WEIHENSTEPHAN · TRIESDORF
University of Applied Sciences

TUMCS

Artificial Intelligence for Biotechnology

Hands-on: NumPy and Matplotlib

Prof. Dr. Dominik Grimm

TUM Campus Straubing for Biotechnology and Sustainability

Weihenstephan-Triesdorf University of Applied Sciences

Technical University of Munich, TUM Department of Informatics

NumPy

Learning objectives of this lecture:

- **Basic Array Methods**
- **Indexing of NumPy Arrays**
- **Linear Algebra with NumPy**

NumPy Arrays

Motivation

Let's say we have two Python lists `a` and `b` for which we want to compute the element-wise multiplication of each element and store it in a new list `c`.

Python core language

```
a = [1,2,3,4]

b = [2,4,7,9]

c = []

for i in range(len(a)):

        c.append(a[i] * b[i])
```

Using NumPy arrays

```
a = np.array([1,2,3,4])
b = np.array([2,4,7,9])
c = a * b
```

👉 **Advantages:** less error-prone, closer to the standard mathematical notation, highly optimized & precompiled C code (much faster than the alternative with for loops for large n)

NumPy 1D-Array Creation

1-dimensional NumPy arrays can be initialized with a list of values using the `np.array` constructor:

Example

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

Every NumPy array has a type (e.g. `int64`, `float32`, etc.). The `dtype` attribute of an array allows you to access the type of an array, or allows to specify the type when creating an array:

Example

```
a = np.array([1,2,3,4])
print(a)
print(a.dtype)
b = np.array([1,2,3,4], dtype=np.float32)
print(b)
print(b.dtype)
```

Output

```
[1 2 3 4]
int64
[1. 2. 3. 4.]
float32
```

NumPy Array Creation Routines

Initialize an array from a sequence using the `np.arange` and `np.linspace` methods.

np.arange is the NumPy equivalent of range:

```
a = np.arange(7)
print(a)
```

```
[0 1 2 3 4 5 6] #this is an array not a list, 7 is exclusive
```

```
b = np.arange(3,10,2)
print(b)
```

```
[3 5 7 9] #start at 3 until 10 (exclusive) with a stride of 2
```

np.linspace allows you to create an evenly spaced array of x numbers (e.g. 5):

```
a = np.linspace(1,20,5)
print(a)
```

```
[ 1.  5.75 10.5 15.25 20. ] #create a sequence of 5 values
starting at 1 until 20 (inclusive)
```

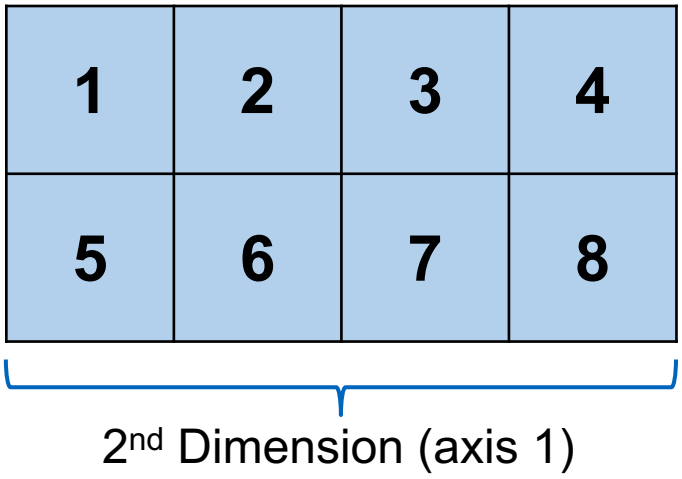
NumPy N-Dimensional Array Creation

N-dimensional NumPy arrays can be initialized with a list of values using the `np.array` constructor:

Example 2-dimensional Array

```
a = np.array( [[1,2,3,4], [5,6,7,8]] )
print(a)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

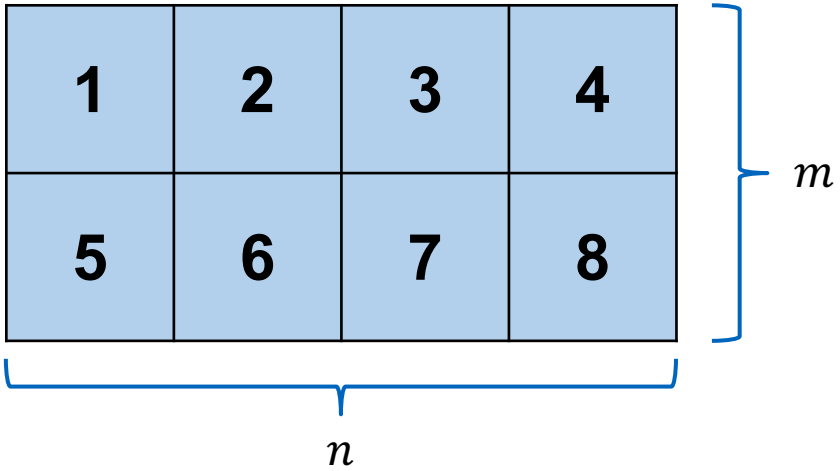


You can create NumPy arrays with up to 32 axis!

Linear Algebra

Matrix is a 2-dimensional NumPy Array

```
A = np.array( [[1,2,3,4], [5,6,7,8]] )
```



$\boldsymbol{A} \in \mathbb{R}^{m \times n}$,
where m is the number of rows
and n is the number of columns

NumPy N-Dimensional Array Creation

N-dimensional NumPy arrays can be initialized with a list of values using the `np.array` constructor:

Example 2-dimensional Array

```
a = np.array( [[1,2,3,4], [5,6,7,8]] )
print(a)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

Return the number of elements in an array

```
print(a.size)
```

```
8
```

Return the number of dimensions of an array

```
print(a.ndim)
```

```
2
```

Return the number of elements for each dimension

```
print(a.shape)
```

```
(2,4)
```

Array Indexing

Indexing NumPy arrays is the same as indexing list.

Example 1-Dimensional Array

```
a = np.arange(7)
print(a[:2]) } [0 1]
```

Example 2-Dimensional Array (Indexing arrays with more than one dimension (axis) are separated by commas)

```
a = np.array([[1,2,3,4], [5,6,7,8]])
print(a[0,0]) #upper left
print(a[-1,-1]) #lower right
print(a[1,2]) #second row, third column
print(a[0,:]) #entire first row     } [1 2 3 4]
print(a[:,0]) #entire first column  } [1 5]
print(a[:,:2]) #first two columns   } [[1 2]
                                      [5 6]]
```

1	2	3	4
5	6	7	8

Advanced Array Indexing

Filtering NumPy Arrays with Boolean Operators

Example

```
a = np.array([[1,2,3,4], [5,6,7,8]])
mask = (a <= 5)
print(mask)
```

```
[[ True True True True]
 [ True False False False]]
```

```
print(a[mask])
```

```
[1 2 3 4 5]
```

Operations in NumPy Arrays

Universal Functions – `ufuncs`

Example: Sum up all elements in a list

Core Python

```
a = [1,2,3,4,5,6]
sum = 0
for elem in a:
    sum += elem
print(sum)
```

