Röntgenspektrum

In [1]:

```
#import modules
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from scipy.stats import chi2

plt.style.use('seaborn-white')
plt.rcParams['errorbar.capsize']=2
%matplotlib inline
```

In [2]:

```
#converter
def string2float (valstr):
    return float (valstr.decode("utf-8").replace(',','.'))

#fit functions
#Linear function
def linear(x, a, c):
    return x*a+c
#gausssian function
def gaussian(x, A, mu, sig, y0):
    return y0 + (A*np.exp(-(x-mu)**2/(2*sig**2)))
#function to approximate the data and to take the inconstant underground into account
def overkill(x, a, c, A1, mu1, sig1, A2, mu2, sig2):
    return linear(x, a, c)+gaussian(x, A1, mu1, sig1, 0)+gaussian(x, A2, mu2, sig2, 0)
```

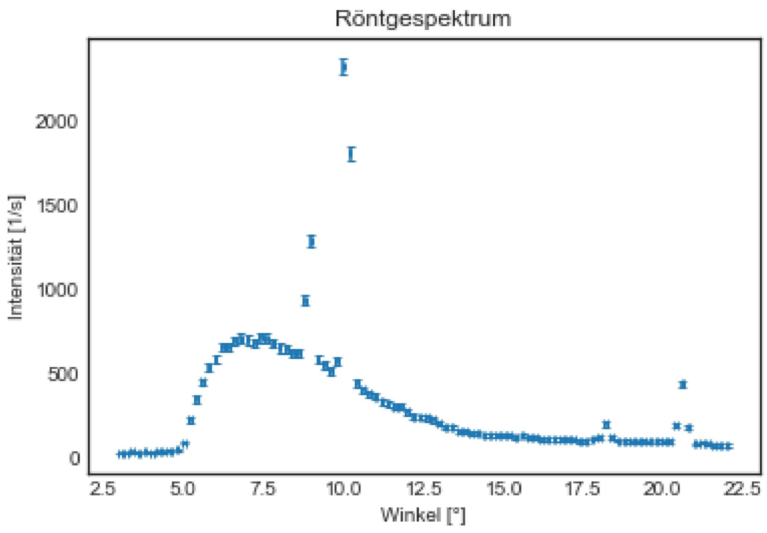
Abschätzen der Planckschen Konstante

In [3]:

```
#load data
angle, inten= np.loadtxt('Daten/01_03_2018 09_38_46.txt',
                         skiprows=0,
                         converters={0:string2float, 1:string2float},
                         unpack=True)
inten_err = np.sqrt(inten)
angle_err = 0.05

#plot data
plt.figure('data, total')
plt.errorbar(angle, inten, inten_err, angle_err, linestyle = 'None')
plt.title('Röntgespektrum')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
```

