Übung 3: Matplotlib

Name: Sahin Cingöz
Gruppe: C

Matplotlib ist ein Python-Modul, das zur Erstellung von Diagrammen und Darstellungen in verschiedenen Formaten verwendet wird. Die Beliebtheit von Matplotlib steigt aufgrund des wachsenden Interesses an der Programmiersprache Python stetig. Mittels Matplotlib lassen sich publikationsreife Diagramme und Darstellungen in verschiedenen Formaten erzeugen.

In dieser Übung sollen Sie sich mit den wichtigsten Plot-Routinen vertraut machen. Das Paket matplotlib ist das populärste Paket zur grafischen Darstellung in Python. Vieles funktioniert ähnlich wie in MAPLE. Mit dem Befehl:

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

lässt sich das Untermodul pyplot aus der matplotlib Bibliotek importieren, mit dem wir uns im ersten Teil dieser Übung beschäftigen wollen. Pyplot bietet eine prozedurale Schnittstelle für die objektorientierte Plot-Bibliothek von Matplotlib. In Python-Code wird häufig die Abkürzung "plt" verwendet, um auf Pyplot zuzugreifen. Diese Konvention dient dazu, den Code für andere Programmierer leichter verständlich zu machen und sollte daher beachtet werden. Weitere Informationen zu Pyplot finden Sie in dem Tutorial unter https://realpython.com/python-matplotlib-guide/.

In dieser Übung werden Sie nützliche Funktionen von **Numpy** anwenden. **NumPy** ist ein Akronym für "Numerisches Python" (englisch: "Numeric Python" oder "Numerical Python"). Dabei handelt es sich um ein Erweiterungsmodul für Python. Das Paket wird mit dem Befehl:
`import numpy as np`
importiert. Üblicherweise wird NumPy in np umbenannt.

Aufgabe 1 : Einführung in pyplot

In diesem 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, 0, 17, 36, 3.7])

plt.show()
```

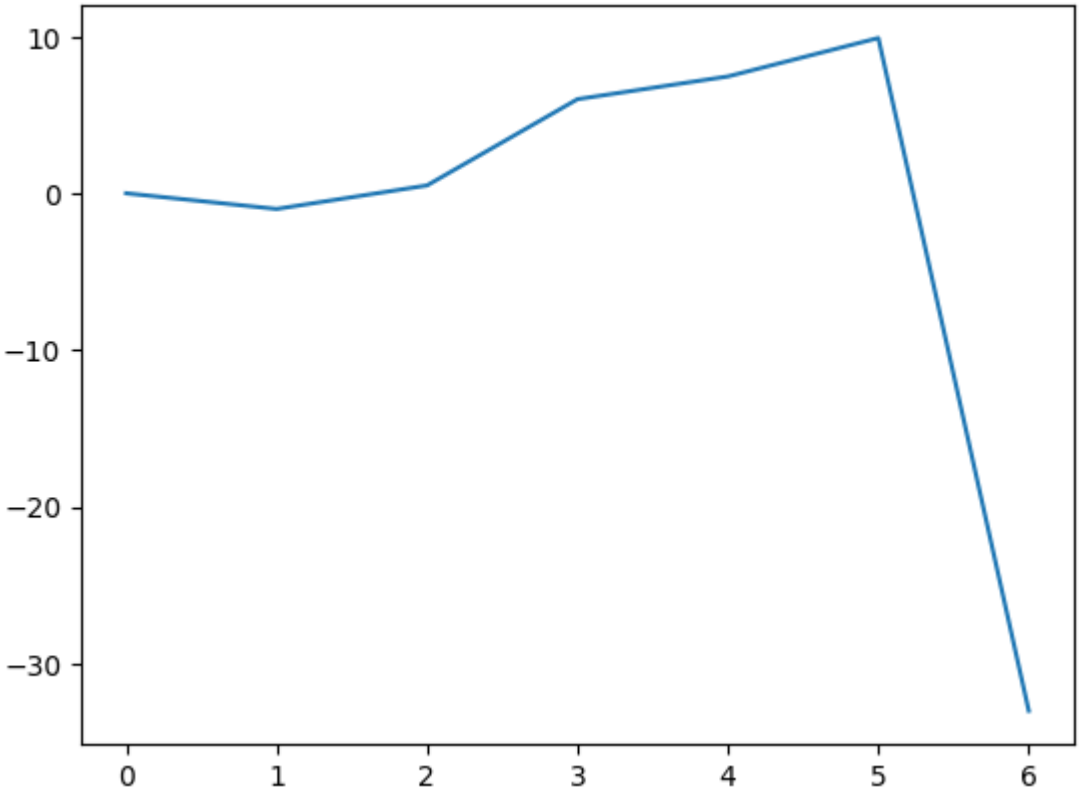
oder

```
y = [-1, -4.5, 0, 17, 36, 3.7]

plt.plot(y)

plt.show()
```

a) Führen Sie diese Befehle mit der Liste *[0,-1,0.5,6,7.45,9.9, -33]* aus. (0.1 Punkte)

In [4]:
```
import matplotlib.pyplot as plt
y = [0,-1,0.5,6,7.45,9.9,-33]
plt.plot(y)
plt.show
```

Out[4]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



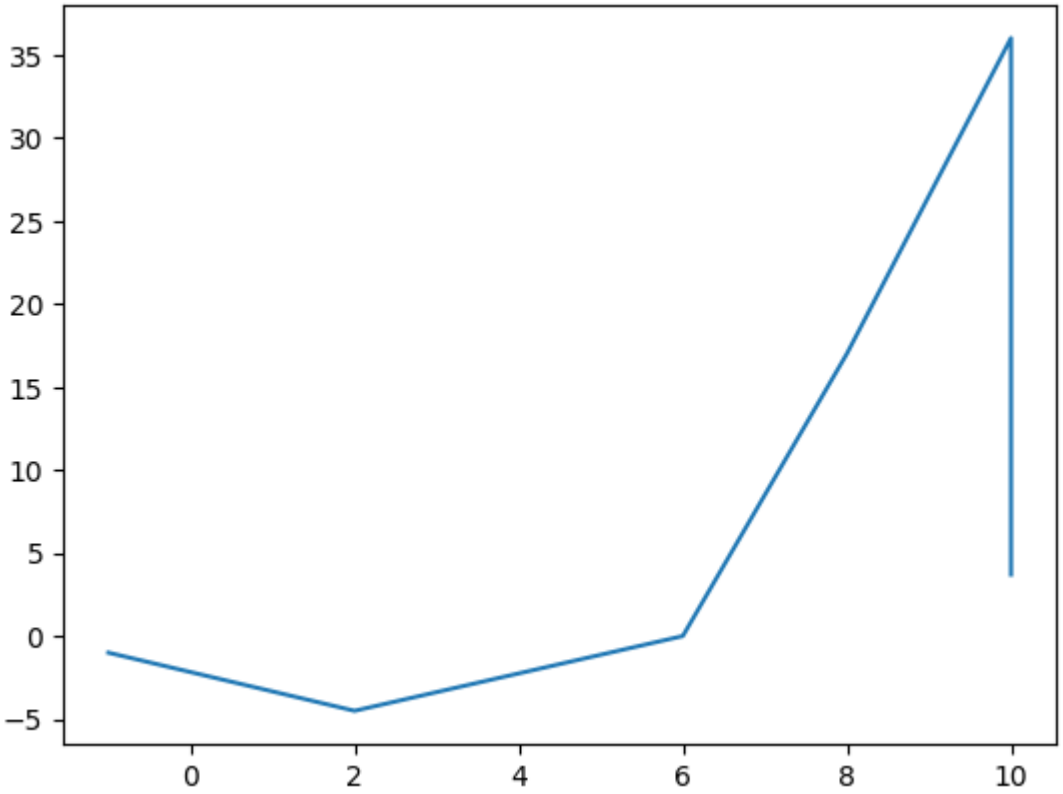
Auch X-Werte können explizit in der Form eine zweite Liste an die plot-Funktion übergeben. Zum Beispiel:

```
x = [-1, 2, 6, 8,10,10]

y = [-1, -4.5, 0, 17, 36, 3.7]

plt.plot(x,y)

plt.show()
```

b) Führen diesen Befehle aus. (0.1 Punkte)

In [5]:
```
import matplotlib.pyplot as plt
```

```
x = [-1, 2, 6, 8,10,10]

y = [-1, -4.5, 0, 17, 36, 3.7]

plt.plot(x,y)

plt.show()
```



Wir können einen Graphen mit diskreten Werten erstellen, indem wir einen Formatstring beim Funktionsaufruf mitübergeben. Zum Beispiel können wir die diskreten Punkte mit blauen Vollkreisen darstellen („bo“). Der Formatstring legt fest, wie die diskreten Punkte dargestellt werden:

```
plt.plot(daten, formatstring)
```

Zeichen für Formatstring finden Sie unter: https://python-course.eu/numerical-programming/formatting-plot-in-matplotlib.php. Zeichen

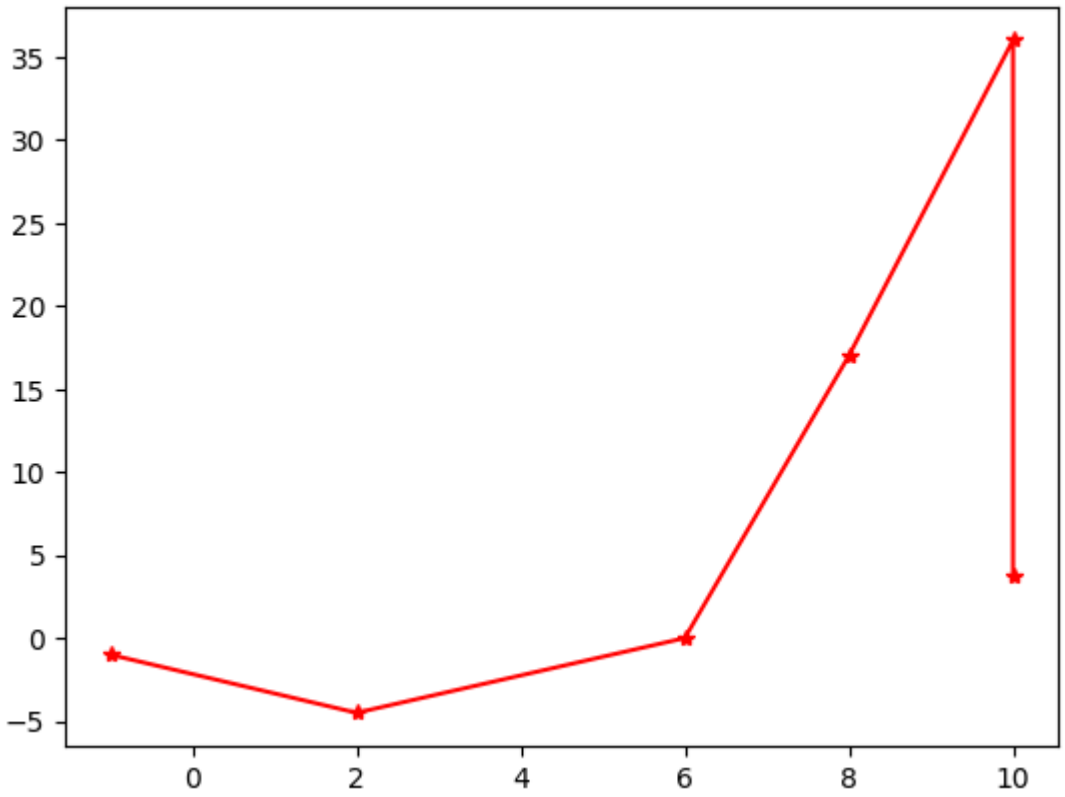
sind kombinierbar.

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

c) Stellen Sie die Daten in Aufgabe 1.b in eine neuen Graph als rote Sterne dar und verbinden Sie die Punkte mit einen roten Linien. (0.1 Punkte)

In [34]:
```
import matplotlib.pyplot as plt
x = [-1, 2, 6, 8,10,10]
y = [-1, -4.5, 0, 17, 36, 3.7]
plt.plot(x,y, "*r-")
plt.show()
```



Die Achsen eines Graphs lassen sich mit den **xlabel-** und **ylabel-** Funktionen von PyPlot benennen. Neben der Achsenbeschreibungen sollten bei Plots die Titel nicht fehlen. Diese lassen sich über die Funktion `plt.title()` hinzufügen. Zum Beispiel:

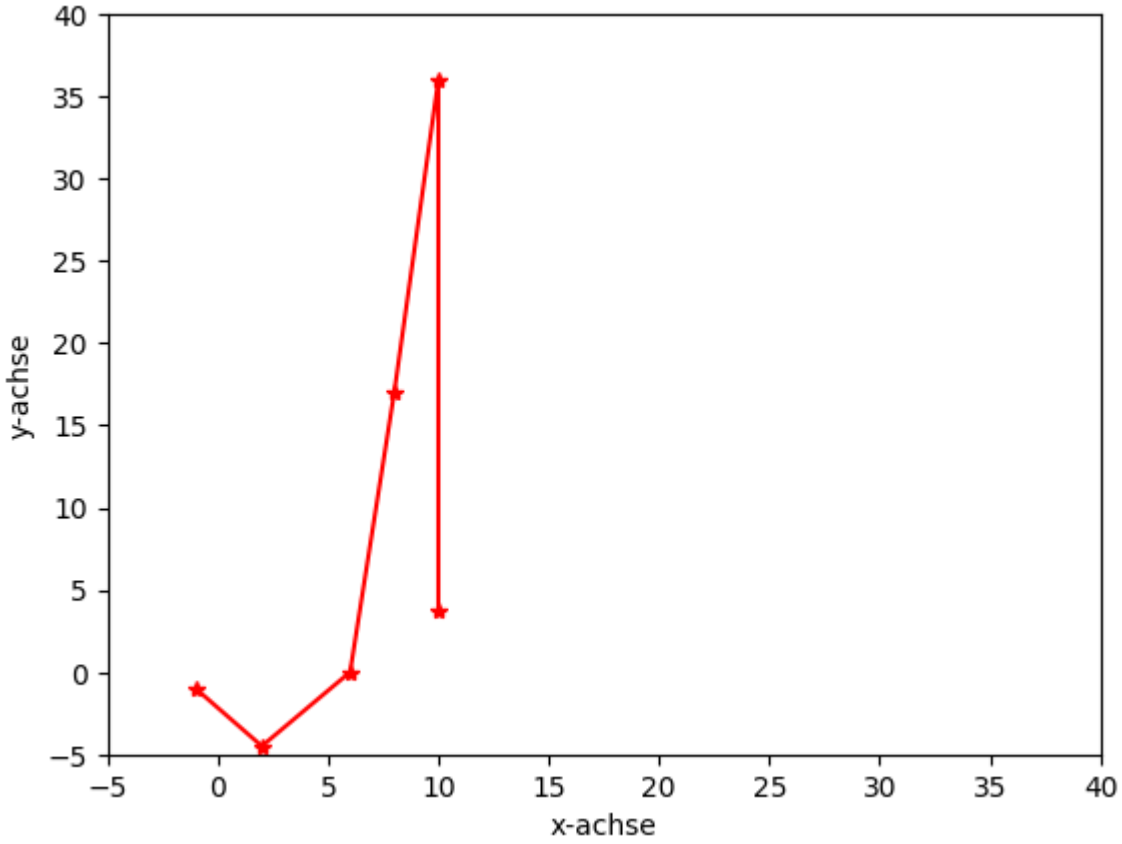
```
plt.xlabel("x")
plt.ylabel("y")
plt.title(„Meinem Graph")
plt.plot(x,y)
plt.show()
```

Die Wertebereiche der Achsen werden mit den Funktionen **xlim-** und **ylim-** definiert. Man kann beide Werte (*xmin* und *xmax*) oder auch nur einen Wert vorgeben.

