Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

FS24 CAS PML - Python

Niklaus Johner

niklausbernhard.johner@bfh.ch

FS24 CAS PML - Python

15. numpy

numpy basics

- *numpy* stellt die Basisobjekte für technisches Rechnen in python zur Verfügung
  - arrays (*ndarray*)
  - matrices (*matrix*)
- es wird meistens als *np* importiert

```
import numpy as np
```

- enthält auch
  - mathematische Funktionen
  - Statistik
  - linear algebra

numpy array

- Klassen Initialisierung

```
l1d = [1, 2, 3, 4, np.nan, 6]
a1d = np.array(l1d)
```

1D

```
l2d = [[1, 2], [3, 4], [np.nan, 6]]
a2d = np.array(l2d)
```

2D

- Vektoren Dimensionen

```
In [10]: print("dim=", a1d.ndim, "shape=", a1d.shape)
    ...: print("dim=", a2d.ndim, "shape=", a2d.shape)
    ...:
dim= 1 shape= (6,)
dim= 2 shape= (3, 2)
```

numpy array

- Andere Initialisierungen

```
a1d = np.zeros(12)  # array of 0s with 12 elements
a2d = np.ones((3, 4))  # array of 1s with 3x4 elements
np.arange(0.0, 1.01, 0.1)  # from 0 to 1 with step 0.1
```

- Array slicing

```
a2d[0, :]  # first line
a2d[:, 0]  # first column
a2d[0, 0]  # first element of first column
a2d[0:2, 0]  # first two elements of first column
```

numpy array operations

- Alle Operationen auf *np.arrays* sind element-wise

```
a1d = np.array([1, 2, 3, 4, np.nan, 6])
a2d = np.array([[1, 2], [3, 4], [np.nan, 6]])
```

```
In [54]: a1d + 3
Out[54]: array([  4.,   5.,   6.,   7.,  nan,   9.])
```

```
In [55]: a2d / 3
Out[55]:
array([[ 0.33333333,  0.66666667],
       [ 1.        ,  1.33333333],
       [        nan,  2.        ]])
```

numpy array operations

- Alle Operationen auf *np.arrays* sind element-wise

```
a1d = np.array([1, 2, 3, 4, np.nan, 6])
a2d = np.array([[1, 2], [3, 4], [np.nan, 6]])
```

```
In [56]: a1d * a1d
Out[56]: array([  1.,   4.,   9.,  16.,  nan,  36.])
```

```
In [57]: a2d * a2d
Out[57]:
array([[  1.,   4.],
       [  9.,  16.],
       [ nan,  36.]])
```

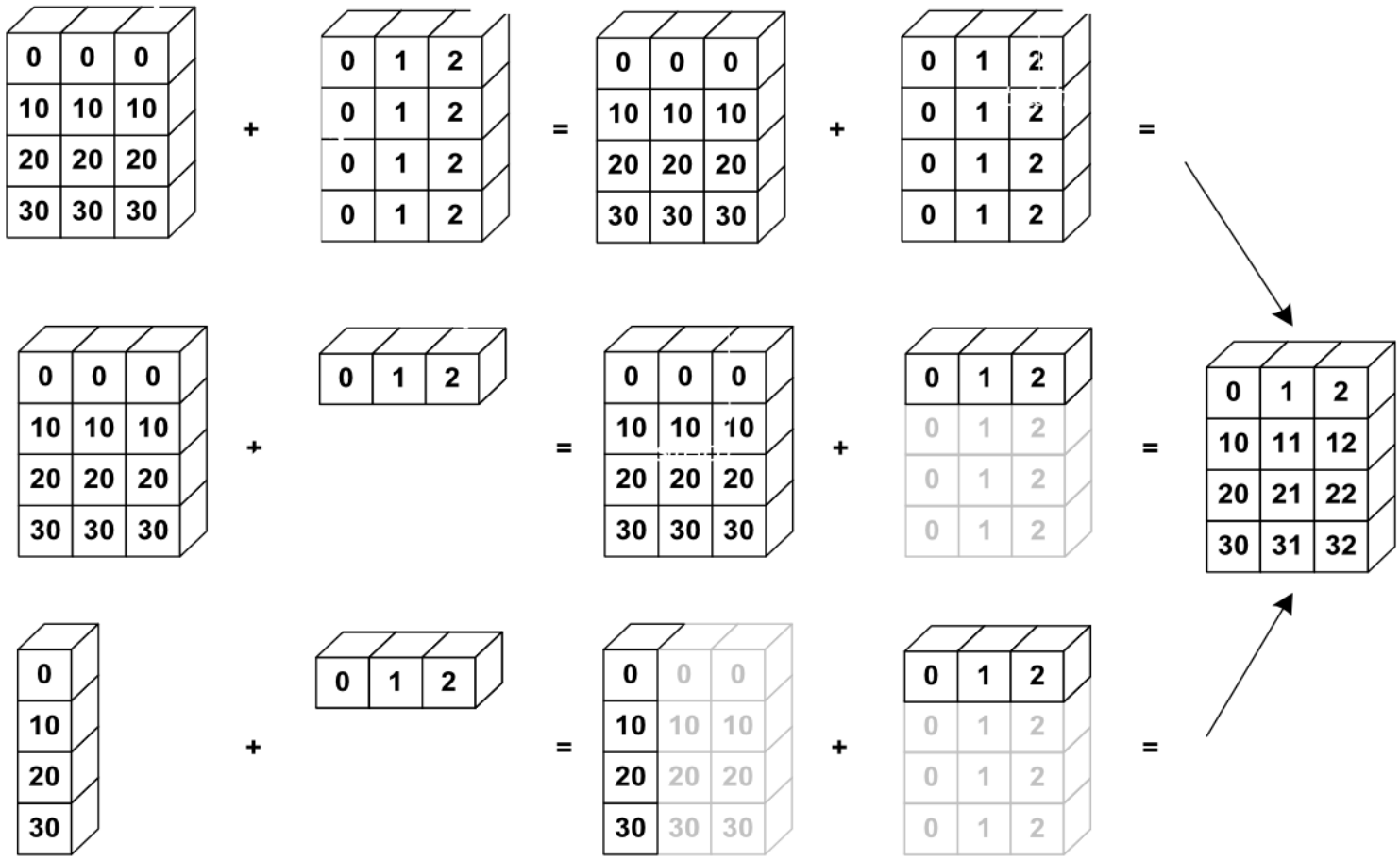
numpy array operations

- Alle Operationen auf *np.arrays* sind element-wise

```
a1d = np.array([1, 2, 3, 4, np.nan, 6])
a2d = np.array([[1, 2], [3, 4], [np.nan, 6]])
```

```
In [59]: a1d[:2] * a2d
Out[59]:
array([[  1.,   4.],
       [  3.,   8.],
       [ nan,  12.]])
```

Broadcasting



http://www.scipy-lectures.org/_images/numpy_broadcasting.png

Statistische Funktionen

- Die meisten Funktionen gibt es als Methoden von *np.array* und als Funktionen in *np*

```
In [68]: a2d = np.array([[1, 2], [3, 4], [5, 6]])

In [69]: np.mean(a2d)
Out[69]: 3.5

In [70]: a2d.mean()
Out[70]: 3.5
```

- Sie nehmen auch ein fakultatives *axis* argument, die Achse auf welcher die Funktion gerechnet wird.

```
In [106]: np.mean(a2d, axis=0)
Out[106]: array([ 3.,  4.])
```

Statistische Funktionen

- Als Funktionen in *np*

```
np.mean(a2d, axis=0)    # mean
np.std(a2d, axis=0)     # stdandard deviation
np.min(a2d, axis=0)     # min value
np.max(a2d, axis=0)     # max value
np.sort(a2d, axis=0)    # return sorted array
```

```
np.corrcoef(a2d)    # correlation coefficients
```

Statistische Funktionen

- Als Methoden von *np.array*

```
a2d.mean(axis=0)     # mean
a2d.std(axis=0)      # stdandard deviation
a2d.min(axis=0)      # min value
a2d.max(axis=0)      # max value
a2d.sort(axis=0)     # sorted array in place
```

Vergleiche sind auch element-wise

- Vergleich generiert ein boolean array

```
In [3]: a1d = np.array([1, 2, 3, 4, 5, 6])

In [4]: a1d < 3
Out[4]: array([ True,  True, False, False, False, False], dtype=bool)
```

- Boolean arrays kann man als Masken verwenden

```
In [5]: a1d[a1d < 3]
Out[5]: array([1, 2])
```

Undefinierte Elemente

- Das undefinierte Element: `np.nan`
- Test für undefinierte Elemente: `np.isnan(a1d)`
- Wird auch Element-wise angewendet

```
In [6]: a1d = np.array([1, 2, 3, 4, np.nan, 6])
   ...: np.isnan(a1d)
   ...:
Out[6]: array([False, False, False, False,  True, False], dtype=bool)
```

Zusätzliche Folien

numpy Matrizen Multiplikation

```
a2d = np.array([[1, 2], [3, 4], [np.nan, 6]])
m2d = np.matrix(a2d)
```

- Für Matrizen Multiplikation
  - Entweder np.dot

```
In [64]: np.dot(a2d,a2d.T)
Out[64]:
array([[  5.,  11.,  nan],
       [ 11.,  25.,  nan],
       [ nan,  nan,  nan]])
```

  - Oder mit *matrix* Objekten arbeiten

```
In [65]: m2d * m2d.T
Out[65]:
matrix([[  5.,  11.,  nan],
        [ 11.,  25.,  nan],
        [ nan,  nan,  nan]])
```

Zufallszahlengenerator

- Mit numpy kann man arrays von Zufallszahlen generieren
  - Zahlen zwischen 0 und 1:

```
In [22]: np.random.random(3) #3 random numbers in [0,1)
Out[22]: array([ 0.02142362,  0.40388068,  0.75375359])
```

  - Zahlen aus einer normalen Verteilung

```
In [23]: np.random.normal(loc=0.0, scale=1.0, size=3)
Out[23]: array([ 0.1979848 ,  1.08893364,  1.75919085])
```

  - Auswahl aus einem array

```
In [24]: np.random.choice([1,3], 5)
Out[24]: array([3, 1, 3, 3, 3])
```

Logische Funktionen

- *np.any* Funktion testet ob irgend ein Element *True* ist

```
In [9]: a2d = np.array([[1, 2], [3, 4], [np.nan, 6]])
```

```
In [11]: np.any(np.isnan(a2d))
Out[11]: True
```

- Es nimmt auch *axis* als keyword argument

```
In [13]: np.any(np.isnan(a2d),axis=1)
Out[13]: array([False, False,  True], dtype=bool)
```

- Ein Array von boolean kann man invertieren

```
In [9]: np.logical_not(np.isnan(a1d))
Out[9]: array([ True,  True,  True,  True, False,  True], dtype=bool)
```