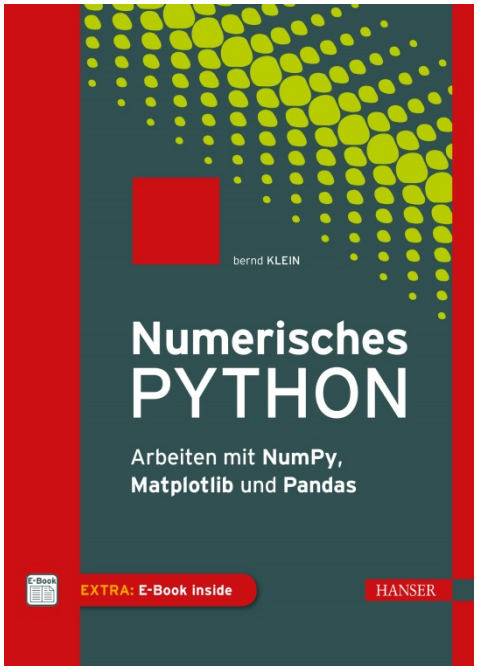
HANSER



Leseprobe

zu

„Numerisches Python“

von Bernd Klein

Print-ISBN: 978-3-446-45076-9
E-Book-ISBN: 978-3-446-45363-0

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-45076-9
sowie im Buchhandel

Inhalt

Vorwort

Eine der treibenden Kräfte in der weltweiten Softwareentwicklung wird wohl am besten durch die beiden populären Begriffe „Big Data" und „Maschinelles Lernen" beschrieben. Immer mehr Institute und Firmen betätigen sich in diesen Feldern. Für diese und auch für individuelle Personen, die in diesen Bereichen tätig werden wollen, ist eine der bedeutendsten Fragen – wenn nicht gar die bedeutendste Frage –, was die geeignetste Programmiersprache zu diesem Zweck ist. In vielen Umfragen wird Python als beste oder auch als beliebteste Programmiersprache genannt.

Python war ursprünglich nicht für numerische Probleme ausgerichtet gewesen. Die Erfolgsstory von Python wurde erst möglich durch die Module NumPy, SciPy, Matplotlib und Pandas. Dieses Buch bietet eine umfassende Einführung in die Module NumPy, Matplotlib und Pandas, setzt aber grundlegende Kenntnisse von Python voraus. Somit ergänzt es in idealer Weise das Buch „Einführung in Python 3: Für Ein- und Umsteiger" von Bernd Klein.

Brigitte Bauer-Schiewek, Lektorin

12 Einführung

Matplotlib ist eine Bibliothek zum Plotten wie GNUplot. Der Hauptvorteil gegenüber GNUplot ist die Tatsache, dass es sich bei Matplotlib um ein Python-Modul handelt. Aufgrund des wachsenden Interesses an der Programmiersprache Python steigt auch die Popularität von Matplotlib kontinuierlich.

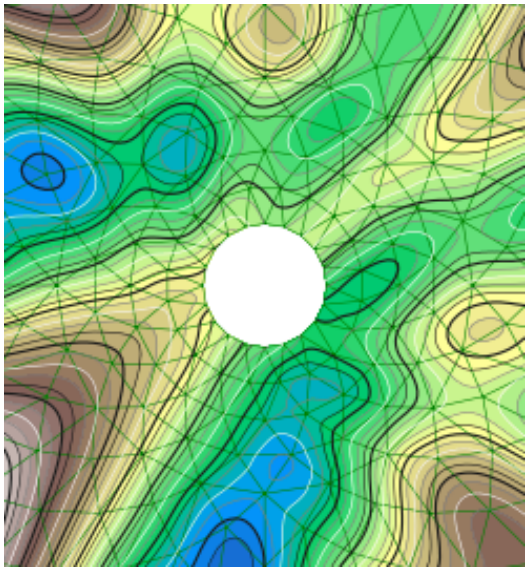


Bild 12.1 Tricontouring

Ein anderer Grund für die Attraktivität von Matplotlib liegt in der Tatsache, dass es als gute Alternative, wenn es ums Plotten geht, für MATLAB angesehen wird, wenn es in Verbindung mit NumPy und SciPy benutzt wird. Während es sich bei MATLAB um kostspielige Closed-Source-Software handelt, ist die Software von Matplotlib frei, kostenlos und quelloffen. Außerdem kann in Matplotlib objektorientiert programmiert werden. Es kann auch in allgemeinen GUIs wie wxPython, Qt und GTK+ verwendet werden. Mit der „pylab“-Erweiterung wird die Möglichkeit geboten, noch MATLAB-ähnlicher zu programmieren. Davon wird jedoch im Allgemeinen abgeraten, da dies zu einem unsauberen Programmierstil führt, auch wenn es dadurch MATLAB-Nutzern extrem leicht gemacht wird zu wechseln.