```
plt.xlim(xmin, xmax)
plt.xlim(left=xmin) oder plt.xlim(right=xmax)

plt.ylim(ymin,ymax)
plt.ylim(bottom=ymin) oder plt.ylim(top=ymax)
```

d) Bezeichnen Sie die Achsen des Graphs von Aufgabe 1.c . Stellen Sie die Werte der X- und Y-Achsen im Bereich zwischen -5 und 40. (0.1 Punkte)

In [38]:
```
import matplotlib.pyplot as plt
x = [-1, 2, 6, 8,10,10]
y = [-1, -4.5, 0, 17, 36, 3.7]
plt.xlabel("x-achse")
plt.ylabel("y-achse")
plt.xlim(-5, 40)
plt.ylim(-5, 40)
plt.plot(x,y, "*r-")
plt.show()
```

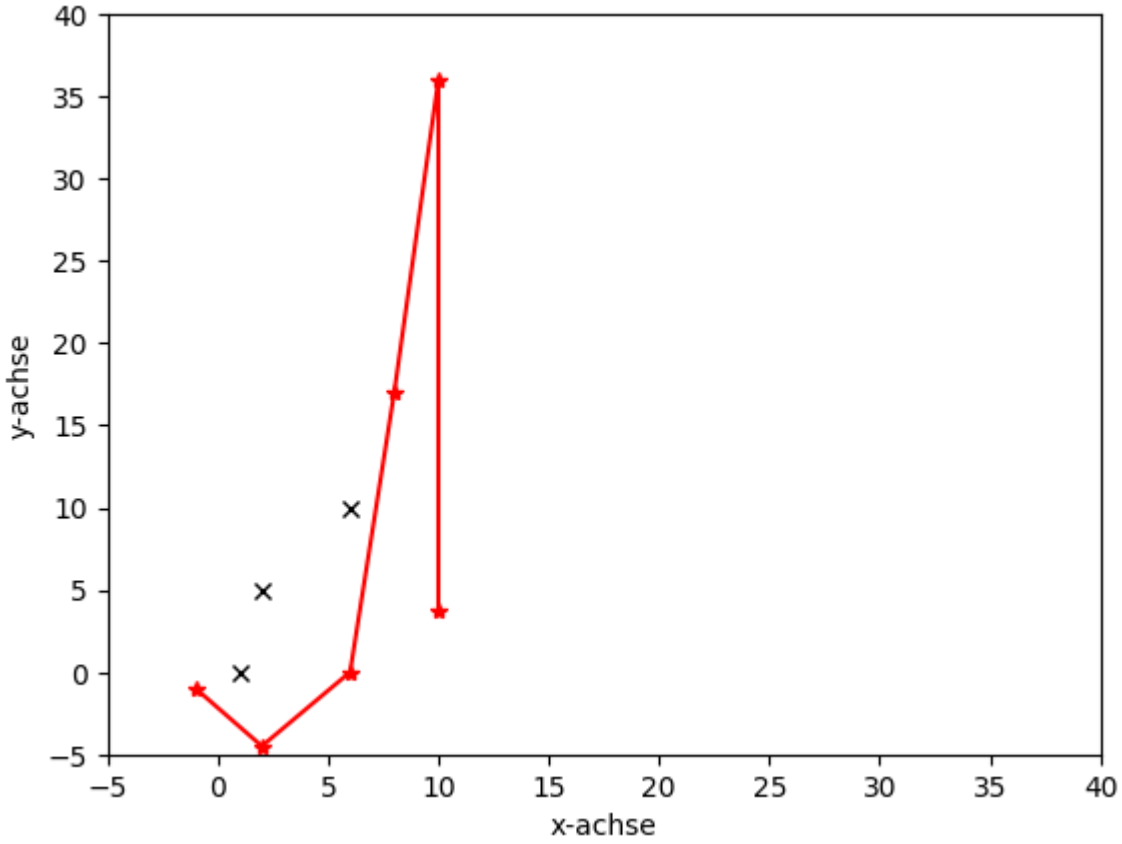


Mit *PyPlot* ist es auch möglich eine beliebige Anzahl von (x, y, formatstring)-Gruppen in einer **plot**-Funktion zu spezifizieren. Zum Beispiel:

```
x1=[1,2,3,4,5]
y1=[2,4,1,4,6]
x2=[-1,0,6,7]
y2=[9,3,5,0]
plt.plot(x1,y1,”ro”,y2,x2,”bo”)
plt.show()
```

e) Ergänzen Sie den Graph in Aufgabe 1.c mit den folgende Wertepaaren (1,0)(2,5) und (6,10) dargestellt als schwarze Kreuze.(0.1 Punkte)

In [37]:
```
import matplotlib.pyplot as plt
x = [-1, 2, 6, 8,10,10]
y = [-1, -4.5, 0, 17, 36, 3.7]
x1 = [1,2,6]
y1 = [0,5,10]
plt.xlabel("x-achse")
plt.ylabel("y-achse")
plt.xlim(-5, 40)
plt.ylim(-5, 40)
plt.plot(x,y, "*r-",x1,y1, "kx")
plt.show()
```



Aufgabe 2. Kraft-Weg Verlauf

a) Nutzen Sie die Daten aus der Tabelle 1 und stellen den Kraft-Weg Verlauf für Probe 1 in einem Grid mit beschrifteten Achsen dar. (0.1 Punkte)

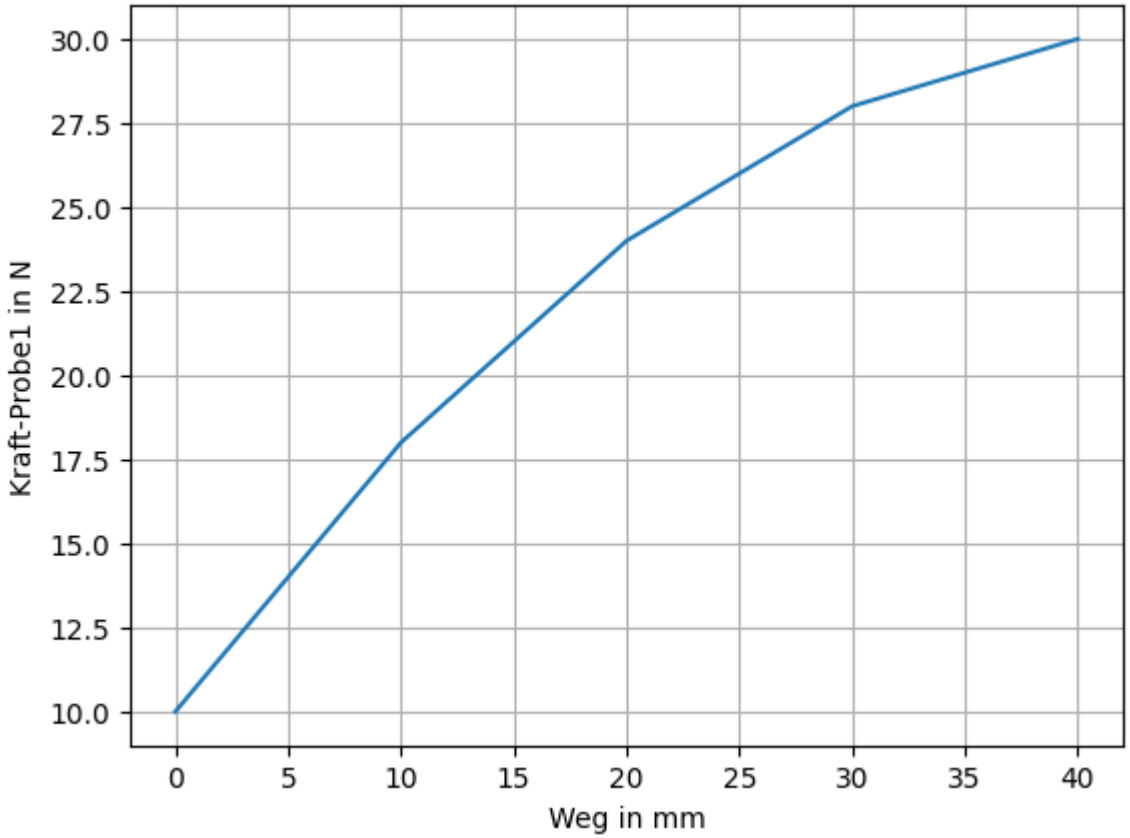
Sie können die Darstellung verbessern in dem Sie die **grid** (Gitter) Funktion verwenden: `plt.grid()` .

Tabelle 1

Weg(mm)	Kraft-Probe 1 (N)	Kraft-Probe 2 (N)

Weg(mm)	Kraft-Probe 1 (N)	Kraft-Probe 2 (N)
0	10	0
10	18	10
20	24	20
30	28	30
40	30	40

In [40]:
```
Weg = [0,10,20,30,40]
Probe1 = [10,18,24,28,30]
plt.xlabel("Weg in mm")
plt.ylabel("Kraft-Probe1 in N")
plt.plot(Weg,Probe1)
plt.grid(True)
plt.show()
```

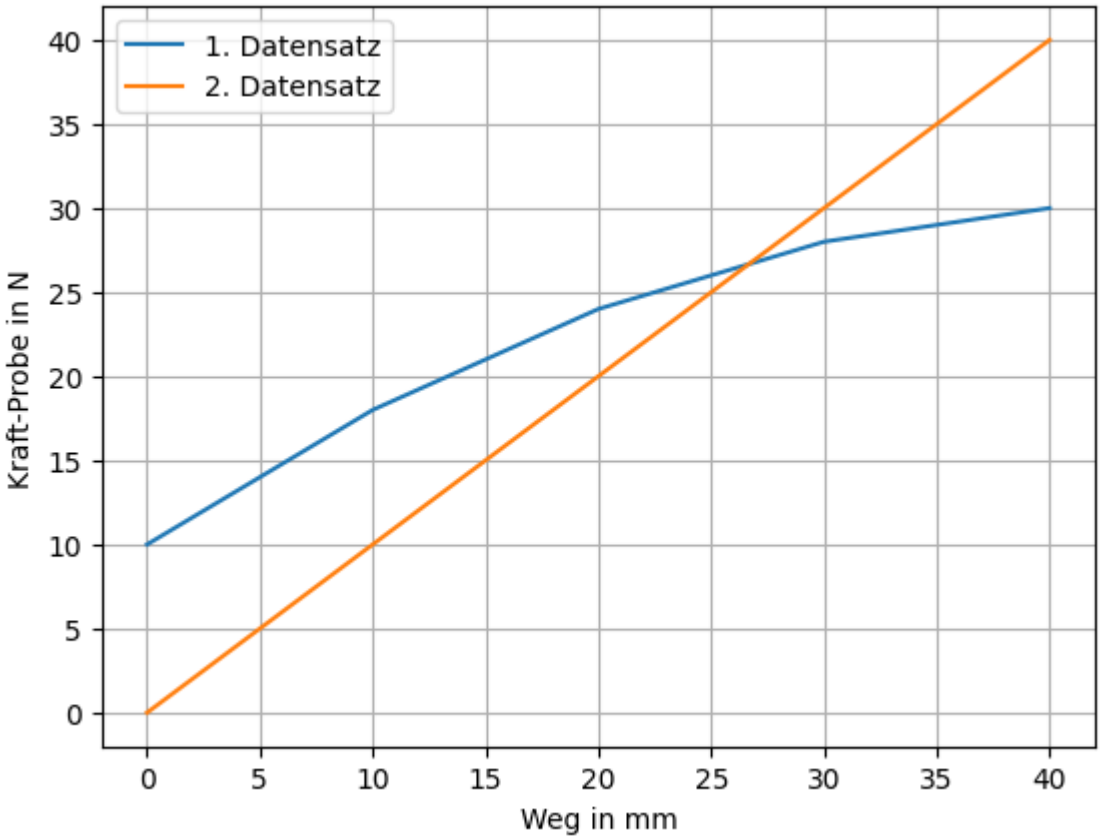


b) Plotten Sie nun einen zweiten Verlauf (Probe 2) dazu und beschriften Sie Beide. (0.1 Punkte)

Hinweis, mit *label="name"* können Sie den Linien eine Name (z.B, `plt.plot(x1,y1,label="1. Datensatz"))`) geben und mit `plt.legend()` diesen anzeigen.

In [49]:
```
Weg = [0,10,20,30,40]
Probe1 = [10,18,24,28,30]
Probe2 = [0,10,20,30,40]
plt.xlabel("Weg in mm")
plt.ylabel("Kraft-Probe in N")
plt.plot(Weg, Probe1, label="1. Datensatz")
```

