SciPy & Pandas

Felix Maurer

Universität Hamburg

20. Mai 2019

Inhaltsverzeichnis

SciPy: Übersicht

- Python-Modul: erweitert Python um komplexere mathematische Funktionen
  - Fokus: Wissenschaftliches Rechnen, Visualisierung
  - Teil des SciPy Stacks
  - Erweiternd auf NumPy aufgebaut
- Open Source
- Kostenlos
- Installation: `pip install scipy`

Scipy: Wichtige Submodule

Ich beschränke mich heute auf:

- `scipy.integrate`
- `scipy.linalg`
- `scipy.interpolate`

SciPy: Wichtige Submodule: scipy.integrate

quad: `scipy.integrate.quad(func, a, b)`

- func: Zu integrierende Funktion
- a: Anfang des Integrals
- b: Ende des Integrals

Beispiel: $\int_0^5 x^2 dx$

```
In [15]: import scipy.integrate

In [13]: scipy.integrate.quad(lambda x: x**2,0,5)
Out[13]: (41.66666666666666, 4.625929269271485e-13)
```

SciPy: Wichtige Submodule: scipy.integrate (2)

dblquad: `scipy.integrate.dblquad(func, a, b, gfun, hfun)`

- func: Zu integrierende Funktion
- a: Anfang des äußeren Integrals
- b: Ende des äußeren Integrals
- gfun: Funktion, die unteres Ende des inneren Integrals beschreibt
- hfun: Funktion, die oberes Ende des inneren Integrals beschreibt

Beispiel: $\int_{y=0}^{1} \int_{x=0}^{2y} (x+y) dx dy$

```
In [15]: import scipy.integrate

In [17]: scipy.integrate.dblquad(lambda x,y: x+y,0,1, lambda x: 0, lambda x: 2*x)
Out[17]: (1.3333333333333335, 4.421626967161769e-14)
```

SciPy: Wichtige Submodule: scipy.linalg

inv: `scipy.linalg.inv(A)`

- A: Quadratische Matrix

Beispiel:

$$\text{Sei } A = \begin{pmatrix} 4 & 2 \\ 3 & 1 \end{pmatrix}. \text{ Was ist } A^{-1}? \ (A \cdot A^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix})$$

```
In [50]: import scipy.linalg

In [51]: A= np.array([[4,2],[3,1]])
         print(A)

         [[4 2]
          [3 1]]

In [52]: scipy.linalg.inv(A)

Out[52]: array([[-0.5,  1. ],
                [ 1.5, -2. ]])
```

SciPy: Wichtige Submodule: scipy.linalg (2)

det: `scipy.linalg.det(A)`

- A: Quadratische Matrix

Beispiel: $det \begin{pmatrix} 4 & 2 \\ 3 & 1 \end{pmatrix}$

```
In [50]: import scipy.linalg

In [51]: A= np.array([[4,2],[3,1]])
         print(A)

         [[4 2]
          [3 1]]

In [53]: scipy.linalg.det(A)

Out[53]: -2.0
```

SciPy: Wichtige Submodule: scipy.linalg (3)

det: `scipy.linalg.solve(A, b)`

- A: Quadratische Matrix eines Gleichungssystems
- b: Rechte Seite der Gleichung

Bestimmt x eines Gleichungssystem der Form $A \times x = b$

Beispiel: $\begin{pmatrix} 4 & 2 \\ 3 & 1 \end{pmatrix} \times x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

```
In [50]: import scipy.linalg

In [58]: A= np.array([[4,2],[3,1]])
         b= np.array([[1],[2]])
         print(A)
         print('.....')
         print(b)

         [[4 2]
          [3 1]]
         .....
         [[1]
          [2]]

In [59]: scipy.linalg.solve(A,b)

Out[59]: array([[ 1.5],
                [-2.5]])
```

SciPy: Wichtige Submodule: scipy.interpolate

interp1d: `scipy.interpolate.interp1d(x, y, kind)`

- x: Werte aus Definitionsmenge
- y: Zugehörige Werte aus Zielmenge
- kind: Art der Annäherung (linear, quadratisch, kubisch etc.)