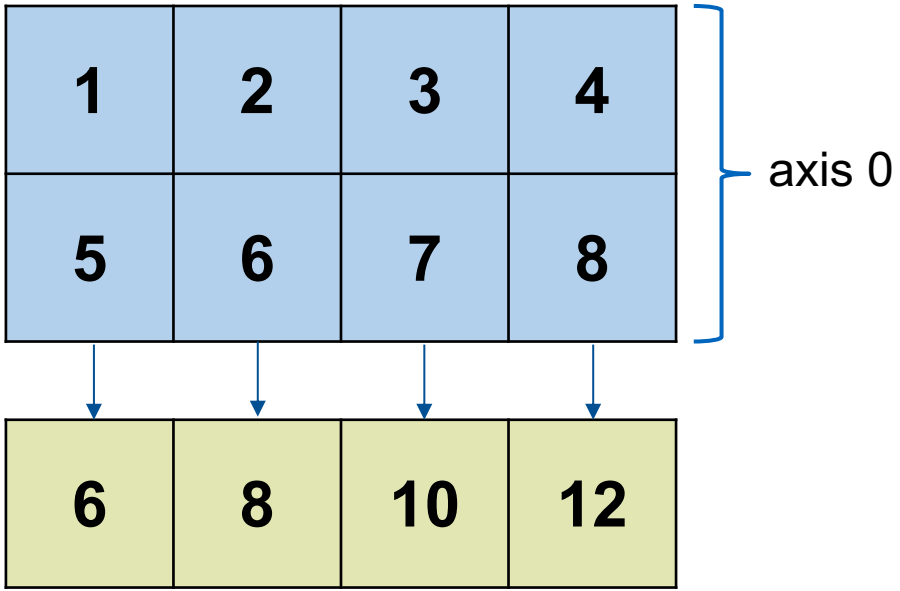
NumPy ufuncs

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

Operations in NumPy Arrays

Sum up elements along a given dimension

Sum up all columns

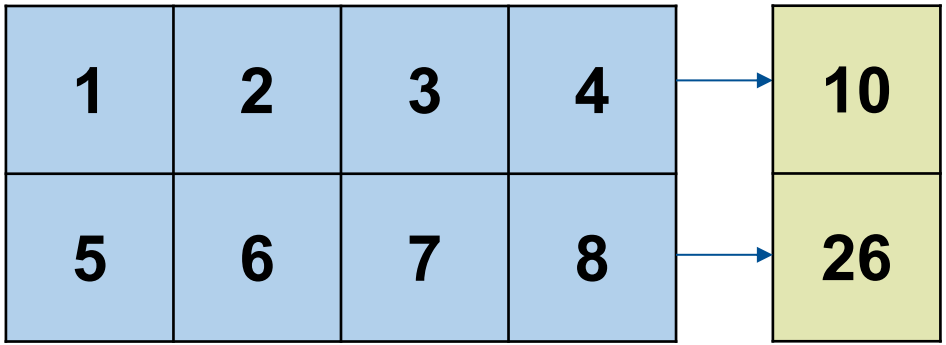



```
a = np.array([[1,2,3,4],
              [5,6,7,8]])
csum = np.sum(a, axis=0)
#or csum = a.sum(axis=0)


print(csum)
```

Operations in NumPy Arrays

Sum up elements along a given dimension

Sum up all rows




```
a = np.array([[1,2,3,4],
              [5,6,7,8]])
csum = np.sum(a, axis=1)
#or csum = a.sum(axis=1)
print(csum)
```

Other useful ufuncs:

np.mean(), np.std(), np.min(), np.var(), np.max(), np.argmin(), np.argmax(), etc…

Linear Algebra

Transpose a Matrix: A^T

```
A = np.array( [[1,2,3,4], [5,6,7,8]] )
At = A.transpose() #or A.T
print(At)
```

1	2	3	4
5	6	7	8

1	5
2	6
3	7
4	8

- Row-vector transposed results in a column-vector
- Column-vector transposed results in a row-vector

Linear Algebra

NumPy arrays can be interpreted as vectors and matrices

Row Vector $\boldsymbol{a} \in \mathbb{R}^{1\times n}$

6	8	10	12

```
a = np.array([6,8,10,12])
print(a)  } [6,8,10,12]
a = np.array([[6,8,10,12]]) #alternative which is helpful for linear algebra
print(a)  } [[6,8,10,12]]
```

Column Vector $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$

10
26

```
b = np.array([[10],[26]])
print(b) } [[10],
           [26]]
```

Linear Algebra

NumPy arrays can be interpreted as vectors and matrices

Row Vector → Column Vector

```
a = np.array([6,8,10,12])
a = a[:, np.newaxis]
print(a)
```

```
[[ 6]
 [ 8]
 [10]
 [12]]
```

(Alternative)

```
a = np.array([[6,8,10,12]])
a = a.transpose()
print(a)
```

6
8
10
12

Column Vector → Row Vector

```
b = np.array([[10],[26]])
b = b.flatten()
print(b)
```

```
[10,26]
```

10	26

Special Matrices

Useful and commonly used array construction routines

`np.ones` and **`np.zeros`** to construct arrays containing ones or zeros:

```
a = np.ones((2,3))
print(a)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

```
b = np.zeros((3, 2))
print(b)
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
```

`np.eye` to construct an identity matrix:

```
a = np.eye(3)
print(a)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

`np.diag` to construct a diagonal matrix:

```
a = np.diag([1,2,3])
print(a)
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Linear Algebra

Hadamard Product: Elementwise multiplication of two equally sized matrices:

$$\boldsymbol{A} \circ \boldsymbol{B} = (a_{ij} \cdot b_{ij}) = \begin{pmatrix} a_{11} \cdot b_{11} & \cdots & a_{1n} \cdot b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} \cdot b_{m1} & \cdots & a_{mn} \cdot b_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