Out[3]:

```
Text(0,0.5,'Intensität [1/s]')
```



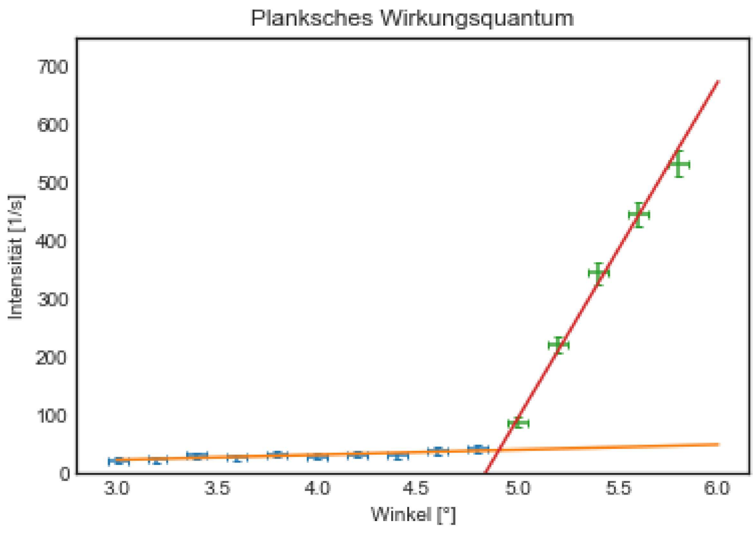
In [4]:

```
#fit underground
maskPlanck1 = (angle<5)
anglePlanck1 = angle[maskPlanck1]
intPlanck1 = inten[maskPlanck1]
intPlanck1_err = inten_err[maskPlanck1]
popt1, pcov1 = curve_fit(linear, anglePlanck1, intPlanck1, sigma = intPlanck1_err)
perr1 = np.sqrt(np.diag(pcov1))
print('Untergrund:')
print('Steigung = %.1e +- %.1e'%(popt1[0], perr1[0]))
print('y_Achse = %.0e +- %.0e'%(popt1[1], perr1[1]))

#fit start
maskPlanck2 = [(angle>=5)&(angle<6)]
anglePlanck2 = angle[maskPlanck2]
intPlanck2 = inten[maskPlanck2]
intPlanck2_err = inten_err[maskPlanck2]
popt2, pcov2 = curve_fit(linear, anglePlanck2, intPlanck2, sigma = intPlanck2_err)
perr2 = np.sqrt(np.diag(pcov2))
print('Linearer Anstieg:')
print('Steigung = %.2e +- %.1e'%(popt2[0], perr2[0]))
print('y_Achse = %.2e +- %.1e'%(popt2[1], perr2[1]))
```

```
Untergrund:
Steigung = 8.7e+00 +- 1.6e+00
y_Achse = -3e+00 +- 6e+00
Linearer Anstieg:
Steigung = 5.80e+02 +- 2.5e+01
y_Achse = -2.80e+03 +- 1.3e+02
```

In [5]:

```
#plot fit
x = np.linspace(angle[0], 6, 300)
plt.figure('planck')
plt.ylim(0, 750)
plt.errorbar(anglePlanck1, intPlanck1, intPlanck1_err, angle_err, linestyle = 'None', l
abel='Untergrund, Messwerte')
plt.plot(x, linear(x, *popt1), marker='', label = 'Untergrund, Fit')
plt.errorbar(anglePlanck2, intPlanck2, intPlanck2_err, angle_err, linestyle = 'None', l
abel='Linearer Anstieg, Messwerte')
plt.plot(x, linear(x, *popt2), marker='', label = 'Linearer Anstieg, Fit')
plt.title('Planksches Wirkungsquantum')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
plt.savefig('Diagramme/Planck1.pdf')
```



K-alpha und K-beta Peaks im LiF Kristall

1. Ordnung

In [6]:

```
#Load data, first order
angleD, intD= np.loadtxt('Daten/01_03_2018 09_55_32.txt',
                         skiprows=0,
                         converters={0:string2float, 1:string2float},
                         unpack=True)
intD_err = np.sqrt(intD)
```

