# Python Einführungskurs

## Einführung in die Numerik
## Sommersemester 2022

28. April 2022

# 3. Übung

## numpy, pyplot, scipy

# NumPy

- Arbeiten mit Matrizen und Vektoren
- Lineare Algebra, Fourier-Transformation
- Effiziente Operationen auf Matrizen

```
>>> import numpy as np
>>> a = np.array([1,2,4,-0.9])
>>> print(a)
>>> a[2:4]
>>> a[1] = -2
>>> M = np.array([[1,2,3],[4,5,6]])
>>> print(M)
```

Matrizen als Listen von Zeilenvektoren

# Spezielle Matrizen

## Erzeugung spezieller Matrizen

- `np.zeros(shape)` : Null-array
- `np.ones(shape)` : array bestehend aus Einsen
- `np.eye(N)` : Einheits-array
- `np.arange(N)` : Vektor von 1 bis N

```
>>> np.zeros(2)
>>> np.zeros((2,3))
>>> np.eye(2)
```

# Hilfe im Internet

- Bereits gesehen: `help(np.eye)`
- Online-Dokumentation [1]
  - Auflistung aller Optionen einer Funktion
- Andere Online-Ressourcen:
  - tutorial, quickstart, intro to ...
  - Fehler googlen
  - `https://stackoverflow.com/`

[1] `https://numpy.org/doc/stable/reference/generated/numpy.eye.html`

# array-Methoden

- Viele vorprogrammierte Methoden
- Tippe `A.` TAB, um alle anzuzeigen

```
>>> A = np.array([[1,2,3],[4,5,6])
>>> A.shape
>>> A.sum()
>>> A.min()
```

# Matrix-Multiplikation

- `A@B` oder `np.matmul(A,B)` gewöhnl. Multiplikation
  - Matrix*Matrix oder Matrix*Vektor
- `np.dot(A,B)` gewöhnl. Multiplikation
  - Matrix*Matrix oder Matrix*Vektor oder Skalar*Matrix
  - bei komplexen Matrizen wird `A` kompl.konj.
- `A*B` oder `np.multiply(A,B)` elementweise Multiplikation

Bei 3-dimensionalen Arrays unterscheiden sich `np.matmul` und `np.dot` vom Verhalten

# NumPy Funktionen

- `+`, `-`, `*`, `/`, `**`: Elementweise Add., Subtr., Mult., Div., Potenz
- `<`, `<=`, `>`, `>=`, `==`, `!=`: Elementweise logische Vergleiche
- `np.sin()`, `np.exp()`, ...: Elemtw. Sinus, Exponentialfunktion, ...

Dimensionen zweier arrays muss gleich sein.

# Übungen

Mit Hilfe der Referenz [2] lösen

1. Erstelle ein `np.array` $A$ mit shape `(3,3)`
2. Berechne elementweise den **sin** von $A$
3. Berechne das Minimum und die Position des Minimums der Matrix $A$
4. Berechne die Spaltensummen von $A$
5. Berechne (elementweise) die aufgerundete Wurzel der Einträge von $A$

[2] `https://numpy.org/doc/stable/reference/routines.math.html`

# np.linalg

- `np.linalg.solve(A,b)` : Löse LGS
- `np.linalg.det(A)` : Determinante
- `np.linalg.matrix_rank(A)` : Rang
- `np.linalg.inv(A)` : Inverse
- `np.linalg.svd(A)` : Singulärwertzerlegung
- `np.linalg.eig(A)` : Eigenwerte, -vektoren

und vieles mehr...