Mittels Matplotlib kann man Diagramme und Darstellungen in verschiedenen Formaten erzeugen, die dann in Veröffentlichungen verwendet werden können.

Eine andere Besonderheit besteht in der steilen Lernkurve, was sich darin zeigt, dass die Benutzerinnen und Benutzer sehr schnelle Fortschritte bei der Einarbeitung machen. Auf der offiziellen Webseite steht dazu Folgendes: „Matplotlib versucht, Einfaches einfach und Schweres möglich zu machen. Man kann mit nur wenigen Codezeilen Plots, Histogramme,

Leistungsspektren, Balkendiagramme, Fehlerdiagramme, Streudiagramme (Punktwolken) und so weiter erzeugen.“ [1]

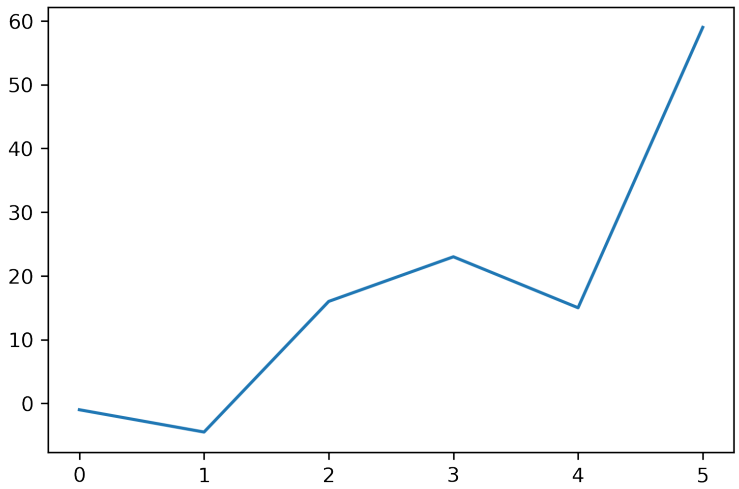
12.1 Ein erstes Beispiel

Wir werden mit einem einfachen Graphen beginnen. So einfach, dass es nicht mehr einfacher geht. Ein Graph in Matplotlib ist eine zwei- oder dreidimensionale Zeichnung, die mithilfe von Punkten, Kurven, Balken oder anderem einen Zusammenhang herstellt. Es gibt zwei Achsen: die horizontale x-Achse für die unabhängigen Werte und die vertikale y-Achse für die abhängigen Werte.

Wir werden im Folgenden das Untermodul `pyplot` verwenden. `pyplot` stellt eine prozedurale Schnittstelle zur objektorientierten Plot-Bibliothek von Matplotlib zur Verfügung. Die Kommandos von `pyplot` sind so gewählt, dass sie sowohl in den Namen als auch in ihren Argumenten MATLAB ähnlich sind.

Es ist allgemein üblich, `matplotlib.pyplot` in `plt` umzubenennen. In unserem ersten Beispiel werden wir die `plot`-Funktion von `pyplot` benutzen. Wir übergeben an die `plot`-Funktion eine Liste von Werten. `plot` betrachtet und benutzt die Werte dieser Liste als y-Werte. Die Indizes dieser Liste werden automatisch als x-Werte genommen.

```
import matplotlib.pyplot as plt

plt.plot([-1, -4.5, 16, 23, 15, 59])
plt.show()
```