```
A = np.array( [[1,2,3,4], [5,6,7,8]] )
B = np.random.random((2,4)) # Create Random Matrix with size 2 x 4
C = A*B
```

1	2	3	4
5	6	7	8

$\circ$

1	2	3	4
5	6	7	8

2	4	9	16
25	36	49	64

Linear Algebra

Dot Product of two Vectors

Let $\boldsymbol{r} \in \mathbb{R}^{1\times n}$ and $\boldsymbol{c} \in \mathbb{R}^{1\times n}$ be two row vectors. The dot product is defined as:

$$\boldsymbol{r} \cdot \boldsymbol{c}^T = (r_1, \dots, r_n) \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix} = \sum_{i=1}^{n} r_i c_i$$

```
r = np.array([[1,2,3,4]])
c = np.array([[5,6,7,8]])
scalar = r.dot(c.T)
print(scalar)
```
} `[[70]]`

Linear Algebra

Dot Product of two Matrices

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times k}$ be two matrices. The dot product is defined as:

$$\boldsymbol{A} \cdot \boldsymbol{B} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{pmatrix} = \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix} \begin{pmatrix} c_1 & \cdots & c_k \end{pmatrix} = \begin{pmatrix} r_1 \cdot c_1 & \cdots & r_1 \cdot c_k \\ \vdots & \ddots & \vdots \\ r_m \cdot c_1 & \cdots & r_m \cdot c_k \end{pmatrix}$$

r_1, r_m, c_1, c_k

```
A = np.array([[1,2,3,4], [5,6,7,8]])
B = np.array([[1,2], [5,6], [1,2], [5,6]])
C = A.dot(B)
print(C)
```

```
[[34  44]
 [82 108]]
```

Linear Algebra

Examples Dot Product

```
A = np.array([[1,2,3,4], [5,6,7,8]]) #Matrix of size 2 x 4
b = np.array([9,8,7,6]) # row vector
b = b[:, np.newaxis] #row vector to column vector
```

Example: $c = b^T b$

```
bt = b.transpose() # transpose b
c = bt.dot(b)
print(c) # → [[230]]
```

Example: $c = Ab$

```
c = A.dot(b)
print(c) # → [[70],
              [190]]
```

Example: $C = bb^T$

```
C = b.dot(bt)
print(C) # →  [[81 72 63 54]
               [72 64 56 48]
               [63 56 49 42]
               [54 48 42 36]]
```

Matplotlib

Learning objectives of this lecture:

- **How to create basic plots (line charts, scatter & bar plots, histograms)**

Matplotlib

Matplotlib is a highly customizable plotting library for Python

Tutorials: https://matplotlib.org/tutorials/index.html

How to use Matplotlib in Jupyter Notebooks?

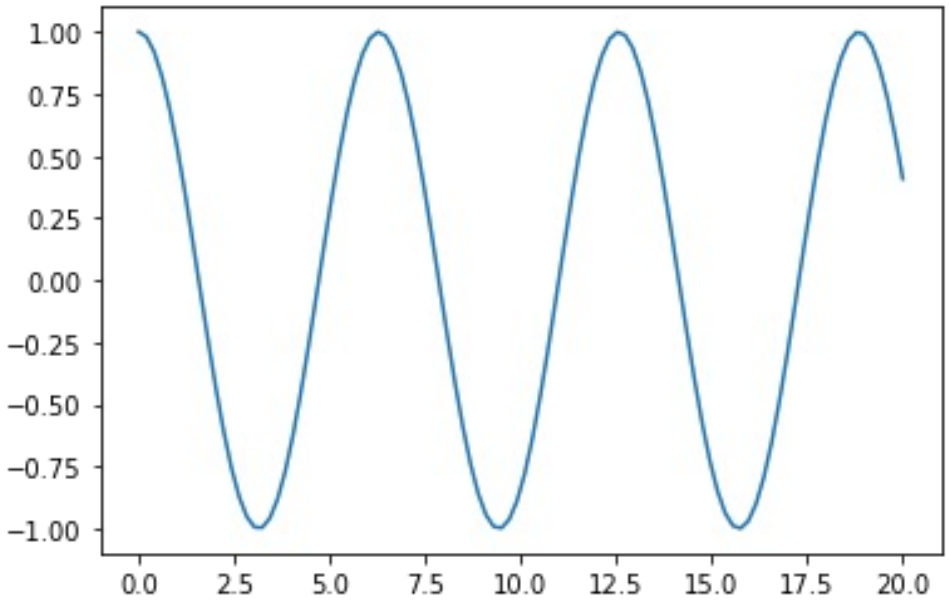
Import library and specify “inline” plotting. This is important such that the plots are shown in the Jupyter environment:

```
%matplotlib inline

import pylab as pl
```

Add these two lines at the top in your Jupyter Notebook and run the cell to import the library!

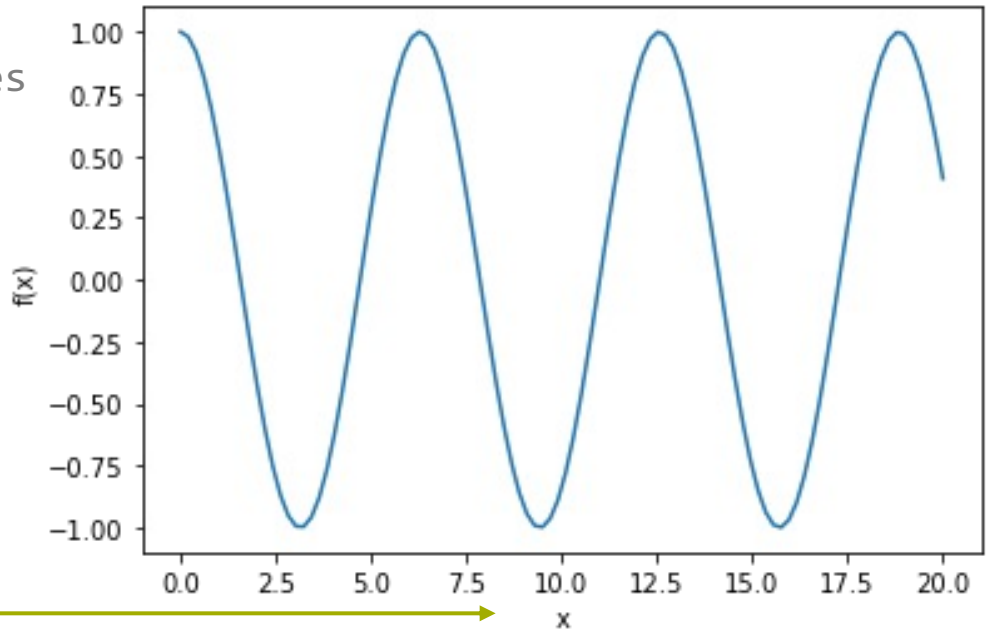
Line Plots

```
#Create some x values

x = np.linspace(0,20,100)

#plot cosine for the given x-values

pl.plot(x, np.cos(x))
```



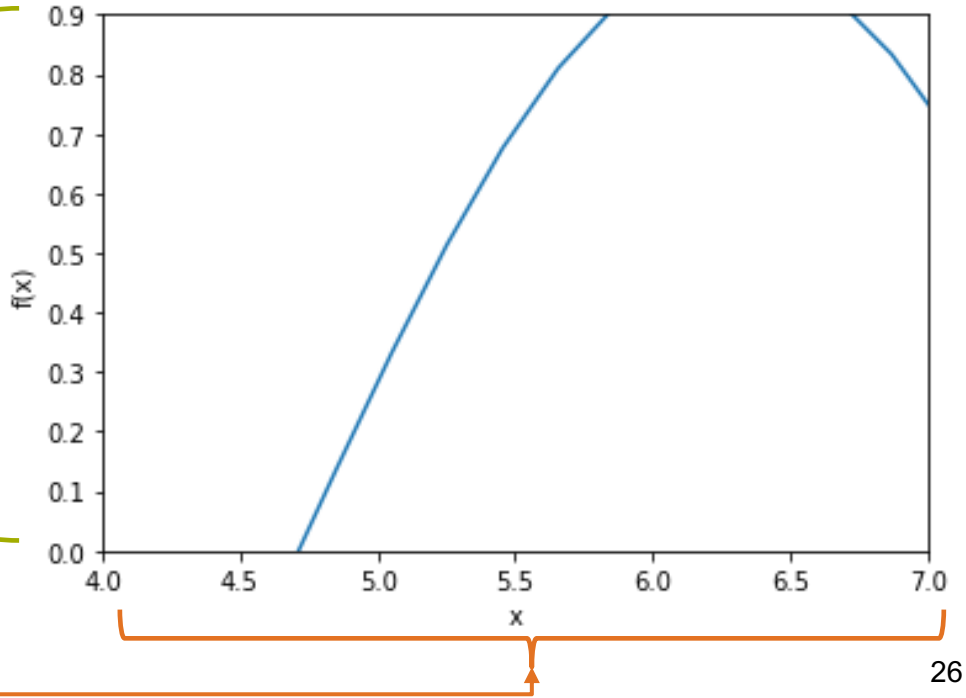
Line Plots

```
#Create some x values

x = np.linspace(0,20,100)

