

NumPy - Introduction

Micha Tracewicz

2020-03-07

1 What is NumPy?

NumPy is general purpose Python package used for array-processing. It provides a high-performance multidimensional array object, and tools for working with these arrays.

2 Basics

2.1 Introduction

In NumPy the main way of representing data are multi dimensional arrays. They are represented by ndarray class. It represents table of elements (all of which are of the same type). Those arrays are indexed by tuples of positive integers. Number of dimensions for given array is called rank. Tuple of integers representing size of array along each dimension is called shape.

2.2 Creating arrays

Example code presenting how to create ndarrays:

```
import numpy as np

# Creating a rank 1 Array
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)

# Creating a rank 2 Array
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Array with Rank 2: \n", arr)

# Creating an array from tuple
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)

# Creating an array filled with 0
arr = np.zeros(10)
print("\nArray created using "
      "zeros:\n", arr)

# Creating an array filled with 1
arr = np.ones(10)
print("\nArray created using "
      "ones:\n", arr)

# Creating an array filled with from 2.0 to 10.0 containing 5 elements
arr = np.linspace(2, 10, 5)
```

```

print("\nArray created using "
      "linspace:\n", arr)

# Creating an array filled random numbers (max numebr is 9, because it
# will generate numebtrs lower then given n))
np.random.seed(0)
arr = np.random.randint(10, size = 6)
print("\nArray created using "
      "random:\n", arr)

```

2.3 Accessing elements of arrays

Elements of those arrays are accesed using square brackes. When using ndarrays one can also use slices.

```

import numpy as np

# Initial Array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])
print("Initial Array: ")
print(arr)

# Printing a range of Array
# with the use of slicing method
sliced_arr = arr[:2, ::2]
print ("Array with first 2 rows and"
      " alternate columns(0 and 2):\n", sliced_arr)

# Printing elements at
# specific Indices
Index_arr = arr[[1, 1, 0, 3],
                 [3, 2, 1, 0]]
print ("\nElements at indices (1, 3), "
      "(1, 2), (0, 1), (3, 0):\n", Index_arr)

```

3 Operations

NumPy provides a set of usefull buildin functions.

```

import numpy as np

# Defining Array 1
a = np.array([[1, 2],

```

```

        [3, 4]])

# Defining Array 2
b = np.array([[4, 3],
              [2, 1]])

# Adding 1 to every element
print ("Adding 1 to every element:", a + 1)

# Subtracting 2 from each element
print ("\nSubtracting 2 from each element:", b - 2)

# sum of array elements
# Performing Unary operations
print ("\nSum of all array "
        "elements: ", a.sum())

# Adding two arrays
# Performing Binary operations
print ("\nArray sum:\n", a + b)

# Multiplying two arrays
print ("\nArray sum:\n", a * b)

# Getting dot product
print ("\nArray sum:\n", a @ b)

# Transposition
print ("\nMatrix transposition:\n", a,a.T)

```

4 Data types

Every ndarray has associated data type, known as dtype. It provides information about arrays layout. This example should illustrate how NumPy tries to guess data and how user can force it to use particular one.

```

import numpy as np

# Integer datatype
# guessed by Numpy
x = np.array([1, 2])
print("Integer Datatype: ",x.dtype)

# Float datatype
# guessed by Numpy
x = np.array([1.0, 2.0])
print("\nFloat Datatype: ",x.dtype)

```

```
# Forced Datatype
x = np.array([1, 2], dtype = np.int64)
print("\nForcing a Datatype: ",x.dtype)
```

5 List of NumPy methods:

- all()
- any()
- take()
- put()
- apply_along_axis()
- apply_over_axes()
- argmin()
- argmax()
- nanargmin()
- nanargmax()
- amax()
- amin()
- insert()
- delete()
- append()
- around()
- flip()
- fliplr()
- flipud()
- triu()
- tril()
- tri()
- empty()

- `empty_like()`
- `zeros()`
- `zeros_like()`
- `ones()`
- `ones_like()`
- `full_like()`
- `diag()`
- `diagflat()`
- `diag_indices()`
- `asmatrix()`
- `bmat()`
- `eye()`
- `roll()`
- `identity()`
- `arange()`
- `place()`
- `extract()`
- `compress()`
- `rot90()`
- `tile()`
- `reshape()`
- `ravel()`
- `isinf()`
- `isrealobj()`
- `isscalar()`
- `isneginf()`
- `isposinf()`
- `iscomplex()`

- `isnan()`
- `iscomplexobj()`
- `isreal()`
- `isfinite()`
- `isfortran()`
- `exp()`
- `exp2()`
- `fix()`
- `hypot()`
- `absolute()`
- `ceil()`
- `floor()`
- `degrees()`
- `radians()`
- `npv()`
- `fv()`
- `pv()`
- `power()`
- `float_power()`
- `log()`
- `log1()`
- `log2()`
- `log10()`
- `dot()`
- `vdot()`
- `trunc()`
- `divide()`
- `floor_divide()`

- `true_divide()`
- `random.rand()`
- `random.randn()`
- `ndarray.flat()`
- `expm1()`
- `bincount()`
- `rint()`
- `equal()`
- `not_equal()`
- `less()`
- `less_equal()`
- `greater()`
- `greater_equal()`
- `prod()`
- `square()`
- `cbrt()`
- `logical_or()`
- `logical_and()`
- `logical_not()`
- `logical_xor()`
- `array_equal()`
- `array_equiv()`
- `sin()`
- `cos()`
- `tan()`
- `sinh()`
- `cosh()`
- `tanh()`

- $\arcsin()$
- $\arccos()$
- $\arctan()$
- $\arctan2()$