SciPy: Wichtige Submodule: scipy.interpolate (Beispiel 1)

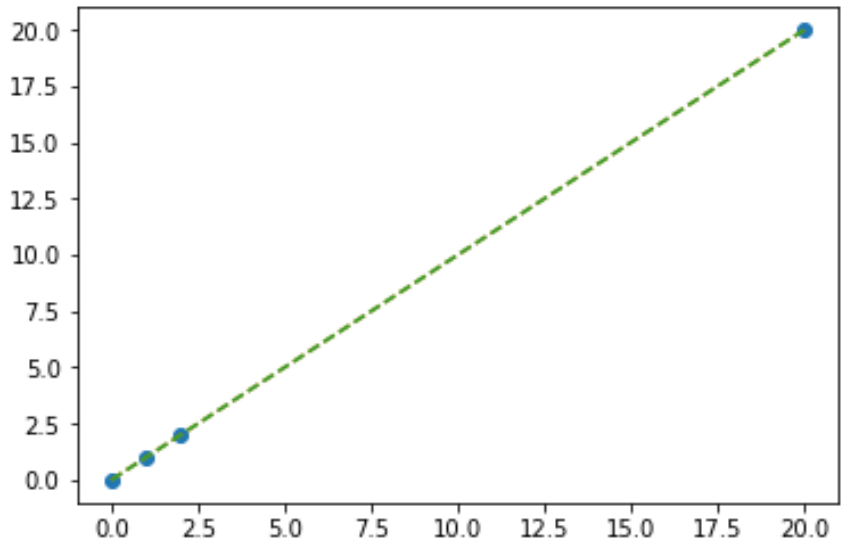
Beispiel 1:

x	y
0	0
1	1
2	2
20	20

In [87]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate
```

In [154]:
```
x = [0,1,2,20]
y = [0,1,2,20]
f = interp1d(x, y, kind='linear')
f2 = interp1d(x, y, kind='quadratic') #in diesem Fall identisch
```

In [155]:
```
xnew = np.linspace(0, 20, num=20)
plt.plot(x, y, 'o', xnew, f(xnew), '--', xnew, f2(xnew), '--')
plt.show()
```



SciPy: Wichtige Submodule: scipy.interpolate (Beispiel 2)

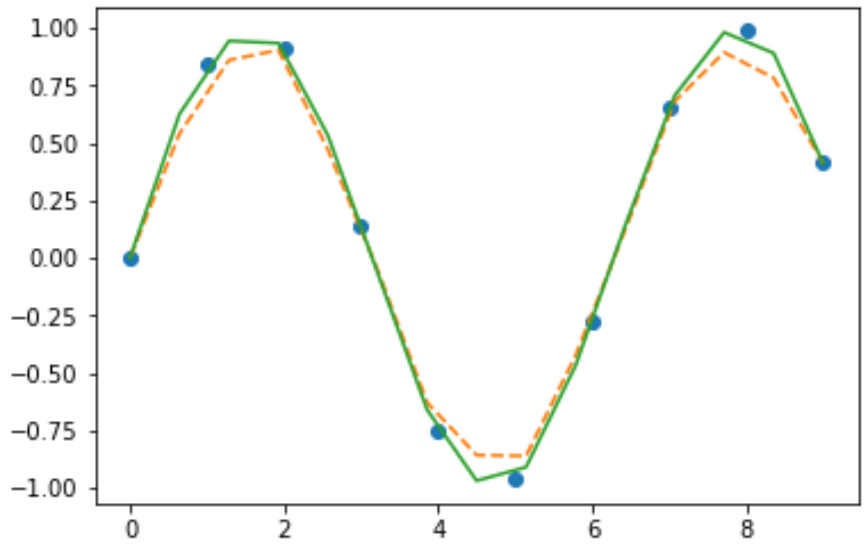
Beispiel 2:

x	y
0	0
1	0.841
2	0.909
3	0.141
4	-0.757
5	-0.959
6	-0.279
7	0.657
8	0.989
9	0.412

In [233]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate
```