```
plt.plot(Weg, Probe2, label="2. Datensatz")
plt.legend()
plt.grid(True)
plt.show()
```

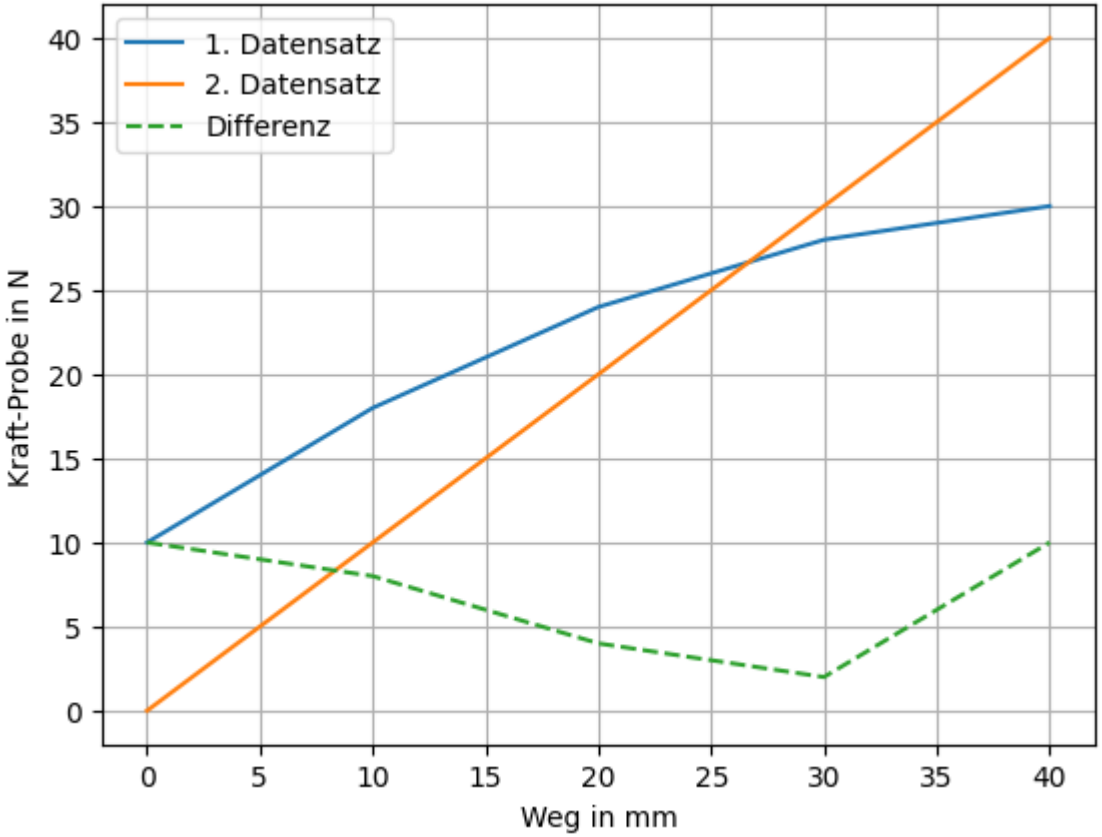


c) Stellen Sie nun die Differenz aus beiden Kraftmessungen dar. (0.1 Punkte)

Hinweis, mit klassischen Python-Listen mit denen wir bisher gearbeitet haben können wir keine Rechenoperationen durchführen. Ein möglicher Weg ist stattdessen ein **numpy** Array zu verwenden

In [55]:
```
import numpy as np
Weg = [0,10,20,30,40]
Probe1 = [10,18,24,28,30]
```

```
Probe2 = [0,10,20,30,40]

probe1_array = np.array(Probe1)
probe2_array = np.array(Probe2)

differenz = np.abs(probe1_array - probe2_array)
plt.xlabel("Weg in mm")
plt.ylabel("Kraft-Probe in N")
plt.plot(Weg, Probe1, label="1. Datensatz")
plt.plot(Weg, Probe2, label="2. Datensatz")
plt.plot(Weg, differenz, '--', label="Differenz")
plt.legend()
plt.grid(True)
plt.show()
```

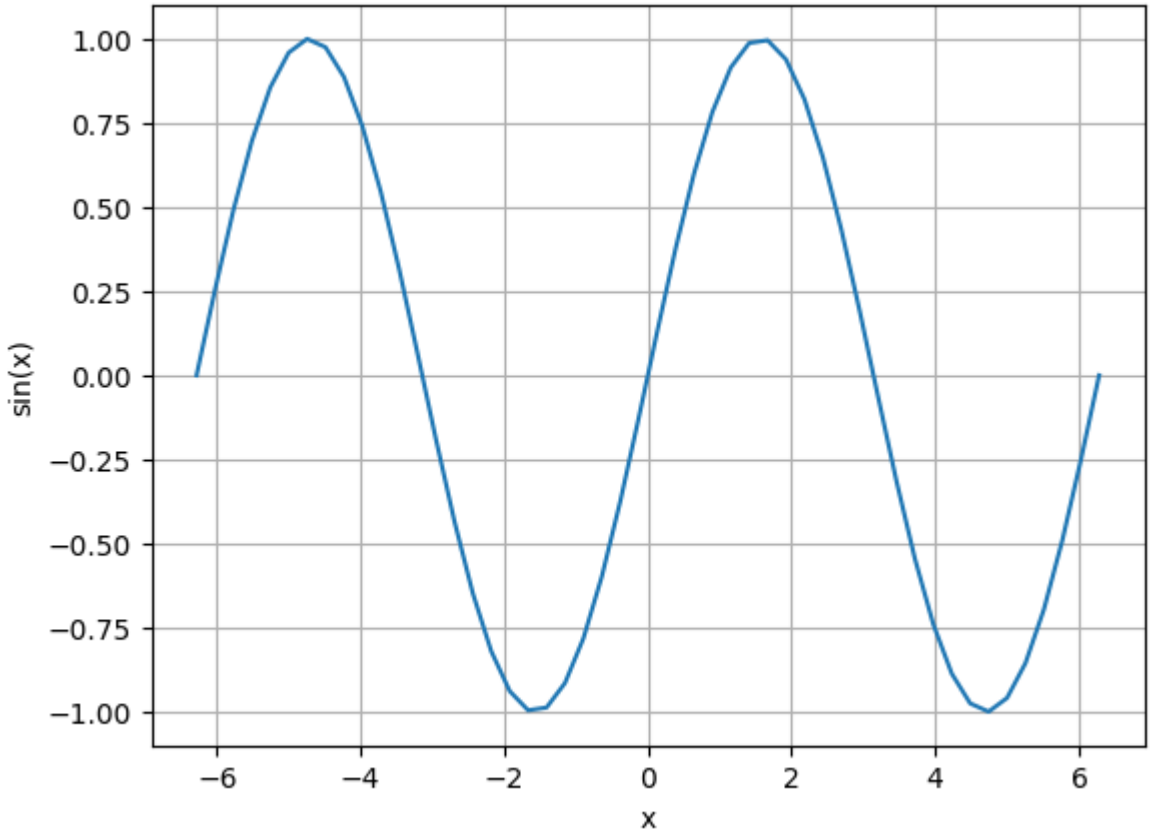


Aufgabe 3 : Funktionen in Pyplot.

a) Mit Hilfe der numpy Bibliotek , plotten Sie das Sinus Funktion in der Interval [-2π, 2π] . (0.1 Punkte)

Hinweise: die NumPy-Funktion **linspace** wird dazu benutzt, gleichmäßig verteilte Werte innerhalb eines spezifizierten geschlossene Intervalls zu erzeugen. Zum Beispiel `np.linspace(-np.pi , np.pi, 50, endpoint=True)` wobei n=50 der Anzahl von Werten zwischen Anfangswert und Endwert entspricht.

In [56]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2 * np.pi , 2 * np.pi, 50, endpoint=True)
y = np.sin(x)

plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(True)
plt.show()
```

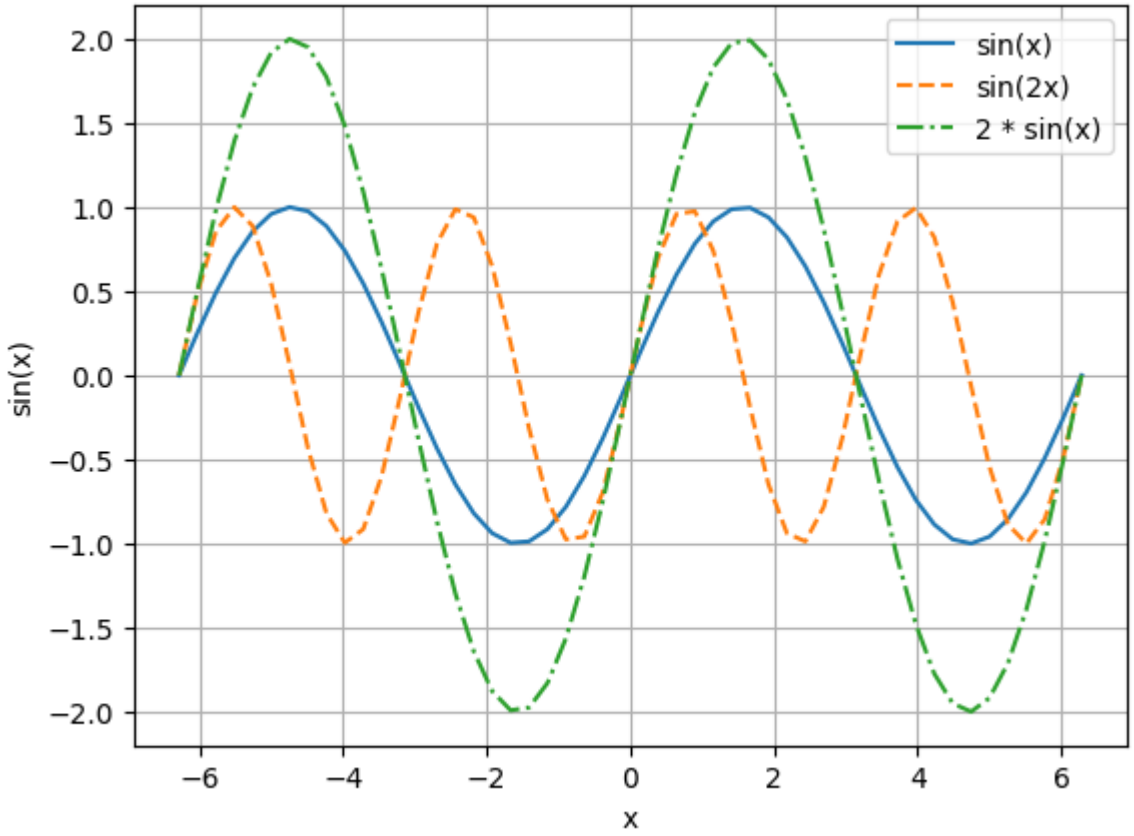


b) Erstelle zwei neue Sinusfunktionen: Eine mit der gleichen Amplitude, aber doppelter Frequenz und eine andere Funktion mit doppelter Amplitude und gleicher Frequenz. (0.1 Punkte)