# Nützliche Funktionen

- `np.linspace(a,b,num=50)` gibt array von äquidistanten Stützpunkten
- `A.reshape(shape)` bringt `A` in die vorgegebene shape

```
>>> A = np.arange(15).reshape(3,5)
>>> B = np.linspace(2,5).reshape(-1,5)
>>> C = np.linspace(2,5,num=15).reshape(-1,5)
```

# Arrays kombininieren

- `np.hstack(a,b)` kombiniert arrays horizontal
- `np.vstack(a,b)` kombiniert arrays vertikal
- `np.concatenate((a,b),axis=0)` kombiniert arrays entlang der übergebenen Achse

```
>>> # A, C wie auf der vorherigen folie
>>> D = np.hstack((A,C))
>>> E = np.vstack((A,C))
>>> F = np.concatenate((A,C),axis=0)
```

# matplotlib

- Package zum Plotten von Daten
- Sehr vielseitig, hier nur eine kleine Auswahl

# Plotte Punkte

```
import matplotlib.pyplot as plt
```

- `plt.plot(x,y)`
  - `x,y` sind `np.array`s
  - Punkte $(x(i), y(i))$ werden geplottet
  - `label='name'` für Beschriftung
  - `color='red'` für manuelle Farbwahl
- Beispiel: `simple_plot.py`

# Logarithmische Plots

- `plt.loglog(x,y)` logarithmisch in `x,y`
- `plt.semilogx(x,y)` logarithmisch in `x`, linear in `y`
- `plt.semilogy(x,y)` logarithmisch in `y`, linear in `x`

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 6, 100)
y = np.exp(x)
plt.semilogy(x, y)
plt.show()
```

# Linien-Stil

Verschiedene Linienstile:

- `'solid'`
- `'dashed'`
- `'dotted'`
- `'dashdot'`

Verwendung:

```
>>> plt.plot(x,y,linestyle='dashed')
```

# Outlook

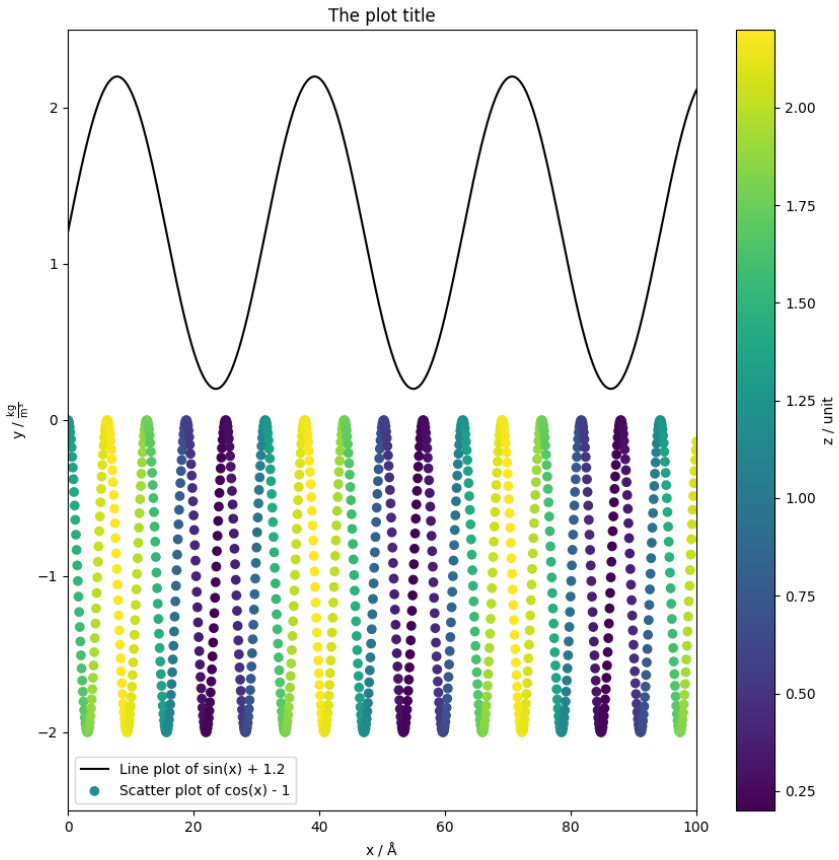


Abbildung: Komplizierter plot, generiert durch `advanced_plot.py`

# scipy

- Lösen von Optimierungsaufgaben
- Interpolation
- Integration
- sparse Matrizen
- gewöhnliche Differentialgleichungen

# Andere nützliche packages

- `time` zum Messen der Programmlaufzeit
- `tensorflow` für neuronale Netzwerke
- `os` für Dateizugriff, u.Ä.

# Danke für Eure Aufmerksamkeit