Project Work

Matrices and their Operations in Python

Contents

Abstract

Matrices are a part of Linear Algebra which is used everywhere in Computer Science. It is used in computer graphics to create 2D/3D models, animations, etc. It is used in cryptography (making data secure) by using matrices to store data and a key matrix to encrypt it and its inverse to decrypt it. In this project we will look at operations on matrices in Python.

1 What is a Matrix?

A matrix is a rectangular array of data[1] arranged in rows and columns.
A matrix looks like the following:

$$A = \begin{bmatrix} 1 & 10 & 12 \\ 2 & 20 & 22 \end{bmatrix}_{2\times 3}$$

A matrix is represented by capital letters and the subscript represents Number of rows $\times$ Number of columns and the matrix A is called *a 2 by 3 matrix*.
In Python, to enter a matrix, we use nested lists like:

```
Matrix=[
  [1,10,12],
  [2,10,22]
]
```

1.1 Accessing a Matrix

A matrix a is generally written as $A = [a_{ij}]_{m\times n}$ where $1 \leq i \leq m \wedge 1 \leq j \leq n$. Thus, if we know the location of an element say, *element in row 2 and column 1*, we can write it as a_{21}. In Python as well, if we need to find the *element in row i and column j*, we can return it as:

```python
def find_element(matrix, row, column):
  row_index=row-1 #we need to use row-1 as indexes begin from 0
  column_index=column-1
  return matrix[row_index][column_index]
```

Example. We can also access the elements of a matrix, either row or column wise.

Solution. To print the matrix row-wise:

```python
def row_wise(matrix):
  for i in range(len(matrix)):
    for j in range(len(matrix[0])):
      print(matrix[i][j],end="\t")
  print("\n")
```

If matrix is $A = \begin{bmatrix} 1 & 10 & 12 \\ 2 & 20 & 22 \end{bmatrix}$, then, the output is,

```
1       10      12

2       20      22
```

To print the matrix column-wise:

```python
def column_wise(matrix):
  for i in range(len(matrix[0])):
    for j in range(len(matrix)):
      print(matrix[j][i],end="\t")
  print("\n")
```

The output in this case is:

```
1       2

10      20

12      22
```

1.2 Null Matrix

A null matrix is one such that,

$$a_{ij} = 0 \quad \forall\ i, j$$

Such a matrix is represented by O.

[1]Data may be of any form, like numbers, expressions or alphabets.

Example. Creating a null matrix of a given order.

Solution.

```
def null(rows,columns):
  null_matrix=[[0 for i in range(columns)] for i in range(rows)] #loop over columns then over rows
  return null_matrix
```

1.3 Upper and Lower Triangular Matrices

The upper triangular matrix is a matrix in which all the entries below the diagonal are zero, i.e.,

$$a_{ij} = 0 \quad \forall\, i \geq j$$

Or,

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

The lower triangular matrix is a matrix in which all the entries above the diagonal are zero[2], i.e.,

$$a_{ij} = 0 \quad \forall\, i \leq j$$

Or,

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Example. Creating an upper and lower triangular matrix.

Solution. First for an upper triangular matrix,

```
def upper_triangular(matrix):
    rows=len(matrix)
    columns=len(matrix[0])
    upper_matrix=null(rows,columns) #null matrix
    for i in range(rows):
        for j in range(columns):
            if i>=j: #Condition for a null matrix
                upper_matrix[i][j]+=matrix[i][j]
            else:
                continue
    return upper_matrix
```

If the matrix is $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, then, the output is,

```
[[1, 0, 0], [4, 5, 0]]
```

Now, for a lower triangular matrix,

```
def lower_triangular(matrix):
  rows=len(matrix)
  columns=len(matrix[0])
  lower_matrix=null(rows,columns) #null matrix
  for i in range(rows):
    for j in range(columns):
      if j>=i: #Condition for a null
        lower_matrix[i][j]+=matrix[i][j]
      else:
        continue
  return lower_matrix
```

[2]These definitions are open for discussion. Some authors claim that the matrix must be square, while some do not restrict the matrix. Even though a 'triangle' would not be formed in the case of rectangular matrices, it is acceptable. We have chosen not to restrict the matrices to only square matrices.

The output in this case is,

```
[[1, 2, 3], [0, 5, 6]]
```

1.4 Transpose of a Matrix

The transpose of a matrix $A = [a_{ij}]_{m\times n}$ is given by,

$$A^T = [a_{ji}]_{n\times m}$$

Example. Create another matrix which is the transpose of a given matrix.

Solution.

```python
def transpose(matrix):
  rows=len(matrix)
  columns=len(matrix[0])
  transpose_matrix=null(columns, rows) #null matrix
  for i in range(rows):
    for j in range(columns):
      transpose_matrix[j][i]+=matrix[i][j] #definition of transpose

  return transpose_matrix
```

If the matrix is $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$

, the output is,

```
[[1, 4, 1], [2, 5, 1], [3, 6, 1]]
```

1.5 Addition of Two Matrices

Given two matrices $A = [a_{ij}]_{n_1\times m_1}$ and $B = [b_{ij}]_{n_2\times m_2}$, the sum $A + B$ is defined only if $(n_1, m_1) = (n_2, m_2) = (n, m)$ (Say), i.e. the matrices have the same order.

$$C = A + B = [a_{ij} + b_{ij}]_{n\times m}$$

Example. Find the sum of two matrices.

Solution. It returns the sum of the matrices if it exists, else, returns -1.

```python
def Matrix_Sum(matrix1,matrix2):
    if len(matrix1)==len(matrix2) and len(matrix1[0])==len(matrix2[0]): #Checking for compatibility for addition
        rows, columns=len(matrix1), len(matrix1[0])
        sum_matrix=null(rows,columns) #null matrix
        for i in range(rows):
            for j in range(columns):
                sum_matrix[i][j]+=matrix1[i][j]+matrix2[i][j] #definition of sum of matrices
        return sum_matrix
    else:
        return -1
```

If the input matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 2 & 2 & 2 \end{bmatrix}$, the output is,

```
[[2, 4, 6], [8, 10, 12], [3, 3, 3]]
```

If the matrices are $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, the output is,

-1