Plotten Sie alle drei Funktionen in einen Graph. Benutzen Sie unterschiedliche Linienstile für jede Kurve. Hinweis, sie Können jede einzelne Funktion mit **plt.plot** plotten und alle drei gleichzeitig mit **plt.show** anzeigen lassen. Der Linienstil eines Plots kann durch die Parameter `linestyle=` oder `ls=` der **plot**-Funktion eingestellt werden. Sie können auf folgende Werte gesetzt werden: '-', '--', '-.', ':', 'None', ' ', ''.

In [66]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2 * np.pi , 2 * np.pi, 50, endpoint=True)
```

```
y = np.sin(x)
y1 = np.sin(2 * x)
y2 = 2 * np.sin(x)
plt.plot(x,y, label="sin(x)", ls='-')
plt.plot(x,y1, label="sin(2x)", ls='--')
plt.plot(x,y2, label="2 * sin(x)", ls='-.')
plt.legend()
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(True)
plt.show()
```



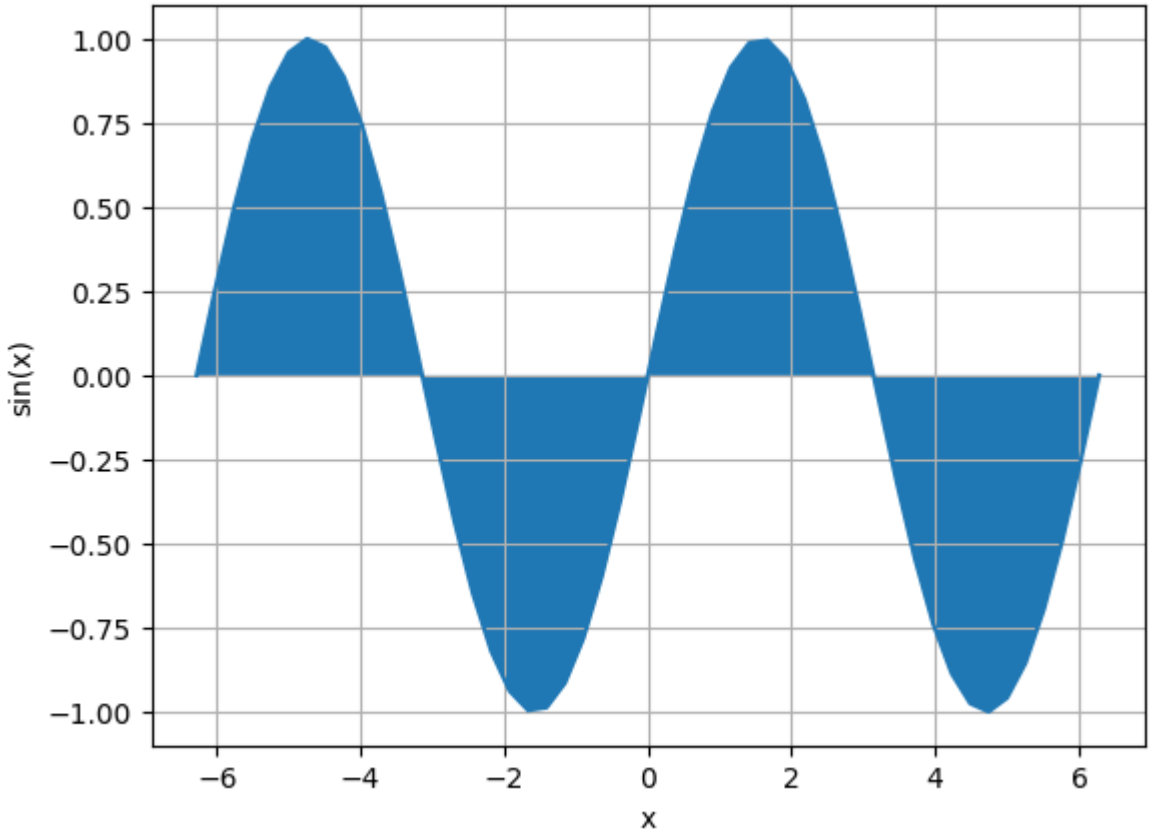
Mit der PyPlot-Funktion `fill_between` ist es möglich, Flächen zwischen Kurven oder Achsen zu schraffieren oder einzufärben. Zum Beispiel, `plt.fill_between(X,0,F,color='blue', alpha=0.1)` färbt die Fläche zwischen y=0 (untere Grenze) und y=F (obere

Grenze) entlang die X-Werte in durchsichtige blau.

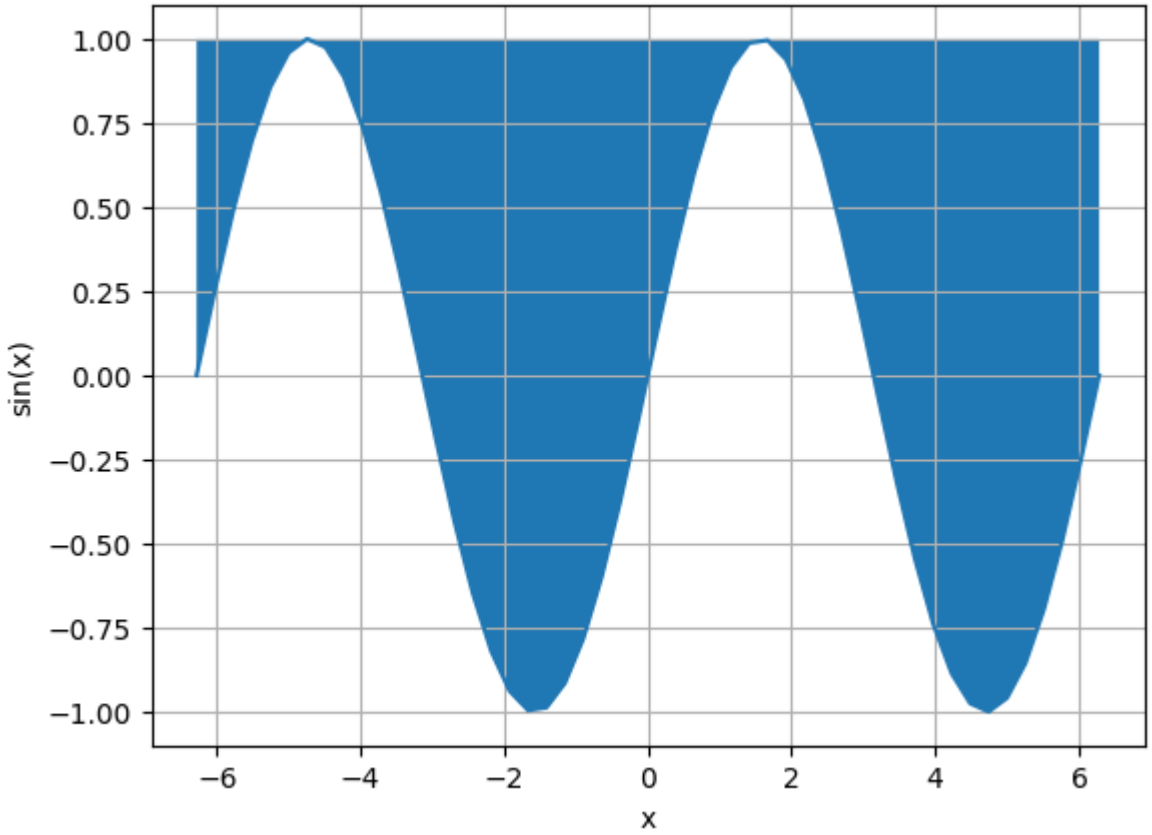
c) Erstellen Sie zwei Graphen der Sinus-Funktion von Aufgabe 4a. (0.2 Punkte)

Im ersten Graph befüllen Sie die Fläche zwischen der Y-Achse bei y=0 und die Sinus Kurve. Im zweiten Graph befüllen Sie die Fläche zwischen Y-Achse bei y=Ymax und die Sinus Kurve.

In [70]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2 * np.pi , 2 * np.pi, 50, endpoint=True)
y = np.sin(x)

plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.fill_between(x,0,np.sin(x))
plt.grid(True)
plt.show()
```



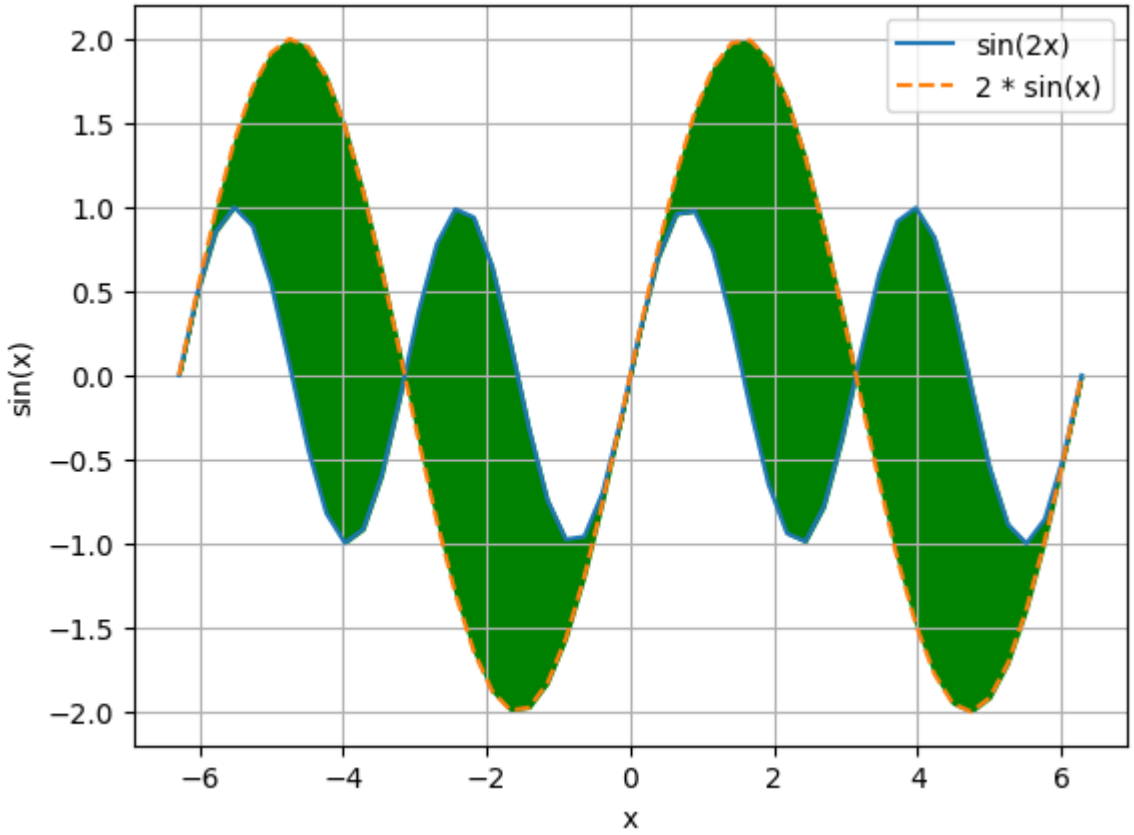
In [71]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2 * np.pi , 2 * np.pi, 50, endpoint=True)
y = np.sin(x)
ymax = np.max(y)
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.fill_between(x,y,ymax)
plt.grid(True)
plt.show()
```



d) Erstellen Sie einen Graphen mit beiden Sinus-Funktionen aus Aufgabe 4b und färben Sie die Fläche zwischen den beide Kurven Grün. (0.2 Punkte)

In [77]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2 * np.pi , 2 * np.pi, 50, endpoint=True)

y1 = np.sin(2 * x)
y2 = 2 * np.sin(x)
plt.plot(x,y1, label="sin(2x)", ls='-')
plt.plot(x,y2, label="2 * sin(x)", ls='--')
plt.legend()
```

```
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.fill_between(x, y1, y2, color="green")
plt.grid(True)
plt.show()
```



Aufgabe 4 : Gruppiertes Säulendiagramm

Die GDcG hat 2021 eine ausführliche Statistik sämtlicher Chemiestudiengänge durchgeführt (für mehr Information, siehe (https://www.gdch.de/ausbildung-karriere/karriere-und-beruf/hochschulstatistiken/statistik-chemiestudiengaenge.html). Die Tabelle 2 beschreibt die Anzahl der Studienanfängerinnen und -anfänger aller Chemiestudiengänge (inkl. Wirtschaftschemie) zwischen 2012 und 2021. Stellen Sie diese Informationen mit Hilfe der `plt.bar( )` Methode in einem Säulendiagramm dar.