#plot cosine for the given x-values

pl.plot(x, np.cos(x))

#add labels

pl.ylabel("f(x)")

pl.xlabel("x")
```



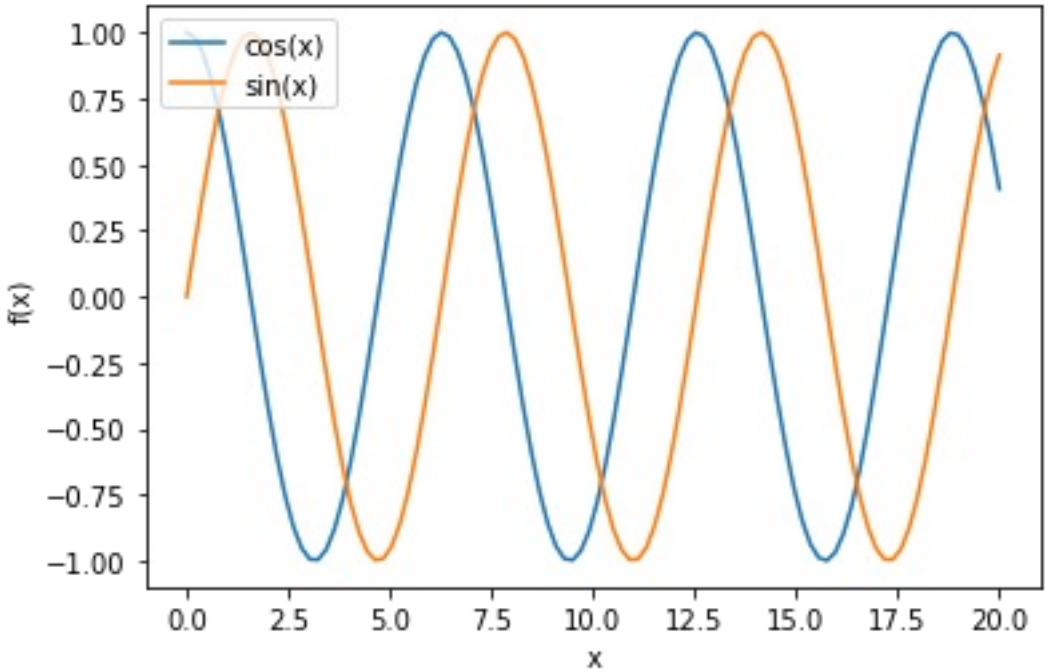
Line Plots

```
#Create some x values

x = np.linspace(0,20,100)

#plot cosine for the given x-values

pl.plot(x, np.cos(x))

#add labels

pl.ylabel("f(x)")

pl.xlabel("x")

#change axis range

pl.ylim([0,0.9])

pl.xlim([4,7])
```



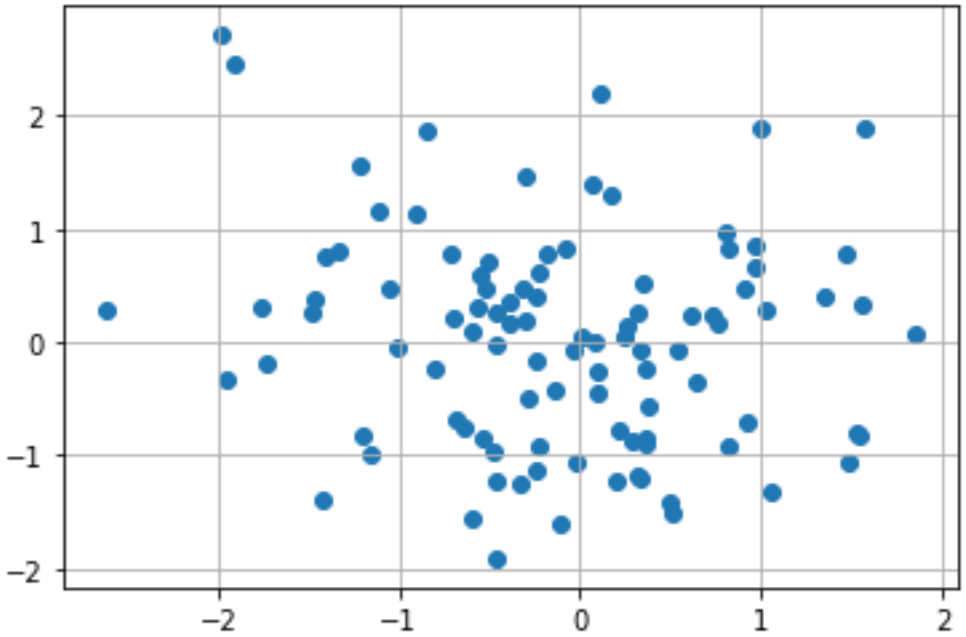
Several Line Plots

```
#Create some x values

x = np.linspace(0,20,100)

#plot cosine for the given x-values

pl.plot(x,np.cos(x),label="cos(x)")

pl.plot(x,np.sin(x),label="sin(x)")

#add labels

pl.ylabel("f(x)")

pl.xlabel("x")

#add legend

pl.legend(loc="upper left")
```



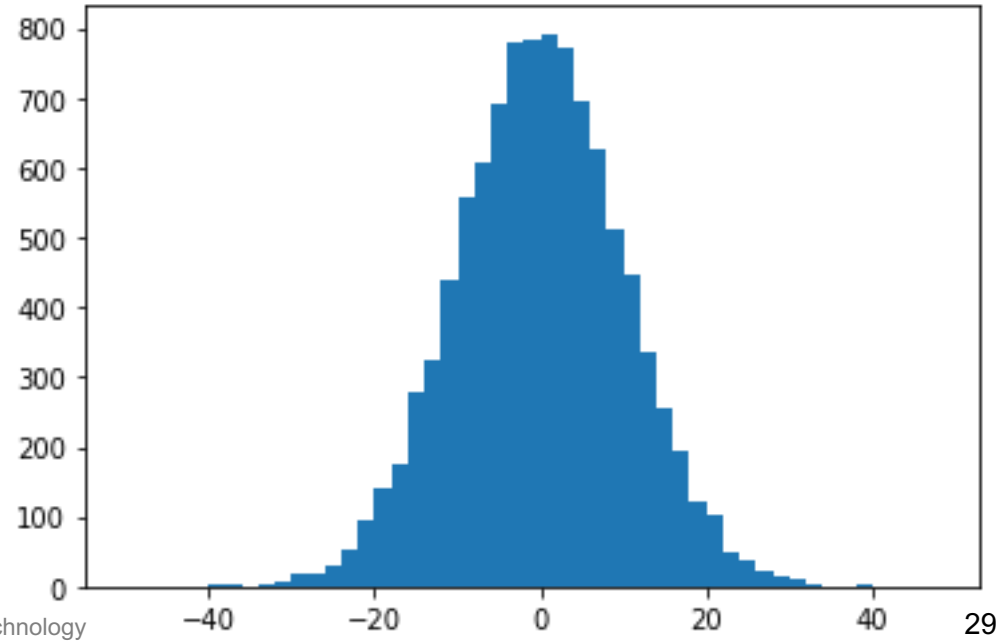
Scatter Plots

```
#Seed for random generator (Seeds are used to reproduce random numbers)

#Create samples from a standard normal distribution

np.random.seed(42)

x = np.random.randn(100)

y = np.random.randn(100)

pl.scatter(x,y)

