Python library - NumPy (*)

Numpy is a library for scientific computing. It provides a multidimensional array object (**ndarray** also known by the alias **array**) and support for fast operations on arrays. Following are the main differences between NumPy arrays and Python data types

- Fixed size arrays
- The data type of array elements must be the same (can be objects)
- More efficient and easier to implement operations
- Vectorised approach is more compact and easier to understand

(*) https://numpy.org/doc/stable/user/index.html

Python library - NumPy

ndarray is a multidimensional array class, known also by the name array, with elements of the same type and indexed by a tuple of non-negative integers. Following are listed some of the attributes of **ndarray**.

ndarray.ndim	Number of dimensions (axes) of the array
ndarray.shape	Size of the array in each dimension (tuple of integers)
ndarray.size	Total number of the elements of the array
ndarray.dtype	Object type of the elements of the array (numpy.int32, numpy.float64)
ndarray.itemsize	Number of bytes of each element of the array (4 bytes for numpy.int32)
ndarray.data	Container of the elements of the array (no need to use for indexed access)

Python library - NumPy

NumPy supports a great variety of numerical types. The primitive types are closely related to those in C.

numpy.int8	One byte integer (int16, int32, int64)
numpy.uint8	One byte unsigned integer (uint16, uint32, uint64)
numpy.float32	Four bytes float (float64 – builtin Python float)
numpy.complex64	Complex number (4 bytes real and imaginary componentes)
numpy.complex128	Complex number (8 bytes real/imaginary – builtin Python complex)

The data type can be defined with **dtype**: a = np.array([[1,2],[3,4]], dtype=np.float64)

Or **dtype** can be used to get the data type: a.dtype
dtype('float64')

astype can be used to convert the data type: a.astype(int)

float64 is the default data type

Python library - NumPy

Creating arrays

There are several ways for creating numpy arrays:

- Conversion from Python data types (lists, tuples, …)
  - **numpy.array**([[1,-2],[5,3]])

- Using numpy methods
  - **numpy.arange**(initial, final, step) works as Python range
  - **numpy.ones**((dim1,dim2)) creates a dim1xdim2 size array with ones
  - **numpy.zeros**((dim1,dim2)) creates a dim1xdim2 size array with zeros
  - **numpy.linspace**(initial, final, n) creates **n** of elements between initial and final values
  - **numpy.random.rand**(n) creates **n** random elements (uniform in [0, 1[)
  - **numpy.eye**(n) creates an identity matrix with **n** ones in the diagonal

- Importing data
  - **numpy**.genfromtxt('person.csv', skip_header=1, delimiter=";")

Python library - NumPy

Creating arrays - examples

```
In [1]: #%% NumPy  - Python scientific library
        import numpy as np

In [2]: # arange() works similar to range()
        np.arange(1, 9, 2)

Out[2]: array([1, 3, 5, 7])

In [3]: # reshape defines the array dimensions
        np.arange(12).reshape(3,4)

Out[3]: array([[ 0,  1,  2,  3],
               [ 4,  5,  6,  7],
               [ 8,  9, 10, 11]])

In [4]: # creates an array from a list
        np.array([[1,-2],[5,3]])

Out[4]: array([[ 1, -2],
               [ 5,  3]])
```

```
In [5]: # creates an array with zeros
        np.zeros((2,3))

Out[5]: array([[0., 0., 0.],
               [0., 0., 0.]])

In [6]: # creates an array of type int with ones
        np.ones((2,3), dtype=int)

Out[6]: array([[1, 1, 1],
               [1, 1, 1]])

In [7]: # creates an array from a csv file
        np.genfromtxt('data.csv', delimiter=";")

Out[7]: array([[ 1.,  2.,  3.,  4.],
               [ 5.,  6.,  7.,  8.],
               [ 9., 10., 11., 12.],
               [13., 14., 15., 16.]])
```

Python library - NumPy

Array operations examples - All arithmetic operates elementwise

```
In [8]: # Product of two arrays in Python
        a = [1,2,3]
        b = [6,5,4]
        c = list()
        for i in range(len(a)):
            c.append(a[i] * b[i])
        print(c)

        [6, 10, 12]
```

```
In [9]: # Product of two arrays unsing NumPy
        # arrays must be declared
        a = np.array([1,2,3])
        b = np.array([6,5,4])
        c = a * b
        c

Out[9]: array([ 6, 10, 12])
```

Basic operations

```
In [10]: a - b

Out[10]: array([-5, -3, -1])

In [11]: b**2

Out[11]: array([36, 25, 16], dtype=int32)

In [12]: 5 * np.sqrt(a)

Out[12]: array([5.        , 7.07106781, 8.66025404])

In [13]: a > 2

Out[13]: array([False, False,  True])
```

Python library - NumPy

Universal functions (ufuncs) (*)

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating-point, or complex values
sqrt	Compute the square root of each element (equivalent to arr ** 0.5)
square	Compute the square of each element (equivalent to arr ** 2)
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)
ceil	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
floor	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
rint	Round elements to the nearest integer, preserving the dtype
modf	Return fractional and integral parts of array as a separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non-inf, non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of not x element-wise (equivalent to ~arr).