In [239]:
```
x = [0,1,2,3,4,5,6,7,8,9]
y = [0.0, 0.841, 0.909, 0.141, -0.757,
     -0.959, -0.279, 0.657, 0.989, 0.412]

f = interp1d(x, y, kind='linear')
f2 = interp1d(x, y, kind='quadratic')
```

In [240]:
```
xnew = np.linspace(0, 9, num=15)
plt.plot(x, y, 'o', xnew, f(xnew), '--', xnew, f2(xnew), '-')
plt.show()
```



SciPy: Wichtige Submodule: scipy.interpolate (Beispiel 2)

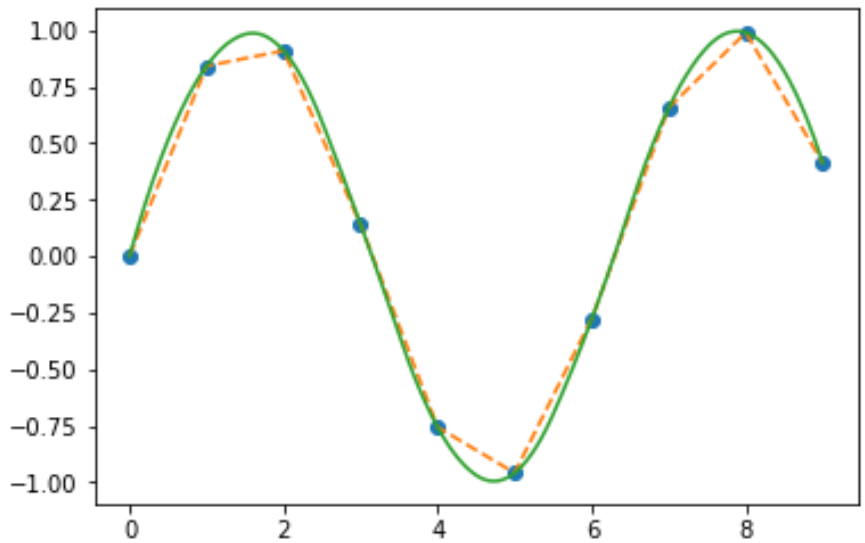
Beispiel 2:

x	y
0	0
1	0.841
2	0.909
3	0.141
4	-0.757
5	-0.959
6	-0.279
7	0.657
8	0.989
9	0.412

In [233]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate
```

In [239]:
```
x = [0,1,2,3,4,5,6,7,8,9]
y = [0.0, 0.841, 0.909, 0.141, -0.757,
     -0.959, -0.279, 0.657, 0.989, 0.412]

