termin2

December 18, 2024

Gruppe:
Luca Kässner
Marlene Conrad
Yassin Starzetz

1 Kapitel 2 Teil 1

Übung 1:

Erstellen Sie zwei Arrays, in welchen Sie die Werte aus der Tabelle speichern, wobei das erste Array die x-Werte und das zweite die y-Werte enthalten soll.

x	y
20.1	5.3
8.5	3.2
1.7	9.8
4.7	7.8
6.6	21.9

[42]:
```
import numpy as np

a = np.array([20.1, 8.5, 1.7, 4.7, 6.6]) # ein-dimensionales Array aller x Werte
b = np.array([5.3, 3.2, 9.8, 7.8, 21.9]) # ein-dimensionales Array aller y Werte
print("x-Werte: ", a)
print("y-Werte: ", b)
```

```
x-Werte:  [20.1  8.5  1.7  4.7  6.6]
y-Werte:  [ 5.3  3.2  9.8  7.8 21.9]
```

Übung 2: Betrachten Sie das Array `C=(10, 20, 30, 40, 50, 60, 70, 80, 90, 100)` und formulieren Sie Vermutungen, was die folgenden Befehle ausgeben werden. Vergleichen Sie Ihre Vermutung mit den Ergebnissen und schreiben Sie Ihre Schlussfolgerungen in einer Markdown-Zelle.

- C[9]
- C[-1]
- C[1:3]
- C[1:]
- C[:5]

- C[0:6:2]
- C[::2]

Vermutung - C[9] gibt das 9. Element wieder [100] - C[-1] gibt das 9. Element wieder [100] - C[1:3] gibt das 1. bis 2. Element wieder [20, 30] - C[1:] gibt alles nach dem 1. Element wieder [20, 30, 40, 50, 60, 70, 80, 90, 100] - C[:5] gibt alles vor dem 5. Element wieder [10, 20, 30, 40, 50] - C[0:6:2] gibt vom 0. bis 6. Element jedes zweite wieder [10, 30, 50] - C[::2] gibt jedes zweite Element wieder [10, 30, 50, 70, 90]

[43]:
```
a = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100]) # Array mit 10 Elementen
print("C[9]    ", a[9])
print("C[-1]   ", a[-1])
print("C[1:3]  ", a[1:3])
print("C[1:]   ", a[1:])
print("C[:5]   ", a[:5])
print("C[0:6:2]", a[0:6:2])
print("C[::2]  ", a[::2])
```

```
C[9]     100
C[-1]    100
C[1:3]   [20 30]
C[1:]    [ 20  30  40  50  60  70  80  90 100]
C[:5]    [10 20 30 40 50]
C[0:6:2] [10 30 50]
C[::2]   [10 30 50 70 90]
```

Ich lag wohl richtig mit meinen Vermutungen :)

Übung 3:

Erstellen Sie ein Array mit 20 Zahlen zwischen 1 und 10.

[44]:
```
a = np.linspace(1,10, 20) # generiert ein Array mit 20 Zahlen zwischen 1 und 10
print(a)
print("Länge: ", a.shape[0])
```

```
[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
Länge:  20
```

Übung 4:

Wir beginnen mit einem Beispiel. Sie möchten die Körpergröße Ihres Freundes wissen und haben dafür eine Messreihe aufgenommen:

- 180 cm
- 179 cm
- 181 cm
- 182 cm
- 178 cm

- 179.7 cm

Die Standardabweichung der Stichprobe σ kann bekanntlich mithilfe der folgenden Formel berechnet werden:

$$\sigma^2_{sample} = \frac{1}{N-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Hierbei sind x_i die Werte der individuellen Datenpunkte, $\overline{x}$ der Mittelwert der Datenreihe und N die Anzahl der Datenpunkte.

Versuchen Sie, anstelle die Formel zu nutzen, den Mittelwert und die Standardabweichung direkt mit den Numpy-Funktionen **mean** und **std** zu berechnen. Geben Sie das Ergebnis im Format {: .3f} aus.

Achtung: Die Numpy-Funktion **std** dividiert durch `N-ddoF`. Dabei ist `N` die Anzahl der Elemente und `ddof` die Anzahl der Freiheitsgrade ("Delta Degrees of Freedom"). Wird nichts anderes angegeben, setzt Python `ddoF` standardmäßig auf null. Ist allerdings bspw... der Term `N-1` im Nenner gewünscht, so muss im Argument der Funktion `ddoF=1` spezifiziert werden.

[45]:
```
a = np.array([180, 179, 181, 182, 178, 179.7])
std = np.std(a, ddof=1)
print(f"Die Standardabweichung beträgt: {std:.3f}cm")
```

```
Die Standardabweichung beträgt: 1.420cm
```

Übung 5:

Berechnen Sie nun die Unsicherheit des Mittelwertes:

$$\sigma_{\overline{x}} = \frac{t * \sigma_{sample}}{\sqrt{N}}$$

Da die Anzahl der Messungen N klein ist, muss der Studentfaktor t mit einbezogen werden. Für $N = 6$ ist $t = 1{,}1$. Um die Unsicherheit des Mittelwertes zu berechnen, wird die Länge N des Arrays benötigt. Diese kann mithilfe des **len()**-Befehls ermittelt werden.
Geben Sie das Ergebnis im {: .2f}-Format aus.

[46]:
```
import math

def mean_deviation(deviation: float, t: float) -> float:
    """function to determine the deviation of elements in an array to the␣
↪mean"""
    return t*deviation/math.sqrt(np.shape(a)[0])

a = np.array([180, 179, 181, 182, 178, 179.7]) # Array der Messwerte
std = np.std(a, ddof=1) # Standardabweichung
s_faktor = 1.1 # Studentenfaktor
```

```python
md = mean_deviation(std, s_faktor)
print(f"Unsicherheit des Mittelwerts: {md:.2f}")
```

```
Unsicherheit des Mittelwerts: 0.64
```

