Python Matrices and NumPy Arrays

TUYEN NGOC LE

Outline

Python Matrix

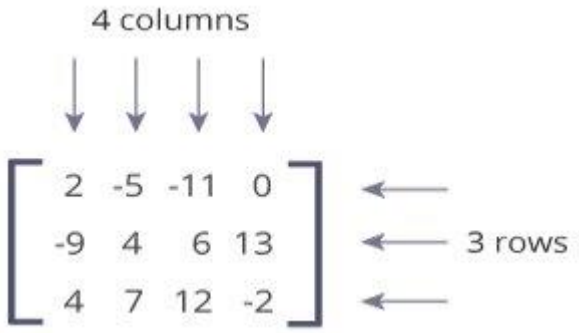
NumPy Array

Matrix as Numpy Array

Python Matrix

- A matrix is a two-dimensional data structure where numbers are arranged into rows and columns.
- Python doesn't have a built-in type for matrices. However, we can treat a list of a list as a matrix.

```
A = [[2, -5, -11, 0],
[-9, 4, 6, 13],
[4, 7, 12, -2]]
print(A)
```



This matrix is a 3x4 (pronounced "three by four") matrix

Access each row, column, element

```
A = [[1, 4, 5, 12],
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]]

print("A =", A)
print("A[1] =", A[1])      # 2nd row
print("A[1][2] =", A[1][2])   # 3rd element of 2nd row
print("A[0][-1] =", A[0][-1])   # Last element of 1st Row

column = [];        # empty list
for row in A:
  column.append(row[2])

print("3rd column =", column)

for row in A:
    print(type(row))
    for i in range(len(row)):
        print(row[i])
```

Access each row, column, element

```
A = [[1, 4, 5, 12],
     [-5, 8, 9, 0],
     [-6, 7, 11, 19]]

# iterate through rows
for i in range(len(A)):
    # iterate through columns
    for j in range(len(A[0])):
         print(A[i][j])
```

Add Two Matrices

```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

```
# Program to add two matrices using list comprehension

X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[X[i][j] + Y[i][j]  for j in range(len(X[0]))] for i in range(len(X))]

for r in result:
   print(r)
```

Transpose a Matrix

```
# Program to transpose a matrix using a nested loop

X = [[12,7],
    [4 ,5],
    [3 ,8]]

result = [[0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[j][i] = X[i][j]

for r in result:
   print(r)
```

```
X = [[12,7],
    [4 ,5],
    [3 ,8]]

result = [[X[j][i] for j in range(len(X))] for i in range(len(X[0]))]

for r in result:
   print(r)
```

Matrix Multiplication

```
# 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]

# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
       # iterate through rows of Y
       for k in range(len(Y)):
           result[i][j] += X[i][k] * Y[k][j]

for r in result:
   print(r)
```

```
# Program to multiply two matrices using list comprehension

# 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]

# result is 3x4
result = [[sum(a*b for a,b in zip(X_row,Y_col)) for Y_col in zip(*Y)] for X_row in X]

for r in result:
   print(r)
```

For larger matrix operations we recommend optimized software packages like NumPy which is several (in the order of 1000) times faster than the above code.

built-in function zip()

Iterate over several iterables in parallel, producing tuples with an item from each one.

```
for item in zip([1, 2, 3], ['sugar', 'spice', 'everything nice']):
    print(item)
```

```
(1, 'sugar')
(2, 'spice')
(3, 'everything nice')
```

NumPy Array

- NumPy is a package for scientific computing which has support for a powerful N-dimensional array object.
- NumPy provides multidimensional array of numbers (which is actually an object)

```
A = [[1 2 3]
     [3 4 5]]
B = [[1.1 2.  3. ]
     [3.  4.  5. ]]
C = [[1.+0.j 2.+0.j 3.+0.j]
     [3.+0.j 4.+0.j 5.+0.j]]
```

NumPy Array Creation

```
numpy.array(object, dtype = None, *, copy = True, order = 'K', subok = False, ndmin = 0, like = None)
```

object	**(*array_like*) An array, any object exposing the array interface, an object whose __array__ method returns an array, or any (nested) sequence. If object is a scalar, a 0-dimensional array containing object is returned.**
dtype	(***data-type, optional***) The desired data-type for the array. If not given, then the type will be determined as the minimum type required to hold the objects in the sequence
copy	(***bool, optional***) If true (default), then the object is copied. Otherwise, a copy will only be made if __array__ returns a copy, if obj is a nested sequence, or if a copy is needed to satisfy any of the other requirements (dtype, order, etc.).
order	(***{'K', 'A', 'C', 'F'}, optional***) Specify the memory layout of the array. If object is not an array, the newly created array will be in C order (row major) unless 'F' is specified, in which case it will be in Fortran order (column major). If object is an array the following holds.
subok	(***bool, optional***) If True, then sub-classes will be passed-through, otherwise the returned array will be forced to be a base-class array (default).
ndmin	(***int, optional***) Specifies the minimum number of dimensions that the resulting array should have. Ones will be prepended to the shape as needed to meet this requirement.
like	(***array_like, optional***) Reference object to allow the creation of arrays which are not NumPy arrays. If an array-like passed in as like supports the __array_function__ protocol, the result will be defined by it. In this case, it ensures the creation of an array object compatible with that passed in via this argument.

NumPy Array Creation

```
numpy.array(object, dtype = None, *, copy = True, order = 'K', subok = False, ndmin = 0, like = None)
```

```
import numpy as np
A = np.array([[1, 2, 3], [3, 4, 5]])
print(A)

B = np.array([[1.1, 2, 3], [3, 4, 5]]) # Array of floats
print(B)

C = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex) # Array of complex numbers
print(C)
```

NumPy Array Creation

Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation.

The function **zeros** creates an array full of zeros, the function **ones** creates an array full of ones, and the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype of the created array is float64.

```
import numpy as np
a = np.zeros((3,4))
print(a)
b = np.ones((3,4))
print(b)
```

NumPy Array Creation

To create sequences of numbers, NumPy provides a function analogous to **range** that returns arrays instead of lists.

```
import numpy as np
a = np.arange( 10, 30, 5 )
print(a)
b = np.arange( 0, 2, 0.3 ) # # it accepts float arguments
print(b)
```

NumPy Array Creation

When arange is used with floating point arguments, it is generally not possible to predict the number of elements obtained, due to the finite floating point precision. For this reason, it is usually better to use the function linspace that receives as an argument the number of elements that we want, instead of the step:

```
import numpy as np
from numpy import pi
a = np.linspace( 0, 2, 9 )      # 9 numbers from 0 to 2
x = np.linspace( 0, 2*pi, 100 ) # useful to evaluate function at lots of points (100)
f = np.sin(x)
print(a)
print(x)
print(f)
```

NumPy Array Creation

When arange is used with floating point arguments, it is generally not possible to predict the number of elements obtained, due to the finite floating point precision. For this reason, it is usually better to use the function linspace that receives as an argument the number of elements that we want, instead of the step:

```
import numpy as np
from numpy import pi
a = np.linspace( 0, 2, 9 )      # 9 numbers from 0 to 2
x = np.linspace( 0, 2*pi, 100 ) # useful to evaluate function at lots of points (100)
f = np.sin(x)
print(a)
print(x)
print(f)
```

Attributes of an array object

ndim	the number of axes (dimensions) of the array.
shape	the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim
size	the total number of elements of the array. This is equal to the product of the elements of shape.
dtype	an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples
itemsize	the size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize
data	the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities

Attributes of an array object

```
import numpy as np
A = np.array([[1, 4, 5, 12],
   [-5, 8, 9, 0],
   [-6, 7, 11, 19]])
print(A.ndim)
print(A.shape) # A.shape[0], A.shape[0]
print(A.dtype)
print(A.itemsize)
print(A.data)
```

Basic Operations of an array object

```
import numpy as np
A = np.array([[1, 4, 5, 12],
   [-5, 8, 9, 0],
   [-6, 7, 11, 19]])
print(A.ndim)
print(A.shape) # A.shape[0], A.shape[0]
print(A.dtype)
print(A.itemsize)
print(A.data)
```

Matrix as Numpy Array

Access matrix elements, rows and columns

```
import numpy as np
A = np.array([[1, 4, 5, 12],
   [-5, 8, 9, 0],
   [-6, 7, 11, 19]])
# Access elements of a two-dimensional array
# First element of first row
print("A[0][0] =", A[0][0])
# Third element of second row
print("A[1][2] =", A[1][2])
# Last element of last row
print("A[-1][-1] =", A[-1][-1])
# Access rows of a Matrix
print("A[0] =", A[0]) # First Row
print("A[2] =", A[2]) # Third Row
print("A[-1] =", A[-1]) # Last Row (3rd row in this case)
# Access columns of a Matrix
print("A[:,0] =",A[:,0]) # First Column
print("A[:,3] =", A[:,3]) # Fourth Column
print("A[:,-1] =", A[:,-1]) # Last Column (4th column in this case)
```

Access matrix elements, rows and columns

```
import numpy as np
A = np.array([[1, 4, 5, 12],
   [-5, 8, 9, 0],
   [-6, 7, 11, 19]])
# Slicing of a Matrix
print(A[:2, :4])  # two rows, four columns
print(A[:1,])  # first row, all columns
print(A[:,2])  # all rows, second column
print(A[:, 2:5])  # all rows, third to the fifth column
# Access elements
for i in range(A.shape[0]):
   for j in range(A.shape[1]):
      print(A[i,j])
```

Matrix Operations

```
import numpy as np
# Addition of Two Matrices
A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B      # element wise addition
print(C)
# Multiplication of Two Matrices
D = A.dot(B)
print(D)
# Transpose of a Matrix
E = A.transpose()
print(E)
```

Exercise

Create a Python project to perform some simple statistics on a list (or numpy.array) of values, such as:

- Computes the average.
- Computes the mean deviation. Mean Deviation is average of distance of each value from that mean(average).
- Counts the occurrence of a value in a list of values.
- Computes the variance. Variance is useful to see how the list of values varied against the average.
- Computes the standard deviation. Standard Deviation is useful to give an idea about range of normal values(i.e. location of most of values).
- Computes the median. Median is the middle value in a sorted list of values.
- Returns the max value.
- Returns the min value.
- Returns the difference between max and min values.
- Returns summation of all values in a list.
- Returns the length of list.
- Sorts the list.

References

- https://realpython.com/what-can-i-do-with-python/
- https://www.w3schools.com/python/python_getstarted.asp
- https://www.programiz.com/python-programming/first-program
- https://www.geeksforgeeks.org/
- https://www.w3resource.com/python-exercises/list/

Thank you for listening!