TECHNISCHE UNIVERSITÄT DRESDEN

PYTHONKURS FÜR INGENIEUR:INNEN

Numerische Datenauswertung

Folien: Carsten Knoll

Dresden, WiSe 2020/21

Datenauswertung

Was macht man, wenn man (Mess-) Daten auswertet?

- Daten laden
- Relevante Daten (bzw. Abschnitte) heraussuchen
- Aus vorhandenen Größen neue bestimmen
- Informationsdichte erhöhen
- Grafisch darstellen

Übersicht

- Lernziele:
  - Daten Laden / Speichern
  - Produktiver Umgang mit Numpy Arrays
  - Überblick, welche Algorithmen schon implementiert sind

Daten Laden / Speichern

```python
import numpy as np

array1  = np.load("file1.npy") # Binaerformat
array2  = np.loadtxt("file2.dat") # TXT-Format

# Rechnungen

np.save("ergebnis.bin", result)
```

Weitere Möglichkeiten (z.B. für `*.wav`-Dateien oder Matlab-Format: siehe `scipy.io.*`)

Array-Indizierung

Indizierungsmöglichkeiten von Numpy-Arrays (Wdh.):

- `int`-Zahlen

Array-Indizierung

Indizierungsmöglichkeiten von Numpy-Arrays (Wdh.):

- `int`-Zahlen
- “Slices” (`x[1:10]`)

Array-Indizierung

Indizierungsmöglichkeiten von Numpy-Arrays (Wdh.):

- `int`-Zahlen
- “Slices” (`x[1:10]`)
- `int`-arrays

```
x = np.array([7.5, 8.1, 12.4])
indices = np.array([1, 2, 2, 1, 0])
y = x[indices] # -> array([8.1, 12.4, 12.4, 8.1, 7.5])
```

Array-Indizierung

Indizierungsmöglichkeiten von Numpy-Arrays (Wdh.):

- `int`-Zahlen
- “Slices” (`x[1:10]`)
- `int`-arrays

```
x = np.array([7.5, 8.1, 12.4])
indices = np.array([1, 2, 2, 1, 0])
y = x[indices] # -> array([8.1, 12.4, 12.4, 8.1, 7.5])
```

- `boolean`-arrays

```
x = np.arange(20)

# Alle Werte die kleiner 10 sind negieren:

x[x<10]*=-1
```

Numerisches Differenzieren

- Für jedes Array-Element: Nachfolger *minus* Vorgänger
- Für Näherung der ersten Ableitung einer Funktion: noch durch Δx teilen

→ „Differenzenquozient“: $\frac{df(x)}{dx} \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$

- Ergebnis ist um eins kürzer als Eingangsdaten

```
x = np.arange(20) # [0, 1, 2, ...]
xd = np.diff(x) # [1, 1, 1, ...]
```

- Auch höhere Ableitungsordnungen möglich, siehe Doku

- Gegenteilige Operation: `np.cumsum(x)` (Kummulative Summe $\hat{=}$ diskretes Integral)

Häufige gebrauchte Tools: Filter und FFT

- Filterung (Tiefpass, Gleitender Mittelwert, ...)

→ Paket: scipy.signal

- Darstellung des Frequenzspektrums

→ Paket: numpy.fft (Fast Fourier Transform)

Beides erfordert Hintergrundwissen (darum hier nicht Thema)

Interpolation (1)

- Interpolation (Zwischenwerte ergänzen, Abtastrate ändern, ...)