(*) Python for Data Analysis, Wes McKinney, O'Reilly

Python library - NumPy

Binary universal functions (ufuncs) (*)

Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum; fmax ignores NaN
minimum, fmin	Element-wise minimum; fmin ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argument
greater, greater_equal, less, less_equal, equal, not_equal	Perform element-wise comparison, yielding boolean array (equivalent to infix operators >, >=, <, <=, ==, !=)
logical_and	Compute element-wise truth value of AND (&) logical operation.
logical_or	Compute element-wise truth value of OR (|) logical operation.
logical_xor	Compute element-wise truth value of XOR (^) logical operation.

(*) Python for Data Analysis, Wes McKinney, O'Reilly

Python library - NumPy

Basic array statistical methods (*)

Method	Description
sum	Sum of all the elements in the array or along an axis; zero-length arrays have sum 0
mean	Arithmetic mean; invalid (returns NaN) on zero-length arrays
std, var	Standard deviation and variance, respectively
min, max	Minimum and maximum
argmin, argmax	Indices of minimum and maximum elements, respectively
cumsum	Cumulative sum of elements starting from 0
cumprod	Cumulative product of elements starting from 1

(*) Python for Data Analysis, Wes McKinney, O'Reilly

Python library - NumPy

Random functions (*)

Function	Description
seed	Seed the random number generator
permutation	Return a random permutation of a sequence, or return a permuted range
shuffle	Randomly permute a sequence in-place
rand	Draw samples from a uniform distribution
randint	Draw random integers from a given low-to-high range
randn	Draw samples from a normal distribution with mean 0 and standard deviation 1 (MATLAB-like interface)
binomial	Draw samples from a binomial distribution
normal	Draw samples from a normal (Gaussian) distribution
uniform	Draw samples from a uniform [0, 1) distribution

(*) Python for Data Analysis, Wes McKinney, O'Reilly

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Python library - NumPy

linalg functions (*)

Function	Description
diag	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
dot	Matrix multiplication
trace	Compute the sum of the diagonal elements
det	Compute the matrix determinant
eig	Compute the eigenvalues and eigenvectors of a square matrix
inv	Compute the inverse of a square matrix
solve	Solve the linear system Ax = b for x, where A is a square matrix
lstsq	Compute the least-squares solution to Ax = b

np.transpose(ma)	Returns the transpose of the matrix
ma.flatten()	Transforms the matrix in a one-dimensional numpy array

(*) Python for Data Analysis, Wes McKinney, O'Reilly

Python library - NumPy

Matrix operations - examples

Matrix/vector operations

```
In [15]: ma = np.array([[1,-2],[5,3]])
         ma

Out[15]: array([[ 1, -2],
                [ 5,  3]])

In [16]: mb = np.array([[-1,2],[4,1]])
         mb

Out[16]: array([[-1,  2],
                [ 4,  1]])

In [17]: # Matrix product
         ma @ mb
         np.dot(ma,mb)

Out[17]: array([[-9,  0],
                [ 7, 13]])

In [18]: # Matrix transpose
         np.transpose(ma)

Out[18]: array([[ 1,  5],
                [-2,  3]])

In [19]: # Matrix determinant
         np.linalg.det(ma)

Out[19]: 13.0

In [20]: # Matrix inverse
         np.linalg.inv(ma)

Out[20]: array([[ 0.23076923,  0.15384615],
                [-0.38461538,  0.07692308]])

In [21]: # Matrix flatten
         ma.flatten()

Out[21]: array([ 1, -2,  5,  3])
```

Python library - NumPy

Python and NumPy execution time comparison

In [22]:
```
#%% Python and numpy time comparison
import numpy as np
import time
a = range(1, 10000001)
b = range(1,10000001)
c = list()
t1 = time.perf_counter()
for i in range(len(a)):
    c.append(a[i] / b[i])
t2 = time.perf_counter()
print(f'Python time: {t2 - t1:0.4}')
```

```
Python time: 4.774
```

In [23]:
```
a = np.arange(1,1000001)
b = np.arange(1,1000001)
t1 = time.perf_counter()
c = a / b
t2 = time.perf_counter()
print(f'NumPy time: {t2 - t1:0.4}')
```

```
NumPy time: 0.1221
```