Tabelle 2

Jahr	Gesamt	weibl%
2012	6095	37
2013	6755	37
2014	7003	38
2015	7345	40
2016	7019	40
2017	7174	42
2018	6433	43
2019	5746	45
2020	5671	45
2021	5129	47

Hinweise:

• Säulen- oder Balkendiagrammen lassen sich mit der Funktion plt.bar( ) darstellen. Die allgemeine Syntax ist:

```
plt.bar(x, height, width=0.8, bottom=None, align='center', label=" ")
```

Um Säulendiagramme übereinander zu stapeln, fügen wir alle Datensätze, die wir stapeln wollen, hinzu und übergeben die Summe als den Parameter bottom an die Methode plt.bar(). Zum Beispiel:

```
plt.bar(x,height1, bottom=0)
plt.bar(x,height2, bottom=height1)
```

• Mit der Funktion `plt.legend( )` ist es möglich Legenden in ein Plot zu platzieren.

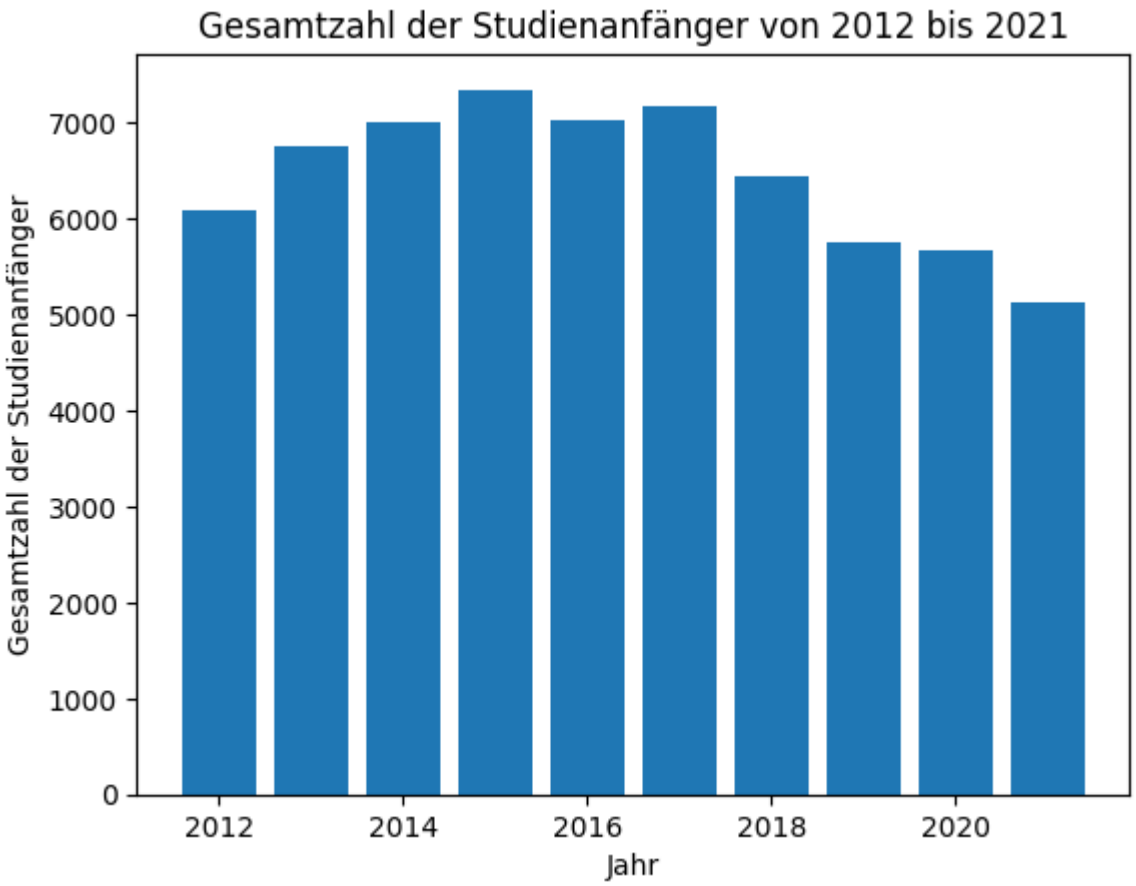
• Um die Spalten eines *csv* Datei lesen zu können, benutzen Sie:

```
a,b,c =np.loadtxt('gdch.csv', delimiter=',',unpack=True)
```

a) Erstellen Sie ein einfaches Säulendiagramm, dass die Gesamtzahl der Studienanfängern darstellt. (0.1 Punkte)

In [91]:
```
import matplotlib.pyplot as plt
import numpy as np

a,b,c =np.loadtxt('gdch.csv', delimiter=',', unpack=True)

plt.bar(a,b, width=0.8, bottom=None, align='center')
plt.title("Gesamtzahl der Studienanfänger von 2012 bis 2021")
plt.ylabel("Gesamtzahl der Studienanfänger")
```

```
plt.xlabel("Jahr")
plt.show()
```

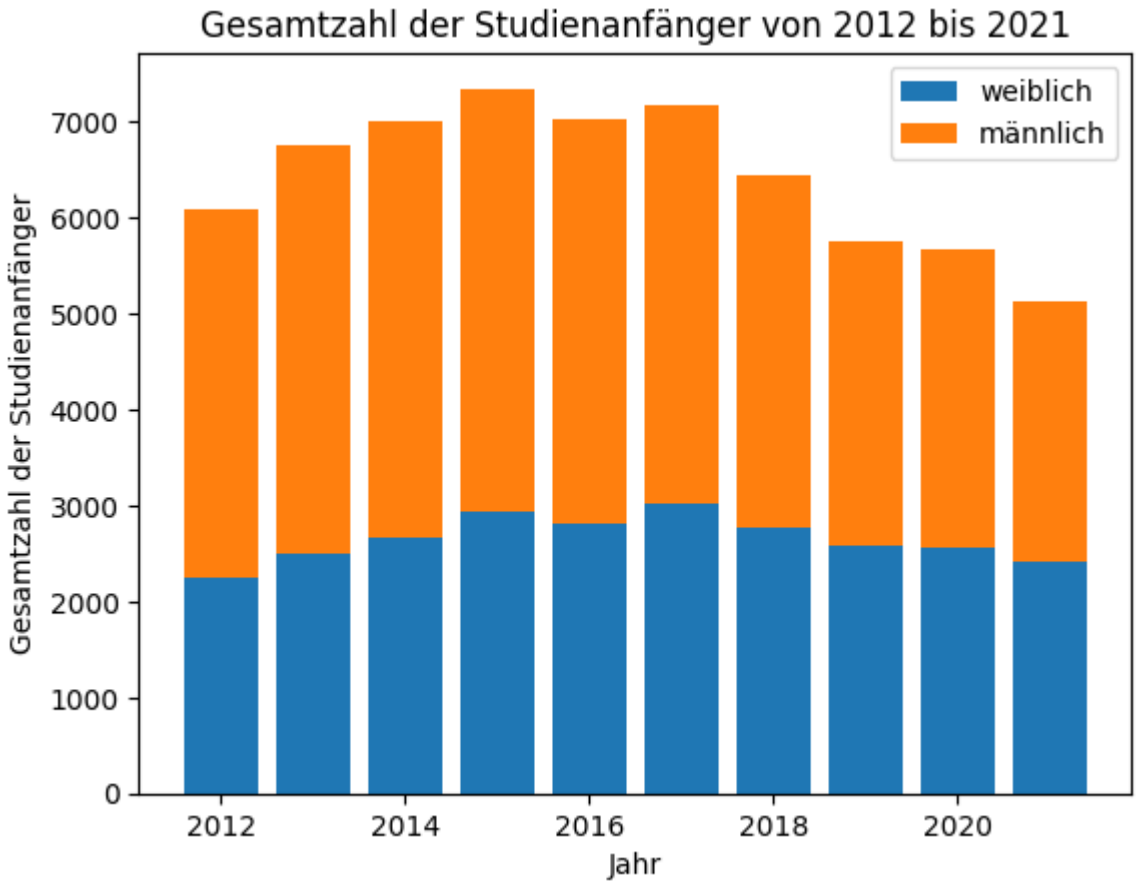


b) Erstelle Sie ein gestapeltes Säulendiagramm, in dem die Anzahl der männlichen und weiblichen Studienanfänger dargestellt wird. Denken Sie daran, die Achsen zu beschriften. (0.2 Punkte)

In [103…

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

a,b,c =np.loadtxt('gdch.csv', delimiter=',', unpack=True)
b_w = b * (c/100)
```

```
b_m = b - b_w
plt.bar(a,b_w, width=0.8, bottom=0, align='center', label="weiblich")
plt.bar(a,b_m, width=0.8, bottom=b_w, align='center', label="männlich")
plt.title("Gesamtzahl der Studienanfänger von 2012 bis 2021")
plt.ylabel("Gesamtzahl der Studienanfänger")
plt.xlabel("Jahr")
plt.legend()
plt.show()
```



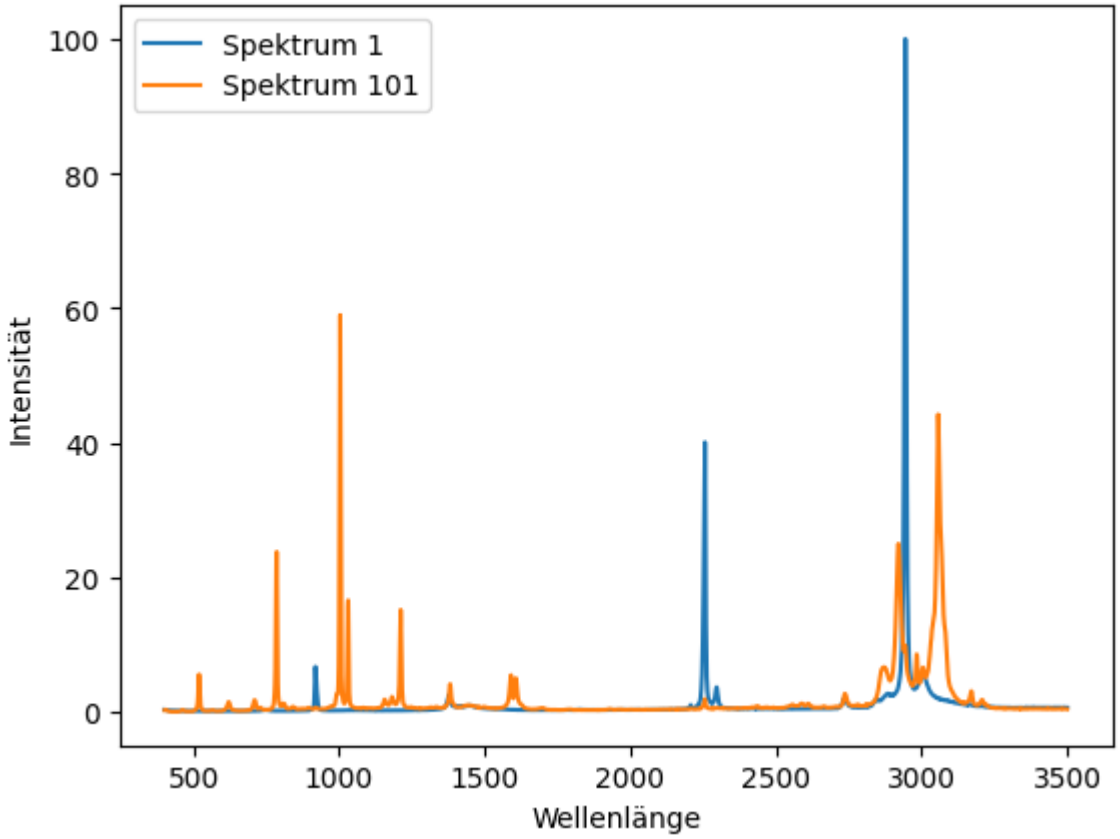
Aufgabe 5 : Darstellung von Schwingungsspektren

a) Laden Sie die Datei Mischungsspektren.cvs hoch und plotten Sie Spektrum 1 und Spektrum 101 gemeinsam. (0,1 Punkte)

Es handelt sich um ein Datensatz aus einer hypotetische spektroskopische Studie. Er enthält 101 Infrarotspektren, die Zeitiche Entwickung einer Reaktionsmischung beschreibt. Die 1.Spalte enthält die Wellenzahlen (450-3500 cm-1) und die restlichen Spalten die entsprechenden IR-Intensitäten der Mischung die in Zeitschriten von 1 Sekunden gemessen wurden. Nutzen Sie dafür alle befehlen die Sie bisher gelernt haben (Achsen und Spektren beschriften).

In [107…

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

data = np.loadtxt('Mischungsspektren.csv', delimiter=',')
wellenlänge = data[:, 0]
spektrum1 = data[:, 1]
spektrum101 = data[:, 100]

plt.plot(wellenlänge, spektrum1, label="Spektrum 1")
plt.plot(wellenlänge, spektrum101, label="Spektrum 101")
plt.xlabel("Wellenlänge")
plt.ylabel("Intensität")
plt.legend()
plt.show()
```



b) Plotten Sie den Konzentrations Verlauf der Edukte und den der Produkte. (0,2 Punkte)