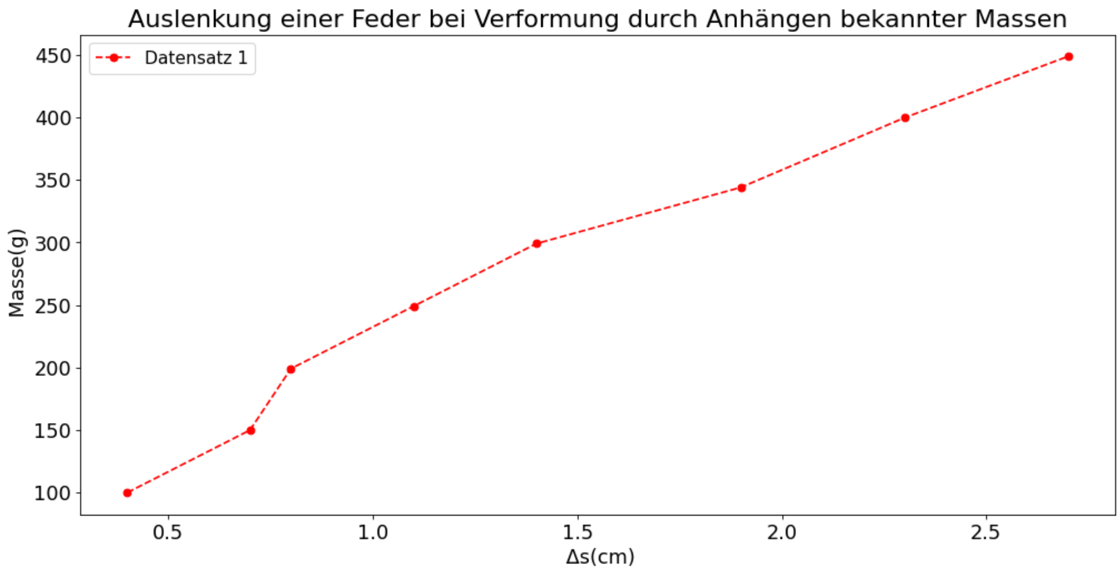
Übung 6:

Die folgenden Daten beschreiben den Betrag der Auslenkung einer Feder bei Verformung durch Anhängen bekanntermassen.

Erstellen Sie ein Diagramm für diese Daten und fügen Sie einen Titel, eine Legende und Achsenbeschriftungen hinzu. Wählen Sie zusätzlich den Linienstil, die Markierungssymbole und deren Farbe aus.

Δs(cm)	Masse(g)
0.4	100.0
0.7	150.0
0.8	199.1
1.1	249.1
1.4	299.1
1.9	344.1
2.3	399.8
2.7	448.9

[47]:
```python
import matplotlib.pyplot as plt

x = [0.4, 0.7, 0.8, 1.1, 1.4, 1.9, 2.3, 2.7] # Δs(cm)
y = [100.0, 150.0, 199.1, 249.1, 299.1, 344.1, 399.8, 448.9] # Masse(g)

plt.figure(figsize=(15, 7), dpi=80)
plt.plot(x,y,'ro--', label="Datensatz 1")
plt.xlabel(' Δs(cm)',fontsize=16)  # Beschriftung der x-Achse
plt.ylabel(' Masse(g)',fontsize=16)   # Beschriftung der y-Achse
plt.title('Auslenkung einer Feder bei Verformung durch Anhängen bekannter␣
 ↪Massen', fontsize=20) # Titel
plt.legend(fontsize=14) # Anzeige einer Legende im Diagramm
plt.show() # Ausgabe des Diagramms
#plt.scatter()
```



2 Kapitel 2 Teil 2

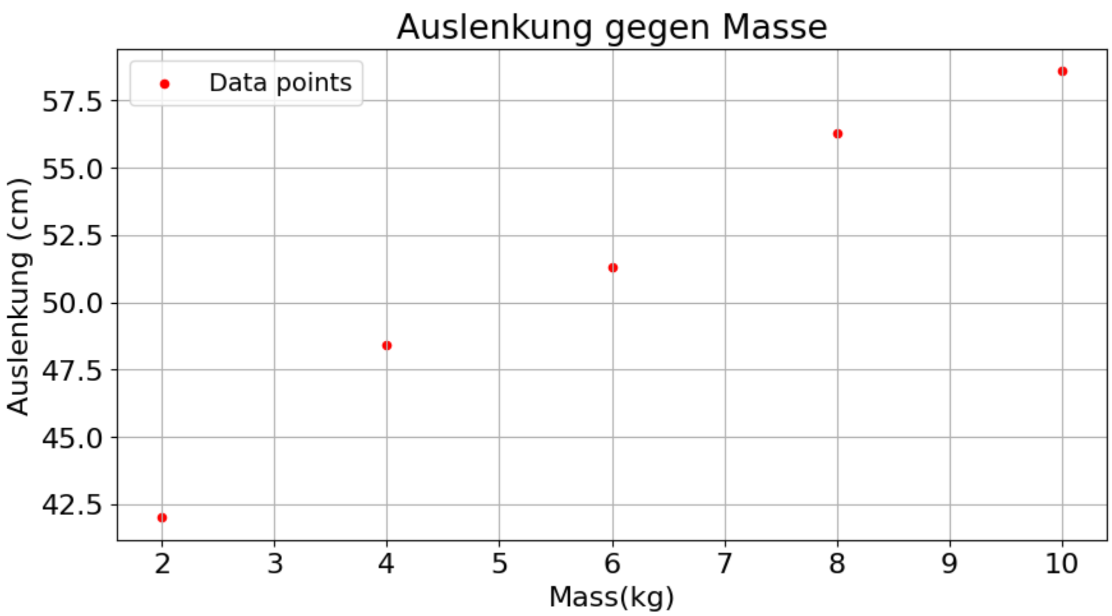
[48]:
```python
# benötigte Module
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import curve_fit

# Plot-Parameter
plt.rcParams['figure.figsize'] = (10,5) # Größe der Abbildung
plt.rcParams['font.size'] = 16          # Schriftgröße des Textes
plt.rcParams['legend.fontsize'] = 14    # Schriftgröße der Legende
#plt.rcParams['lines.linestyle'] = '--' # Linienart
#plt.rcParams['lines.linewidth'] = 2    # Linienstärke

# Panda DataFrame
feder = {'Auslenkung (cm)': [42.0,48.4,51.3,56.3,58.6],
         'Masse (kg)': [2,4,6,8,10]} # die Daten werden mit einem Dictionary␣
 ↪definiert
df_feder = pd.DataFrame(data=feder)  # erzeugt ein Panda DataFrame

# Struktur des DataFrames
df_feder.head()
```

[48]:

	Auslenkung (cm)	Masse (kg)
0	42.0	2
1	48.4	4

```
2              51.3             6
3              56.3             8
4              58.6            10
```

