auswertung

November 4, 2018

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        from scipy.optimize import curve_fit
        from scipy.stats import chi2
        from math import sqrt
```

SpectraSuite verwendet Kommata als Dezimaltrennzeichen, die durch Punkte ersetzt werden müssen:

```
In [2]: def comma_to_float(valstr):
            return float(valstr.decode("utf-8").replace(',', '.'))
```

1 Analyse des Sonnenlichts

Zunächst werden die Daten mit und ohne Fenster geladen. Die ersten 17 Zeilen sowie alle Zeilen, die mit '>' beginnen, sind Header und Kommentare und werden deshalb ausgelassen.

```
In [3]: lamb_og, inten_og = np.loadtxt('data/sonne_ohne_glas.txt',
                                       skiprows = 17,
                                       converters = {0:comma_to_float, 1:comma_to_float},
                                       comments = '>',
                                       unpack = True)
```

```
In [4]: lamb_mg, inten_mg = np.loadtxt('data/sonne_durch_glas.txt',
                                       skiprows = 17,
                                       converters = {0:comma_to_float, 1:comma_to_float},
                                       comments = '>',
                                       unpack = True)
```

Die beiden Intensitätsverteilungen werden nun in ein gemeinsames Diagramm eingezeichnet:

```
In [77]: plt.plot(lamb_og, inten_og, label="Ohne Fenster", linewidth = 1)
         plt.plot(lamb_mg, inten_mg, label="Mit Fenster", linewidth = 1)
         plt.title("Gemessenes Sonnenspektrium mit und ohne Fenster")
         plt.xlabel("Wellenlänge [nm]")
         plt.ylabel("Intensität [b.E.]")
```

```
plt.legend()
plt.grid()
plt.ylim((0, 65000))
plt.xlim((250, 900))
plt.savefig("figures/sonnenlicht_mit_ohne_glas.pdf", format = "pdf")
```

1.1 Absorption von Glas

Die Absorption von Glas ist gegeben durch

$$A(\lambda) = 1 - \frac{I_{mG}(\lambda)}{I_{oG}(\lambda)}$$

```
In [6]: A = 1 - inten_mg/inten_og
```

Die Absorption wird nun als Funktion der Wellenlänge dargestellt:

```
In [7]: plt.plot(lamb_mg, A)
        plt.title("Absorption von Glas")
        plt.xlabel("Wellenlänge [nm]")
        plt.ylabel('Absorption [b.E.]')
        plt.ylim((0, 1))
        plt.xlim((320, 800))
        plt.savefig("figures/absorption_glas.pdf", format = "pdf")
```

1.2 Fraunhoferlinien

In einem interaktiven Plot werden die Fraunhoferlinien abgelesen:

```
In [8]: %matplotlib widget
        plt.plot(lamb_og, inten_og)
        plt.title("Sonnenspektrum")
        plt.xlabel("Wellenlänge [nm]")
        plt.ylabel("Intensität [b.E.]")
        plt.ylim((0, 65000))
        plt.xlim((350, 800))
        plt.savefig("figures/sonnenspektrum.pdf", format = "pdf")
```

Die abgelesenen Wellenlängen werden nun in einer neuen Abbildung dargestellt:

```
In [76]: %matplotlib inline
         plt.plot(lamb_og, inten_og, linewidth = 1)
         plt.title("Sonnenspektrum mit Fraunhoferlinien")
         plt.xlabel("Wellenlänge [nm]")
         plt.ylabel("Intensität [b.E.]")
         plt.ylim((0, 65000))
         plt.xlim((350, 800))

         # abgelesene Werte:
```

```
fraunhofer_linien = [430.2, 485.9, 517.3, 526.7, 589.1, 656.0, 687.0, 719.4, 760.3]
for linie in fraunhofer_linien:
    # vertikale Gerade einzeichnen
    plt.axvline(x = linie, linewidth = 1, color = "orange")

plt.savefig("figures/fraunhofer.pdf", format = "pdf")
```

2 Natriumspektrum

Zunächst werden die Datensätze wie oben geladen:

```
In [81]: # Integrationszeit so gewählt, dass hellste Linie nicht in Sättigung
         lamb_gesamt_1, inten_gesamt_1 = \
             np.loadtxt('data/natrium_komplett_1.txt',
                        skiprows = 17,
                        converters = {0:comma_to_float, 1:comma_to_float},
                        comments = '>',
                        unpack = True)

         # Längere Integrationszeit, damit wesentlicher Teil des Spektrums sichtbar
         lamb_gesamt_2, inten_gesamt_2 = \
             np.loadtxt('data/natrium_komplett_2.txt',
                        skiprows = 17,
                        converters = {0:comma_to_float, 1:comma_to_float},
                        comments = '>',
                        unpack = True)

         # Integrationszeit für 400 - 540 nm optimiert
         lamb_schwach, inten_schwach = \
             np.loadtxt('data/natrium_niedrige_intensitaet.txt',
                        skiprows = 17,
                        converters = {0:comma_to_float, 1:comma_to_float},
                        comments = '>',
                        unpack = True)
```

Zunächst wird das gesamte Spektrum geplottet:

```
In [11]: plt.plot(lamb_gesamt_1, inten_gesamt_1, linewidth = 1)
         plt.title("Natriumspektrum (gesamt)")
         plt.xlabel("Wellenlänge [nm]")
         plt.ylabel("Intensität [b.E.]")
         plt.yscale("log")
         plt.ylim((10, 60000))
         plt.xlim((350, 800))
         plt.savefig("figures/natrium_komplett.pdf", format = "pdf")
```

Um die Linien im Bereich 600nm - 850nm besser zu erkennen, wird dieser Abschnitt extra geplottet, wobei der dazu besser geeignete Datensatz verwendet wird (höhere Integrationszeit).

```
In [12]: plt.plot(lamb_gesamt_2, inten_gesamt_2, linewidth = 1)
         plt.title("Natriumspektrum (langwellig)")
         plt.xlabel("Wellenlänge [nm]")
         plt.ylabel("Intensität [b.E.]")
         plt.yscale("log")
         plt.ylim((40, 60000))
         plt.xlim((600, 850))
         plt.savefig("figures/natrium_langwellig.pdf", format = "pdf")
```

Schließlich wird der Bereich von 300nm bis 540nm mit dem dafür optimierten Datensatz geplottet.

```
In [13]: plt.plot(lamb_schwach, inten_schwach, linewidth = 1)
         plt.title("Natriumspektrum (kurzwellig)")
         plt.xlabel("Wellenlänge [nm]")
         plt.ylabel("Intensität [b.E.]")
         plt.yscale("log")
         plt.ylim((250, 60000))
         plt.xlim((300, 540))
         plt.savefig("figures/natrium_schwach.pdf", format = "pdf")
```

2.1 Zuordnung der gefundenen Linien zu Serien

Einige benötigte Naturkonstanten:

```
In [14]: E_Ry = -13.605 # [eV], Rydbergenergie
         h = 4.13567e-15 # [eV s], Planckssches Wirkungsquantum
         c = 2.9979e8 # [m/s], Lichtgeschwindigkeit
```

2.1.1 1. Nebenserie ($md \rightarrow 3p$)

Wir nehmen an, dass die Linie bei 818,8 nm zu m = 3 gehört. Daraus ergibt sich:

```
In [44]: E_3p = E_Ry/3**2 - h*c/818.8e-9
         E_3p_ = E_Ry/3**2 - h*c/((818.8 + 1.4)*1e-9)
         print("E_3p = ", E_3p, "+-", abs(E_3p - E_3p_))

E_3p =  -3.0258734440237745 +- 0.0025846006928795795
```

Der Fehler wird durch Einsetzen einer Grenze des Fehlerbereichs berechnet.

Nun können die erwarteten Wellenlängen berechnet werden, wobei sich die Fehler wieder durch Einsetzen ergeben:

```
In [40]: for m in range(3, 13):
             l = h*c/(E_Ry/m**2 - E_3p)*1e9
             l_ = h*c/(E_Ry/m**2 - E_3p_)*1e9
             Dl = abs(l - l_)
             print('m={m:2d}, lambda={l:6.2f}+-{Dl:6.2f}'.format(m=m, l=l, Dl=Dl))
```

```
m= 3, lambda=818.80+-  1.40
m= 4, lambda=569.89+-  0.68
m= 5, lambda=499.60+-  0.52
m= 6, lambda=468.22+-  0.46
m= 7, lambda=451.14+-  0.42
m= 8, lambda=440.70+-  0.41
m= 9, lambda=433.82+-  0.39
m=10, lambda=429.03+-  0.38
m=11, lambda=425.56+-  0.38
m=12, lambda=422.95+-  0.37
```

2.1.2 2. Nebenserie ($ms \to 3p$)

Genau analog zur 1.Nebenserie (nur, dass diesmal der Korrekturterm nicht vernachlässigt wird):

```
In [45]: E_3s = E_3p - h*c/588.8e-9
         E_3s_ = E_3p_ - h*c/((588.8 + 1.0)*1e-9)
         print("E_3s = ", E_3s, "+-", abs(E_3s - E_3s_))

E_3s =  -5.1315672437860025 +- 0.006154783466297609


In [46]: Delta_s = sqrt(E_Ry/E_3s) - 3
         Delta_s_ = sqrt(E_Ry/E_3s_) - 3
         print("Delta_s = ", Delta_s, "+-", abs(Delta_s - Delta_s_))

Delta_s =  -1.371738099720573 +- 0.000977345016534903


In [47]: for m in range(4, 10):
             l = h*c/(E_Ry/(m - Delta_s)**2 - E_3p)*1e9
             l_ = h*c/(E_Ry/(m - Delta_s_)**2 - E_3p_)*1e9
             Dl = abs(l - l_)
             print('m={m:2d}, lambda={l:6.2f}+-{Dl:6.2f}'.format(m=m, l=l, Dl=Dl))

m= 4, lambda=485.37+-  0.52
m= 5, lambda=460.77+-  0.46
m= 6, lambda=446.70+-  0.43
m= 7, lambda=437.83+-  0.41
m= 8, lambda=431.85+-  0.39
m= 9, lambda=427.62+-  0.39
```

2.1.3 Hauptserie ($mp \to 3s$)

```
In [48]: Delta_p = sqrt(E_Ry/E_3p) - 3
         Delta_p_ = sqrt(E_Ry/E_3p_) - 3
         print("Delta_p = ", Delta_p, "+-", abs(Delta_p - Delta_p_))
```

```
Delta_p =  -0.879570229013229 +- 0.0009061809392800768
```

```
In [49]: for m in range(4, 10):
             l = h*c/(E_Ry/(m - Delta_p)**2 - E_3s)*1e9
             l_ = h*c/(E_Ry/(m - Delta_p_)**2 - E_3s_)*1e9
             Dl = abs(l - l_)
             print('m={m:2d}, lambda={l:6.2f}+-{Dl:6.2f}'.format(m=m, l=l, Dl=Dl))
```

```
m= 4, lambda=271.88+-  0.38
m= 5, lambda=261.68+-  0.35
m= 6, lambda=255.95+-  0.33
m= 7, lambda=252.39+-  0.32
m= 8, lambda=250.02+-  0.31
m= 9, lambda=248.35+-  0.31
```

2.2 Bestimmung von Serienenergien und Korrekturtermen

2.2.1 1. Nebenserie

Die zugeordneten Wellenlängen sind:

```
In [64]: wellenl = np.array([818.8, 568.1, 497.7, 466.3, 450.0, 438.6, 433.0, 429.7, 426.3])
         fehler = np.array([1.4, 1.0, 0.7, 0.9, 2.0, 1.0, 1.2, 1.0, 1.3])
         quantenz = np.arange(3, 12)
```

Die folgende Funktion, die die Wellenlänge angibt, soll an diese Daten gefittet werden:

```
In [65]: def fit_func_1(m, _E_Ry, _E_3p, D_d):
             return h*c/(_E_Ry/(m - D_d)**2 - _E_3p)*1e9
```

Dazu werden Startwerte für die Parameter E_{Ry}, E_{3p} und Δ_d gewählt:

```
In [66]: para = [-13.6, -3, -0.02]
         popt, pcov = curve_fit(fit_func_1, quantenz, wellenl, sigma = fehler, p0 = para)
```

Nun werden die Ergebnisse des Fits mit den jeweiligen Fehlern ausgegeben:

```
In [67]: print("E_Ry = ", popt[0], ", Standardfehler = ", np.sqrt(pcov[0][0]))
         print("E_3p = ", popt[1], ", Standardfehler = ", np.sqrt(pcov[1][1]))
         print("D_d = ", popt[2], ", Standardfehler = ", np.sqrt(pcov[2][2]))
```

```
E_Ry =  -12.966162642316942 , Standardfehler =  0.2995829353625529
E_3p =  -3.024395644570556 , Standardfehler =  0.005166940588024771
D_d =  0.07012734717231944 , Standardfehler =  0.02991574849036398
```

Schließlich werden noch die χ^2- und χ^2_{red}-Werte berechnet:

```
In [74]: chi2_ = np.sum((fit_func_1(quantenz, *popt) - wellenl)**2/fehler**2)
         dof = len(quantenz) - 3
         chi2_red = chi2_/dof
         print("chi2 = ", chi2_)
         print("chi2_red = ", chi2_red)

chi2 =  4.605958153596782
chi2_red =  0.7676596922661303
```

```
In [75]: prob = round(1 - chi2.cdf(chi2_, dof), 2)*100
         print("Wahrscheinlichkeit:", prob, "%")

Wahrscheinlichkeit: 60.0 %
```

Die zugeordneten Wellenlängen werden gemeinsam mit der gefitteten Funktion in ein Diagramm eingetragen:

```
In [70]: plt.errorbar(quantenz, wellenl, fehler, fmt = ".")
         plt.xlabel("Quantenzahl")
         plt.ylabel("Wellenlänge [nm]")
         plt.title("1. Nebenserie des Na-Atoms")
         x = np.linspace(2.8, 12.2, 100)
         plt.plot(x, fit_func_1(x, *popt))
         plt.savefig("figures/nebenserie_1.pdf", format = "pdf")
```