

Carsten Knoll

Chair of Fundamentals of Electrical Engineering

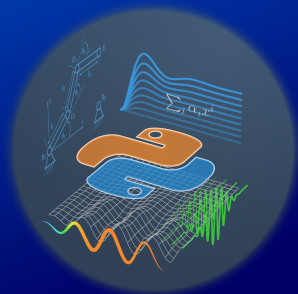
Python for Engineers

Pythonkurs für Ingenieur:innen

Data Processing and Analysis
Numerische Datenauswertung

Dresden (Online), 2024-11-22

<https://tu-dresden.de/pythonkurs>
<https://python-fuer-ingenieure.de>



Data Analysis: Overview

What do you do when you evaluate (measurement) data?

- load data
- select relevant data (or sections)
- determine new variables from existing ones
- increase information density
- visualize results

Data Analysis: Overview

What do you do when you evaluate (measurement) data?

- load data
- select relevant data (or sections)
- determine new variables from existing ones
- increase information density
- visualize results

Learning objectives:

- load / save data
- productive handling of Numpy arrays
- overview, which algorithms are already implemented
- overview on Pandas

Loading / Saving Data

```
import numpy as np

# ... do important calculations then save the result
result_array = np.arange(21).reshape(7, 3)

# same array but with automatic determination of column number
result_array2 = np.arange(21).reshape(7, -1)

np.save("result.npy", result_array) # binary format
np.savetxt("result.dat", result_array) # txt format

# ... in another file ...

array1 = np.load("result.npy") # binary format
array2 = np.loadtxt("result.dat") # txt format
```

∃ other possibilities (e.g. for *.wav files or Matlab format: see `scipy.io.*`).

Array Indexing

Basic indexing options of Numpy arrays (see also course03):

- integer numbers (`x[5]`) and “slices” (`x[3:10]`)

Array Indexing

Basic indexing options of Numpy arrays (see also course03):

- integer numbers (`x[5]`) and “slices” (`x[3:10]`)

Advanced indexing option 1: `int`-arrays

- can have arbitrary length; must contain integer values between $-n$ and $+n - 1$
- values are interpreted as indices; values can be repeated and omitted

```
x = np.array([10, 11, 12, 13])  
ids = np.array([1, 2, 2, 1, 0 -2])  
y = x[ids] # -> array([11, 12, 12, 11, 10, 12])
```

Array Indexing

Basic indexing options of Numpy arrays (see also course03):

- integer numbers (`x[5]`) and “slices” (`x[3:10]`)

Advanced indexing option 1: `int`-arrays

- can have arbitrary length; must contain integer values between $-n$ and $+n - 1$
- values are interpreted as indices; values can be repeated and omitted

```
x = np.array([10, 11, 12, 13])
ids = np.array([1, 2, 2, 1, 0 -2])
y = x[ids] # -> array([11, 12, 12, 11, 10, 12])
```

Advanced indexing option 2: `boolean` arrays

- length must be the same as indexed array (`x`)
- can only contain values `True` and `False`
- length of result equals number of `True`

```
ids = np.array([True, False, False, True])
x[ids] # -> [10, 13] (only first and last value were selected)

# negate all values that are less than 12:
x[x<12]*=-1 # -> [-10, -11, 12, 13]
```

Numerical Differentiation

- reminder: $\frac{df(x)}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\delta x}$ ("difference quotient")
- `numpy.diff` for each array element: calc *successor minus predecessor*.
- for approximation of the first derivative of a function: **you** have divide by Δx .
- `numpy.diff(x)` → result is one element shorter than input data `x`

```
import numpy as np
x = np.arange(20) # array([0, 1, 2, 3,...])
xd = np.diff(x) # array([1, 1, 1, ...])
```

- higher derivative orders also possible, see [doc](#)
- opposite operation: `np.cumsum(x)` (cumulative sum $\hat{=}$ discrete integral).

Useful Engineering Tools: Filter and FFT

- filtering (low pass, moving average, ...).

→ package: `scipy.signal`

- Representation of the frequency spectrum

→ package: `numpy.fft` (Fast Fourier Transform)

“most important algorithm” of the information age

both require quite some background knowledge (therefore not covered here)

Interpolation (1)

- Interpolation (generate intermediate values, change sampling rate, ...)

