Recap and language details for Python exercises

Lab 6 - NumPy

NumPy

Credits and infos

Slides adapted from a notebook by Stefano Zingaro (tutor two years ago), who adapted from Volodymyr Kuleshov and Isaac Caswell from the `CS231n` Numpy and Python tutorial by Justin Johnson.

Some slides are also adapted from Prof. Martini ones. **Please refere to that slides for a more comprehensive view on the material**

- Virtuale uses NumPy 1.23.1 (the last stable)
- examples will always assume `import numpy as np`

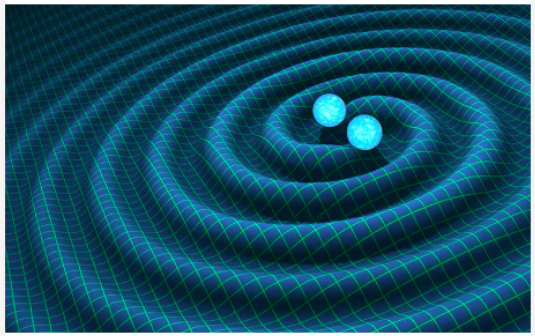
NumPy

- Numpy is the core library for scientific computing in Python.
- It provides a high-performance multidimensional array object, and tools for working with these arrays.
- If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy.
- It is at the basis of top notch scientific computations

FIRST IMAGE OF A BLACK HOLE

How NumPy, together with libraries like SciPy and Matplotlib that depend on NumPy, enabled the Event Horizon Telescope to produce the first ever image of a black hole

DETECTION OF GRAVITATIONAL WAVES



In 1916, Albert Einstein predicted gravitational waves; 100 years later their existence was confirmed by LIGO scientists using NumPy.

SPORTS ANALYTICS

Cricket Analytics is changing the game by improving player and team performance through statistical modelling and predictive analytics. NumPy enables many of these analyses.

Arrays

- A numpy array is a grid of values, all of the same type, and is **indexed by a tuple of nonnegative integers**.
- The number of dimensions is the *rank* of the array;
- The **shape of an array is a tuple of integers** giving the size of the array along each dimension.
- We can initialize numpy arrays from nested Python lists, and access elements using square brackets.

Arrays

```
1  import numpy as np
2
3  a = np.array([1, 2, 3])   # Create a rank 1 array
4  print(type(a))  # Prints "<class 'numpy.ndarray'>"
5  print(a.shape)            # Prints "(3,)"
6  print(a[0], a[1], a[2])   # Prints "1 2 3"
7  a[0] = 5                  # Change an element of the array
8  print(a)                  # Prints "[5, 2, 3]"
9
10 b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 arr.
11 print(b.shape)                     # Prints "(2, 3)"
12 print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Creating arrays

```
1  a = np.zeros((2,2))   # Create an array of all zeros
2  print(a)              # Prints "[[ 0.  0.]
3                        #          [ 0.  0.]]"
4  print(type(a[0, 0]))  #<class 'numpy.float64'>
5
6  b = np.ones((1,2))    # Create an array of all ones
7  print(b)              # Prints "[[ 1.  1.]]"
8
9  c = np.full((2,2), 7)  # Create a constant array
10 print(c)                # Prints "[[ 7.  7.]
11                        #          [ 7.  7.]]"
12
13 d = np.eye(2)          # Create a 2x2 identity matrix
14 print(d)               # Prints "[[ 1.  0.]
15                        #          [ 0.  1.]]"
16
17 e = np.random.random((2,2))  # Array w. random val.
18 print(e)                     # Who knows??
```

Creating arrays

- With a range (ints or floats) `np.arange(first,last,step)`
- Random range: `np.random.randint(first,last,number)`
- With a function

```
1 def g(i):
2     return i % 10
3
4 a = np.fromfunction(g,(20,))
5 #what is a?
```

NB: you will be in trouble if you need to use a boolean or an **if** inside `g`...

Datatypes

- Every numpy array is a grid of elements of the same type.
- Numpy provides a large set of numeric datatypes that you can use to construct arrays.
- Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

```
>>> np.zeros((2,2))
array([[0., 0.],
       [0., 0.]])
>>> np.zeros((2,2), dtype=int)
array([[0, 0],
       [0, 0]])
```

You can read all about numpy datatypes in the documentation.

Reshaping arrays

- `A.reshape(shape)`, shape is a tuple, must match the size of A, returns a view on A

```
>>> np.random.randint(1,10,100).reshape((10,10))
array([[5, 2, 8, 6, 4, 7, 8, 8, 6, 3],
       [9, 8, 8, 8, 9, 3, 2, 5, 6, 8],
       [3, 7, 6, 9, 9, 3, 7, 4, 7, 1],
       [2, 2, 1, 6, 8, 7, 3, 3, 8, 8],
       [3, 8, 6, 8, 7, 1, 6, 2, 4, 4],
       [5, 9, 9, 9, 7, 6, 7, 1, 1, 1],
       [8, 2, 8, 7, 4, 5, 5, 8, 6, 7],
       [7, 4, 2, 7, 4, 3, 2, 2, 7, 5],
       [9, 7, 8, 8, 2, 8, 7, 4, 8, 1],
       [7, 7, 5, 7, 7, 3, 4, 2, 8, 5]])
```

Array indexing

Numpy offers several ways to index into arrays.

Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array