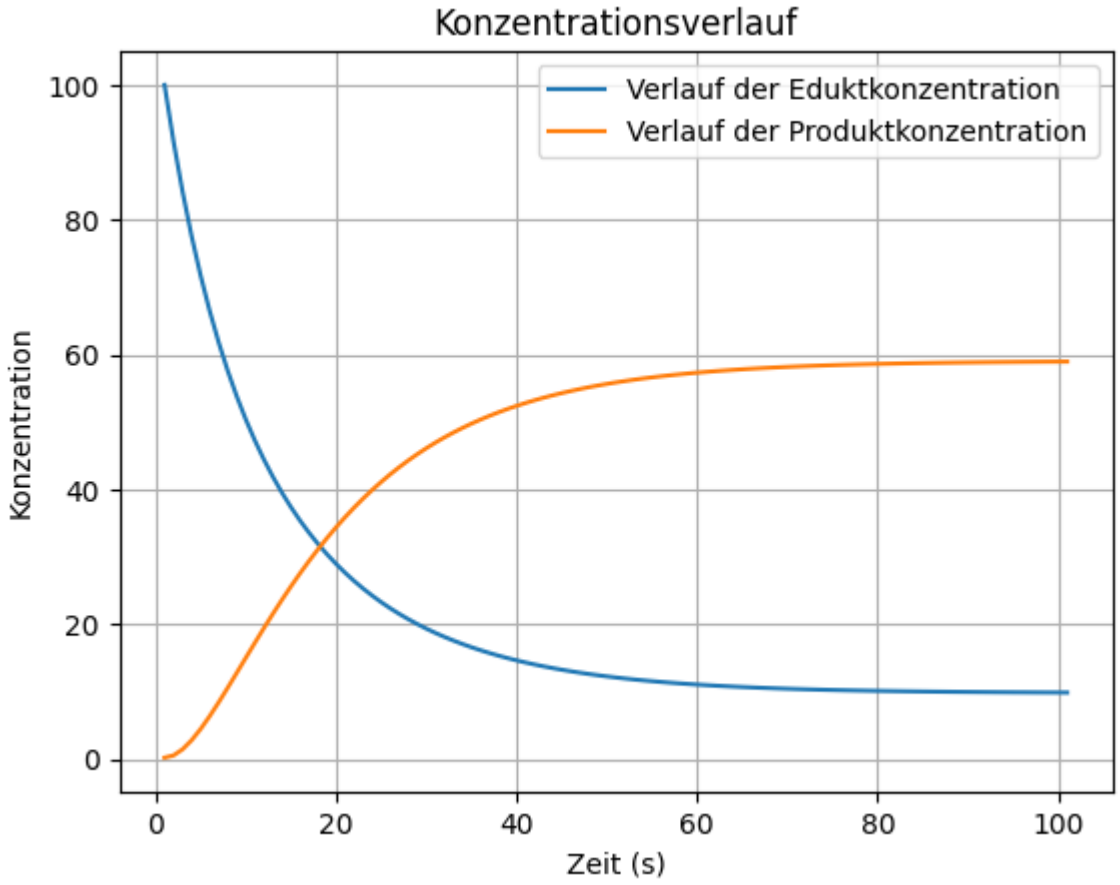
Nehmen Sie dabei an, das die Konzentration einer Spezies proportional ist zu der maximalen IR Intensität. Die NumPy Funktion `np.argmax(Array)` gibt den Index des größten Elements in einer Array wieder.

In [149…

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

data2 = data[:, 1:]
max = np.argmax(data2, axis=0)
print(max)
konzentration_edukt = max[:18]
konzentration_produkt = max[19:]
```

```
print(konzentration_edukt)
print(konzentration_produkt)

intensität_edukt = data2[2033, :]
intensität_produkt = data2[482, :]

time = np.arange(1, 102)

plt.plot(time, intensität_edukt, label="Verlauf der Eduktkonzentration")
plt.plot(time, intensität_produkt, label="Verlauf der Produktkonzentration")

plt.title("Konzentrationsverlauf")
plt.xlabel("Zeit (s)")
plt.ylabel("Konzentration")
plt.legend()
plt.grid(True)

plt.show()
```

```
[2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
 2033 2033 2033 2033  482  482  482  482  482  482  482  482  482  482
  482  482  482  482  482  482  482  482  482  482  482  482  482  482
  482  482  482  482  482  482  482  482  482  482  482  482  482  482
  482  482  482  482  482  482  482  482  482  482  482  482  482  482
  482  482  482  482  482  482  482  482  482  482  482  482  482  482
  482  482  482  482  482  482  482  482  482  482  482  482  482  482
  482  482  482]
[2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033 2033
 2033 2033 2033 2033]
[482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482
 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482
 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482
 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482
 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482 482
 482 482 482 482 482 482 482 482 482 482]
```



b) Bestimen Sie die Geschwindigkeitkonstant (k) diese Reaktion unter der Annahme dass der Verbrauch der Eduktes eine Reaktion 1.Ordnung folgt. (0,3 Punkte)

Für die Berechnung der Geschwindigkeitkonstante können Sie das Konzept der Halbwertzeit ($ \tau = ln(2)/k $) benutzen.

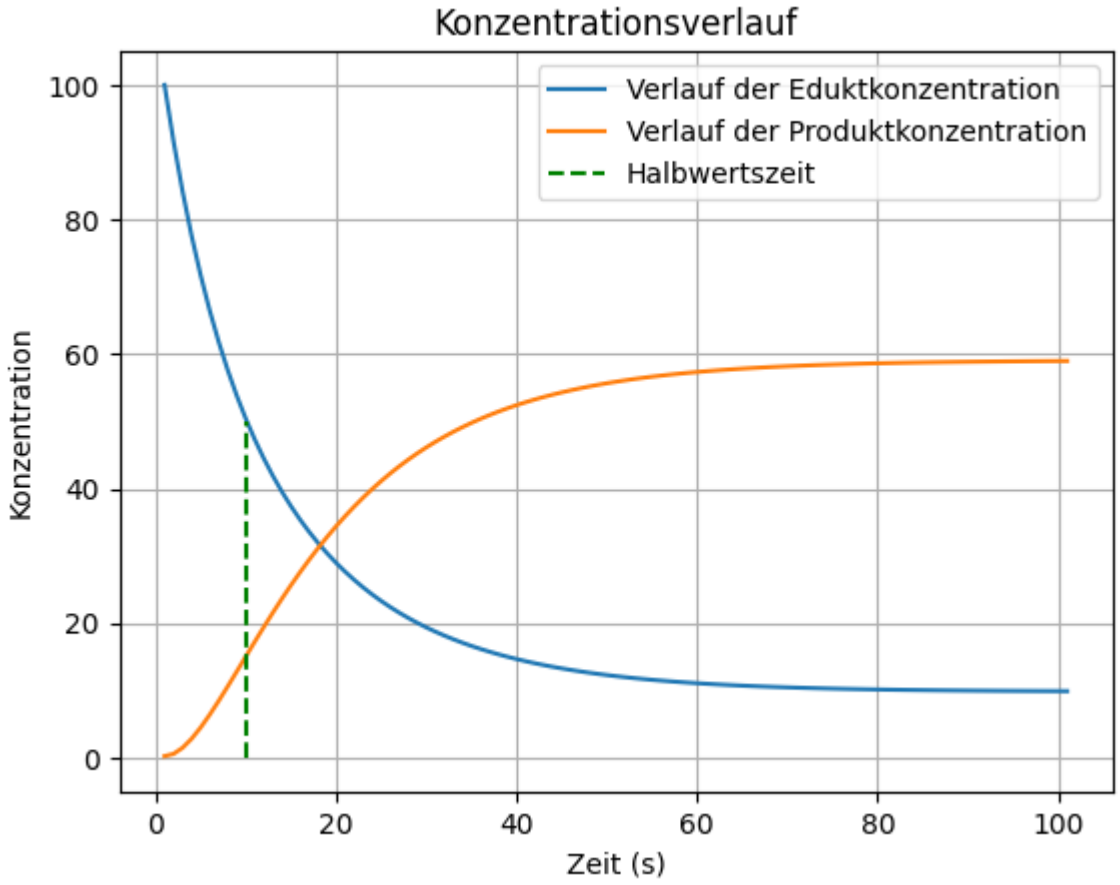
Stellen Sie das Wertepaar (τ, Intensität(τ)) im Konzentrationsverlauf Diagramm grafisch dar.

Horizontalen und Vertikalen Linien lassen sich mit `plt.hlines(y,xmin,xmax)` und `plt.vlines(x,ymin,ymax)` dasrstellen (siehe https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hlines.html)

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
data = np.loadtxt('Mischungsspektren.csv', delimiter=',')
data2 = data[:, 1:]
max = np.argmax(data2, axis=0)
#print(max)
konzentration_edukt = max[:18]
konzentration_produkt = max[19:]
#print(konzentration_edukt)
#print(konzentration_produkt)

intensität_edukt = data2[2033, :]
intensität_produkt = data2[482, :]
tau = intensität_edukt[0] / 2
print(tau)
k = np.log(2) / tau
print("Die Geschwindigkeitskonstante k:",k)
time = np.arange(1, 102)

x = np.where((intensität_edukt >= 50) & (intensität_edukt <= 51))[0]
print(x)
plt.plot(time, intensität_edukt, label="Verlauf der Eduktkonzentration")
plt.plot(time, intensität_produkt, label="Verlauf der Produktkonzentration")
plt.vlines(10, 0, 50, colors='green', linestyle= 'dashed', label="Halbwertszeit")
plt.title("Konzentrationsverlauf")
plt.xlabel("Zeit (s)")
plt.ylabel("Konzentration")
plt.legend()
plt.grid(True)

plt.show()
```

```
50.0
Die Geschwindigkeitskonstante k: 0.013862943611198907
[9]
```



In [ ]:

Aufagbe 6: Scatter Plots und Histogramm

`plt.scatter()` ist die einfachste und standarisierte Methode, um Daten als Punkte in der Matplotlib darzustellen. Wir übergeben die X- und Y- Koordinaten der Datenpunkte als Argumente an die `plt.scatter()` - Methode, um ein Streudiagramm zu erzeugen. Das Streudiagramm lässt sich anpassen in dem man die Parameter `color` und `marker` (siehe Zeichen Tabelle in Aufgabe 1) in der `plt.scatter()` Methode ändert.

Zum Beispiel:

```
x=[3,1,6,3,0,2]
y=[4,1,1,2,8,7]
plt.scatter[x,y,color="red",marker="o"]
plt.plot()
```

Ein **Histogramm** zeigt die Verteilung von numerischen Daten an. Es nutzt kontinuierliche Intervalle, um die Daten in bestimmte Abschnitte einzuteilen. Der Hauptunterschied zwischen einem Histogramm und einem Balkendiagramm ist der Typ der verarbeiteten Daten: Während Histogramme kontinuierliche Daten verarbeiten, stellen Balkendiagramme kategoriale Daten dar.