f = interp1d(x, y, kind='linear')
f2 = interp1d(x, y, kind='quadratic')
```

In [242]:
```
xnew = np.linspace(0, 9, num=1000)
plt.plot(x, y, 'o', xnew, f(xnew), '--', xnew, f2(xnew), '-')
plt.show()
```



Vergleich: SciPy

Alternative: MATLAB

- Anwendung zur Visualisierung und Verarbeitung von Daten
- eigene Programmiersprache

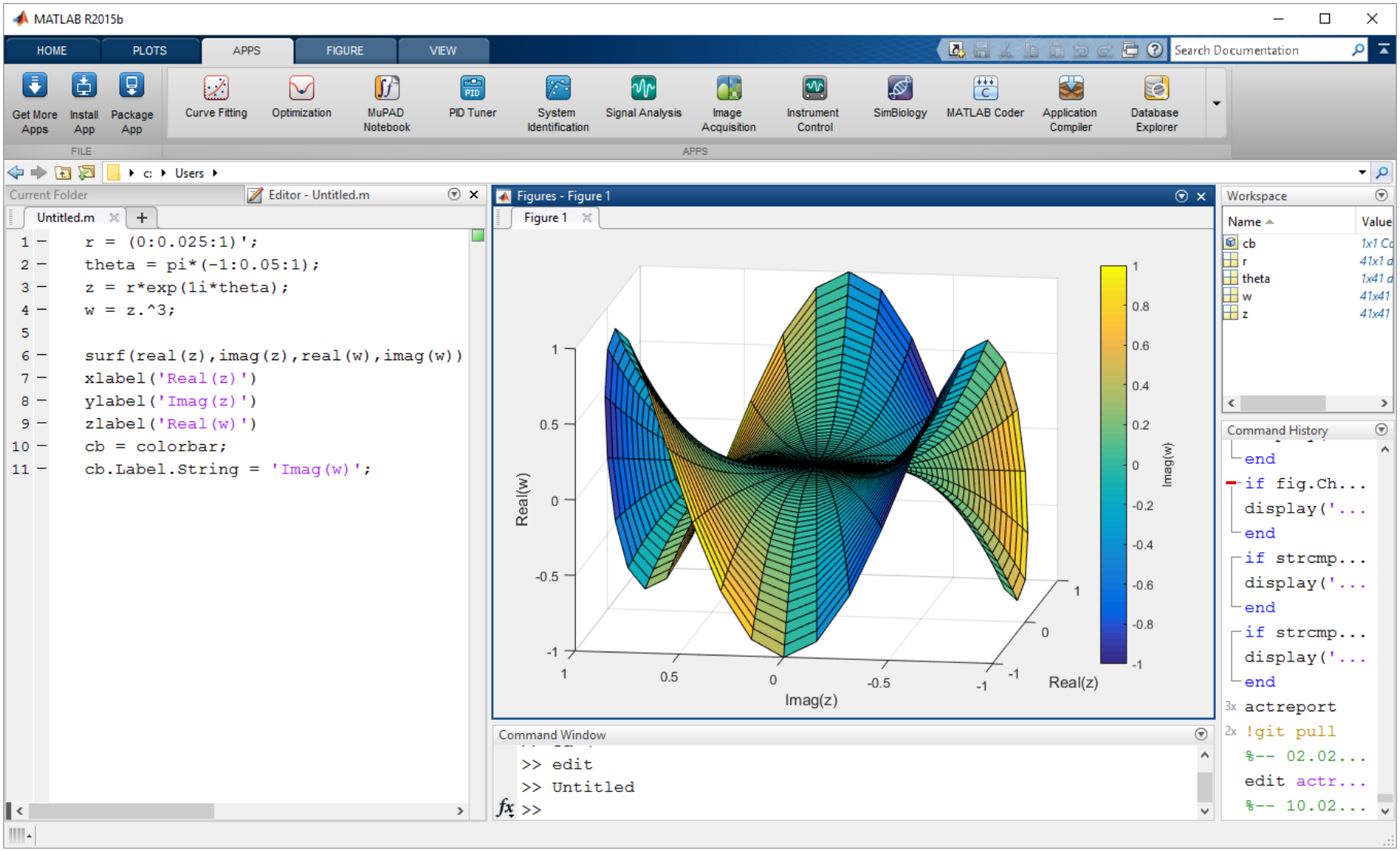
Vorteile gegenüber SciPy:

- GUI
- weit verbreitet

Nachteile gegenüber SciPy:

- Nicht Open-Source, kostenpflichtig
- Deutlich langsamer (Python zwar langsam, SciPy-Stack jedoch C-Code mit Wrapper)
- Kein Python $\Rightarrow$ weniger Modularität, andere Syntax

Erläuterung: MATLAB-GUI



Quelle: wikipedia.com/wiki/matlab

Pandas: Übersicht

- Ermöglicht effiziente, tabellarische Verwaltung von Daten in Python
  - Statt (NumPy-)Arrays zweidimensionale Tabellen ("Spreadsheets")
  - Datensätze leichter verwalten
  - Teil des Scipy-Stacks, also kompatibel mit NumPy und SciPy
- Name → "Panel Data"
- Open-Source
- Kostenlos
- Installation: `pip install pandas`

Pandas: Wichtige Funktionen: Datenstrukturen (1)

Series: `pandas.Series(data, index)`

- data: iterierbare Datenmenge (Array, Liste...)
- index Zeilenbezeichner (optional)