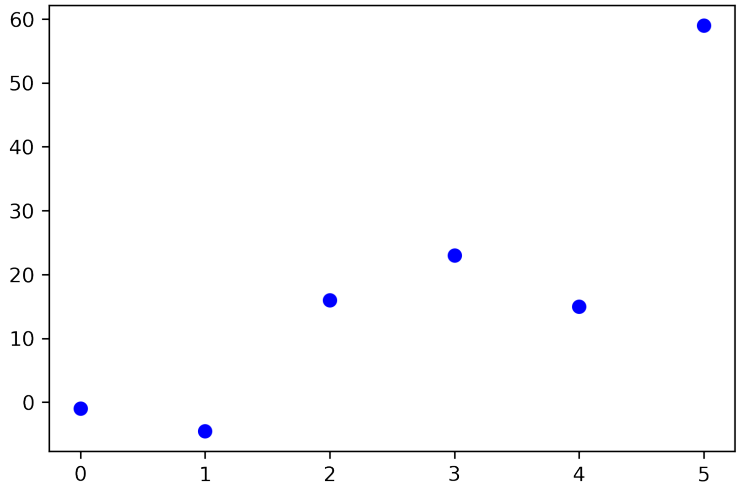


[1] „Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.“

Wir sehen einen zusammenhängenden Graphen, obwohl wir nur diskrete Werte für die Ordinate, allgemein auch als Y-Achse bezeichnet, zur Verfügung gestellt hatten. Als Werte für die Abszisse, also die X-Achse, wurden die Indizes genommen.

Indem wir einen Formatstring beim Funktionsaufruf mit übergeben, können wir einen Graphen mit diskreten Werten erzeugen, in unserem Fall mit blauen Vollkreisen. Der Formatstring definiert, wie die diskreten Punkte darzustellen sind:

```
import matplotlib.pyplot as plt

plt.plot([-1, -4.5, 16, 23, 15, 59], "ob")
plt.show()
```



12.2 Der Formatparameter von pyplot.plot

In unserem vorigen Beispiel hatten wir `ob` als Formatparameter genutzt. Er besteht aus zwei Zeichen. Das erste definiert den Linienstil oder die Darstellung der diskreten Werte, die Markierungen (englisch „markers“). Mit dem zweiten Zeichen wählt man die Farbe für den Graphen aus. Die Reihenfolge der beiden Zeichen hätte aber auch umgekehrt sein können, d.h. wir hätten auch `bo` schreiben können. Falls kein Formatparameter angegeben wird, wie es in unserem ersten Beispiel der Fall war, wird `b-` als Default-Wert benutzt, d.h. eine durchgehende blaue Linie wird ausgegeben.

Die folgenden Zeichen werden in einem Formatstring akzeptiert, um den Linienstil oder die Marker zu steuern:

Zeichen	Beschreibung
'-' (Bindestrich)	durchgezogene Linie
'–' (zwei Bindestriche)	gestrichelte Linie
'-.'	Strichpunkt-Linie
':'	punktierte Linie
'.'	Punkt-Marker
','	Pixel-Marker
'o'	Kreis-Marker
'v'	Dreiecks-Marker, Spitze nach unten
'^'	Dreiecks-Marker, Spitze nach oben
'<'	Dreiecks-Marker, Spitze nach links
'>'	Dreiecks-Marker, Spitze nach rechs
'1'	tri-runter-Marker
'2'	tri-hoch-Marker
'3'	tri-links Marker
'4'	tri-rechts Marker
's'	quadratischer Marker
'p'	fünfeckiger Marker
'*'	Stern-Marker
'h'	Sechseck-Marker1
'H'	Sechseck-Marker2
'+'	Plus-Marker
'x'	x-Marker
'D'	rautenförmiger Marker
'd'	dünner rautenförmiger Marker
'|'	Marker in Form einer vertikalen Linie
'_'	Marker in Form einer horizontalen Linie

Die folgenden Farbabkürzungen sind möglich:

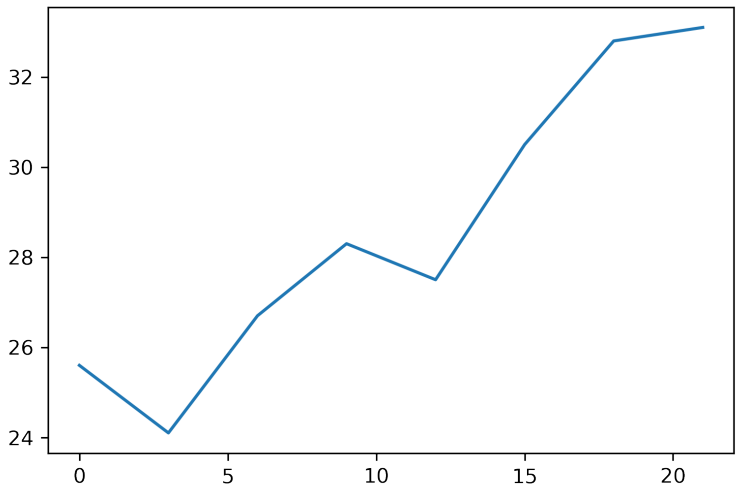
Zeichen	Farbe
'b'	blau
'g'	grün
'r'	rot
'c'	cyan
'm'	magenta
'y'	gelb
'k'	schwarz
'w'	weiß