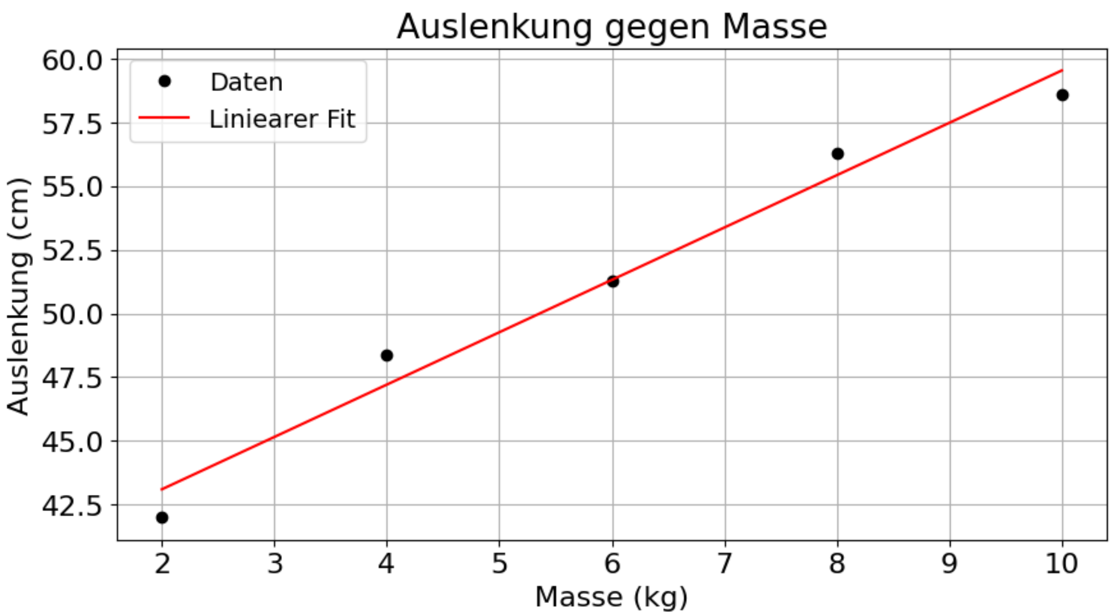
[49]:
```
# Plot Darstellung
df_feder.plot.scatter('Masse (kg)','Auslenkung (cm)', color='red',label='Data␣
 ↪points')
plt.grid(True)
plt.legend()
plt.xlabel('Mass(kg)')
plt.ylabel('Auslenkung (cm)')
plt.title("Auslenkung gegen Masse")
plt.show()
```



2.0.1 Polyfit

[50]:
```
# Polyfit
x_feder = df_feder['Masse (kg)']
y_feder = df_feder['Auslenkung (cm)']
b, a = np.polyfit(x_feder,y_feder,deg=1) # Parameter Anstieg und y-Schnittpunkt

# Plot Darstellung
plt.plot(x_feder, y_feder,'ko', label='Daten')
plt.plot(x_feder, b*x_feder+a,'r',label='Liniearer Fit')
plt.grid(True)
plt.legend()
```

```
plt.xlabel('Masse (kg)')
plt.ylabel('Auslenkung (cm)')
plt.title("Auslenkung gegen Masse")
plt.show()

print("Die beste Ausgleichsgerade hat folgende Parameter: ")
print("===========y=bx+a===========\n")
print("b = {:5.2f} cm/kg".format(b))
print("a = {:5.2f} cm".format(a))
```




```
Die beste Ausgleichsgerade hat folgende Parameter:
===========y=bx+a===========

b =  2.06 cm/kg
a = 38.99 cm
```

2.0.2 Curve Fit

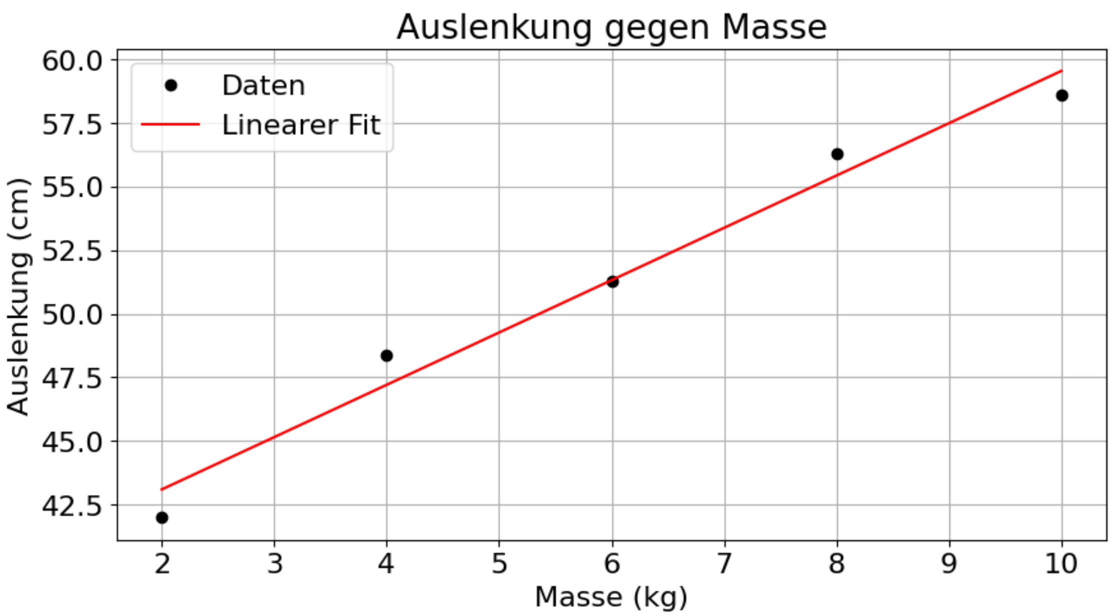
[51]:
```
def linear_fit(x,b,a):
    return b*x+a

parameter, parameter_kovarianzen = curve_fit(linear_fit, x_feder, y_feder)
# parameter[0]=b und parameter[1]=a
# die Kovarianzmatrix wird der Variablen "parameter_kovarianzen" zugewiesen

anstieg = parameter[0]
intercept = parameter[1]
```

```
print(f"Der Anstieg der linear_fit Funktion beträgt:         {anstieg:5.2f} cm/
 ↪kg")
print(f"Der Achsenabschnitt der linear_fit Funktion beträgt: {intercept:5.2f}␣
 ↪cm\n")

u_anstieg = np.sqrt(parameter_kovarianzen [0][0])  # Unsicherheit des Anstieges
u_intercept= np.sqrt(parameter_kovarianzen [1][1]) # Unsicherheit des␣
 ↪Achsenabschnitts
print("Unsicherheit des Anstieges        u_b={:5.2f} cm/kg".format(u_anstieg))
print("Unsicherheit des Achsenabschnitts u_a={:5.2f} cm".format(u_intercept))
```

```
Der Anstieg der linear_fit Funktion beträgt:          2.06 cm/kg
Der Achsenabschnitt der linear_fit Funktion beträgt: 38.99 cm

Unsicherheit des Anstieges        u_b= 0.19 cm/kg
Unsicherheit des Achsenabschnitts u_a= 1.25 cm
```