```
In [4]: import pandas
        import numpy as np
```

```
In [11]: pandas.Series(np.random.randn(3), ['rand1', 'rand2', 'rand3'])
```

```
Out[11]: rand1   -0.055388
         rand2    0.446007
         rand3    0.511584
         dtype: float64
```

Pandas: Wichtige Funktionen: Datenstrukturen (2)

Series: `pandas.DataFrame(data, index, columns)`

- data: Zweidimensionale Datenmenge
- index: Zeilenbezeichner (optional)
- columns: Spaltenbezeichner (optional)

```
In [4]: import pandas
        import numpy as np
```

```
In [15]: pandas.DataFrame(np.random.randn(3,3), columns=['rand1', 'rand2', 'rand3'])
```

Out[15]:

	rand1	rand2	rand3
0	1.061519	1.799274	-0.894694
1	0.913425	1.871492	-1.093921
2	0.687447	2.048325	1.289202

Pandas: Wichtige Funktionen: DataFrames (1)

DataFrame: Basics

Sei `df` ein DataFrame.

- `df.to_numpy()`: DataFrame → Numpy-Array
- `df[a]`, `df.a`: Spalte mit Titel 'a' als Series
- `df[a:b]`: Zeilen zwischen a (inklusiv) und b (exklusiv)
- `df.head(a)`: Alle Zeilen bis Index a (exklusiv); default 5
- `df.tail(a)`: Die letzten a Zeilen; default 5
- `df.sort_values(a)`: Zeilen werden aufsteigend nach Spalte 'a' sortiert
- `df.columns`
- `df.index`

Pandas: Wichtige Funktionen: DataFrames (2)

DataFrame: `loc`,`iloc` und `ix`

Sei `df` ein DataFrame.

- `df.loc[a,b]`: In Zeile(n) 'a' und Spalte(n) 'b'
- `df.iloc[a,b]`: In Zeile(n) a und Spalte(n) b
- `df.ix[a,b]`: Hybrid aus `loc` und `iloc`

In [4]:
```
import pandas
import numpy as np
```

In [16]:
```
df = pandas.DataFrame(np.random.randn(3,3), columns=['rand1', 'rand2', 'rand3'])
```

In [40]:
```
print(df.loc[1])
print('..........................')
print(df.iloc[:,1:3])
```

```
rand1   -0.717296
rand2   -0.101526
rand3    0.087465
Name: 1, dtype: float64
..........................
      rand2     rand3
0  0.271276  1.120179
1 -0.101526  0.087465
2 -1.570037  1.229410
```

Pandas: Wichtige Funktionen: DataFrames (3)

DataFrame: `at` und `apply`

Sei `df` ein DataFrame.

- `df.at[a,b] = c`: Element(e) bei [a,b] wird/werden durch c ersetzt.
- `df.apply(func)`: Für jedes Element a $\in$ df wird ein func(a) gegeben.

In [4]:
```
import pandas
import numpy as np
```

In [66]:
```
df = pandas.DataFrame(np.random.randn(3,3), columns=['rand1', 'rand2', 'rand3'])
print(df)
```

```
      rand1     rand2     rand3
0  1.683432 -1.597399 -1.017243
1  0.525464  0.269870  0.094475
2 -0.824175 -0.166633 -0.346628
```

In [68]:
```
df['rand1'] = df['rand1'].apply(np.sqrt)
df.at[:,'rand3'] = 50
print(df)
```

```
      rand1     rand2  rand3
0  1.297472 -1.597399     50
1  0.724889  0.269870     50
2       NaN -0.166633     50
```

Pandas: Wichtige Funktionen: DataFrames (4)

DataFrame: `dropna` und `fillna`

Sei `df` ein DataFrame.

- `df.dropna()`: Entfernt eine Zeile, wenn in ihr Einträge NaN sind.
- `df.fillna(a)`: Ersetzt jedes NaN in df durch a.

In [4]:
```
import pandas
import numpy as np
```

In [76]:
```
df = pandas.DataFrame([[np.nan,np.nan,3],[4,5,6],[7,8,np.nan]],)
print(df)
```

```
     0    1    2
0  NaN  NaN  3.0
1  4.0  5.0  6.0
2  7.0  8.0  NaN
```

In [91]:
```
print(df.dropna())
print()
print(df.fillna(10))
```

```
     0    1    2
1  4.0  5.0  6.0

      0     1     2
0  10.0  10.0   3.0
1   4.0   5.0   6.0
2   7.0   8.0  10.0
```

Pandas: Wichtige Funktionen: DataFrames (5)

DataFrame: `append` und `concat`

Sei `df` ein DataFrame.

- `df.append(a)`: Fügt alle Zeilen aus a (DataFrame) df hinzu.
- `pandas.concat([a,b,c])`: Erstellt aus a, b und c einen DataFrame.

```
In [1]: import pandas
        import numpy as np
```

```
In [14]: df = pandas.DataFrame([['a','b']])
         print(df)
```

```
   0  1
0  a  b
```

```
In [15]: print(df.append(df))
         print()
         print(pandas.concat([df,df,df]))
```

```
   0  1
0  a  b
0  a  b

   0  1
0  a  b
0  a  b
0  a  b
```

Pandas: I/O

Pandas versteht viele verschiedene Dateiformate!

- scv, html, json, Excel, SQL...
- Input: `pandas.read_<Dateiformat>` (Alternative: `DataFrame.from_csv`)
- Output: `pandas.to_<Dateiformat>` (Alternative: `DataFrame.to_csv`)

Vergleich: Pandas vs Vanilla Python (1)

Vergleich mit direkter Python-Implementation:

In [12]:
```
import pandas as pd
import numpy as np
s = pd.DataFrame([[1,11],[2,22],[3,33]],index=['rot', 'gelb', 'gruen'])

def funcPandas(x,y):
    return s.iloc[x,y]

funcPandas(0,1) #example
```

Out[12]: 11

In [13]:
```
dict = {
  "rot": [1,11],
  "gelb": [2,22],
  "gruen": [3,33],
}

def funcPython(x,y):
    if x==0:
        relevant=dict["rot"]
    elif x==1:
        relevant=dict["gelb"]
    elif x==2:
        relevant=dict["gruen"]
    return relevant[y]

funcPython(0,1) #example
```

Out[13]: 11

Vergleich: Pandas vs Vanilla Python (2)

Performancevergleich: `%timeit`

```
In [19]: %timeit funcPandas(0,1)
7.42 µs ± 37.7 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

In [20]: %timeit funcPython(0,1)
108 ns ± 0.14 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

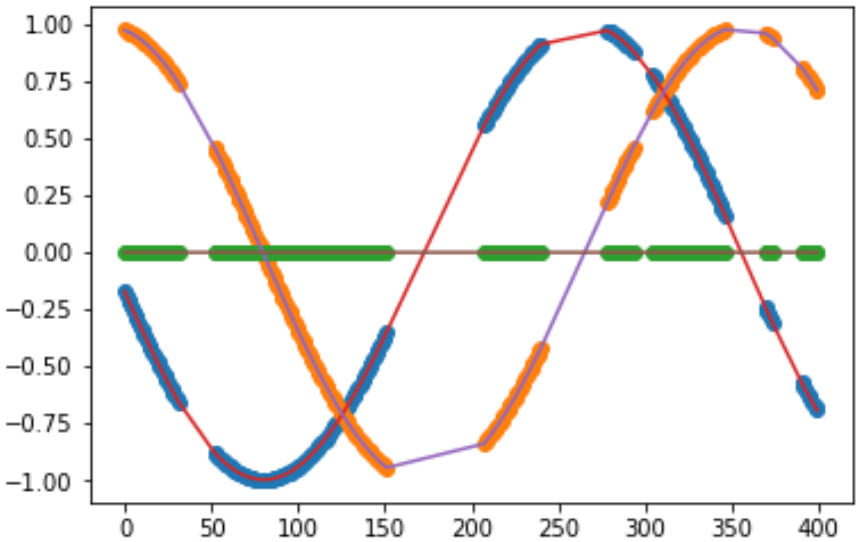
Anwendungsbeispiel: Import von .txt

In [82]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.interpolate as ip
```

In [94]:
```
data = pd.read_csv('/home/felix/Documents/astrodata', sep=" ")
data.columns = ['Day', 'X', 'Y', 'Z']
```

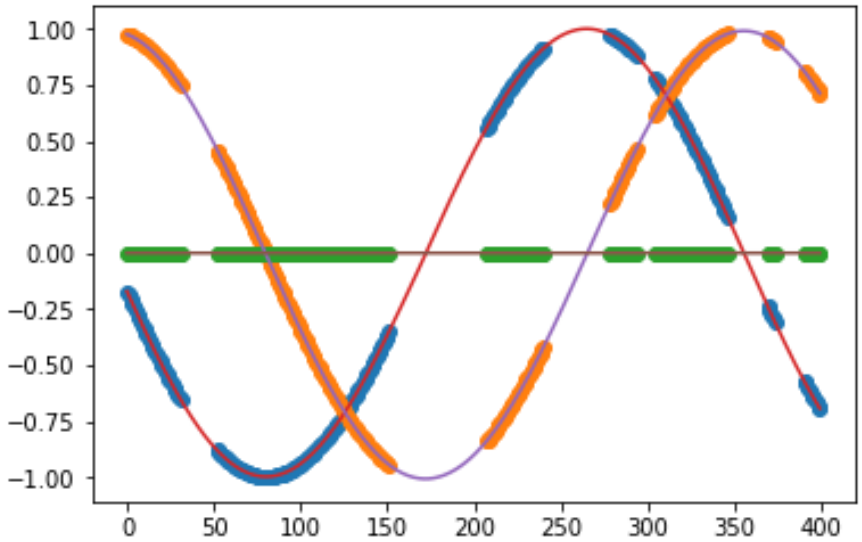
Anwendungsbeispiel: Interpolation (1)

In [89]:
```
fx = ip.interp1d(data.Day,data.X)
fy = ip.interp1d(data.Day,data.Y)
fz = ip.interp1d(data.Day,data.Z)

days = np.arange(0, 400)