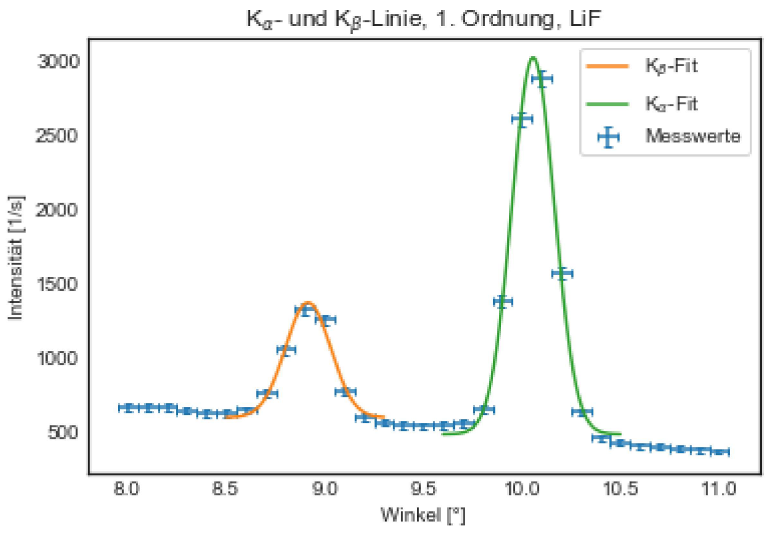
In [7]:

```
#beta line fit
maskB = [(angleD>8.5)&(angleD<9.3)]
angleB = angleD[maskB]
intB = intD[maskB]
intB_err = intD_err[maskB]
poptB, pcovB = curve_fit(gaussian, angleB, intB, sigma = intB_err, p0 = [700, 9, 0.1, 6
00])
perrB = np.sqrt(np.diag(pcovB))

#alpha line fit
maskA = [(angleD>9.6)&(angleD<10.5)]
angleA = angleD[maskA]
intA = intD[maskA]
intA_err = intD_err[maskA]
poptA, pcovA = curve_fit(gaussian, angleA, intA, sigma = intA_err,  p0 = [2500, 10, 0.1
, 560])
perrA = np.sqrt(np.diag(pcovA))

#fit values
print(r'K-beta-Fit:')
print(poptB, '+-', perrB)
print(r'K-alpha-Fit:')
print(poptA, '+-', perrA)
```

```
K-beta-Fit:
[  7.77710907e+02   8.91449290e+00   1.14881194e-01   5.93919406e+02] +- [
  6.96879413e+01   9.41485919e-03   1.45529465e-02   5.17511526e+01]
K-alpha-Fit:
[  2.54412275e+03   1.00574514e+01   1.07847046e-01   4.81319773e+02] +- [
  1.01156076e+02   3.96992220e-03   4.57176272e-03   3.63719108e+01]
```

In [8]:

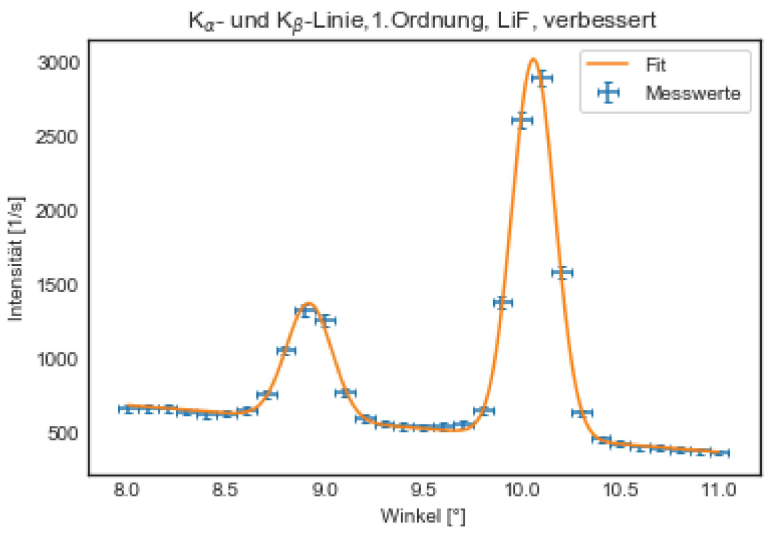
```
#plot data
xB = np.linspace(8.5, 9.3, 200)
xA = np.linspace(9.6, 10.5, 200)
plt.figure('first order')
plt.errorbar(angleD, intD, intD_err, angle_err, linestyle = 'None', label = 'Messwerte'
)
plt.plot(xB, gaussian(xB, *poptB), marker = '', label = r'K$_\beta$-Fit')
plt.plot(xA, gaussian(xA, *poptA), marker = '', label = r'K$_\alpha$-Fit')
plt.legend(frameon=True)
plt.title(r'K$_\alpha$- und K$_\beta$-Linie, 1. Ordnung, LiF')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
plt.savefig('Diagramme/LiF, 1.Ordnung.pdf')
```