[52]:

```
# Plot Darstellung mit curve_fit
plt.plot(x_feder, y_feder,'ko', label='Daten')
plt.plot(x_feder, anstieg*x_feder+intercept,'r',label='Linearer Fit')
plt.grid(True)
plt.legend(fontsize=16)
plt.xlabel('Masse (kg)', fontsize=16)
plt.ylabel('Auslenkung (cm)',fontsize=16)
plt.title("Auslenkung gegen Masse")
plt.show()

print("Die beste Ausgleichsgerade hat folgende Parameter: ")
print("==========y=bx+a===========\n")
print("b = {:5.2f} cm/kg".format(anstieg))
print("a = {:5.2f} cm".format(intercept))
```




```
Die beste Ausgleichsgerade hat folgende Parameter:
===========y=bx+a===========

b =  2.06 cm/kg
a = 38.99 cm
```

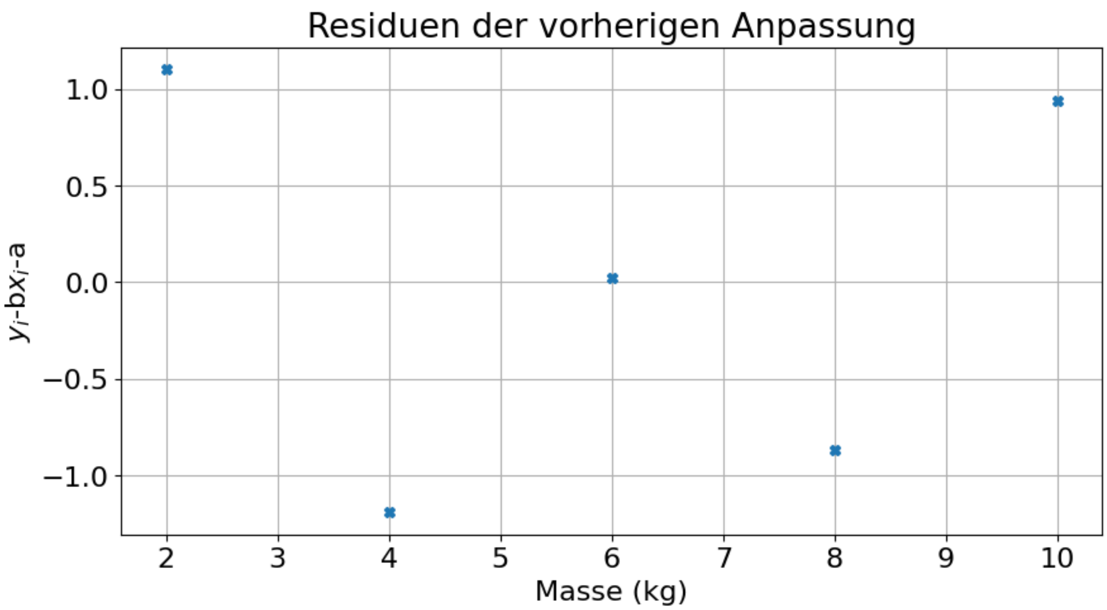
2.0.3 Berechnung des Bestimmtheitsmaßes r^2

[53]:
```
x_data=x_feder # die Daten der DataFrame werden zu x_data hinzugefügt
y_data=y_feder # die Daten der DataFrame werden zu y_data hinzugefügt

residuen = y_data- linear_fit(x_data,anstieg,intercept) # Berechnung der␣
 ↪Residuen
ss_res = np.sum(residuen**2) # hier berechnen wir die Summe der Residuen zum␣
 ↪Quadrat
ss_tot = np.sum((y_data-np.mean(y_data))**2) # Berechnung der Varianz in den␣
 ↪y-Daten
r_quadrat = 1 - (ss_res / ss_tot) # Formel zur Berechnung des␣
 ↪Bestimmtheitsmasses
print ("r² =",r_quadrat)
```

```
r² = 0.9753620343210847
```

[54]:
```
residuen=anstieg*x_data+intercept-y_data
plt.plot(x_data, residuen, 'X',label='Residuen')
plt.title("Residuen der vorherigen Anpassung")
#plt.legend(loc='lower right')
```

```
plt.xlabel('Masse (kg)')
plt.ylabel('$y_i$-b$x_i$-a')
plt.grid()
plt.show()
```



2.0.4 gewichtete kleinste Quadrate

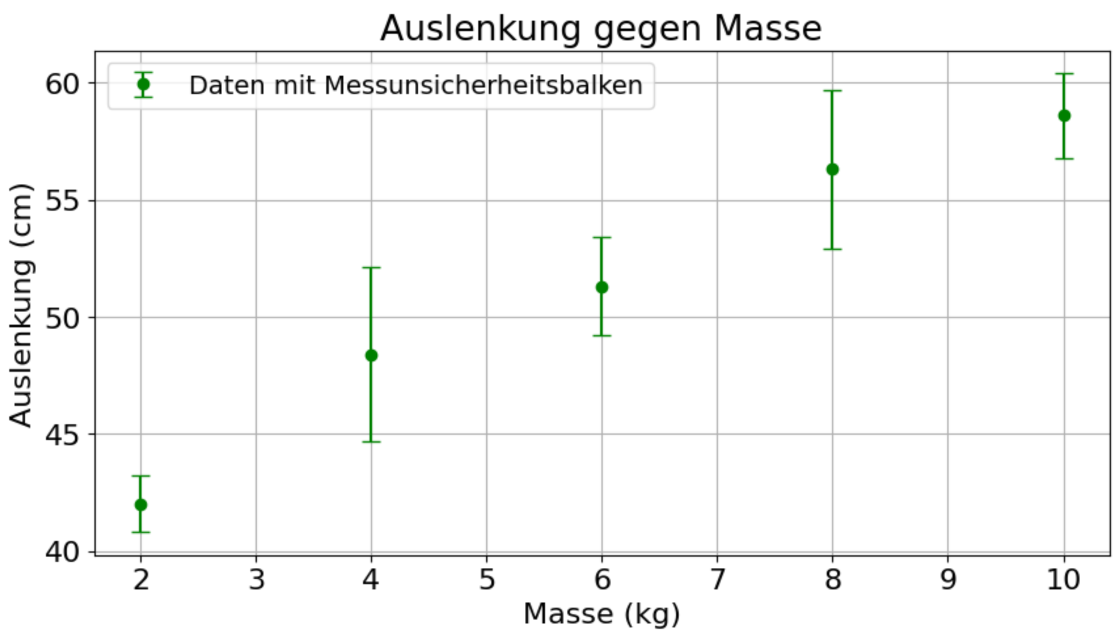
[55]:
```
feder_mit_unsicherheiten = {'Auslenkung (cm)': [42.0,48.4,51.3,56.3,58.6],
                            'Masse (kg)': [2,4,6,8,10],'Unsicherheiten Auslenk.␣
 ↪(cm)':[1.2,3.7,2.1,3.4,1.8]}

# definiert neues DataFrame
df2=pd.DataFrame(data=feder_mit_unsicherheiten)
df2.head()
```