Wie einige sicherlich schon vermutet haben, kann man auch X-Werte an die Plot-Funktion übergeben. Im folgenden Beispiel übergeben wir eine Liste mit den Vielfachen von 3 zwischen 0 und 21 als X-Werte an plot:

```
import matplotlib.pyplot as plt

# die X-Werte:
days = list(range(0, 22, 3))
print(days)
# die Y-Werte:
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]

plt.plot(days, celsius_values)
plt.show()
```

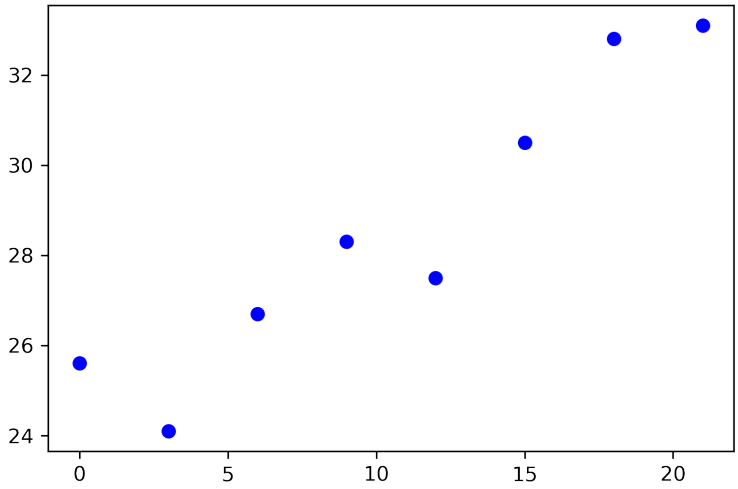
Ausgabe:

```
[0, 3, 6, 9, 12, 15, 18, 21]
```



... und das Ganze wieder mit diskreten Werten:

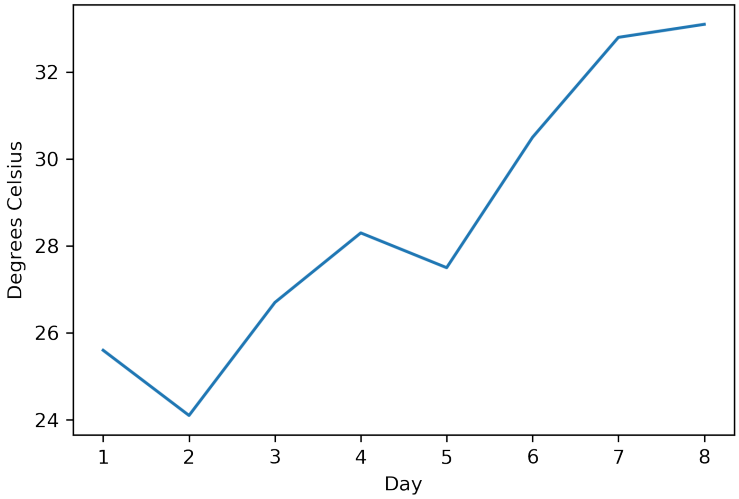
```
plt.plot(days, celsius_values, 'bo')
plt.show()
```



12.3 Bezeichnungen für die Achsen

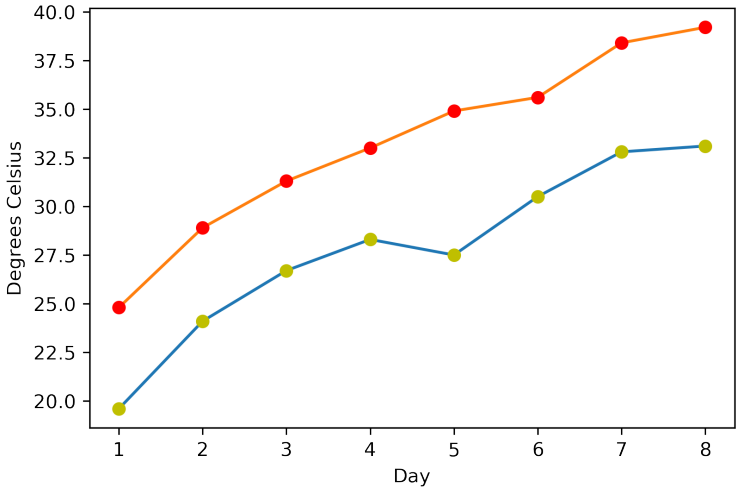
Wir können das Aussehen unseres Graphen verbessern, indem wir die Achsen mit Bezeichnungen versehen. Dazu benutzen wir die `ylabel`- und `xlabel`-Funktionen von `pyplot`.

```
import matplotlib.pyplot as plt

days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]

plt.plot(days, celsius_values)
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.show()
```



Wir können eine beliebige Anzahl von (x, y, fmt)-Gruppen in einer Plot-Funktion spezifizieren. Im folgenden Beispiel benutzen wir zwei verschiedene Listen mit Y-Werten:

```
import matplotlib.pyplot as plt

days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]

plt.xlabel('Day')
plt.ylabel('Degrees Celsius')

plt.plot(days, celsius_min,
  days, celsius_min, "oy",
  days, celsius_max,
  days, celsius_max, "or")

plt.show()
```



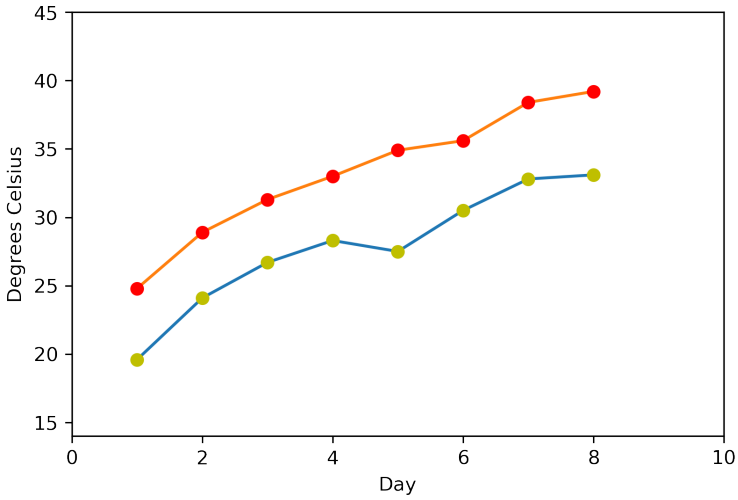
12.4 Abfragen und Ändern des Wertebereichs der Achsen

Mit der Funktion `axis` lässt sich der Wertebereich einer Achse abfragen und ändern. Ruft man `axis` ohne Argumente auf, liefert sie den aktuellen Wertebereich einer Achse zurück:

```
import matplotlib.pyplot as plt

days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]

plt.xlabel('Day')
plt.ylabel('Degrees Celsius')

plt.plot(days, celsius_min,
  days, celsius_min, "oy",
  days, celsius_max,
  days, celsius_max, "or")

print("The current limits for the axes are:")
print(plt.axis())
print("We set the axes to the following values:")
xmin, xmax, ymin, ymax = 0, 10, 14, 45
print(xmin, xmax, ymin, ymax)
plt.axis([xmin, xmax, ymin, ymax])
plt.show()
```

Ausgabe:

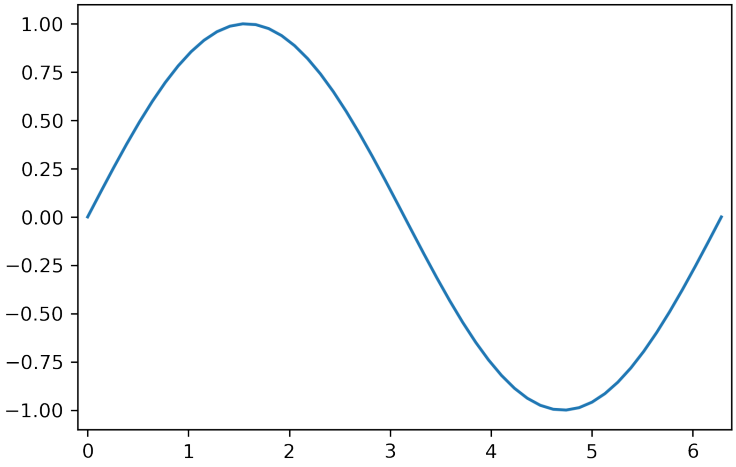
```
The current limits for the axes are:
(0.6499999999999999, 8.35, 18.62, 40.18)
We set the axes to the following values:
0 10 14 45
```



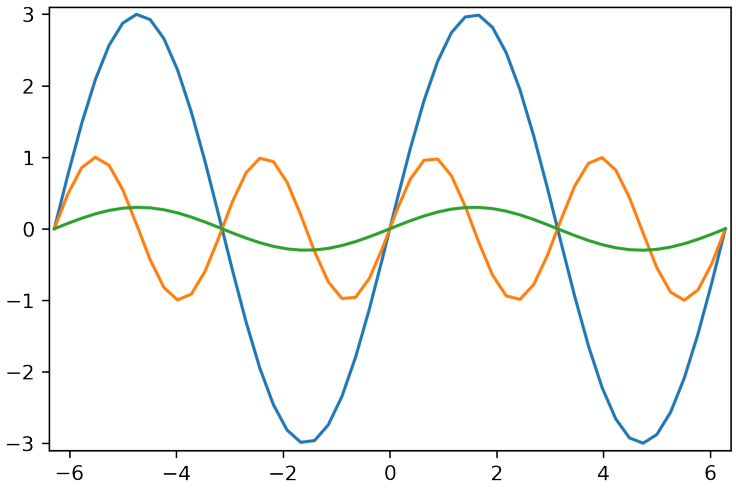
12.5 „linspace“ zur Definition von X-Werten

Im folgenden Beispiel werden wir die NumPy-Funktion `linspace` verwenden. `linspace` wird dazu benutzt, gleichmäßig verteilte Werte innerhalb eines spezifizierten Intervalls zu erzeugen. Wir haben `linspace` ausführlich in unserem NumPy-Kapitel behandelt.

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

X = np.linspace(0, 2 * np.pi, 50, endpoint=True)
F = np.sin(X)
plt.plot(X,F)
startx, endx = -0.1, 2*np.pi + 0.1
starty, endy = -1.1, 1.1
plt.axis([startx, endx, starty, endy])
plt.show()
```

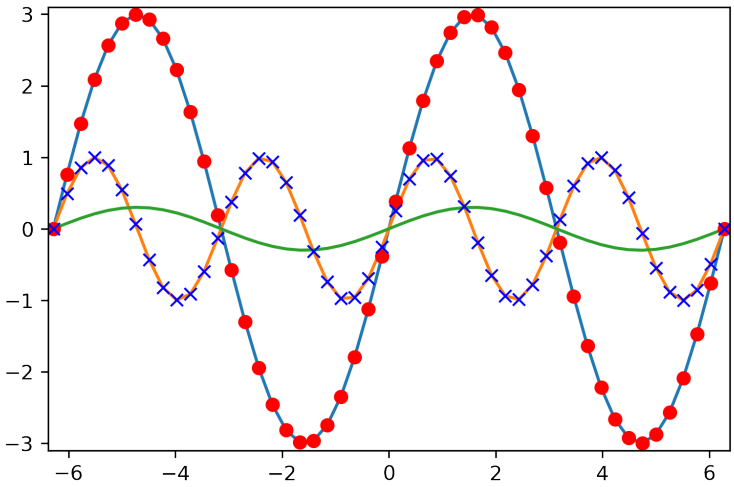



```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-2 * np.pi, 2 * np.pi, 50, endpoint=True)
F1 = 3 * np.sin(X)
F2 = np.sin(2*X)
F3 = 0.3 * np.sin(X)

startx, endx = -2 * np.pi - 0.1, 2*np.pi + 0.1
starty, endy = -3.1, 3.1
plt.axis([startx, endx, starty, endy])

plt.plot(X,F1)
plt.plot(X,F2)
plt.plot(X,F3)
plt.show()
```



Das nächste Beispiel ist im Prinzip nichts Neues. Wir fügen lediglich zwei weitere Plots mit diskreten Punkten hinzu:

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

X = np.linspace(-2 * np.pi, 2 * np.pi, 50, endpoint=True)
F1 = 3 * np.sin(X)
F2 = np.sin(2*X)
F3 = 0.3 * np.sin(X)
startx, endx = -2 * np.pi - 0.1, 2*np.pi + 0.1
starty, endy = -3.1, 3.1

plt.axis([startx, endx, starty, endy])
plt.plot(X,F1)
plt.plot(X,F2)
plt.plot(X,F3)
plt.plot(X, F1, 'ro')
plt.plot(X, F2, 'bx')
plt.show()
```



12.6 Linienstil ändern

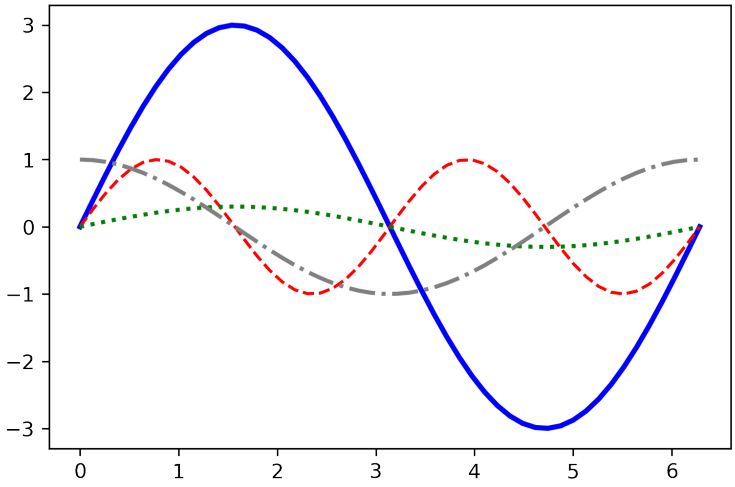
Der Linienstil eines Plots kann durch die Parameter `linestyle` oder `ls` der `plot`-Funktion beeinflusst werden. Sie können auf einen der folgenden Werte gesetzt werden:

'-', '–', '-.', ':', 'None', ' ', ''

Wir können mit `linewidth`, wie der Name impliziert, die Linienstärke oder Liniendicke setzen:

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

X = np.linspace(0, 2 * np.pi, 50, endpoint=True)
```

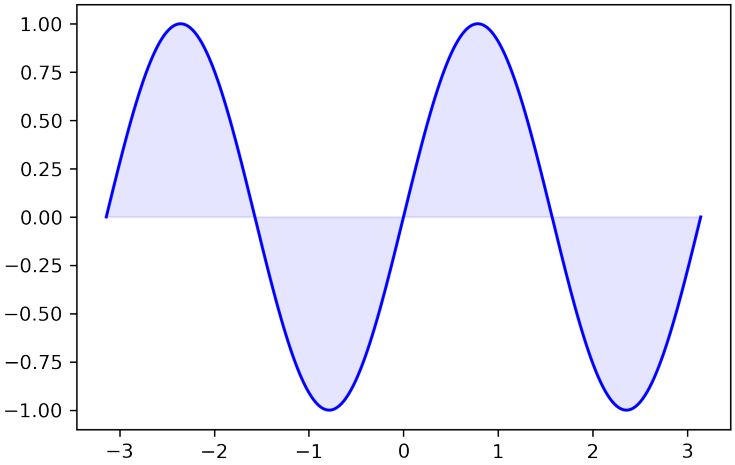
```
F1 = 3 * np.sin(X)
F2 = np.sin(2*X)
F3 = 0.3 * np.sin(X)
F4 = np.cos(X)

plt.plot(X, F1, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, F2, color="red", linewidth=1.5, linestyle="--")
plt.plot(X, F3, color="green", linewidth=2, linestyle=":")
plt.plot(X, F4, color="grey", linewidth=2, linestyle="-.")
plt.show()
```



12.7 Flächen einfärben

Mit der pyplot-Funktion `fill_between` ist es möglich, Flächen zwischen Kurven oder Achsen zu schraffieren oder einzufärben. Im folgenden Beispiel füllen wir die Flächen zwischen der X-Achse und dem Graph der Funktion `sin(2*X)`:

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

n = 256
X = np.linspace(-np.pi,np.pi,n,endpoint=True)
Y = np.sin(2*X)

plt.plot (X, Y, color='blue', alpha=1.00)
plt.fill_between(X, 0, Y, color='blue', alpha=.1)
plt.show()
```



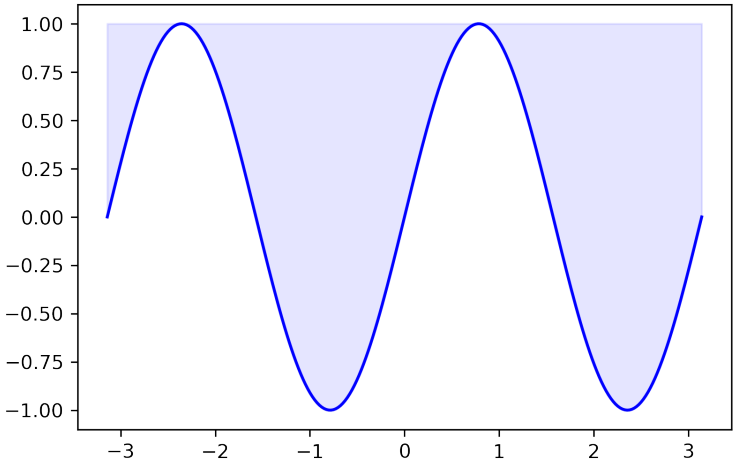
Die allgemeine Syntax von `fill_between`:

```
fill_between(x, y1, y2=0, where=None, interpolate=False, **kwargs)
```

Die Parameter von `fill_between`:

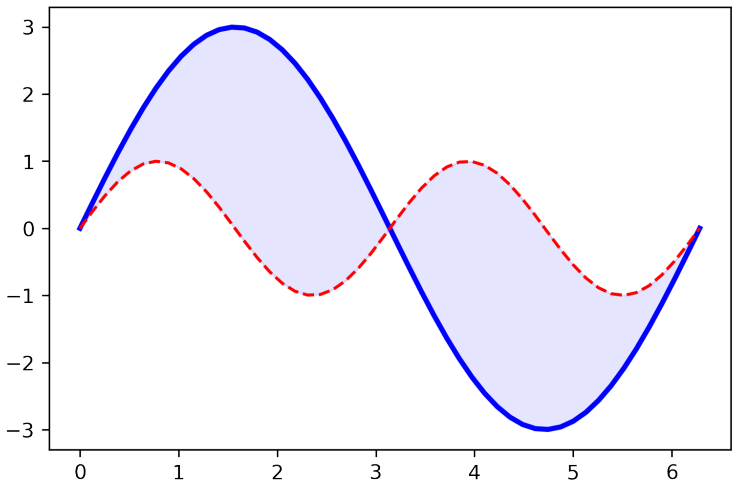
Parameter	Bedeutung
x	Ein Array mit N Elementen mit X-Werten
y1	Ein Array mit N Elementen (oder ein Skalar) von Y-Daten
y2	Ein Array mit N Elementen (oder ein Skalar) von Y-Daten
where	Wenn auf None gesetzt, wird per Default alles gefüllt. Wenn es nicht auf „None“ gesetzt wird, so wird ein numpy boolean-Array erwartet mit N Elementen. Es werden nur dann die Regionen eingefärbt, bei denen where==True ist.
interpolate	Wenn True, so wird zwischen zwei Linien interpoliert, um den genauen Schnittpunkt zu finden. Andernfalls werden die Start- und Endwerte nur als explizite Werte auf der Region erscheinen.
kwargs	Schlüsselwortargumente, die an PolyCollection übergeben werden.

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

 n = 256
 X = np.linspace(-np.pi,np.pi,n,endpoint=True)
 Y = np.sin(2*X)

 plt.plot (X, Y, color='blue', alpha=1.00)
 plt.fill_between(X, Y, 1, color='blue', alpha=.1)
 plt.show()
```



Im nächsten Beispiel füllen wir die Flächen zwischen den Funktionen F1 und F2:

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

X = np.linspace(0, 2 * np.pi, 50, endpoint=True)
F1 = 3 * np.sin(X)
F2 = np.sin(2*X)


plt.plot(X, F1, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, F2, color="red", linewidth=1.5, linestyle="--")
plt.fill_between(X, F1, F2, color='blue', alpha=.1)
plt.show()
```



Stichwortverzeichnis