In [9]:

```
#overkill fit
popt, pcov = curve_fit(overkill, angleD, intD, sigma = intD_err, p0 = [-30, 1000, 2500,
 10, 0.1, 700, 9, 0.1])
perr = np.sqrt(np.diag(pcov))

#fit values
print('Zusammengefasster Fit:')
print(popt, '+-', perr, 'in der Reihenfolge')
print('Steigung und y-Achsenabschnitt der Gerade, Normierung, Mittelwert und Sigma der
 Gaußfunktionen für k-alpha/K-beta Linie')

#plot data again
plt.figure('first order, better')
x = np.linspace(8, 11, 300)
plt.errorbar(angleD, intD, intD_err, angle_err, linestyle = 'None', label = 'Messwerte'
)
plt.plot(x, overkill(x, *popt), marker = '', label = 'Fit')
plt.legend(frameon=True)
plt.title(r'K$_\alpha$- und K$_\beta$-Linie,1.Ordnung, LiF, verbessert')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
plt.savefig('Diagramme/LiF, 1.Ordnung, verbessert,.pdf')
```

```
Zusammengefasster Fit:
[ -1.04899585e+02   1.52300155e+03   2.54981790e+03   1.00594573e+01
   1.08925782e-01   7.84901651e+02   8.92028522e+00   1.15628919e-01] +- [
  4.22835246e+00   4.13962024e+01   3.73118089e+01   1.48176443e-03
   1.34948940e-03   2.56574109e+01   3.86705778e-03   3.84152295e-03] in d
er Reihenfolge
Steigung und y-Achsenabschnitt der Gerade, Normierung, Mittelwert und Sigm
a der Gaußfunktionen für k-alpha/K-beta Linie
```



In [10]:

```
#fit quality
chisquare=np.sum(((overkill(angleD,*popt)-intD)**2/intD_err**2))
dof=intD.size-len(popt)
chisquare_red=chisquare/dof
prob=round(1-chi2.cdf(chisquare,dof),2)*100
print('Chi^2 =', chisquare)
print('Chi^2 reduziert =', chisquare_red)
print('Fitwahrscheinlichkeit =', prob, '%')
```

```
Chi^2 = 16.498930828
Chi^2 reduziert = 0.717344818608
Fitwahrscheinlichkeit = 83.0 %
```

2. Ordnung

In [11]:

```
#Load data, second order
angleD2, intD2= np.loadtxt('Daten/01_03_2018 10_10_41.txt',
                          skiprows=0,
                          converters={0:string2float, 1:string2float},
                          unpack=True)
intD2_err = np.sqrt(intD2)
```