[55]:

	Auslenkung (cm)	Masse (kg)	Unsicherheiten Auslenk. (cm)
0	42.0	2	1.2
1	48.4	4	3.7
2	51.3	6	2.1
3	56.3	8	3.4
4	58.6	10	1.8

[56]:
```
# Parameter aus dem Dataframe
x2= df2['Masse (kg)']
y2=df2['Auslenkung (cm)']
yerr2=df2['Unsicherheiten Auslenk. (cm)']
```

```
# Erstellen Sie ein Diagramm, welches die Daten mit Fehlerbalken anzeigt
plt.errorbar(x2,y2,yerr2,fmt='go',label='Daten mit Messunsicherheitsbalken',␣
↪capsize=5)
plt.grid(True)
plt.legend(loc="upper left")  # Ort der Legende
plt.xlabel('Masse (kg)')
plt.ylabel('Auslenkung (cm)')
plt.title("Auslenkung gegen Masse")
plt.show()
```



[57]:
```
params, params_covariance = curve_fit(linear_fit, x2,␣
↪y2,sigma=yerr2,absolute_sigma=True)
print(params) # optimale Werte für die Parameter, welche die Summe der␣
↪quadrierten Residuen minimieren.
print(params_covariance)   # geschätzte Kovarianz von params
```

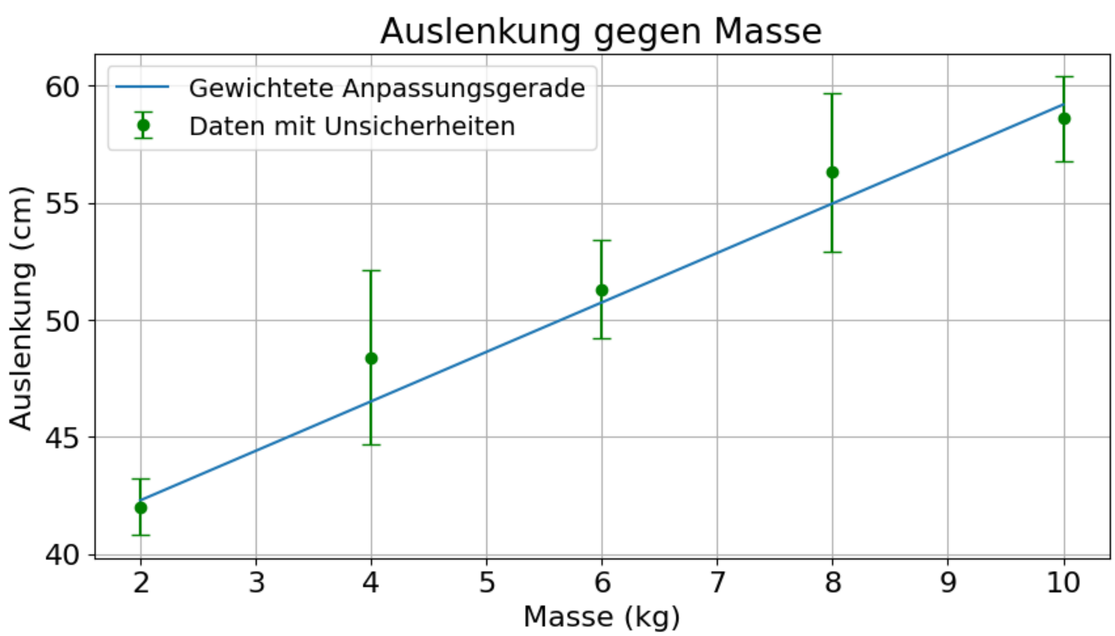
```
[ 2.11443674 38.05309837]
[[ 0.06652116 -0.32653028]
 [-0.32653028  2.32256645]]
```

[58]:
```
anstieg = params[0]             # Anstieg b
achsenabschnitt = params[1]    # Achsenabschnitt a
u_anstieg= np.sqrt(params_covariance[0][0])          # Unsicherheit des␣
↪Anstieges
u_achsenabschnitt = np.sqrt(params_covariance[1][1])   # Unsicherheit des␣
↪Achsenabschnittes
```

```
print("Anstieg=", anstieg, "cm/kg\n", "u_Anstieg=",u_anstieg, "cm/kg\n",␣
↪"Achsenabschnitt=", achsenabschnitt,"cm\n",␣
↪"u_Achsenabschnitt=",u_achsenabschnitt, "cm\n")
```

```
Anstieg= 2.1144367388233323 cm/kg
 u_Anstieg= 0.25791696530219727 cm/kg
 Achsenabschnitt= 38.053098371458425 cm
 u_Achsenabschnitt= 1.523996865194833 cm
```

[59]:
```
# Diagramm mit den Daten und dazugehörigen Fehlerbalken
plt.errorbar(x2,y2,yerr2,fmt='go',label='Daten mit Unsicherheiten', capsize=5)

# Diagramm inklusiv der besten Anpassungsgerade an die Daten
plt.plot(x2,anstieg*x2+achsenabschnitt,label='Gewichtete Anpassungsgerade')
plt.grid()
plt.legend()
plt.xlabel('Masse (kg)')
plt.ylabel('Auslenkung (cm)')
plt.title("Auslenkung gegen Masse")
plt.show()

print('Die Steigung ist b ={:5.1f} cm/kg, mit der Unsicherheit {:5.1f} cm/kg'.
↪format(anstieg,u_anstieg))
print('Der Achsenabschnitt ist a= {:5.1f} cm, mit der Unsicherheit {:5.1f} cm'.
↪format(achsenabschnitt,u_achsenabschnitt))
```



```
Die Steigung ist b =  2.1 cm/kg, mit der Unsicherheit   0.3 cm/kg
Der Achsenabschnitt ist a=  38.1 cm, mit der Unsicherheit   1.5 cm
```

2.0.5 reduziertes Chi-Quadrat

[60]:
```python
def reduziertes_chi_quadrat(x, y, yerr, f, *args):
    """
    reduzierte Chi-Quadrat-Funktion
    x, y und yerr sind Numpy-Arrays, die sich auf die Daten x, y und yerr␣
↪beziehen
    f ist die Funktion, die wir anpassen.
    args sind die Argumente der Funktion, die wir angepasst haben.
    """
    return 1/(len(x)-len(args))*np.sum((f(x, *args)-y)**2/yerr**2)
```

Übung 1:

In einem Experiment wird ein Stein mit der Anfangsgeschwindigkeit v senkrecht nach oben geworfen. Ein Student führt sieben Würfe durch, um die Beziehung zwischen v^2 und h zu überprüfen. Erwartet wird die Form: $v^2 = 2gh$

Die Daten des Studenten werden in der folgenden Zelle in einem Pandas DatenFrame angezeigt.

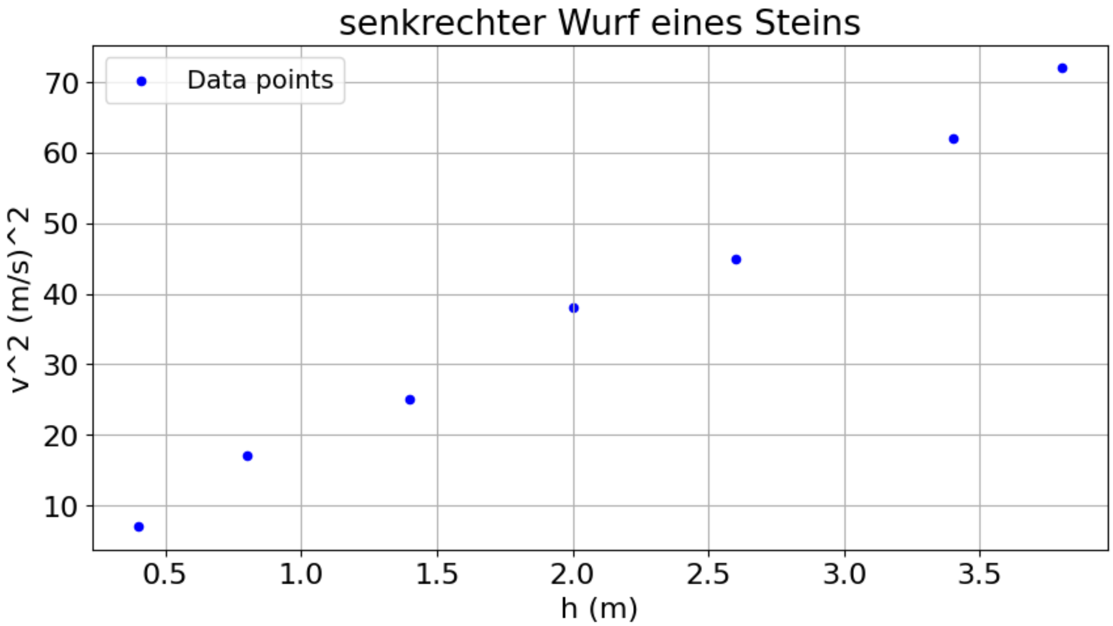
Weisen Sie h und v^2 den Variablen x und y zu und stellen Sie diese einschließlich der gegebenen Messunsicherheiten in einem $v^2(h)$-Diagramm graphisch dar.

[61]:
```python
# DataFrame
# Definieren die Daten mit einem Dictionary und einer Liste
stein = {'h (m)': [0.4,0.8,1.4,2.0,2.6,3.4,3.8], 'v^2 (m/s)^2':␣
↪[7,17,25,38,45,62,72], 'Unsicherheit von v^2 (m/s)^2':[3,3,3,4,5,5,6]}
stein_df=pd.DataFrame(data=stein)
stein_df.head()
```

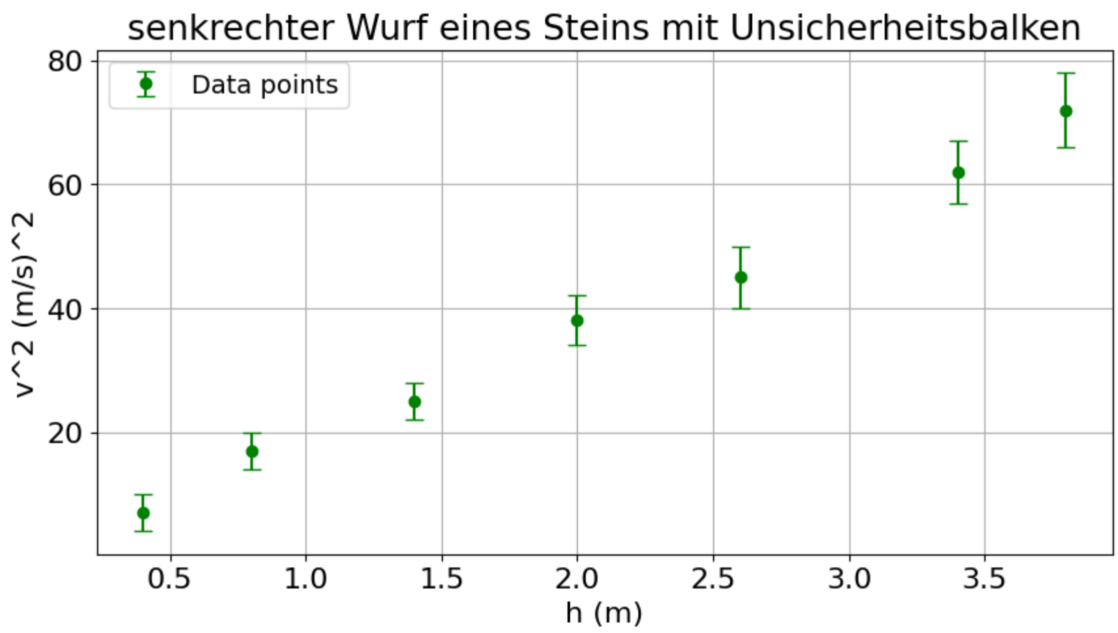
[61]:

	h (m)	v^2 (m/s)^2	Unsicherheit von v^2 (m/s)^2
0	0.4	7	3
1	0.8	17	3
2	1.4	25	3
3	2.0	38	4
4	2.6	45	5

[62]:
```python
# Plot Darstellung
stein_df.plot.scatter('h (m)','v^2 (m/s)^2', color='blue',label='Data points')
plt.grid(True)
plt.legend()
plt.xlabel('h (m)')
plt.ylabel('v^2 (m/s)^2')
plt.title("senkrechter Wurf eines Steins")
plt.show()
```