→ Paket: scipy.interpolate

Listing: example-code/interpolation1.py

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

# Ausgangs-Daten
x = [1, 2, 3, 4]
y = [2, 0, 1, 3]

plt.plot(x, y, "bx")  # Blaue Kreuze
plt.savefig("interpolation0.pdf")

# Interpolator-Funktion erstellen (linear)
fnc1 = ip.interp1d(x, y)

# Höherer x-Auflösung erreichen durch Auswertung von fnc1
xx = np.linspace(1, 4, 20)
plt.plot(xx, fnc1(xx), "r.-")  # rote Punkte, solide Linie
plt.savefig("interpolation1.pdf")
plt.show()
```

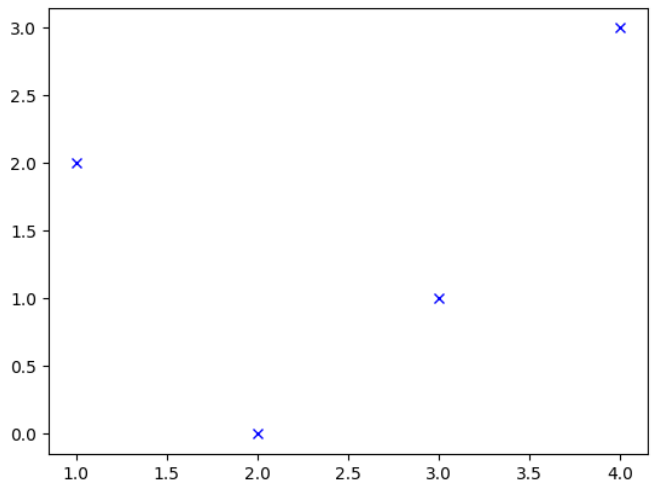
Interpolation (1)

- Interpolation (Zwischenwerte ergänzen, Abtastrate ändern, ...)

→ Paket: scipy.interpolate

Listing: example-code/interpolation1.py

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

# Ausgangs-Daten
x = [1, 2, 3, 4]
y = [2, 0, 1, 3]

plt.plot(x, y, "bx")  # Blaue Kreuze
plt.savefig("interpolation0.pdf")

# Interpolator-Funktion erstellen (linear)
fnc1 = ip.interp1d(x, y)

# Höherer x-Auflösung erreichen durch Auswertung von fnc1
xx = np.linspace(1, 4, 20)
plt.plot(xx, fnc1(xx), "r.-")  # rote Punkte, solide Linie
plt.savefig("interpolation1.pdf")
plt.show()
```



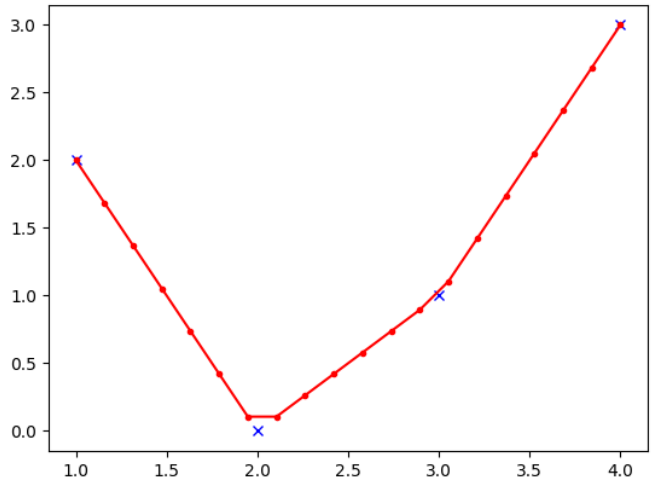
Interpolation (1)

- Interpolation (Zwischenwerte ergänzen, Abtastrate ändern, ...)