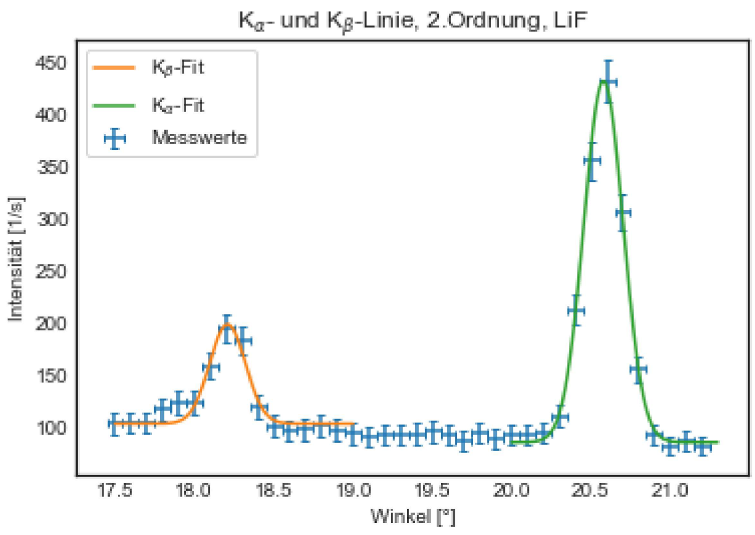
In [12]:

```
#beta line fit
maskB2 = [(angleD2>17.5)&(angleD2<19)]
angleB2 = angleD2[maskB2]
intB2 = intD2[maskB2]
intB2_err = intD2_err[maskB2]
poptB2, pcovB2 = curve_fit(gaussian, angleB2, intB2, sigma = intB2_err, p0 = [100, 18.3
, 0.2, 100])
perrB2 = np.sqrt(np.diag(pcovB2))

#alpha line fit
maskA2 = [(angleD2>20)&(angleD2<21.3)]
angleA2 = angleD2[maskA2]
intA2 = intD2[maskA2]
intA2_err = intD2_err[maskA2]
poptA2, pcovA2 = curve_fit(gaussian, angleA2, intA2, sigma = intA2_err,  p0 = [350, 20.
5, 0.2, 100])
perrA2 = np.sqrt(np.diag(pcovA2))

#fit values
print(r'K-beta-Fit:')
print(poptB2, '+-', perrB2)
print(r'K-alpha-Fit:')
print(poptA2, '+-', perrA2)
```

```
K-beta-Fit:
[  95.25283612   18.21022855    0.11323165  103.54948265] +- [ 9.92118079
0.01220244  0.01282603  3.02268    ]
K-alpha-Fit:
[  3.46443657e+02   2.05802276e+01   1.23755331e-01   8.57866636e+01] +- [
  8.44103700e+00   2.84605024e-03   2.92252739e-03   2.20139464e+00]
```

In [13]:

```
#plot data
xB2 = np.linspace(17.5, 19, 200)
xA2 = np.linspace(20, 21.3, 200)
plt.figure('second order')
plt.errorbar(angleD2, intD2, intD2_err, angle_err, linestyle = 'None', label = 'Messwer
te')
plt.plot(xB2, gaussian(xB2, *poptB2), marker = '', label = r'K$_\beta$-Fit')
plt.plot(xA2, gaussian(xA2, *poptA2), marker = '', label = r'K$_\alpha$-Fit')
plt.legend(frameon=True)
plt.title(r'K$_\alpha$- und K$_\beta$-Linie, 2.Ordnung, LiF')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
plt.savefig('Diagramme/LiF, 2.Ordnung.pdf')
```



In [14]:

```
#fit quality, beta-Peak
chisquare=np.sum(((gaussian(angleB2,*poptB2)-intB2)**2/intB2_err**2))
dof=intB2.size-len(poptB2)
chisquare_red=chisquare/dof
prob=round(1-chi2.cdf(chisquare,dof),2)*100
print('Chi^2 =', chisquare)
print('Chi^2 reduziert =', chisquare_red)
print('Fitwahrscheinlichkeit =', prob, '%')

#fit quality, alpha-Peak
chisquare=np.sum(((gaussian(angleA2,*poptA2)-intA2)**2/intA2_err**2))
dof=intA2.size-len(poptA2)
chisquare_red=chisquare/dof
prob=round(1-chi2.cdf(chisquare,dof),2)*100
print('Chi^2 =', chisquare)
print('Chi^2 reduziert =', chisquare_red)
print('Fitwahrscheinlichkeit =', prob, '%')
```

```
Chi^2 = 6.95145525988
Chi^2 reduziert = 0.695145525988
Fitwahrscheinlichkeit = 73.0 %
Chi^2 = 2.2679239262
Chi^2 reduziert = 0.283490490775
Fitwahrscheinlichkeit = 97.0 %
```

Bestimmen der Planckschen Konstante

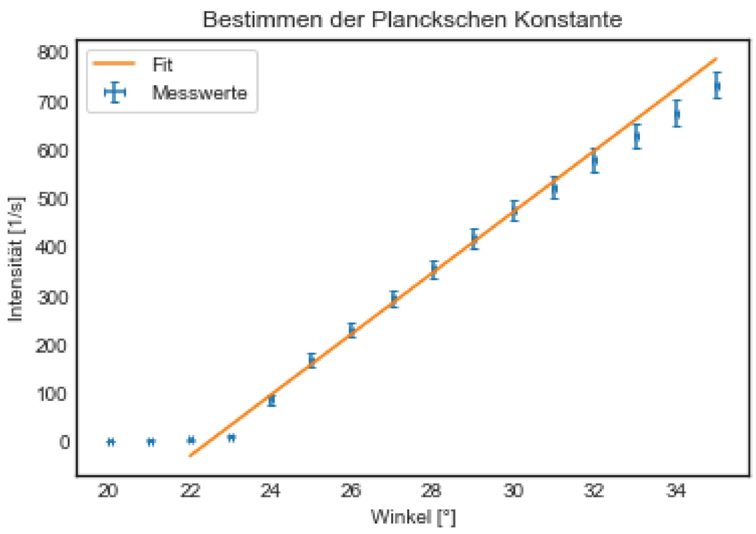
In [15]:

```
#load data
U = np.arange(20, 36)
U_err = 0.05
I = np.array([1.3, 2.55, 4.9, 11.55, 87.2, 168.9, 230.0, 293.7, 355.4, 419.2, 475.6, 52
2.6, 580.5, 630.5, 676.5, 733.2])
I_err = np.sqrt(I)

#linear fit
popt, pcov = curve_fit(linear, U[4:-3], I[4:-3], sigma = I_err[4:-3])
perr = np.sqrt(np.diag(pcov))
```

In [16]:

```
#plot data and fit
x = np.linspace(22, 35, 200)
plt.figure('plack2')
plt.errorbar(U, I, I_err, U_err, linestyle = 'None', label = 'Messwerte')
plt.plot(x, linear(x, *popt), label = 'Fit')
plt.legend(frameon=True)
plt.title('Bestimmen der Planckschen Konstante')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
print('Fit-Parameter:')
print(popt, '+-', perr)
plt.savefig('Diagramme/Planck2.pdf')
```

```
Fit-Parameter:
[   62.75632578 -1409.42846244] +- [   1.46224156   38.82422388]
```



K-alpha und K-beta Peaks im NaCl Kristall

In [17]:

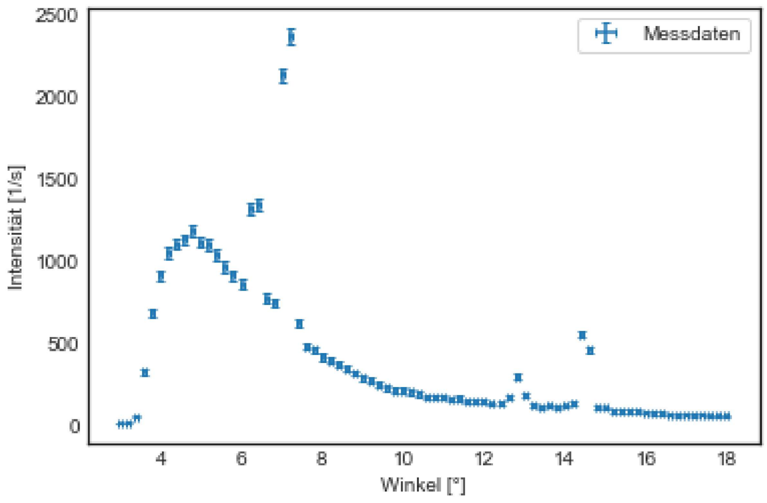
```
#load data
angleD, intD= np.loadtxt('Daten/01_03_2018 10_40_33.txt',
                         skiprows=0,
                         converters={0:string2float, 1:string2float},
                         unpack=True)
intD_err = np.sqrt(intD)
```

In [18]:

```
plt.figure('nacl')
plt.errorbar(angleD, intD, intD_err, angle_err, label = 'Messdaten', linestyle = 'None'
)
plt.legend(frameon=True)
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
```

Out[18]:

```
Text(0,0.5,'Intensität [1/s]')
```



1. Ordnung

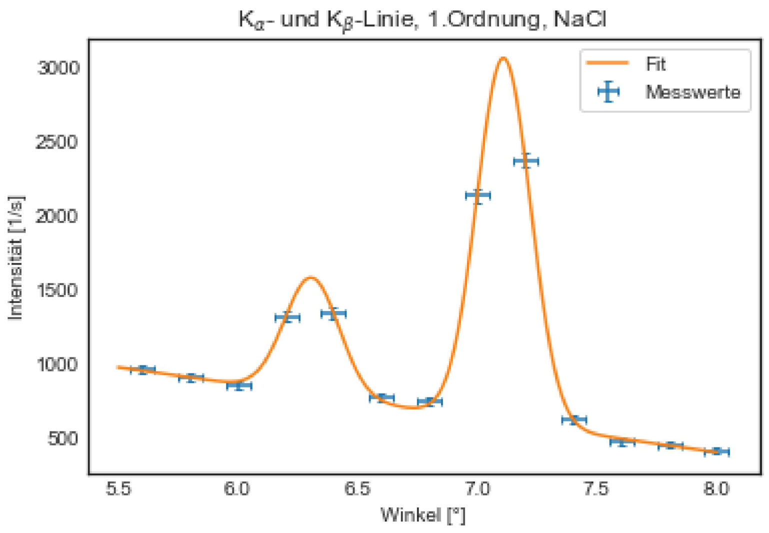
In [19]:

```
#data
angleD1 = angleD[13:26]
intD1 = intD[13:26]
intD1_err = intD_err[13:26]
```

In [20]:

```
#overkill fit
popt1, pcov1 = curve_fit(overkill, angleD1, intD1, sigma = intD1_err, p0 = [-250, 2000,
 500, 6.5, 0.2, 1600, 7.3, 0.2])
perr1 = np.sqrt(np.diag(pcov1))

#fit values
print('Zusammengefasster Fit:')
print(popt1, '+-', perr1)
print('in der Reihenfolge')
print('Steigung und y-Achsenabschnitt der Gerade, Normierung, Mittelwert und Sigma der
 Gaußfunktionen für k-alpha/K-beta Linie')

#plot data again
plt.figure('first order, nacl')
x1 = np.linspace(5.5, 8, 300)
plt.errorbar(angleD1, intD1, intD1_err, angle_err, linestyle = 'None', label = 'Messwer
te')
plt.plot(x1, overkill(x1, *popt1), marker = '', label = 'Fit')
plt.legend(frameon=True)
plt.title(r'K$_\alpha$- und K$_\beta$-Linie, 1.Ordnung, NaCl')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
plt.savefig('Diagramme/NaCl, 1.Ordnung.pdf')
```

```
Zusammengefasster Fit:
[ -2.28572756e+02   2.23715244e+03   7.86695060e+02   6.30888607e+00
   1.14006668e-01   2.44807877e+03   7.11069847e+00   1.11849325e-01] +- [
  7.08609226e+00   5.18506960e+01   5.87731665e+01   4.16980442e-03
   1.11508593e-02   6.32796630e+01   1.65758821e-03   3.25254071e-03]
in der Reihenfolge
Steigung und y-Achsenabschnitt der Gerade, Normierung, Mittelwert und Sigm
a der Gaußfunktionen für k-alpha/K-beta Linie
```



In [21]:

```
#fit quality
chisquare=np.sum(((overkill(angleD1,*popt1)-intD1)**2/intD1_err**2))
dof=intD1.size-len(popt1)
chisquare_red=chisquare/dof
prob=round(1-chi2.cdf(chisquare,dof),2)*100
print('Chi^2 =', chisquare)
print('Chi^2 reduziert =', chisquare_red)
print('Fitwahrscheinlichkeit =', prob, '%')
```

```
Chi^2 = 2.04009116841
Chi^2 reduziert = 0.408018233683
Fitwahrscheinlichkeit = 84.0 %
```

2. Ordnung

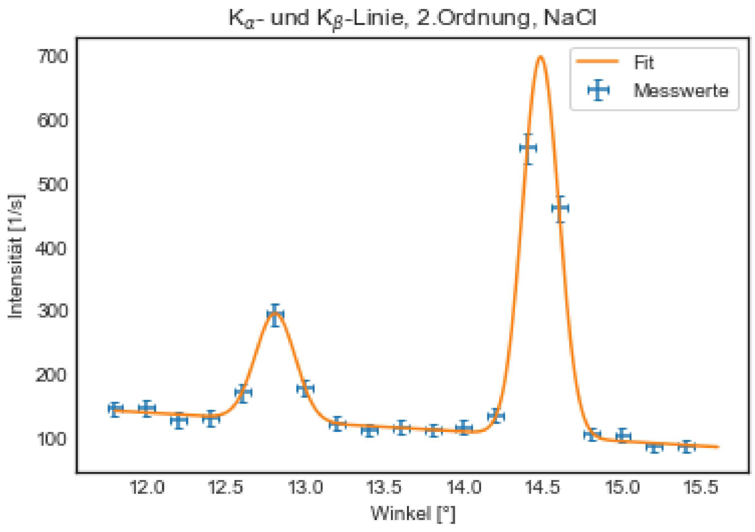
In [22]:

```
#data
angleD2 = angleD[44:63]
intD2 = intD[44:63]
intD2_err = intD_err[44:63]
```

In [23]:

```
#overkill fit
popt2, pcov2 = curve_fit(overkill, angleD2, intD2, sigma = intD2_err, p0 = [-1, 200, 50
, 13, 0.2, 300, 14.5, 0.2])
perr2 = np.sqrt(np.diag(pcov2))

#fit values
print('Zusammengefasster Fit:')
print(popt2, '+-', perr2)
print('in der Reihenfolge')
print('Steigung und y-Achsenabschnitt der Gerade, Normierung, Mittelwert und Sigma der
 Gaußfunktionen für k-alpha/K-beta Linie')

#plot data again
plt.figure('second order, nacl')
x2 = np.linspace(11.8, 15.6, 300)
plt.errorbar(angleD2, intD2, intD2_err, angle_err, linestyle = 'None', label = 'Messwer
te')
plt.plot(x2, overkill(x2, *popt2), marker = '', label = 'Fit')
plt.legend(frameon=True)
plt.title(r'K$_\alpha$- und K$_\beta$-Linie, 2.Ordnung, NaCl')
plt.xlabel('Winkel [°]')
plt.ylabel('Intensität [1/s]')
plt.savefig('Diagramme/NaCl, 1.Ordnung.pdf')
```

```
Zusammengefasster Fit:
[ -1.49722454e+01   3.19484981e+02   1.67957518e+02   1.28112963e+01
  -1.24895206e-01   5.96748420e+02   1.44845932e+01   1.14168372e-01] +- [
  1.37302522e+00   1.89301437e+01   9.55588288e+00   8.49789208e-03
   6.32461379e-03   2.11836992e+01   2.77418797e-03   4.24545070e-03]
in der Reihenfolge
Steigung und y-Achsenabschnitt der Gerade, Normierung, Mittelwert und Sigm
a der Gaußfunktionen für k-alpha/K-beta Linie
```



In [24]:

```
#fit quality
chisquare=np.sum(((overkill(angleD2,*popt2)-intD2)**2/intD2_err**2))
dof=intD2.size-len(popt2)
chisquare_red=chisquare/dof
prob=round(1-chi2.cdf(chisquare,dof),2)*100
print('Chi^2 =', chisquare)
print('Chi^2 reduziert =', chisquare_red)
print('Fitwahrscheinlichkeit =', prob, '%')
```

```
Chi^2 = 3.23255809978
Chi^2 reduziert = 0.293868918162
Fitwahrscheinlichkeit = 99.0 %
```