→ Paket: `scipy.interpolate`

Listing: example-code/01_interpolation.py

```
import numpy as np
import scipy.interpolate as ip
import matplotlib.pyplot as plt

# original data
x = [1, 2, 3, 4]
y = [2, 0, 1, 3]

plt.plot(x, y, "bx") # blue crosses
plt.savefig("interpolation0.pdf")

# create linear interpolator function
fnc1 = ip.interpld(x, y)

# achieve higher x-resolution by evaluation of fnc1
xx = np.linspace(1, 4, 20)
plt.plot(xx, fnc1(xx), "r.-") # red solid line and dots
plt.savefig("interpolation1.pdf")
plt.show()
```

Interpolation (1)

- Interpolation (generate intermediate values, change sampling rate, ...)

→ Paket: `scipy.interpolate`

Listing: example-code/01_interpolation.py

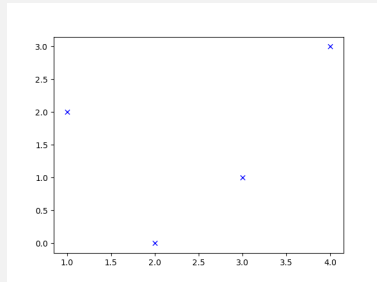
```
import numpy as np
import scipy.interpolate as ip
import matplotlib.pyplot as plt

# original data
x = [1, 2, 3, 4]
y = [2, 0, 1, 3]

plt.plot(x, y, "bx") # blue crosses
plt.savefig("interpolation0.pdf")

# create linear interpolator function
fnc1 = ip.interpld(x, y)

# achieve higher x-resolution by evaluation of fnc1
xx = np.linspace(1, 4, 20)
plt.plot(xx, fnc1(xx), "r.-") # red solid line and dots
plt.savefig("interpolation1.pdf")
plt.show()
```



Interpolation (1)

- Interpolation (generate intermediate values, change sampling rate, ...)

→ Paket: `scipy.interpolate`

Listing: example-code/01_interpolation.py

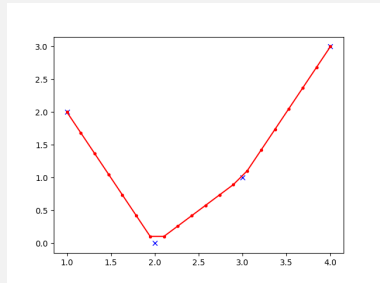
```
import numpy as np
import scipy.interpolate as ip
import matplotlib.pyplot as plt

# original data
x = [1, 2, 3, 4]
y = [2, 0, 1, 3]

plt.plot(x, y, "bx") # blue crosses
plt.savefig("interpolation0.pdf")

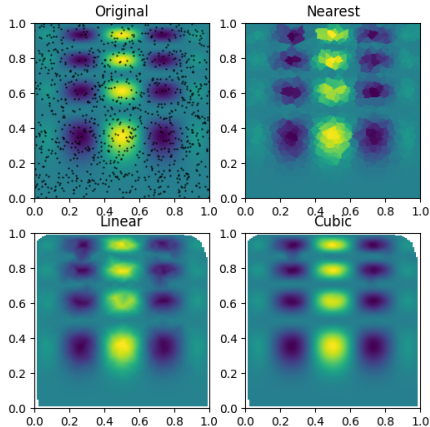
# create linear interpolator function
fnc1 = ip.interpld(x, y)

# achieve higher x-resolution by evaluation of fnc1
xx = np.linspace(1, 4, 20)
plt.plot(xx, fnc1(xx), "r.-") # red solid line and dots
plt.savefig("interpolation1.pdf")
plt.show()
```



Interpolation (2)

- Works also in higher dimensions with irregularly distributed input data (see `example_code/01b_griddata.py`, taken from [docs.scipy.org/...](https://docs.scipy.org/))



Regression (or “Fitting”)

