Introduction to NumPy

Informatics 1 for Biomedical Engineers
Tutor Session 6

KTI, Knowledge Technologies Institute

21. November 2016

Today's Topics

- NumPy overview
- Introduction to NumPy Arrays
- Working with NumPy Arrays
- Basic polynomial fitting

Student Goals

- Get to know the basic features of NumPy, namely arrays
- Know how to select and manipulate data in arrays
- Know how to use polynomials
- Get an overview of the standard functions in NumPy

NumPy Overview

- the NumPy[1] module provides mathematical functionality for python
- works with arrays and matrices
- implements a number of mathematical operations
- functionality comparable to that of MATLAB

[1] http://www.numpy.org

Use NumPy

- NumPy is an additional module → has to be imported

```python
# generic import
import numpy

# import and define 'np' for quicker access
# will be used in the following slides
import numpy as np
```

Arrays

- multidimensional array of objects of the *same* type
- indexed using tuples of positive integers

Array Creation

- create arrays
  - from python lists or sequences
  - generate with functions
  - load from strings or files
- the type is guessed from the input values if it is not specified

Array Creation – from Python Objects

```python
# one-dimensional array
a = np.array([1,2,3,4])

# optional: set the type
b = np.array([1,2,3,4], float)

# two-dimensional float array
c = np.array([(1.1, 2.2, 3.3), (4.4, 5.5, 6.6)])
d = np.array([[1.1, 2.2], [3.3, 4.4], [5.5, 6.6]], float)
```

$$c = \begin{pmatrix} 1.1 & 2.2 & 3.3 \\ 4.4 & 5.5 & 6.6 \end{pmatrix}, \quad d = \begin{pmatrix} 1.1 & 2.2 \\ 3.3 & 4.4 \\ 5.5 & 6.6 \end{pmatrix}$$

Array Creation – Generate with Functions

- Changing array dimensions is a costly operation
- setup empty array if you know how many rows/columns you'll need to improve performance

```
# zero-filled array with 3 rows and 4 columns
np.zeros( (3, 4) )
# array filled with ones
np.ones( (2, 3, 4) )
# empty array (can contain random values)
np.empty( (2, 3) )
# array filled with numbers from a range
# from 10 to 30 with a step size of 5
np.arange( 10, 30, 5 )
```

Array Creation – Load from file

- Numpy defines convenience functions for loading data[2]
- Source can be a string (file name), a list of strings, a generator
- Additional options for defining delimiters, trimming white space, excluding comments, skipping header lines. . .

```
data = np.genfromtxt('file.csv', delimiter=',')
```

[2] https://docs.scipy.org/doc/numpy/user/basics.io.genfromtxt.html

Working with Arrays – Indexing

```
a = np.array([[1, 2, 3], [4, 5, 6]])

# array indexing and slicing is similar to that of lists
# access the content at given coordinates
a[0,2] # 3: first row, third column

# all elements in the row with index 1
a[1,:] # [ 4., 5., 6.]

# all elements in the column with index 2
a[:,2] # [3, 6]

# the last two elements of the last row
a[-1:,-2:] # [[ 5., 6.]]
```

Working with Arrays – Info

```
# shape
a.shape # (2, 3)

# data type
a.dtype # dtype('int64')

# size, i.e. number of elements in the array
a.size # 6

# length, i.e. elements in the first dimension
len(a) # 2

# rank, number of dimensions
a.ndim # 2
```

Working with Arrays – Dimensions

```python
a = np.array([[1, 2], [3, 4]])

# concatenate
b = np.array([[5, 6]])
np.concatenate((a, b), axis=0)
np.concatenate((a, b.T), axis=1)

# transpose
a.transpose()
a.T

# flatten
a.flatten() # array([ 1, 2, 3, 4])
```

Iterating

- you can iterate over arrays as you can with lists
- for multidimensional arrays: iterates over first axis

```python
a = np.array([[1, 2], [3, 4], [5, 6]])
for x in a:
    print(x)
# [1 2]
# [3 4]
# [5 6]
```

Array Mathematics

- arrays can be used for all sorts of mathematical applications
- operations either use the entire array as input
- or perform element-wise operations

Array Mathematics – Entire Array

- e.g. used for statistic analysis
- compute sum, product, minimum, mean, standard deviation... of array elements

```
a = np.array([[1, 2], [3, 4], [5, 6]])

a.sum() # 21 -- with python lists it's sum(list)
a.prod() # 720
a.min() # 1
a.mean() # 3.5
a.std()
# ...
```

Array Mathematics – Element-Wise Application

- operation is applied to each element separately
- e.g. round, floor, ceiling, absolute value, sign

```python
a = np.array([[-1.1, 2.2], [3.3, 4.4], [5.5, 6.6]])
print(np.ceil(a)) # [[ -1. 3.]\n [ 4. 5.]\n [ 6. 7.]]
print(np.rint(a)) # [[-1. 2.]\n [ 3. 4.]\n [ 6. 7.]]
print(np.abs(a)) # [[ 1.1 2.2]\n [ 3.3 4.4]\n [ 5.5 6.6]]
a + a # np.array([[2, 4], [6, 8], [10, 12]])
a * 2 # ([[-2.2, 4.4], [6.6, 8.8], [11.0, 12.2]])
# ...
```

Array Mathematics – Logical Expressions

- check if array elements fulfil a logical expression

```
a = np.array([1, 3, 0])
a > 2 # array([False, True, False], dtype=bool)
np.logical_and(a > 0, a < 3) # array([ True, False, False], dtype=bool)

b = np.array([True, False, True])
c = np.array([False, True, True])
np.logical_or(b, c) # array([ True, True, True], dtype=bool)
a == b # array([False, False, True], dtype=bool)
```

