

A scalar has no dimension.

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

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

Question 1: Represent the previous 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([
  [4.1, 5.3],
  [-3.9, 8.4],
  [6.4, -1.8]
])
print(x)
print(x.shape)    # size of x
print(type(x))    # type of x
print(x.dtype)    # data type of x

y = np.array([4.1, -3.9, 6.4])
print(y)
print(y.shape)    # size of y

z = 5.6
print(z)
print(np.array(z).shape)    # size of z
```

```
[[ 4.1  5.3]
 [-3.9  8.4]
 [ 6.4 -1.8]]
(3, 2)
<class 'numpy.ndarray'>
float64
[ 4.1 -3.9  6.4]
(3,)
5.6
()
```

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:

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

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

In [ ]:
```
import numpy as np

#YOUR CODE HERE

X1 =  np.array([
  [4.1, 5.3],
  [-3.9, 8.4],
  [6.4, -1.8]
])
X2 = np.array([
  [2.7, 3.5],
  [7.3, 2.4],
  [5., 2.8]
])
X = np.add(X1, X2)   # summation of X1 and X2

print(X1)
print(X2)
print(X)

Y1 = np.multiply(X, 4)   # X multiplied by 4
Y2 = np.divide(X, 3)   # X divided by 3

print(X)
print(Y1)
print(Y2)
```

```
[[ 4.1  5.3]
 [-3.9  8.4]
 [ 6.4 -1.8]]
[[2.7 3.5]
 [7.3 2.4]
 [5.  2.8]]
[[ 6.8  8.8]
 [ 3.4 10.8]
 [11.4  1. ]]
[[ 6.8  8.8]
 [ 3.4 10.8]
 [11.4  1. ]]
[[27.2 35.2]
 [13.6 43.2]
 [45.6  4. ]]
[[2.26666667 2.93333333]
 [1.13333333 3.6       ]
 [3.8        0.33333333]]
```

3. Matric-vector multiplication

3.1 Example

Example:

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

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

3.2 Formalization

$$\underset{m \times n}{[A]} \times \underset{n \times 1}{[x]} = \underset{m \times 1}{[y]}$$

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

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

Question 3: Multiply the matrix and vector above in Python

In [ ]:
```
import numpy as np

#YOUR CODE HERE

A =  np.array([
  [4.1, 5.3],
  [-3.9, 8.4],
  [6.4, -1.8]
])
x = np.array([
  [2.7],
  [3.5]
])
y = np.array([
  [29.62],
  [18.87],
  [10.98]
]) # multiplication of A and x
print(A)
print(A.shape)    # size of A
print(x)
print(x.shape)    # size of x
print(y)
print(y.shape)    # size of y
```

```
[[ 4.1  5.3]
 [-3.9  8.4]
 [ 6.4 -1.8]]
(3, 2)
[[2.7]
 [3.5]]
(2, 1)
[[29.62]
 [18.87]
 [10.98]]
(3, 1)
```

4. Matrix-matrix multiplication

4.1 Example

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

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

4.2 Formalization

$$\underset{m \times n}{[A]} \times \underset{n \times p}{[X]} = \underset{m \times p}{[Y]}$$

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:

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

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 above in Python

In [ ]:
```
import numpy as np

#YOUR CODE HERE

A =  np.array([
  [4.1, 5.3],
  [-3.9, 8.4],
  [6.4, -1.8]
])
X = np.array([
  [2.7, 3.2],
  [3.5, -8.2]
])
Y = np.dot(A, X)   # matrix multiplication of A and X
print(A)
print(A.shape)    # size of A
print(X)
print(X.shape)    # size of X
print(Y)
print(Y.shape)    # size of Y
```

```
[[ 4.1  5.3]
 [-3.9  8.4]
 [ 6.4 -1.8]]
(3, 2)
[[ 2.7  3.2]
 [ 3.5 -8.2]]
(2, 2)
[[ 29.62 -30.34]
 [ 18.87 -81.36]
 [ 10.98  35.24]]
(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:

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

$$A \times A^{-1} = I$$

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

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

In [ ]:

```
import numpy as np

#YOUR CODE HERE

A = np.array([
  [2.7, 3.5, 3.2],
  [-8.2, 5.4, -1.7]
])
AT = A.T   # transpose of A

print(AT)
print(A.shape)    # size of A
print(AT.shape)    # size of AT

A = np.array([
  [2.7, 3.5],
  [3.2, -8.2]
])
Ainv = np.linalg.inv(A)     # inverse of A
AAinv = np.dot(A, Ainv)   # multiplication of A and A inverse
print(A)
print(A.shape)    # size of A
print(Ainv)
print(Ainv.shape)    # size of Ainv
print(AAinv)
print(AAinv.shape)    # size of AAinv
```

```
[[ 2.7 -8.2]
 [ 3.5  5.4]
 [ 3.2 -1.7]]
(2, 3)
(3, 2)
[[ 2.7  3.5]
 [ 3.2 -8.2]]
(2, 2)
[[ 0.24595081  0.104979  ]
 [ 0.0959808  -0.0809838 ]]
(2, 2)
[[ 1.00000000e+00  9.02056208e-17]
 [-3.96603366e-18  1.00000000e+00]]
(2, 2)
```

In [ ]: