NumPy, SciPy & Matplotlib – a scientist's best friends

Scientific Programming with Python

Christian Elsasser

Based partially on a talk by Stéfan van der Walt

The Ecosystem of Homo Python Scientificus

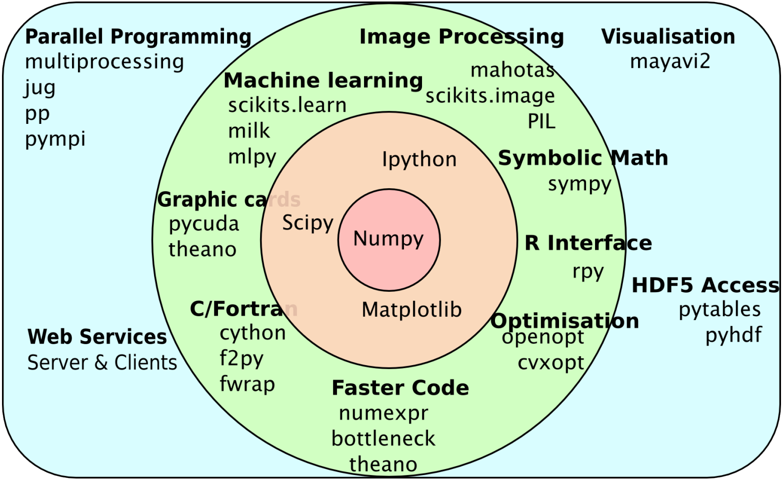


Table of Contents

What is NumPy?

NumPy’s C API

```
ndarray typedef struct PyArrayObject {
    PyObject_HEAD
    char *data;
    int nd;
    npy_intp *dimensions;
    npy_intp *strides;
    PyObject *base;
    PyArray_Descr *descr;
    int flags;
    PyObject *weakreflist;
} PyArrayObject ;
```

`np.__version__` indicates version
`np.show_config()` reveals information about LinAlg calculation

`import numpy as np`

NumPy offers memory-efficient data containers for fast numerical operations, *i.e.* in data manipulation and also in typical linear algebra calculations

Standard Python

```
>>> L = range(1000)
>>> [i**2 for i in L]
```

NumPy

```
>>> import numpy as np
>>> a = np.arange(1000)
>>> a**2
```

$\Rightarrow$ Speed up by a factor of ~ 40

Creating NumPy Arrays

There are several ways to do so

Creating arrays

```
>>> a = np.array([1,2,4])   # [1,2,4]
>>> b = np.arange(1,15,2)   # [1,3,5,7,9,11,13,15]
>>> c = np.linspace(0,1,6)  # [0.0,0.2,0.4,0.6,0.8,1.0]
>>> d = np.ones((3,3))                  # 3x3 array of ones
>>> e = np.zeros((2,5,3))               # 2x5x3 array of zeros
>>> f = np.eye(4)                       # 4x4 unit 'matrix'
>>> g = np.diag(np.array([1,2,3,4])) # diagonal 'matrix'
>>> h = np.random.rand(4)               # array with [0,1]
>>> i = np.random.randn(4,5)            # 4x5 array (norm. dist)
```

Random seed can be set with `np.random.seed(<integer>)`

Basic Operations

Many basic functions/operators can be applied on numPy arrays

Examples

```
>>> a = np.random.rand(3,4)
>>> b = np.random.rand(3,4)

>>> a+b
>>> a*b
>>> a/b
>>> a+3.0
>>> np.exp(b)
>>> a>b
```

All element-wise operations including dedicated functions, called universal functions (ufunc)

`math.exp(a)` $\Rightarrow$ failure as it expects scalar

Data Representation

Data type accessible via `dtype` variable

Datatype

```
>>> a = np.array([1,0,-2],dtype=np.int64)   #[1,0,-2]
>>> b = np.array([1,0,-2],dtype=np.float64) #[1.0,0.0,-2.0]
>>> c = np.array([1,0,-2],dtype=np.bool)    #[True,False,True]
>>> c.dtype # dtype('bool')
```

Data Structure

Information via attributes accessible:

`ndim`	number of dimensons
`nbytes`	data size
`shape`	size of the different dimensions (as a tuple)
`size`	total number of entries
`data`	data representation
`strides`	number of bytes to jump to in-/decrement index by one (as a tuple)
`flags`	among other things if the memory "belongs" to this array

Transpose of arrays can be called by `<array name>.T` $\Rightarrow$ inverts shape and strides (*i.e.* C-contiguous $\leftrightarrow$ F-contiguous)

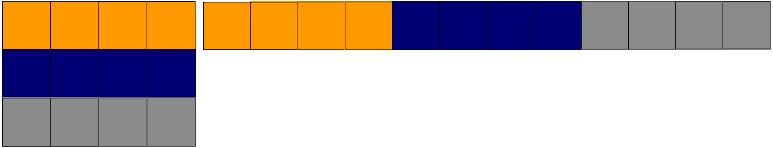
Be aware that many manipulations do not lead to memory duplications. You can force it by the `copy` method.

Data Structure

Strides

Problem of one-dimensional memory to store multi-dimensional arrays:

Logical alignment: *Memory alignment:*



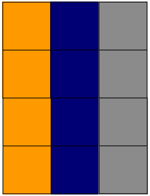
Strides describe logical alignment within the memory

Data Structure

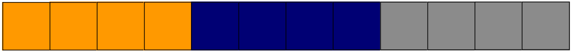
Strides

Problem of one-dimensional memory to store multi-dimensional arrays:

Logical alignment:



Memory alignment:



Transposing the array means to interchange the strides of the different dimensions.

Strides describe logical alignment within the memory

Data Structure

Information via attributes accessible:

`ndim`	number of dimensons
`nbytes`	data size
`shape`	size of the different dimensions (as a tuple)
`size`	total number of entries
`data`	data representation
`strides`	number of bytes to jump to in-/decrement index by one (as a tuple)
`flags`	among other things if the memory “belongs” to this array

Transpose of arrays can be called by `<array name>.T` $\Rightarrow$ inverts shape and strides (*i.e.* C-contiguous $\leftrightarrow$ F-contiguous)

Be aware that many manipulations do not lead to memory duplications. You can force it by the `copy` method.

Get the Data

Reading data from txt/csv/etc. files can be sometimes very painful, especially with complicated/mixed data structure

NumPy offers with the function `loadtxt` an easy way to read in data from text files

`delimiter` defines the columns separation, `comments` the string indicating comments in the text files

Binary files as well as text files are also readable via the function `fromfile`

Get the Data

Complicated data structure are manageable by defining the data type, *e.g.*

Solar.txt (Solar system on June 21, 2014)

```
Sun     332946 2.13E-03 -1.60E-03 -1.20E-04  5.01E-06 ...
Mercury 0.0552 1.62E-01  2.64E-01  6.94E-03 -2.97E-02 ...
Venus   0.8149 3.02E-01  6.54E-01 -8.44E-03 -1.85E-02 ...
Earth   1.00   5.66E-01 -8.46E-01 -9.12E-05  1.40E-02 ...
...
```

Datatype

```
>>> dt = np.dtype([('name','|S7'),('mass',np.float),
 ('position',[('x',np.float),('y',np.float),('z',np.float)]),
 ('velocity',[('x',np.float),('y',np.float),('z',np.float)])])

>>> data = np.loadtxt('Solar.txt',dtype=dt)
```

String in Arrays

String in arrays are in principle not a problem (as seen before), but two things have to be kept in mind

1. Speed reduction due to a different common base type of the objects stored in the array (*i.e.* PyObject)
2. Memory spoiling since the entry size is defined by the maximal length of the stored strings

⇒ if possible, better work with *e.g.* lookup tables

In general you can mix different data type in an array

Mixed datatype

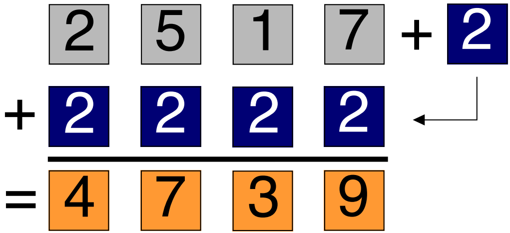
```
>>> na = np.array([2,True,"Hello"],dtype=object)
```

without `dtype=object` the elements would be treated as strings

Broadcasting

Memory-friendly way of combining arrays with different shapes in mathematical operations

Example:

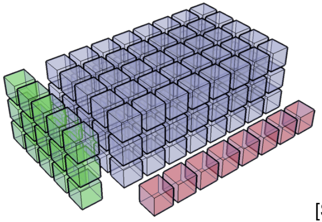


Arrays are alignable if the number of elements in the dimensions match (*i.e.* they are equal or there is only one element)

Details can be found in docstrings `np.doc.broadcasting`

Broadcasting (Example)

Multiplication of a 3×5-array and a 8-element array



[S. v. d. Walt]

Broadcasting

```
>>> a = np.random.rand(3,5)
>>> b = np.random.rand(8)
>>> c = a[...,np.newaxis]*b
>>> c.shape # (3,5,8)
```

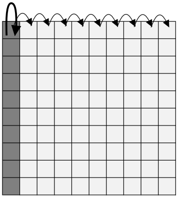
Explicit Broadcasting

NumPy has the method `broadcast_arrays` to align two or more arrays

Explicit Broadcasting

```
>>> d = np.random.rand(1,10)
>>> e = np.random.rand(10,1)
>>> dd,ee = np.broadcast_arrays(d,e)
```

`dd` and `ee` are now 10×10-arrays, but without own data



Broadcasted arrays have a stride of zero $\Rightarrow$ pointer stays while index moves

This concept is a generalisation of the `meshgrid` function in MATLAB

Simple Indexing

NumPy allows to easily select subsets in the array, *e.g.*

Simple indexing

```
>>> a = np.arange(100).reshape(10,10)
>>> a[4:9]      # rows 4 to 8
>>> a[:,3:8]    # columns 3 to 7
>>> a[:,-1]     # the last column
>>> a[2:7,:8]   # rows 2 to 6 and columns upto 7
```

Also repetition of rows or columns are possible, *e.g.*

Simple indexing (continued)

```
>>> a[:,[1,3,1]]
```

All these operations do not create additional memory entries!

Fancy Indexing

NumPy also allows to select subsets via arrays of indices, *e.g.*

Fancy indexing

```
>>> a = np.arange(100).reshape(10,10)
>>> i0 = np.random.randint(0,10,(8,1,8))
>>> i1 = np.random.randint(0,10,(2,8))
>>> a[i0,i1] # creates a 8×2×8 array
```

- First broadcasting of the two index arrays `i0` and `i1`
- Then selecting the elements in `a` according to the broadcasted arrays

Caution: Mixing of indexing types (*e.g.* `b[5:10,i0,:,i1]`) can lead to unpredictable output shapes (and to barely readable code)

Short Break! (5 min)

You are here!

Matrices

NumPy offers a matrix framework for linear algebra calculations, allowing to defining one- and two-dimensional arrays as matrices

Matrices

```
>>> a = np.matrix([[1,2],[3,4]])
>>> b = np.matrix(np.random.rand(4))
>>> c = np.matrix(np.random.rand(3,3))
```

One-dimensional arrays → $1 \times n$ matrices, *i.e.* row vectors

Matrices have some additional functionality (*e.g.* inverse: `a.I`, hermitian: `a.H`)

Linear Algebra

Light version of SciPy's linear algebra implementation at `np.linalg`

Examples of available functionality:

`np.linalg.cholesky`	`np.linalg.det`	`np.linalg.eig`
`np.linalg.eigh`	`np.linalg.qr`	`np.linalg.svd`

The functions are wrappers of the LAPACK linear algebra package

More functionality is embedded in the full SciPy implementation `scipy.linalg`, *e.g.*

Matrix Exponential

```
>>> a = np.matrix([[1,2],[3,4]])
>>> scipy.linalg.expm(a)
```

SciPy

…or where the fun really starts

- Offering a large number of functionality for numerical computation
  - `scipy.linalg` $\rightarrow$ Linear Algebra
  - `scipy.optimize` $\rightarrow$ Numerical optimisation (incl. least square)
  - `scipy.integrate` $\rightarrow$ Numerical integration
  - `scipy.stats` $\rightarrow$ Statistics including a large set of distributions
  - more at http://docs.scipy.org/doc/scipy/reference/
- Eco-system of more advanced packages for data analysis, *e.g.*
  - scikits.learn: Machine-learning algorithms
  - scikits.image: Image processing
  - pytables: data structure (based on HDF5)
  - …

Remark: `import scipy as sp` only imports the most basic tools $\Rightarrow$ `from scipy import stats`

Example: `scipy.stats`

Discrete and continuous distributions, *e.g.*:

`binom`	`poisson`	`norm`
`expon`	`gamma`	`cauchy`
`lognorm`	`beta`	`pareto`

`...`

allowing different operations:

`rvs`	Random variates
`pdf`	Probability density function (continuous)
`pmf`	Probability mass function (discrete)
`cdf`	Cumulative density function (continuous)

...

Remark: Documentation is bad and sometimes the parameter order is not very intuitive

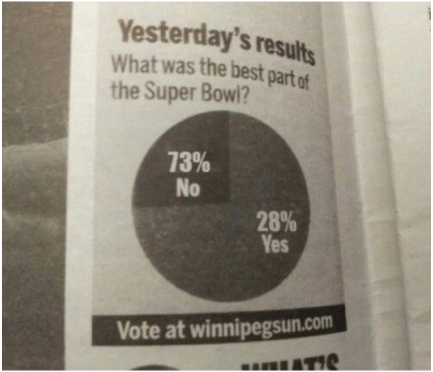
Why Matplotlib might be useful

BusinessInsider and Karl W. Broman (University of Wisconsin - Madison), *Using Microsoft Excel to obscure your data and annoy your readers*

[Fox News]

Why Matplotlib might be useful

BusinessInsider and Karl W. Broman (University of Wisconsin - Madison), *Using Microsoft Excel to obscure your data and annoy your readers*



[Winnipeg Sun]

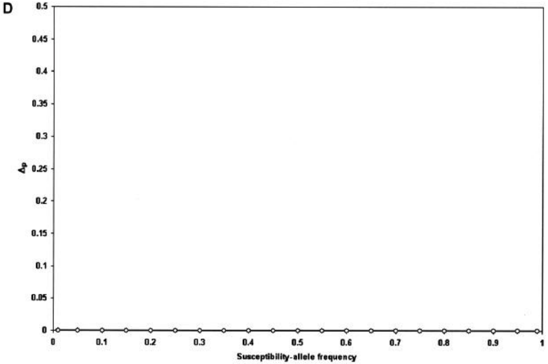
Why Matplotlib might be useful

BusinessInsider and Karl W. Broman (University of Wisconsin - Madison), *Using Microsoft Excel to obscure your data and annoy your readers*

[Wall Street Journal]

Why Matplotlib might be useful

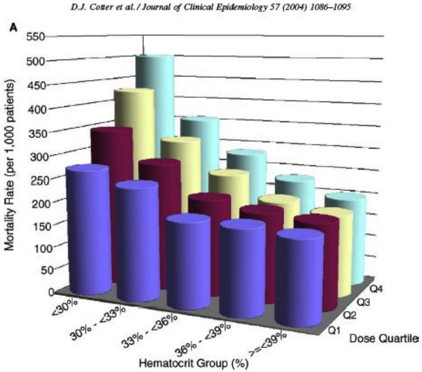
BusinessInsider and Karl W. Broman (University of Wisconsin - Madison), *Using Microsoft Excel to obscure your data and annoy your readers*



[J. K. Wittke-Thompson et al., Am. Jour. of Hum. Gen. 76 (2005) 967-986]

Why Matplotlib might be useful

BusinessInsider and Karl W. Broman (University of Wisconsin - Madison), *Using Microsoft Excel to obscure your data and annoy your readers*



[D. J. Cotter et al., Jour. of Clinical Epid. 57 (2004) 1086-1095]

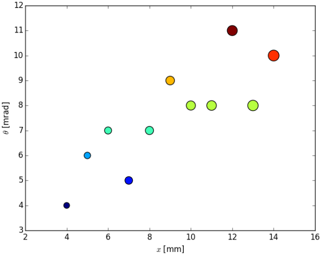
Why Matplotlib might be useful

- Plots can be
  - mis-leading,
  - manipulative,
  - incomprehensible,
  - fussy,
  - just horrible,
  - ...
- "A picture is worth a thousand words" (*i.e.* the text of a complete paper)
- but only if it is good
- Matplotlib might be your friend in data visualisation

Matplotlib on one Slide

Matplotlib Example

```
>>> x = np.array([10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5])
>>> y = np.array([8, 7, 8, 9, 8, 10, 7, 4, 11, 5, 6])
>>> plt.scatter(x, y, c=y, s=20*x)
>>> plt.xlabel(r'$x$ [mm]')
>>> plt.xlabel(r'$\theta$ [mrad]')
>>> plt.show()
```



. . . and many more fancy possibilities, *cf.* http://matplotlib.org

⇒ Almost equal functionality as MATLAB, but with the power of Python behind it . . .

Summary

- NumPy is a very powerful tool for numerical computations and data manipulations
- ...and serves as a basis for many advanced libraries in Python
- SciPy offers a large number of numerical functions and tools $\Rightarrow$ Don't reinvent the wheel, check the SciPy documentation
- Functionality to display measurements due to Matplotlib

- NumPy, SciPy and Matplotlib together with ipython give you a versatile replacement of MATLAB
- `ipython --pylab` (or `from pylab import *`)
- Try it out, try it out, try it out!

References

1. Stéfan van der Walt, *Diving into NumPy*, Advanced Scientific Programming in Python, 2013 (Zurich)
2. Bartosz Teleńczuk, *Introduction to data visualization*, Advanced Scientific Programming in Python, 2013 (Zurich)
3. Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux, *The NumPy array: a structure for efficient numerical computation*, Computing in Science and Engineering (IEEE)
4. http://www.numpy.org
5. http://www.scipy.org
6. http://matplotlib.org

Backup

Data Structure (Advanced)

Further information via the `flags` variable accessible:

C_CONTIGUOUS	dimension ordering C-like
F_CONTIGUOUS	dimension ordering Fortran-like
OWNDATA	responsibility of memory handling
WRITEABLE	data changable
ALIGNED	appropriate hardware alignment
UPDATEIFCOPY	update of base array

C-contiguous:
$a[0, 0], a[0, 1], \ldots, a[0, n], a[1, 0], \ldots, a[m, n]$

F-contiguous:
$a[0, 0], a[1, 0], \ldots, a[m, 0], a[0, 1], \ldots, a[m, n]$

Broadcasting (Dimensional)

This principle can be expanded to multi-dimensional arrays, *e.g.* a 3×4-array and a 1D 4-elements array $\Rightarrow$ adding/multiplying/etc. to each of the three rows the 1D array

Rule: Compare dimensions, starting from the last one. Match when either dimension is one or None, or if dimensions are equal.

`(3,4)`	`(4,1,6)`	`(3,4,1)`	`(3,2,5)`
`(4)`	`(1,3,6)`	`(8)`	`(6)`
`(3,4)`	`(4,3,6)`	`(3,4,8)`	`not OK`

Arrays can be extended to further dimensions by `<array name>[...,np.newaxis]`, *e.g.*

`a.shape` $\rightarrow$ `(3,2)`

$\Rightarrow$ `a[...,np.newaxis,np.newaxis].shape` $\rightarrow$ `(3,2,1,1)`