→ Paket: scipy.interpolate

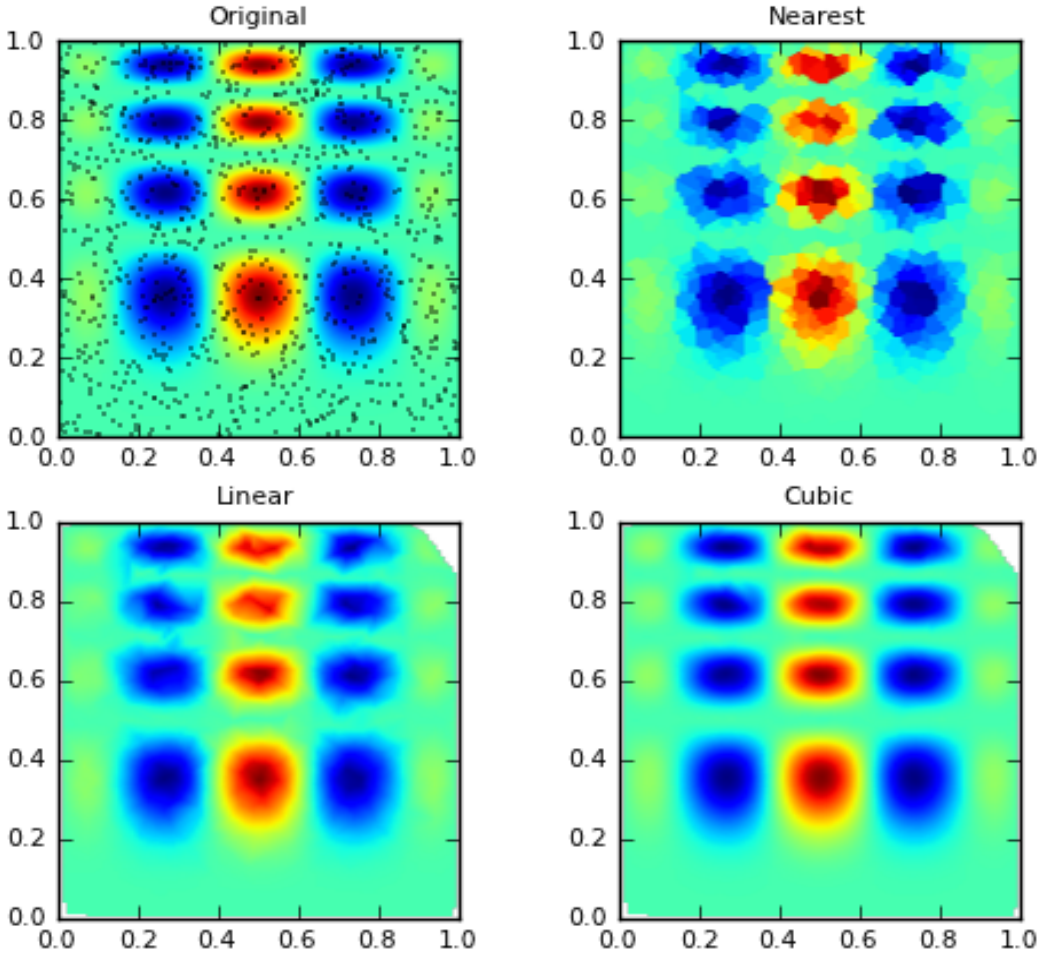
Listing: example-code/interpolation1.py

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

# Ausgangs-Daten
x = [1, 2, 3, 4]
y = [2, 0, 1, 3]

plt.plot(x, y, "bx")  # Blaue Kreuze
plt.savefig("interpolation0.pdf")

# Interpolator-Funktion erstellen (linear)
fnc1 = ip.interp1d(x, y)

# Höherer x-Auflösung erreichen durch Auswertung von fnc1
xx = np.linspace(1, 4, 20)
plt.plot(xx, fnc1(xx), "r.-")  # rote Punkte, solide Linie
plt.savefig("interpolation1.pdf")
plt.show()
```



Interpolation (2)

- Geht auch N-dimensional mit irregulär verteilten Eingangsdaten



Regression (bzw. „Fit“)

→ `numpy.polyfit` und `numpy.polyval`

Listing: example-code/regression1.py

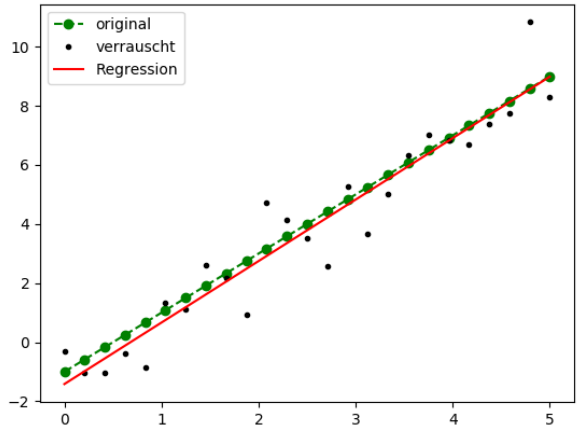
```
import numpy as np
import matplotlib.pyplot as plt

N = 25
xx = np.linspace(0, 5, N)
m, n = 2 , -1  # Doppelzuweisung


# Geradengleichung y = m*x + n auswerten
yy = np.polyval([m, n], xx)
yy_noisy = yy + np.random.randn(N)  # Rauschen addieren

# Regressionsgerade
mr, nr = np.polyfit(xx, yy_noisy, 1)  # fitten
yyr = np.polyval([mr, nr], xx)  # auswerten

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

plt.show()
```

Regression (bzw. „Fit“)

→ `numpy.polyfit` und `numpy.polyval`

Listing: example-code/regression1.py

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

N = 25
xx = np.linspace(0, 5, N)
m, n = 2 , -1  # Doppelzuweisung


# Geradengleichung y = m*x + n auswerten
yy = np.polyval([m, n], xx)
yy_noisy = yy + np.random.randn(N)  # Rauschen addieren

# Regressionsgerade
mr, nr = np.polyfit(xx, yy_noisy, 1)  # fitten
yyr = np.polyval([mr, nr], xx)  # auswerten

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

plt.show()
```

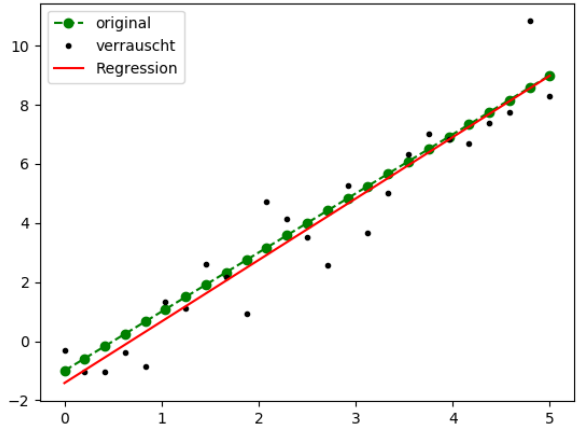


Regression (bzw. „Fit“)

→ `numpy.polyfit` und `numpy.polyval`

Listing: example-code/regression1.py

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

N = 25
xx = np.linspace(0, 5, N)
m, n = 2 , -1  # Doppelzuweisung


# Geradengleichung y = m*x + n auswerten
yy = np.polyval([m, n], xx)
yy_noisy = yy + np.random.randn(N)  # Rauschen addieren

# Regressionsgerade
mr, nr = np.polyfit(xx, yy_noisy, 1)  # fitten
yyr = np.polyval([mr, nr], xx)  # auswerten

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

plt.show()
```



- Höhere Polynomordnungen auch möglich

„Maschinelles Lernen“ (ML)

- Seit einigen Jahren sehr weit verbreitet und teilweise sehr erfolgreich
- In vielen Fällen ist ML eine wohlklingende Umschreibung für **Funktionsapproximation**
- Drei wichtige Teilbereiche
  - **Überwachtes Lernen**
    - Klassifikation (Hund oder Katze? Mozart oder Helene Fischer?)
    - Regression (Wie gut wird Film X der Person Y gefallen?)
  - **Unüberwachtes Lernen** (= automatische Clustererkennung)
  - **Bestärkendes Lernen** (Agent passt Reaktionen auf Umwelt an um Belohnung zu maximieren)

Python ist eine der wichtigsten Sprachen im Bereich ML:
Subjektive Auswahl an Links:

- `https://scikit-learn.org/stable/` (Neuronale Netze, Gauß-Prozesse, u.v.m.)
- `https://github.com/keras-team/keras/` (Neuronale Netze)
- `https://pytorch.org/` (Neuronale Netze)

Zusammenfassung

- Zugriff auf Daten
- Auswahl
- Interpolation
- Regression
- Links