2. MATRIX RELATED

a) Matrix basics

November 27, 2021

1 AIM

Matrices are a vital tool in linear algebra. Here, we will look at their basic properties and functionalities.

2 Operations on matrices

A matrix is a 2D array where elements are divided into rows and columns.

2.1 Defining matrices using lists (not a good idea)

Defining matrix using list...

[1]:
```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(A)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Accessing particular rows, columns and elements...

[7]:
```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print("First row: {0}".format(A[0]))
print("First element: {0}".format(A[0][0]))
```

```
First row: [1, 2, 3]
First element: 1
```

Columns cannot be selected easily using this definition of matrices.

Matrices cannot be operated on using this definition of matrices (list of lists)...

[12]:
```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = [[-1, -2, -3], [-4, -5, -6], [-7, -8, -9]]
print(A + B)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [-1, -2, -3], [-4, -5, -6], [-7, -8, -9]]
```

You cannot use -, * and / for lists.

The above issues that arise from defining a matrix as a list compel us to use numpy to define matrices instead.

2.2 Defining matrices using the numpy library (numerical python)

[3]:
```
import numpy as np
```

2.2.1 As an array

[16]:
```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[21]:
```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[-1, -2, -3], [-4, -5, -6], [-7, -8, -9]])
print(A + B)
print("---")
print(A - B)
print("---")
print(A * B)
print("---")
print(A / B)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
---
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
---
[[ -1  -4  -9]
 [-16 -25 -36]
 [-49 -64 -81]]
---
[[-1. -1. -1.]
 [-1. -1. -1.]
 [-1. -1. -1.]]
```

2.2.2 As a matrix

[18]:
```
B = np.matrix([[-1, -2, -3], [-4, -5, -6], [-7, -8, -9]])
print(A)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[22]:
```
A = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.matrix([[-1, -2, -3], [-4, -5, -6], [-7, -8, -9]])
print(A + B)
print("---")
print(A - B)
print("---")
print(A * B)
print("---")
print(A / B)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
---
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
---
[[ -30  -36  -42]
 [ -66  -81  -96]
 [-102 -126 -150]]
---
[[-1. -1. -1.]
 [-1. -1. -1.]
 [-1. -1. -1.]]
```

Array allows any number of dimensions, while matrix is strictly 2D. Also, the methods available for the latter also differ, and are more specialised for matrix operations.

2.2.3 Accessing rows, columns and elements

[56]:
```
A = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("The matrix:\n{0}".format(A))
print("3rd row:\n{0}".format(A[2]))
print("1st column:\n{0}".format(A[:,0]))
print("Element at 2nd row, 3rd column:\n{0}".format(A[1, 2]))
```

```
The matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
3rd row:
[[7 8 9]]
1st column:
[[1]
 [4]
 [7]]
Element at 2nd row, 3rd column:
```

6

2.2.4 Methods

[4]:
```
A = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A.size) # Gives size of matrix
print(A.shape) # Gives order of matrix i.e. (no. of rows, no. of columns)
# NOTE: These work for arrays also
```

```
9
(3, 3)
```

[6]:
```
N = np.zeros((3,2)) # Makes a matrix of the given order consisting only of zeros
print(N)
M = np.ones((3,6)) # Makes a matrix of the given order consisting only of ones
print(M)
# NOTE: These can also create arrays, ex. np.zeros(4) i.e. an array of 4 zeros
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
[[1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]]
```

[43]:
```
A = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("The matrix: {0}".format(A))
print("The determinant: {0}".format(np.linalg.det(A)))
```

```
The matrix: [[1 2 3]
 [4 5 6]
 [7 8 9]]
The determinant: -9.51619735392994e-16
```

Note that to find the determinant, the matrix must be a square matrix.

[53]:
```
A = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("The matrix:\n{0}".format(A))
print("The inverse:\n{0}".format(np.linalg.inv(A)))
print("The inverse:\n{0}".format(A.getH() / np.linalg.det(A))) # A.getH() gets␣
↪the adjoint of A
```

```
The matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
The inverse:
[[ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]
 [-6.30503948e+15  1.26100790e+16 -6.30503948e+15]
```

```
 [ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]]
The inverse:
[[-1.05083991e+15 -4.20335965e+15 -7.35587939e+15]
 [-2.10167983e+15 -5.25419957e+15 -8.40671930e+15]
 [-3.15251974e+15 -6.30503948e+15 -9.45755922e+15]]
```

2.2.5 Matrix operations

[29]:
```
A = np.matrix([[3, 4, 5], [5, 7, 4], [6, 3, 2]])
B = np.matrix([[-2, 4, 2], [-6, 3, -4], [-3, 5, 2]])
```

[30]:
```
print(A + B)
print(np.add(A, B))
print(A - B)
print(np.add(A, -B))
```

```
[[ 1  8  7]
 [-1 10  0]
 [ 3  8  4]]
[[ 1  8  7]
 [-1 10  0]
 [ 3  8  4]]
[[ 5  0  3]
 [11  4  8]
 [ 9 -2  0]]
[[ 5  0  3]
 [11  4  8]
 [ 9 -2  0]]
```

[31]:
```
print("To do matrix multiplication")
print(A * B) # Apparently only works for square matrices
print(np.dot(A, B))
print("To multiply element-wise")
print(np.multiply(A, B))
```

```
To do matrix multiplication
[[-45  49   0]
 [-64  61 -10]
 [-36  43   4]]
[[-45  49   0]
 [-64  61 -10]
 [-36  43   4]]
To multiply element-wise
[[ -6  16  10]
 [-30  21 -16]
 [-18  15   4]]
```

[34]:
```
print(A / B)
print(np.divide(A, B))
print(np.multiply(A, 1/B))
```

```
[[-1.5         1.          2.5       ]
 [-0.83333333  2.33333333 -1.        ]
 [-2.          0.6         1.        ]]
[[-1.5         1.          2.5       ]
 [-0.83333333  2.33333333 -1.        ]
 [-2.          0.6         1.        ]]
[[-1.5         1.          2.5       ]
 [-0.83333333  2.33333333 -1.        ]
 [-2.          0.6         1.        ]]
```

Note that for matrix multiplication, the matrices must be such that the no. of columns of one matrix must equal the no. of rows in the second matrix.

[25]:
```
print(A**2)
# Repeated multiplication of matrix. Only works with square matrix because of␣
 ↪the condition for matrix multiplication.
```

```
[[59 55 41]
 [74 81 61]
 [45 51 46]]
```

[46]:
```
print("Original")
print(A)
print("Transposed")
print(np.transpose(A))
print(A.transpose())
print(A.T)
```

```
Original
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Transposed
[[1 4 7]
 [2 5 8]
 [3 6 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

3 CONCLUSION

Matrices are far more powerful than lists, when it comes to numerical collections. The wide availablility of functions makes it easy to use for linear algebra.