→ `numpy.polyfit` and `numpy.polyval`

Listing: example-code/02_regression.py

```
import numpy as np
import matplotlib.pyplot as plt

N = 25
xx = np.linspace(0, 5, N)

# cool trick: two assignments in one line
m, n = 2, -1

# evaluate equation of straight line:  $y = m \cdot x + n$ 
yy = np.polyval([m, n], xx)
yy_noisy = yy + np.random.randn(N) # add some random noise

# create linear fit (regression 1st order polynomial)
mr, nr = np.polyfit(xx, yy_noisy, 1) # calculate fit
yyr = np.polyval([mr, nr], xx) # evaluate the function

plt.plot(xx, yy, 'go--', label="original")
plt.plot(xx, yy_noisy, 'k.', label="noisy data")
plt.plot(xx, yyr, 'r-', label="regression")
plt.legend()
plt.savefig("regression.png")

plt.show()
```

Regression (or “Fitting”)

→ `numpy.polyfit` and `numpy.polyval`

Listing: example-code/02_regression.py

```
import numpy as np
import matplotlib.pyplot as plt

N = 25
xx = np.linspace(0, 5, N)

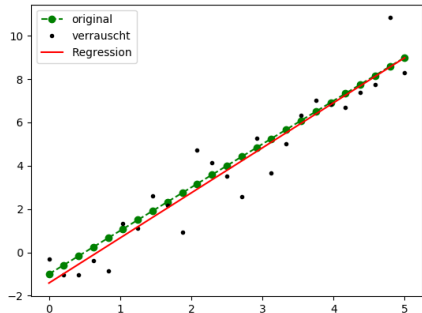
# cool trick: two assignments in one line
m, n = 2, -1

# evaluate equation of straight line:  $y = m \cdot x + n$ 
yy = np.polyval([m, n], xx)
yy_noisy = yy + np.random.randn(N) # add some random noise

# create linear fit (regression 1st order polynomial)
mr, nr = np.polyfit(xx, yy_noisy, 1) # calculate fit
yyr = np.polyval([mr, nr], xx) # evaluate the function

plt.plot(xx, yy, 'go--', label="original")
plt.plot(xx, yy_noisy, 'k.', label="noisy data")
plt.plot(xx, yyr, 'r-', label="regression")
plt.legend()
plt.savefig("regression.png")

plt.show()
```



Regression (or “Fitting”)

→ `numpy.polyfit` and `numpy.polyval`

Listing: example-code/02_regression.py

```
import numpy as np
import matplotlib.pyplot as plt

N = 25
xx = np.linspace(0, 5, N)

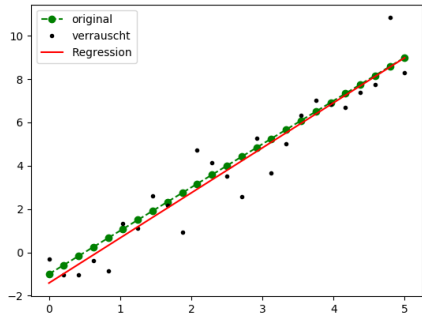
# cool trick: two assignments in one line
m, n = 2, -1

# evaluate equation of straight line:  $y = m \cdot x + n$ 
yy = np.polyval([m, n], xx)
yy_noisy = yy + np.random.randn(N) # add some random noise

# create linear fit (regression 1st order polynomial)
mr, nr = np.polyfit(xx, yy_noisy, 1) # calculate fit
yyr = np.polyval([mr, nr], xx) # evaluate the function

plt.plot(xx, yy, 'go--', label="original")
plt.plot(xx, yy_noisy, 'k.', label="noisy data")
plt.plot(xx, yyr, 'r-', label="regression")
plt.legend()
plt.savefig("regression.png")

plt.show()
```



- higher polynomial orders also possible

“Machine Learning” (ML)

- widely used since 2010's; very successful on some tasks
- can also be understood as **function approximation**
- three important subareas
 - **supervised learning**
 - classification (dog or cat? Mozart or Helene Fischer?)
 - regression (How well will movie X please person Y?)
 - **unsupervised learning** (= automatic cluster detection)
 - **Reinforcing learning** (agent adapts responses to environment to maximize reward)

Python is (arguably) the most important languages in ML:

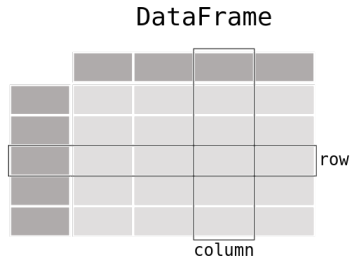
Subjective Link Selection:

- <https://scikit-learn.org> (neural networks, Gaussian processes, and many more).
- <https://pytorch.org> (neural networks)

The Pandas Package

→ **"spreadsheet processing with Python"**

- most important package for "Data Science"
- based on Numpy
- most important data type: `pandas.DataFrame`
 - models a table
 - columns can have names
 - columns can have different data types
- second most important data type: `pandas.Series`
 - models row or column



extensive documentation: <https://pandas.pydata.org/docs/>

Pandas: Create Data Frames

For following code examples see also: `notebooks/pandas_demo.ipynb`

Listing: 03_pandas.py

```
1 import pandas as pd
2 import numpy as np
3
4 # create df from a numpy array:
5 arr = np.random.randn(6, 4)
6 df1 = pd.DataFrame(arr, columns=list("ABCD"))
7
8 # create df from a dict
9 shop_articles = {
10     "weight": [10.1, 5.0, 8.3, 7.2],
11     "color": ["red", "green", "blue", "transparent"],
12     "availability": [False, True, True, False],
13     "price": 8.99 # all have the same price
14 }
15 article_numbers = ["A107", "A108", "A109", "A110"]
16 df2 = pd.DataFrame(shop_articles, index=article_numbers)
17
18 # each has its own data type
19 print("column types:\n", df2.dtypes)showspaces
```

Pandas: Save and Load Data

Listing: 03_pandas.py

```
25 fname = "things.csv"
26
27 # save the data frame as CSV file (Comma Separated Values )
28 # csv file will also contain header information (column labels)
29 df2.to_csv(fname)
30
31
32
33 # Pandas function to load csv-data into DataFrame
34 # (Detects column names automatically)
35 df2_new = pd.read_csv(fname)
36
37 # display(df2_new) # Jupyter-Notebook-specific
```

Pandas: Read/Write Access to Cells

Listing: 03_pandas.py

```
42
43 # access individual values (by verbose index and column):
44 print(df2.loc["A108", "weight"])
45 df2.loc["A108", "weight"] = 3.4 # set new value
46 df2.loc["A108", "weight"] += 2 # increase by two
47
48 # access by numerical indices
49 print(df2.iloc[1, 0]) # row index: 1, column index: 0
50
51 # use slices to change multiple values
52 df2.loc["A108":"A109", "price"] *= 0.30 # 30% discount
53
54 # access multiple columns (-> new df object)
55 print(df2[["price", "weight"]])
56
57 # new column (-> provide a height value for every article)
58 df2["height"] = [10, 20, 30, 40]
59
60 # new row (-> provide a value for every column (weight, color, ...) )
61 df2.loc["X400"] = [15 , "purple" , True , 25.00 , 50]
```

Pandas: Select Data by Boolean Indexing

Listing: 03_pandas.py

```
65 # create Series-object with bool-entries
66 idcs = df2["weight"] > 8
67
68 # use this Series-object for indexing
69 print(df2[idcs])
70
71 # similar statement without intermediate variable:
72 print(df2[df2["weight"] < 8])
```

Pandas: Apply Functions

Listing: 03_pandas.py

```
75 df2.describe()
76 df2["price"].mean()
77 df2["weight"].median()
78 df2["weight"].max()
79
80 print("shorthand notation (if column label is valid python name)")
81 print(df2.weight == df2["weight"])
82 print(all(df2.weight == df2["weight"]))
83
84 # combine function application with boolean indexing
85 df2[df2.weight>8].weight.mean()
86
87 # apply an arbitrary function (here np.diff) to each (selected) column
88 print(df2[["price", "weight"]].apply(np.diff))
```

Summary

- Access to data (`np.load` , `pd.read_csv`)
- Selection (integer indexing, boolean indexing of both arrays and data frames)
- Interpolation
- Regression
- Pandas: DataFrames, r/w cells, rows, columns, boolean indexing, functions
- See docs for more details (links above)