[63]:
```
# Plot Darstellung
x_stein = stein_df['h (m)']
y_stein = stein_df['v^2 (m/s)^2']
u_stein = stein_df['Unsicherheit von v^2 (m/s)^2']
plt.errorbar(x_stein, y_stein, yerr=u_stein,color='green',fmt='o',label='Data␣
 ↪points', capsize=5)
plt.grid(True)
plt.legend(loc='upper left', fontsize=14)
plt.xlabel('h (m)')
plt.ylabel('v^2 (m/s)^2')
plt.title("senkrechter Wurf eines Steins mit Unsicherheitsbalken")
plt.show()
```



Übung 2:

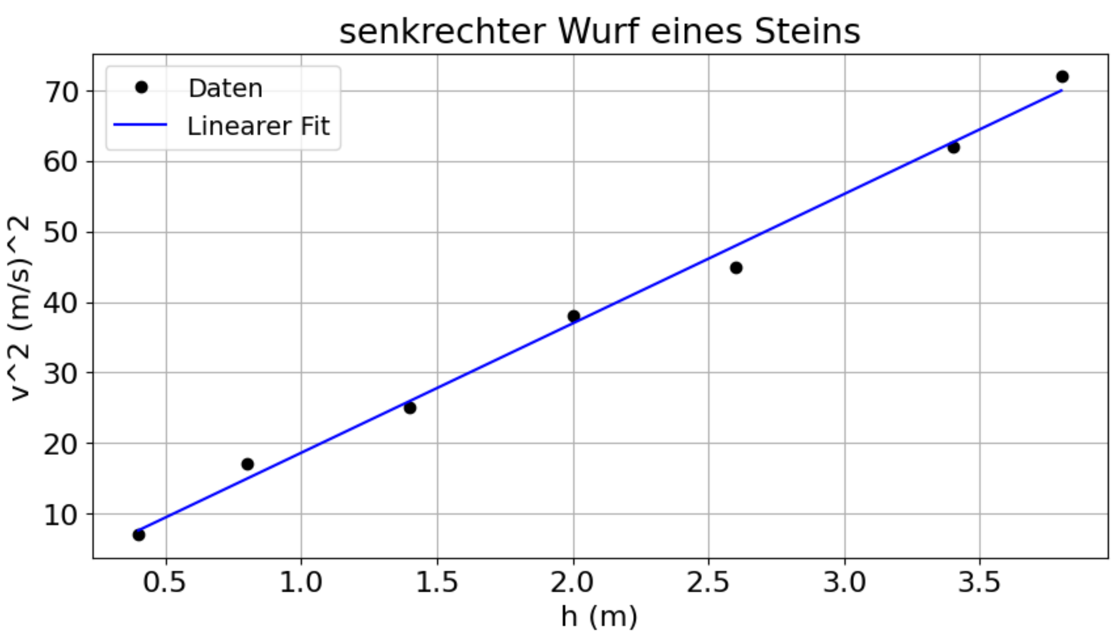
Finden Sie die beste Anpassungsgerade für die Messwerte der vorherigen Übung (Geschwindigkeit des Steins) und tragen Sie diese Daten ein. Ermitteln Sie die Steigung und beurteilen Sie, ob das Ergebnis mit dem Literaturwert von $g = 9,81 m/s^2$ übereinstimmt. Es gilt die Annahme, dass die Unsicherheiten der x-Werte vernachlässigbar und die Unsicherheiten der aller y-Werte gleich sind. Daher sollte hier die Methode der kleinsten Quadrate ohne Gewichtung verwendet werden. Die Daten sind im obigen DataFrame stein_df gespeichert.

[64]:
```
def linear_fit(x,b,a):
    return b*x+a

parameter, parameter_kovarianzen = curve_fit(linear_fit, x_stein, y_stein)
# parameter[0]=b und parameter[1]=a
# die Kovarianzmatrix wird der Variablen "parameter_kovarianzen" zugewiesen

anstieg = parameter[0]
intercept = parameter[1]
print(f"Der Anstieg der linear_fit Funktion beträgt:         {anstieg:5.2f} (m/
 ↪s)^2")
print(f"Der Achsenabschnitt der linear_fit Funktion beträgt: {intercept:5.2f}␣
 ↪m\n")

u_anstieg = np.sqrt(parameter_kovarianzen [0][0])  # Unsicherheit des Anstieges
u_intercept= np.sqrt(parameter_kovarianzen [1][1]) # Unsicherheit des␣
 ↪Achsenabschnitts
print("Unsicherheit des Anstieges        u_b ={:5.2f} cm/kg".format(u_anstieg))
```

```
print("Unsicherheit des Achsenabschnitts u_a ={:5.2f} cm".format(u_intercept))
```

```
Der Anstieg der linear_fit Funktion beträgt:         18.35 (m/s)^2
Der Achsenabschnitt der linear_fit Funktion beträgt:  0.25 m

Unsicherheit des Anstieges        u_b = 0.63 cm/kg
Unsicherheit des Achsenabschnitts u_a = 1.51 cm
```

[65]:
```
# Plot Darstellung mit curve_fit
plt.plot(x_stein, y_stein,'ko', label='Daten')
plt.plot(x_stein, anstieg*x_stein+intercept,'b',label='Linearer Fit')
plt.grid(True)
plt.legend()
plt.xlabel('h (m)')
plt.ylabel('v^2 (m/s)^2')
plt.title("senkrechter Wurf eines Steins")
plt.show()

print("Die beste Ausgleichsgerade hat folgende Parameter: ")
print("===========y=bx+a===========\n")
print("b = {:5.2f} cm/kg".format(anstieg))
print("a = {:5.2f} cm".format(intercept))
```




```
Die beste Ausgleichsgerade hat folgende Parameter:
===========y=bx+a===========

b = 18.35 cm/kg
```

```
a =  0.25 cm
```

Das Ergebnis stimmt nicht mit dem Wert $g = 9,81m/s^2$, wahrscheinlich handelt es sich um eine systematische Abweichung.

Übung 3: Berechnen Sie den Wert von r^2 für das obige Experiment mit dem Stein.

[66]:
```
def residuals(x, y, steigung, start):
    residuen = y - linear_fit(x, steigung, start) # Berechnnung der Residuen
    ss_res = np.sum(residuen**2) # hier berechnen wir die Summe der Residuen␣
 ↪zum Quadrat
    ss_tot = np.sum((y-np.mean(y))**2) # Berechnung der Varianz in den y-Daten
    return 1 - (ss_res / ss_tot) # Formel zur Berechnung des Bestimmtheitsmasses

r_quadrat = residuals(x_stein, y_stein, anstieg, intercept)
print ("r² =",r_quadrat)
```

```
r² = 0.9940719958991769
```

Übung 4:

Ermitteln Sie die **gewichtete** beste Anpassungsgerade für das obige Steinwurf-Experiment und tragen Sie Ihre Ergebnisse ein. Ermitteln Sie die Steigung und diskutieren Sie, ob die Ergebnisse mit dem Wert von $g = 9.81m/s^2$ übereinstimmen. Berechnen Sie auch das reduzierte Chi-Quadrat. Beachten Sie, dass die Daten in dem Datenrahmen stein_df gespeichert sind.