NumPy Constants

- NumPy defines some mathematical constants
  - π
  - e

```python
print(np.pi) # 3.141592653589793
print(np.e) # 2.718281828459045
```

Matrix multiplication

- `np.dot()` computes the dot product of vectors
- and the matrix product of higher-dimensional arrays
- make sure the arrays' shapes are aligned!

```
a = np.array([[0, 1], [2, 3]])
b = np.array([[1, 1], [4, 0]])
np.dot(a, b)
# [[ 4 0]
# [14 2]]
np.dot(b, a)
# [[2 4]
# [0 4]]
```

Polynomial Mathematics

- Find roots of a given polynomial or coefficients for roots
- Evaluate a polynomial for a given *x*
- Integrate or derive a polynomial. Integration constant *C* has default value 0

```
np.poly([4, 2]) # array([ 1., -6., 8.]), i.e. x^2 - 6x + 8
np.roots([ 1., -6., 8.]) # array([ 4., 2.]), i.e. x1 = 4, x2 = 2
np.polyval([1, -2, 0, 2], 4) # 34
np.polyint([1, 1, 1, 1]) # array([ 0.25, 0.33333333, 0.5, 1. , 0. ])
np.polyder([1./4., 1./3., 1./2., 1., 0.]) # array([ 1., 1., 1., 1.])
```

Fitting Polynomials

- Given arrays of values and an order, find a polynomial in the least square sense
- that is, the one where the sum of squared distances from the polynomial curve is minimised
- useful for finding structures in observed data

```
x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [0, 2, 1, 3, 7, 10, 11, 19]
np.polyfit(x, y, 2) # array([ 0.375, -0.88690476, 1.05357143])
```

Outlook

- NumPy is part of SciPy[3] and included in many other mathematical modules
- NumPy can do a lot more....
  - calculate statistic measures such as standard deviation, covariance, correlation coefficient
  - random samples based on normal/...distribution
  - compute matrix trace, eigenvalues, inverse,...
  - compute sine, cosine, inverse sine,...
- Check the reference[4]!

[3] `https://scipy.org/`

[4] `https://docs.scipy.org/doc/numpy/reference/index.html`

Complex Example – Linear Equation Systems

Linear equation systems can be described using matrices

$$\begin{array}{ccccccc} x & & & + & z & = & 6 \\ & & -3y & + & z & = & 7 \\ 2x & + & y & + & 3z & = & 15 \end{array}$$

is equivalent to

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & -3 & 1 \\ 2 & 1 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 6 \\ 7 \\ 15 \end{pmatrix}$$

Complex Example – Solve Equation System

- The system before has the form $AX = B$
- We want to find X, that is $X = A^{-1}B$
- I.e. compute the inverse of A, multiply with B

Complex Example – Sample Solution Part 1

```python
import numpy as np

def solve_equation_system(coefficients, right_side):
    # check if the coefficients and right side match: same number of rows
    if coefficients.shape[0] != right_side.shape[0]:
        print("Dimensions of left and right side of the equation system don’
    # check if the coefficient matrix is square
    elif coefficients.shape[0] != coefficients.shape[1]:
        print("Coefficient matrix must be square.")
    else:
        inverse = np.linalg.inv(coefficients)
        return np.dot(inverse, right_side)
```

Complex Example – Sample Solution Part 2

```python
# equation system:
# x + z = 6
# -3y + z = 7
# 2x + y + 3z = 15

# as matrix
coefficients = np.array([[1, 0, 1], [0, -3, 1], [2, 1, 3]])
right_side = np.array([[6], [7], [15]])
print(solve_equation_system(coefficients, right_side))
```

Student Task – Correlation of two Variables

- Imagine you didn't know yet that height and weight of a person are correlated.
- You have conducted a survey on this
- With a number of variables stored in a numpy array
- You want to analyse the correlation between height and weight of a person, that is check how likely it is that with increasing height the weight is also higher and vice versa.
- Do so using the Pearson Correlation Coefficient

Student Task – Pearson Correlation Coefficient

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

- X and Y are two statistic variables
- their covariance cov is defined as

$$E(X - E(X)(Y - E(Y)) = E(X \cdot Y) - E(X)E(Y)$$

  where E is the expected value. Use the mean for this.
- σ is the standard deviation.
- Values are between -1 and 1. Interpretation: 1 signals strong correlation, 0 no correlation and -1 strong inverse correlation

Student Task – Steps

- Read the file 'height_weight.csv' using the `genfromtext` function
- The height is given in inches and the weight in pounds. Convert this to cm and kg respectively.
  1 inch $\approx$ 2.5cm, 1 pound $\approx$ 0.45kg
- Calculate mean and standard deviation of height and weight
- Compute the correlation
- What does this suggest about the data?

Student Task – Sample Solution Part 1

```python
import numpy as np

# read data from file
# columns: index, height, weight
data = np.genfromtxt('height_weight.csv', delimiter=';', skip_header=1)
data[1, 0:3]

# convert height to cm
data[:,1] = data[:,1] * 2.5

# convert weight to kg
data[:,2] = data[:,2] * 0.45
```

Student Task – Sample Solution Part 2

```python
sd_height = np.std(data[:, 1])
sd_weight = np.std(data[:, 2])

mean_height = np.mean(data[:, 1])
mean_weight = np.mean(data[:, 2])

# mean of height * weight
mean_combined = np.mean(data[:, 1] * data[:, 2])

covariance = mean_combined - mean_height * mean_weight
correlation = covariance / (sd_height * sd_weight)
```