#Add grid to plot

pl.grid()
```



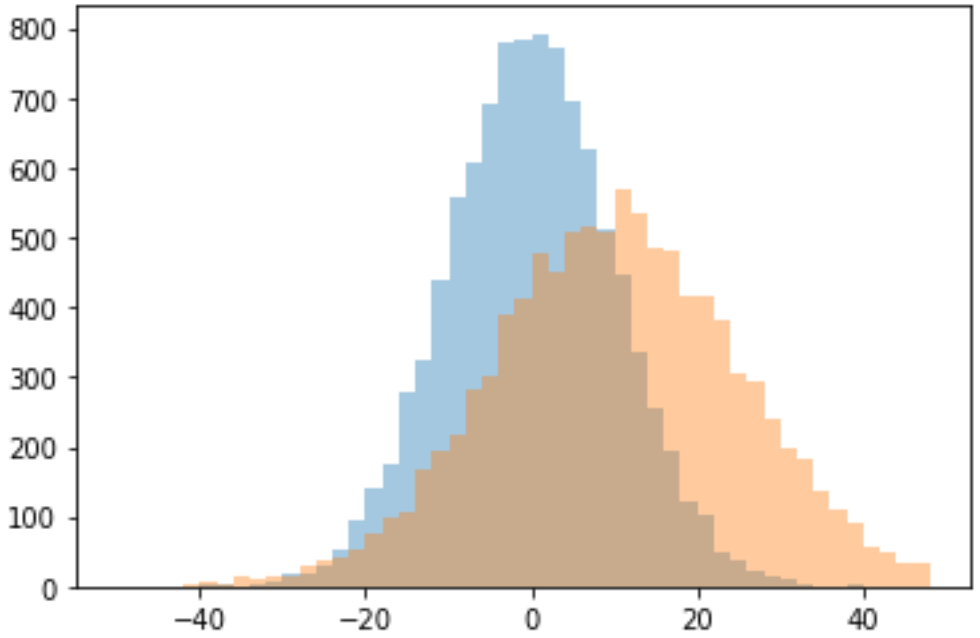
Histograms

```
#Seed for random generator (Seeds are used to reproduce random numbers)

#Create 10000 samples from a standard normal distribution with mean=0 and
std=10

np.random.seed(42)

x = np.random.normal(0,10,10000)

bins = np.arange(-50, 50, 2)

pl.hist(x, bins=bins)
```



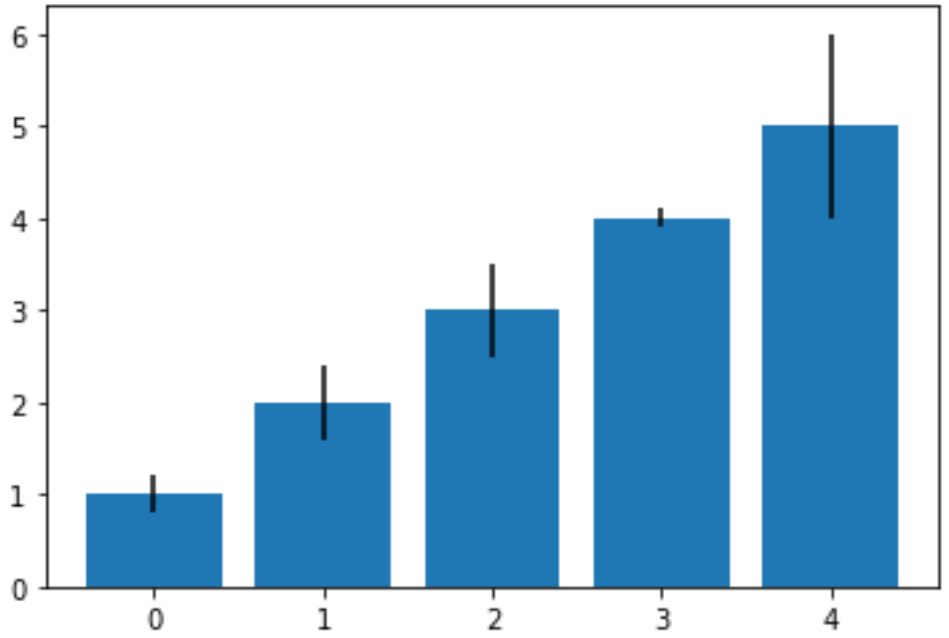
Histograms

```
np.random.seed(42)

x = np.random.normal(0,10,10000)

x2 = np.random.normal(10,15,10000)

bins = np.arange(-50, 50, 2)

pl.hist(x, bins=bins, alpha=0.4)

pl.hist(x2, bins=bins, alpha=0.4)
```



Bar Charts

```
measurments = [1, 2, 3, 4, 5]

stds = [0.2, 0.4, 0.5, 0.1, 1.0]

bar_labels = ['M1', 'M2', 'C1', 'C2', 'Rest']

# plot bars

x_pos = list(range(len(bar_labels)))

pl.bar(x_pos, measurments, yerr=stds)
```



Objectives: What should I know?

You should know, how to

- define NumPy arrays
- index, slice and mask NumPy arrays
- create simple plots

If you **do not** remember basic linear algebra, you should review the basics of vectors, matrices, dot product, transpose, inverse, and how to solve simple linear equations.

Thanks for your attention!

Homepage & Contact Details
Prof. Dr. Dominik Grimm
Petersgasse 18
Raum 00.027


dominik.grimm@tum.de


https://bit.cs.tum.de/


@dg_grimm