Wichtige Anmerkung: Im letzten Beispiel wurde eine Funktion zur Berechnung des reduzierten Chi-Quadrats definiert. Um den Code dieses Notebooks zu optimieren, könnten Sie für die Berechnung des Bestimmtheitsmaßes und für die Erstellung von Diagrammen ebenfalls Funktionen definieren…

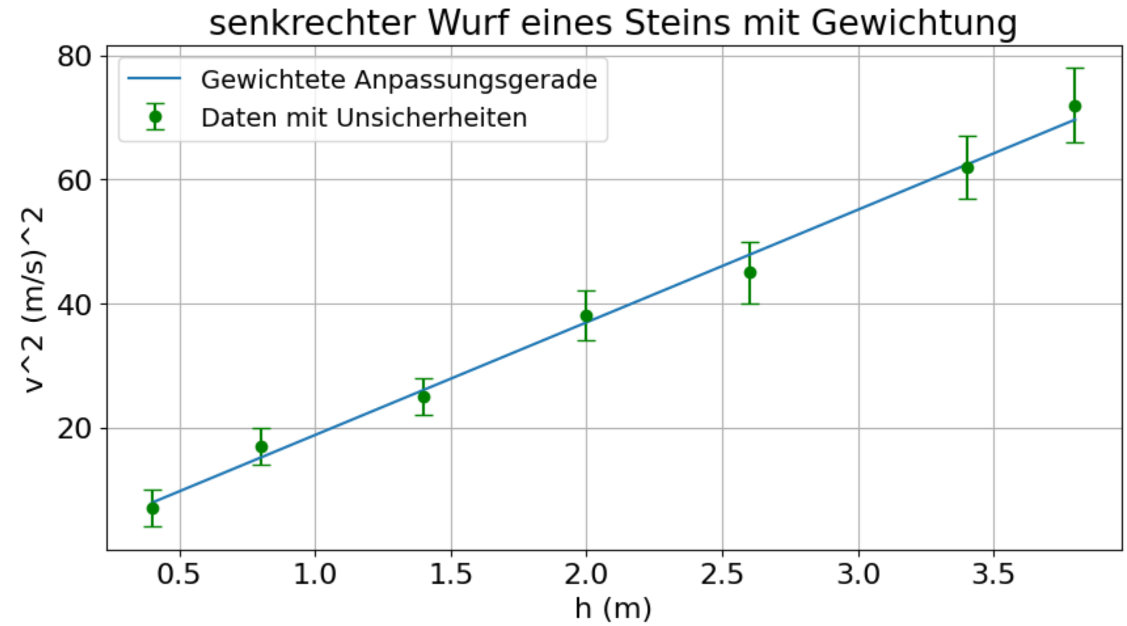
In einem separaten Notebook finden Sie ein Beispiel für die Verwendung der Funktion **curve_fit** in einem physikalischen Experiment. Darin wird die Funktion genutzt, um die Beziehung zwischen der Beleuchtungsstärke einer Glühbirne in Abhängigkeit vom Abstand zu einem Smartphone-Lichtsensor zu ermitteln. Sie können sich dieses Notebook selbstständig ansehen.

[68]:
```
params, params_covariance = curve_fit(linear_fit, x_stein, y_stein,␣
 ↪sigma=u_stein, absolute_sigma=True)

anstieg = params[0]           # Anstieg b
achsenabschnitt = params[1]   # Achsenabschnitt a
u_anstieg = np.sqrt(params_covariance[0][0])            # Unsicherheit des␣
 ↪Anstieges
u_achsenabschnitt = np.sqrt(params_covariance[1][1])    # Unsicherheit des␣
 ↪Achsenabschnittes
print(" Anstieg=", anstieg, "\n", "u_Anstieg=",u_anstieg, "\n",
      "Achsenabschnitt=", achsenabschnitt,"\n",␣
 ↪"u_Achsenabschnitt=",u_achsenabschnitt, "\n")
```

```
 Anstieg= 18.17440549172788
 u_Anstieg= 1.3541521165445563
```

```
Achsenabschnitt= 0.5897272205011221
u_Achsenabschnitt= 2.481060317921474
```

[70]:
```python
# Diagramm mit den Daten und dazugehörigen Fehlerbalken
plt.errorbar(x_stein,y_stein,u_stein,fmt='go',label='Daten mit Unsicherheiten',␣
 ↪capsize=5)

# Diagramm inklusiv der besten Anpassungsgerade an die Daten
plt.plot(x_stein,anstieg*x_stein+achsenabschnitt,label='Gewichtete␣
 ↪Anpassungsgerade')
plt.grid()
plt.legend()
plt.xlabel('h (m)')
plt.ylabel('v^2 (m/s)^2')
plt.title("senkrechter Wurf eines Steins mit Gewichtung")
plt.show()

print('Die Steigung ist        b ={:5.1f} cm/kg, mit der Unsicherheit {:5.1f}␣
 ↪cm/kg'.format(anstieg,u_anstieg))
print('Der Achsenabschnitt ist a ={:5.1f} cm, mit der Unsicherheit {:5.1f} cm'.
 ↪format(achsenabschnitt,u_achsenabschnitt))

def reduziertes_chi_quadrat(x, y, yerr, f, *args):
    """
    reduzierte Chi-Quadrat-Funktion
    x, y und yerr sind Numpy-Arrays, die sich auf die Daten x, y und yerr␣
 ↪beziehen
    f ist die Funktion, die wir anpassen.
    args sind die Argumente der Funktion, die wir angepasst haben.
    """
    return 1/(len(x)-len(args))*np.sum((f(x, *args)-y)**2/yerr**2)

chi_quadrat = reduziertes_chi_quadrat(x_stein, y_stein, u_stein,linear_fit,␣
 ↪*params)
print("Chi-Quadrat: ", chi_quadrat)
```




```
Die Steigung ist        b = 18.2 cm/kg, mit der Unsicherheit   1.4 cm/kg
Der Achsenabschnitt ist a =  0.6 cm, mit der Unsicherheit   2.5 cm
Chi-Quadrat:  0.22848240706988873
```