```
1  a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
2
3  b = a[:2, 1:3]
4  # [[2 3]
5  #  [6 7]]
6
7  # A slice of an array is a view into the same data, so
       modifying it will modify the original array.
8  print(a[0, 1])   # Prints "2"
9  b[0, 0] = 77     # b[0, 0] is a[0, 1]
10 print(a[0, 1])   # Prints "77"
```

Mixing indexing

You can also mix integer indexing with slice indexing.

However, doing so will yield an array of lower rank than the original array.

```
1  a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
2
3  # Mixing integer indexing with slices yields an array
       of lower rank, while using only slices yields an
       array of the same rank as the original array:
4  row_r1 = a[1, :] #Rank 1 view of the second row of a
5  row_r2 = a[1:2, :] #Rank 2 view of the second row of a
6  print(row_r1, row_r1.shape)  # "[5 6 7 8] (4,)"
7  print(row_r2, row_r2.shape)  # "[[5 6 7 8]] (1, 4)"
8
9  # Same for columns
10 col_r1 = a[:, 1]
11 col_r2 = a[:, 1:2]
12 print(col_r1, col_r1.shape) #[ 2  6 10] (3,)
13 print(col_r2, col_r2.shape) #[[ 2],[ 6],[10]] (3, 1)
```

Integer array indexing

- Slicing returns views, array indexing creates copies (unless is used on the left of a =)

```
1  a = nparray([18,10,17,15,10,16,18,17,17,11])
2  b = np.array([0, 2, 0, 1])
3
4  #Select element from each row of a using b indexes
5  c = a[b]
6  #"c now is array([18, 17, 18, 10])"
7  # c is not linked with a (check c.base)
8
9  # Mutate one element from each row of a using the
       indices in b
10 a[b] -= 30
11 # a now is modified as
12 # array([-12, -20, -13,  15,  10,  16,  18,  17,  17,
       11])
```

Example: selecting and permuting columns

```
1  import numpy as np
2
3  a = np.arange(10,20).reshape((2,5))
4  """
5  [[10 11 12 13 14]
6   [15 16 17 18 19]]
7  """
8
9  b =  np.array([3,2,0,4,4])
10
11 print(a[:, b])
12 """
13 [[13 12 10 14 14]
14  [18 17 15 19 19]]
15 """
```

Boolean array indexing

Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

```
1  a = np.array([[1,2], [3, 4], [5, 6]])
2  """Find the elements of a that are bigger than 2;
3  this returns a numpy array of Booleans of the same
4  shape as a, where each slot of bool_idx tells
5  whether that element of a is > 2."""
6  bool_idx = (a > 2)    #  [[False False]
7                        #    [ True  True]
8                        #    [ True  True]]
9  '''We use boolean array indexing to construct a rank 1
       array consisting of the elements of a
     corresponding to the True values of bool_idx'''
10 print(a[bool_idx])  # [3 4 5 6]
11 # We can do all in a single concise statement:
12 print(a[a > 2])      # Prints "[3 4 5 6]"
```

Example

Write a NumPy program to compute an element-wise indication of the sign for all elements in a given array.

```
1 import numpy as np
2 x = np.array([1, 3, 5, 0, -1, -7, 0, 5])
3 x[x>0] = 1
4 x[x<0] = -1
5 #x is modified to
6 #array([ 1,  1,  1,  0, -1, -1,  0,  1])
```

For brevity we have left out a lot of details about numpy array indexing; if you want to know more you should read the documentation.

Newaxis

`np.newaxis` create an axis of length one

```
>>> x=np.array([1,2,3,4]) shape: (4,)
>>> x[np.newaxis,:]
array([[1, 2, 3, 4]]) shape: (1,4)
>>> x[np.newaxis,:].shape
(1, 4)
>>> x[:,np.newaxis] shape: (4,1)
array([[1],
[2],
[3],
[4]])
```

Broadcasting

Powerful mechanism that allows to work with arrays of different shapes when performing arithmetic operations.

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

1. if one arguments has less dimensions add as many 1s as possible to the shape
2. if one argument has size 1 on dimension k stretch that dimension to match the dimension of the other array on that dimension

Universal functions

- Functions that support broadcasting are known as universal functions.
- Universal functions operate in a uniform way on ndarrays, usually element-wise
- You can find the list of all universal functions in the documentation.

Axes

- Axes: directions along dimensions
- Some methods acts on arrays like lists, for example:
  a.sum()
  a.prod()
  a.max()
  a.min()

```
1 x = np.array([[1,2],[3,4]])
2
3 print(np.sum(x))  # Compute sum of all elements;
      prints "10"
4 print(np.sum(x, axis=0))  # Compute sum of each column
      ; prints "[4 6]"
5 print(np.sum(x, axis=1))  # Compute sum of each row;
      prints "[3 7]"
```