2. MATRIX RELATED

b) More on matrices

November 27, 2021

1 AIM

We will be looking at more properties and features of Python matrices that will be useful for linear algebra.

2 Symmetric or skew symmetric

[123]:

```
from numpy import matrix
def checkSymmetry(M, name):
    print("\n{0} =\n{1}".format(name, M))
    # Transpose = Original...?
    print("Is symmetric? {0}".format((False not in (M == M.T))))
    # Negative of transpose = Original...?
    print("is skew symmetric? {0}".format(False not in (M == -M.T)))

# Matrices to be tested
A = matrix([[1, 5, 4], [4, 6, 6], [-5, 1, 5]])
B = matrix([[-1, 3, 9], [-8, 2, 1], [-5, 1, -7]])

# Function call
checkSymmetry(A, "A")
checkSymmetry(B, "B")
```

```
A =
[[ 1  5  4]
 [ 4  6  6]
 [-5  1  5]]
Is symmetric? False
is skew symmetric? False

B =
[[-1  3  9]
 [-8  2  1]
 [-5  1 -7]]
Is symmetric? False
is skew symmetric? False
```

3 Row & column operations and reshaping

[4]:
```python
from numpy import matrix, zeros, reshape, sqrt, random
# Random matrix
M = zeros([6, 5])
for i in range(0, 6):
    for j in range(0, 5):
        M[i, j] = random.randint(1, 25)
print(M)

# Square root of every element of 5th row
tmp = sqrt(M[4])
print("\nSquare root of 5th row's elements...")
for i, e in enumerate(tmp): print(i, ":", e)

# Adding corresponding elements of the 2nd and 4th row
tmp = M[1] + M[3]
print("\nSum of corresponding elements of 2nd and 4th rows...")
for i, e in enumerate(tmp): print(i, ":", e)

# Extracting the second column of the matrix.
# Reshaping this column into a 2 x 3 matrix.
tmp = M[:, 1]
# NOTE:
# All rows, column 2 => 2nd column
# You can given a specified number of rows or columns as well.
# Example: M[2:3, 1:4] => rows 3 to 4, columns 2 to 5.
# This enables you to extract submatrices of various dimensions from M.
print("\nColumn extracted...")
for i in tmp: print(i)
print("Reshaped into a 2 x 3 matrix...")
print(matrix(reshape(tmp, (2, 3))))
# NOTES
# The reshape function returns an array of the given order.
# I have converted this array to a matrix using the matrix class constructor.
# Reshaping of the 1D array above can be also done as tmp.reshape(2,3).
```

```
[[18. 21. 22.  2.  1.]
 [ 7. 24. 14. 21. 15.]
 [14.  6.  9.  9. 10.]
 [20.  3. 10.  6.  7.]
 [ 9.  1. 11.  6. 21.]
 [23.  4.  5. 12. 10.]]

Square root of 5th row's elements…
0 : 3.0
1 : 1.0
```

```
2 : 3.3166247903554
3 : 2.449489742783178
4 : 4.58257569495584

Sum of corresponding elements of 2nd and 4th rows…
0 : 27.0
1 : 27.0
2 : 24.0
3 : 27.0
4 : 22.0

Column extracted…
21.0
24.0
6.0
3.0
1.0
4.0
Reshaped into a 2 x 3 matrix…
[[21. 24.  6.]
 [ 3.  1.  4.]]
```

For more on reshaping a matrix… https://numpy.org/doc/stable/reference/generated/numpy.reshape.html

4 Aggregation & sorting operations for matrices

[3]:
```python
from numpy import matrix, zeros, sum, product, trace, min, max, sort
# NOTE:
# For a multi-dimensional array or matrix...
# - sum from numpy can add all the elements
# - product from numpy can multiply all the elements
# - min and max from numpy can find the minimum and maximum values

# Random matrix
M = zeros([5, 5])
for i in range(0, 5):
    for j in range(0, 5):
        M[i, j] = random.randint(1, 5)
print("The matrix...\n{0}\n".format(M))
print("Sum =", sum(M))
print("Product =",product(M))
print("Trace =", trace(M))
print("Minimum =", min(M))
print("Maximum =", max(M))
print("\nThe matrix with every row sorted...\n{0}".format(sort(M)))
# NOTE
```

```
# Even though we pass the matrix, the sort function only sorts each row, since
↪it only sorts 1D arrays.
```

```
The matrix…
[[3. 2. 1. 2. 2.]
 [4. 3. 1. 4. 3.]
 [1. 4. 2. 1. 1.]
 [2. 2. 2. 4. 4.]
 [1. 3. 3. 4. 3.]]

Sum = 62.0
Product = 382205952.0
Trace = 15.0
Minimum = 1.0
Maximum = 4.0

The matrix with every row sorted…
[[1. 2. 2. 2. 3.]
 [1. 3. 3. 4. 4.]
 [1. 1. 1. 2. 4.]
 [2. 2. 2. 4. 4.]
 [1. 3. 3. 3. 4.]]
```