plt.plot(data.Day,data.X,'o',
         data.Day,data.Y,'o',
         data.Day,data.Z,'o',
         days,fx(x), days,fy(x), days,fz(x))
plt.show()
```



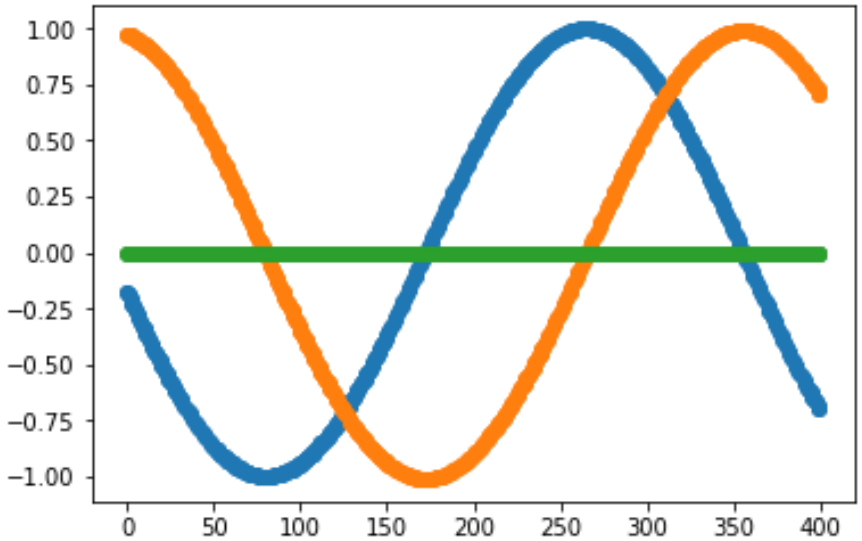
Anwendungsbeispiel: Interpolation (2)

In [90]:
```
fx = ip.interp1d(data.Day,data.X, kind='quadratic')
fy = ip.interp1d(data.Day,data.Y, kind='quadratic')
fz = ip.interp1d(data.Day,data.Z)

days = np.arange(0, 400)

plt.plot(data.Day,data.X,'o',
         data.Day,data.Y,'o',
         data.Day,data.Z,'o',
         days,fx(x), days,fy(x), days,fz(x))
plt.show()
```



Anwendungsbeispiel: Append von weiteren Datensätzen

```
In [91]: data = data.append(pd.read_csv('/home/felix/Downloads/ss19_2.txt', sep=" "))
         plt.plot(data.Day,data.X,'o', data.Day,data.Y,'o', data.Day,data.Z,'o')
         plt.show()
```



Zusammenfassung

- Scipy
  - Komplexere mathematische Operationen, zB. Integrale, Matrixoperationen, Interpolation, Optimisierung...
  - Code teilweise in C → hohe Performanz
- Pandas
  - Datenspeicherung in Panelen, Verwaltung von Daten
  - Selektionsfunktionen nach Spalten-/Zeilenbezeichner
  - Kompatibilität mit anderen Verwaltungsformen
- SciPy-Stack
- Open-Source, kostenlos

Literatur (1)

Alle Websites wurden zuletzt am 19.05.2019 aufgerufen.

- "The SciPy Stack Specification", https://www.scipy.org/stackspec.html
- "SciPy", https://scipy.github.io/devdocs
- Eric van Rees: "pandas and NumPy arrays explained", https://medium.com/@ericvanrees/pandas-series-objects-and-numpy-arrays-15dfe05919d7
- "pandas: powerful Python data analysis toolkit", https://pandas.pydata.org/pandas-docs/stable/
- Jonathan Kinlay: "A Comparison of Programming Languages", http://jonathankinlay.com/2018/10/comparison-programming-languages/
- Shane Lynn: "Using iloc, loc, ix to select rows and columns in Pandas DataFrames", https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/

Literatur (2)

- LaTeX-Template: "Knowledge graph in Neo4J", https://de.overleaf.com/articles/knowledge-graph-in-neo4j/dfmppvtndqhg
- Wikipedia: "NumPy", https://de.wikipedia.org/wiki/NumPy
- Wikipedia: "SciPy", https://de.wikipedia.org/wiki/SciPy
- Wikipedia: "Pandas (Software)", https://de.wikipedia.org/wiki/Pandas_(Software)
- Dharmindar Devsidas: "Python Scientific Programming - SciPy Basic Interpolation", https://www.youtube.com/watch?v=b0jo2g3iKJM
- John Foster: "SciPy - Scientific Toolkit", https://www.youtube.com/watch?v=MtdLd2Irvag
- Harrison Kinsley: "Data Analysis with Python and Pandas Tutorial Introduction", https://www.youtube.com/watch?v=Iqjy9UqKKuo
- "Profiling and Timing Code", https://jakevdp.github.io/PythonDataScienceHandbook/01.07-timing-and-profiling.html