Matplotlib verwendet die `hist()` Funktion, die ein Array von zufälligen oder definierten Werten verwendet, um das Histogramm zu erstellen. Bei einem Histogramm verwenden wir eine Reihe von Intervallen (*Bins*), um den Bereich der gegebenen Werte auf der x-Achse darzustellen. Die y-Achse hingegen repräsentiert die Frequenzinformationen.

Die `hist()` Funktion übernimmt mehrere Parameter, einschließlich:
x: Dies repräsentiert die Array-Sequenz,
bins: Dies ist ein optionaler Parameter, der nicht überlappende Intervalle von Variablen darstellt, die ganze Zahlen oder eine Zeichenfolgenfolge enthalten können.
range: Definiert den oberen und unteren Bereich der Bins
color: Definiert die Farbe der Balken.
alpha: Definiert die Tranzparent-stärke der Balken
rwidth: Setzt die relative Breite der Balken im Histogramm auf die des Bins.
log: Der Log-Parameter definiert eine Log-Skala auf der Achse eines Histogramms.

Zum Beipiel:

```
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27],color="pink")
plt.hist(a, bins = [0,25,50,75,100],color="pink")
plt.xticks([0,25,50,75,100])
plt.ylabel("Frequenz")
plt.xlabel("Bins")
plt.plot()
```

Die Numpy Funktion `count,bin = np.histrogram(X)` hat zwei Rückgabewerte: die das Wertearray des Histogramms angeben

(*counts*) und ein Array des Float-Datentyps, der die Bin-Kanten enthält (*bin*).

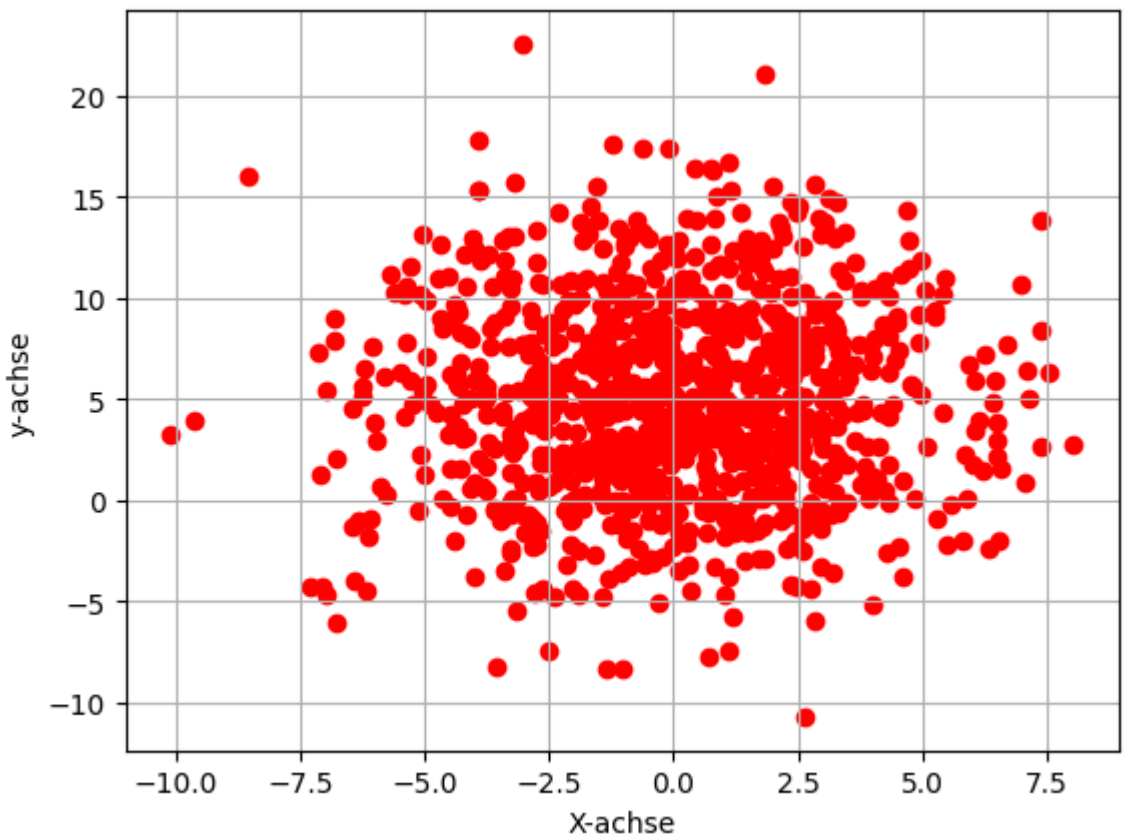
Zufallswerte werden mit der Numpy **random** Funktion generiert. Die Funktion `np.random.random()` erzeugt gleichverteilte Zufallszahlen im Intervall [0,1). Ausgegeben wird diese Zahl als *float*. Optional kann auch der Parameter **size** übergeben werden , um ein NumPy array zu erzeugen (**size** kann dabei entweder ein integer sein und beschreibt dann die Länge eines eindimensionalen NumPy-Arrays, oder ein Tuple, der die Ausmaße des mehrdimensionalen Arrays in jeder Richtung beschreibt). Zusätzlich, kann auch die Numpy `np.random.normal()` Funktion angewendet werden, um zufällige Werte mit einer Normal Verteilung ($ f(x)=1/(\sigma\sqrt(2\pi)) exp(-1/2((x-\mu)/\sigma)^2)$) zu generieren. Die Parameter μ und σ repräsentieren den *Erwartungswert* und die *Standardabweichung* der Verteilung, und werden durch die Parameter **loc** und **scale** in der `np.random.normal()` Funktion übergeben. (https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html)

a) Generieren Sie mit Hilfe von `np.random.normal()` zwei eindimensionale Arrays (X, Y) mit jeweils 1000 Zufallszahlen aus einer Normalverteilung und stellen Sie sie mit Hilfe von `plt.scatter()` als zweidimmensionalen Punktplot dar. (0,1 Punkte)

Nutzen Sie dabei die Mittelwerte $ \mu_x $= 0 und $ \mu_y $= 5 sowie die Standardabweichungen σ_x= 3.0 und σ_y= 5.0. Beschriften Sie bitte die Achsen.

In [98]:
```
import numpy as np
import matplotlib.pyplot as plt
mu_x = 0
mu_y = 5
sigma_x = 3.0
sigma_y = 5.0
x = np.random.normal(mu_x, sigma_x, 1000)
y = np.random.normal(mu_y, sigma_y, 1000)
plt.scatter(x,y, color="red", marker="o")
plt.xlabel("X-achse")
plt.ylabel("y-achse")
plt.grid(True)
plt.plot()
```

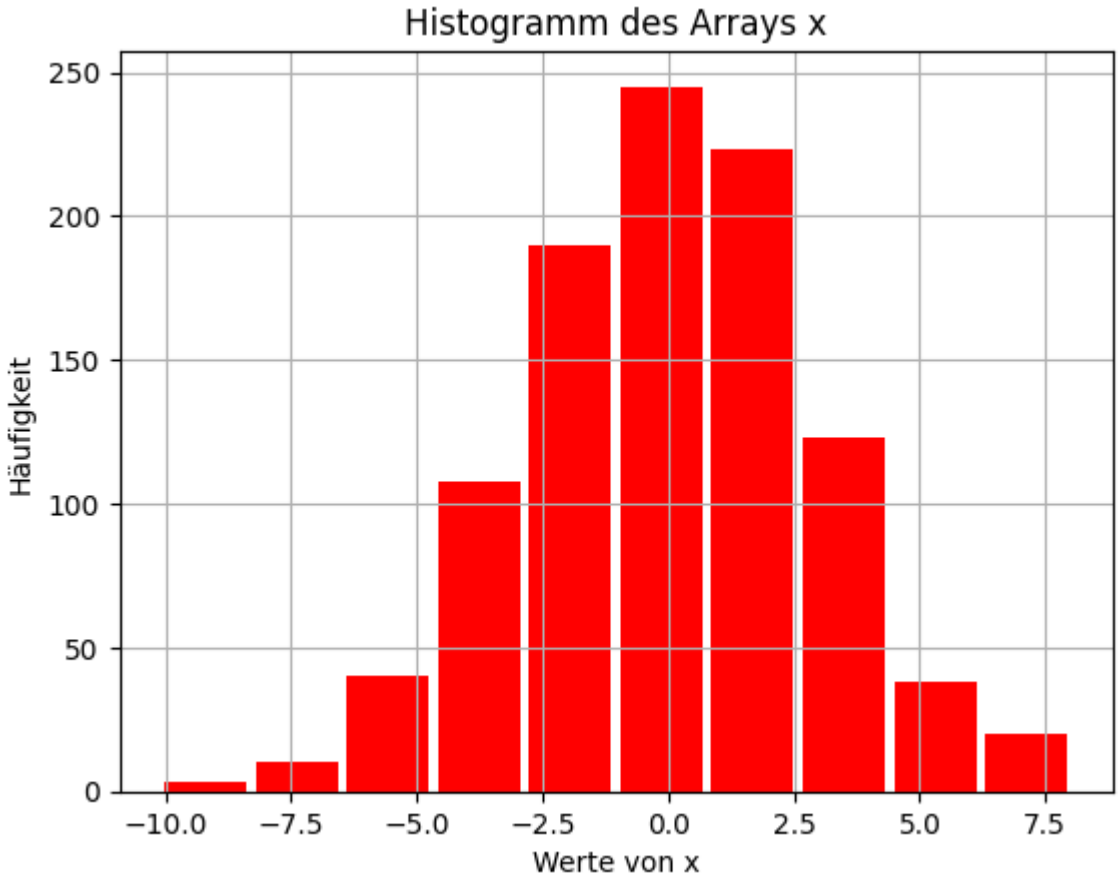
Out[98]:
```
[]
```



b) Erzeugen Sie ein Histogramm für das Array X. (0,1 Punkte)

Welche Bin-edges werden für die Darstellung automatisch werwendet?

In [99]:
```
array, bin_edges, color= plt.hist(x, color='red', rwidth=0.9)
plt.xlabel("Werte von x")
plt.ylabel("Häufigkeit")
plt.title("Histogramm des Arrays x")
plt.grid(True)
plt.show()
print("Die automatisch verwendeten bin_edges sind:")
print(bin_edges)
```



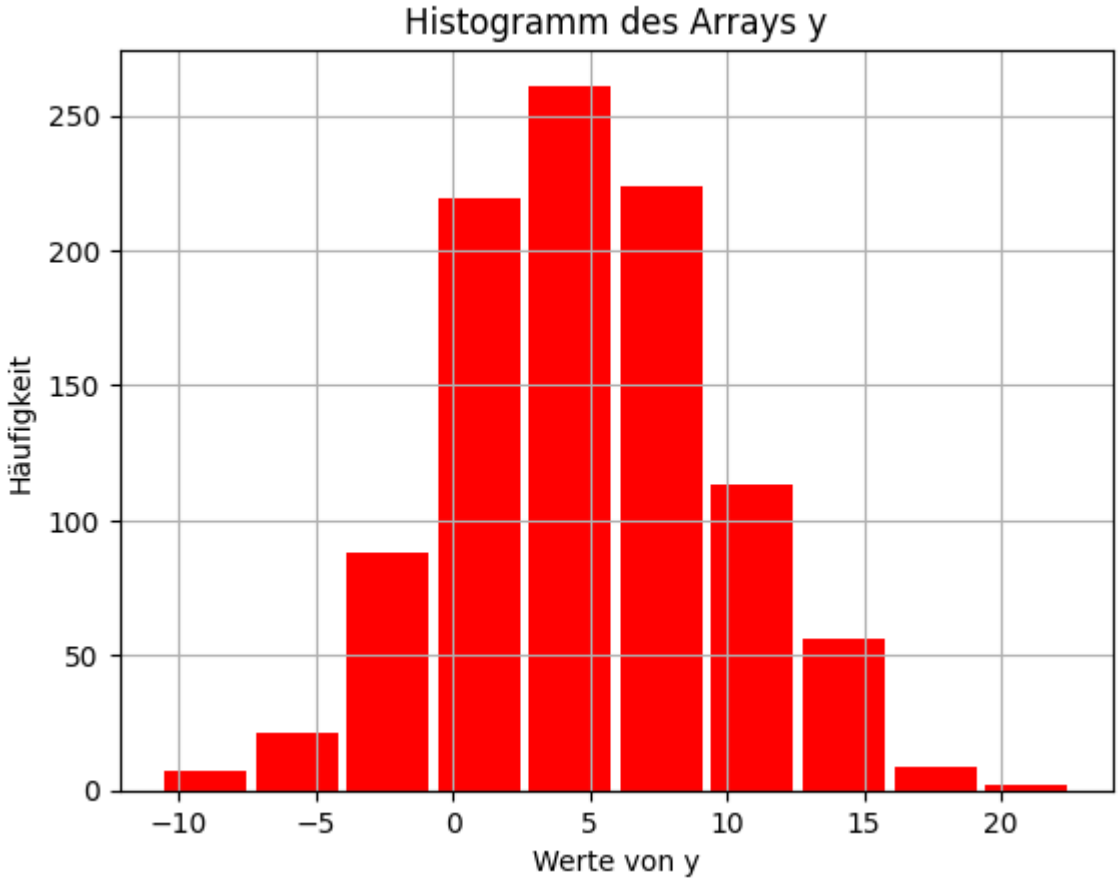

```
Die automatisch verwendeten bin_edges sind:
[-10.10371365  -8.29092213  -6.47813061  -4.66533909  -2.85254757
  -1.03975604   0.77303548   2.585827     4.39861852   6.21141004
   8.02420156]
```

c) Erzeuge ein Histogramm für das Array Y. (0,1 Punkte)

Welche Bin-edges werden für die Darstellung automatisch werwendet? Kann man beide Histogramme miteinander vergleichen?

In [105…
```
array, bin_edges, x= plt.hist(y, color='red', rwidth=0.9)
plt.xlabel("Werte von y")
plt.ylabel("Häufigkeit")
plt.title("Histogramm des Arrays y")
```

```
plt.grid(True)
plt.show()
print("Die automatisch verwendeten bin_edges sind:")
print(bin_edges)
# Um die Histogramme miteinander vergleichen zu können,
# würde man die bin-edges festlegen um identische intervalle für beide diagramme zu erhalten.
```



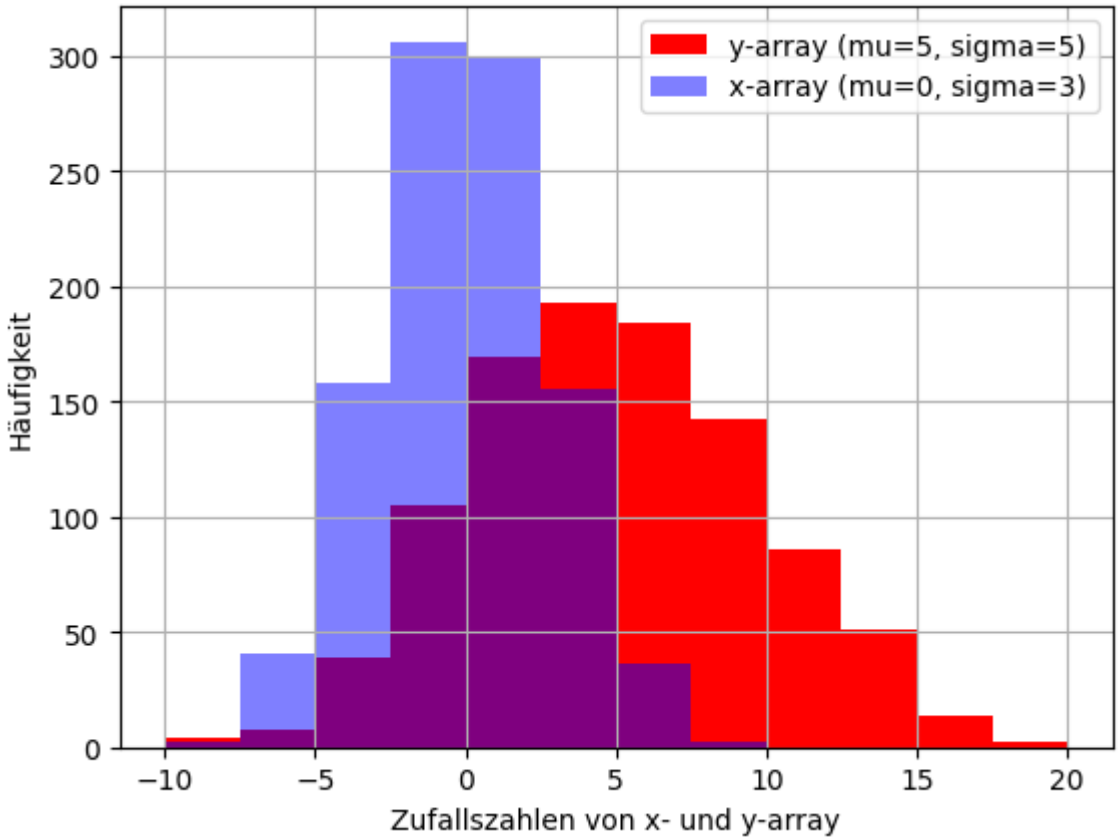

```
Die automatisch verwendeten bin_edges sind:
[-10.69358718  -7.36628524  -4.03898329  -0.71168135   2.6156206
   5.94292254   9.27022449  12.59752643  15.92482838  19.25213033
  22.57943227]
```

d) Erzeugen Sie Histogramme für die Arrays X und Y und stellen Sie sie gemeinsam in einer

Abbildung dar. Benutzen Sie dieses Mal die gleiche Bin-Array. (0,2 Punkte)

Wie würden sie die Histogramme voneinander unterscheiden? Fügen Sie die entsprechenden Legenden ein.

In [104…
```
plt.hist(y, bins=[-10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20], color='red', label="y-array (mu=5,
plt.hist(x, bins=[-10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20], color='blue',alpha=0.5, label="x-ar
plt.xlabel("Zufallszahlen von x- und y-array")
plt.ylabel("Häufigkeit")
plt.grid(True)
plt.legend()
plt.show()
print("Um beide histogramme zu unterscheiden wurde jeweils eine andere Farbe verwendet und eins der beiden Histogra
```




```
Um beide histogramme zu unterscheiden wurde jeweils eine andere Farbe verwendet und eins der beiden Histogramme durc
hsichtiger dargestellt.
```

Aufgabe 7: Plotten von Funktionen von zwei Veränderlichen

Zum Plotten von Funktionen von 2 Veränderlichen z=f(x,y) braucht Matplotlib drei Matrizen, welche jeweils die x-, y- und z-Werte zu jedem Datenpunkt enthalten. Zur Erzeugung dieser Matrizen kann man die Numpy-Funktion `meshgrid()` benutzen:
Zum Beispiel, die 2D Gaussian Funktion $f(x,y)=exp(-x^2-y^2)$ lässt sich mit den folgenden Befehlen darstellen:

```
def f(x,y):
   return (np.exp(-x**2 - y**2)

n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X, Y = np.meshgrid(x, y)

ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, f(X,Y))
plt.show()
```

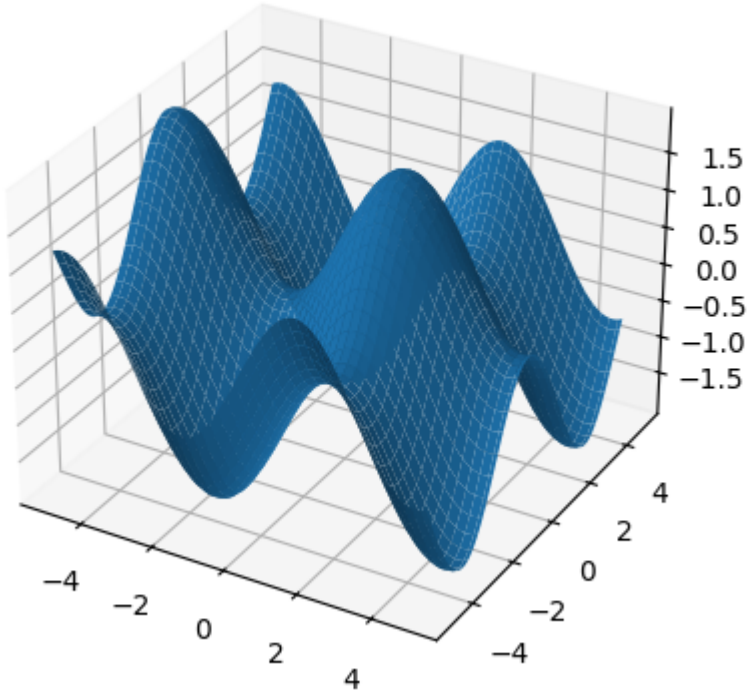
Die Methode `plt.contour(X,Y,Z,[levels])` zeichnet aus den X- Y- und Z- Daten eine Karte von "Höhenlinien", d.h. Linien, entlang derer Z- Wert konstant bleibt. Die Positionen der Konturlinien werden im Array `[levels]` in aufsteigender Reihenfolge angegeben. Zwischen den Gitterpunkten wird intrapoliert. Die Methode `plt.contourf(X,Y,Z,[levels])` erzeugt ebenfalls Höhenlinien, füllt aber die Flächen zwischen diesen Linien farbig aus. Für mehr Informationen siehe https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.contourf.html

a) Stellen Sie die Funktion $ f(x,y)= sin(x)+cos(y)$ im Intervall [-5, 5] in ein dreidimensionales Diagramm dar. (0,1 Punkte)

Setzen Sie eine Auflösung (n) von 256 Werten ein.

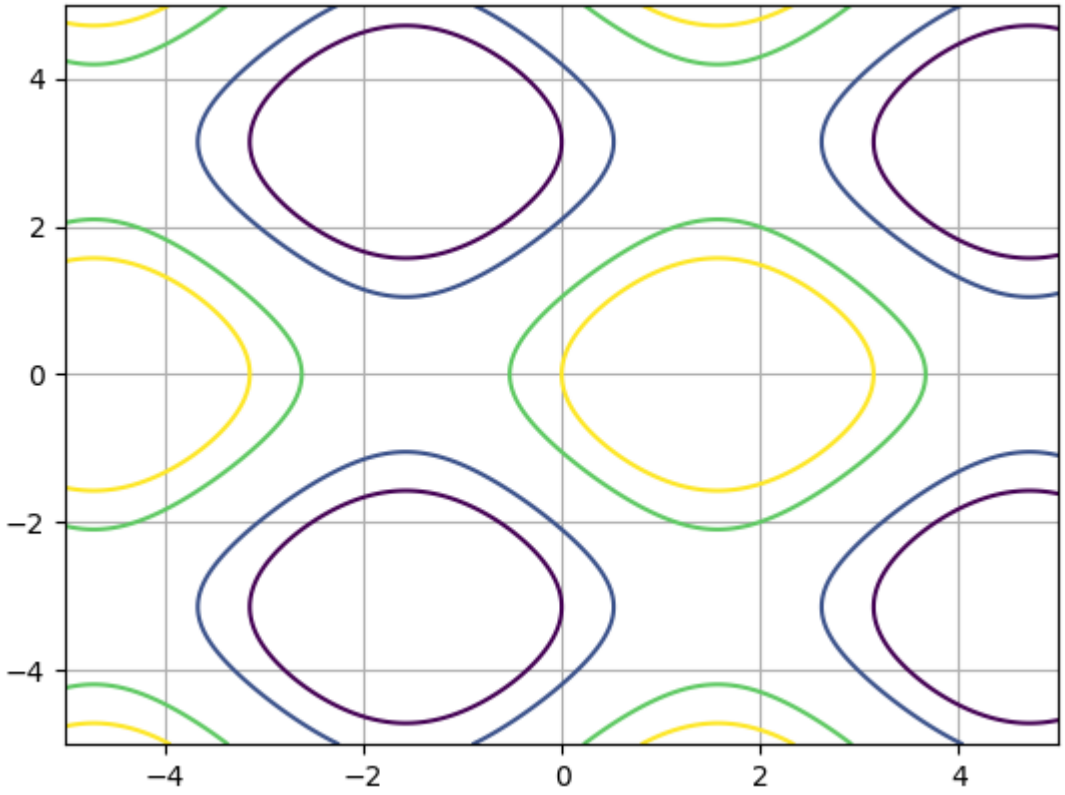
In [93]:
```
import numpy as np
import matplotlib.pyplot as plt
def f(x,y):
   return (np.sin(x) + np.cos(y))

n = 256
x = np.linspace(-5, 5, n)
y = np.linspace(-5, 5, n)
h = np.meshgrid(x, y)
```

```
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, f(X,Y))
plt.grid(True)
plt.show()
```



b) Stellen Sie das Kontourplot der Funktion $ f(x,y)= sin(x)+cos(y)$ unter berücksichtigung der folgenden Ebenen: [-1,-0.5,0,0.5,1]. (0,1 Punkte)

In [91]:
```
plt.contour(X, Y, f(X,Y), [-1,-0.5,0.5,1])
plt.grid(True)
plt.show()
```



In [ ]: