analyze-20190117-n37

April 9, 2019

```
In [1]: # 2019/04/08
        # Code to analyze 20190117-n37 data
```

```
In [2]: import numpy as np
        import pandas as pd
        from uncertainties import unumpy, ufloat

        import os
        import glob

        import matplotlib.pyplot as plt
        from pandas.plotting import register_matplotlib_converters
        register_matplotlib_converters()
        import seaborn as sns

        from lmfit.models import Model, ConstantModel, PolynomialModel, GaussianModel, VoigtMod

        sns.set_context('talk')
```

1 Some constant variables

```
In [3]: # List of files to glob
        POL_LIST = glob.glob('n37-dimer_states-pol-Pol*_out.csv')+['n37-dimer_states-pol-Repet
        UNPOL_LIST = glob.glob('n37-dimer_states-pol-Unpol*_out.csv')+['n37-dimer_states-unpol-

        SET_NAME = '20190117-n37'
        POL_NAME = '20190117-n37p'
        UNPOL_NAME = '20190117-n37u'

        ATOMIC = 1671.24
        ZEEMAN = 1.60839
        ORDER = -1

        SATURATION_A = ufloat(44.0442405, 1.65366925)
        SATURATION_B = ufloat(-0.05852900, 0.00456716)
        NUM_EXPOSURES = 1000 # Number of excitation loops
```

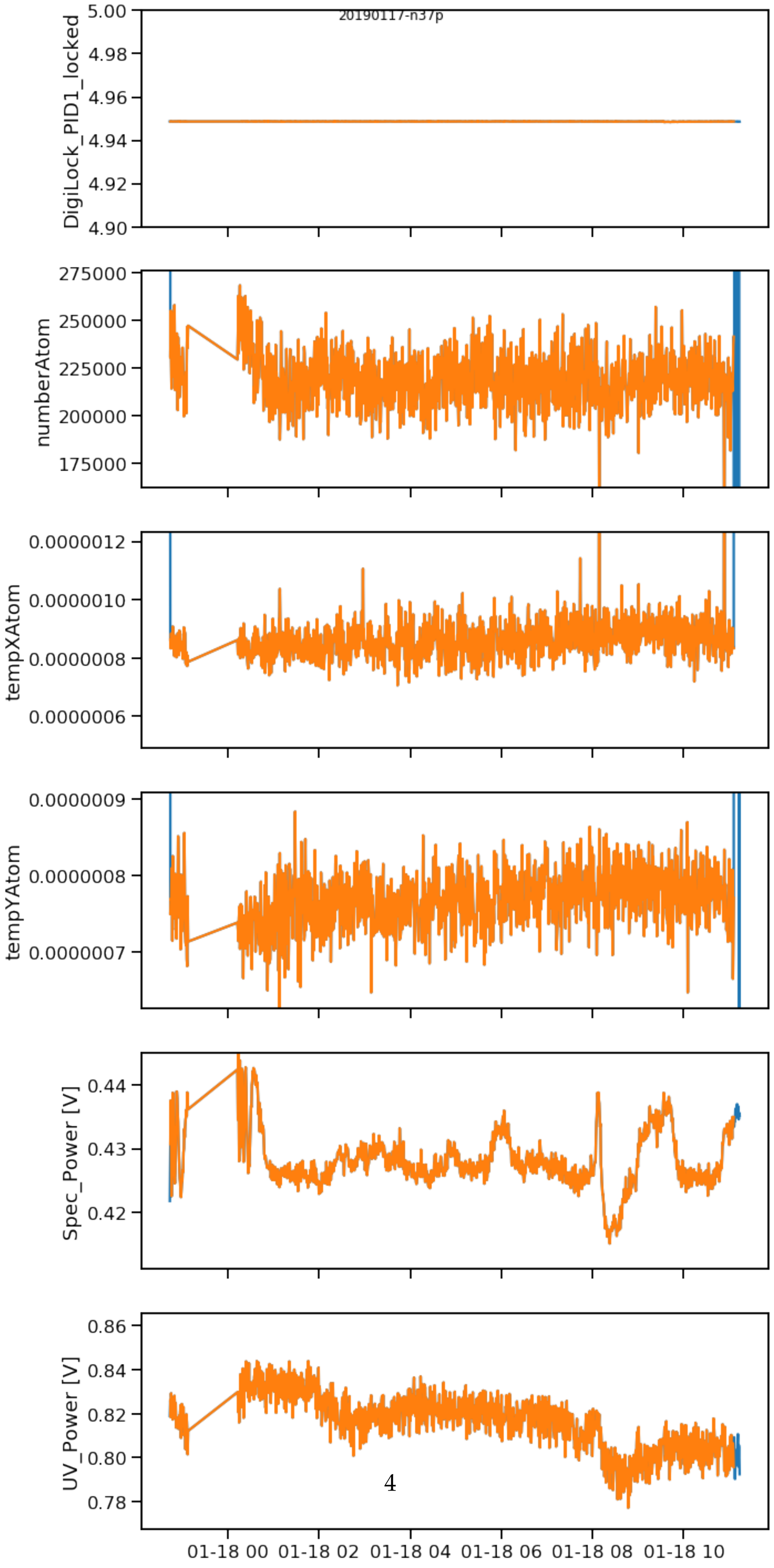
2 Retrieve files and do some preprocessing

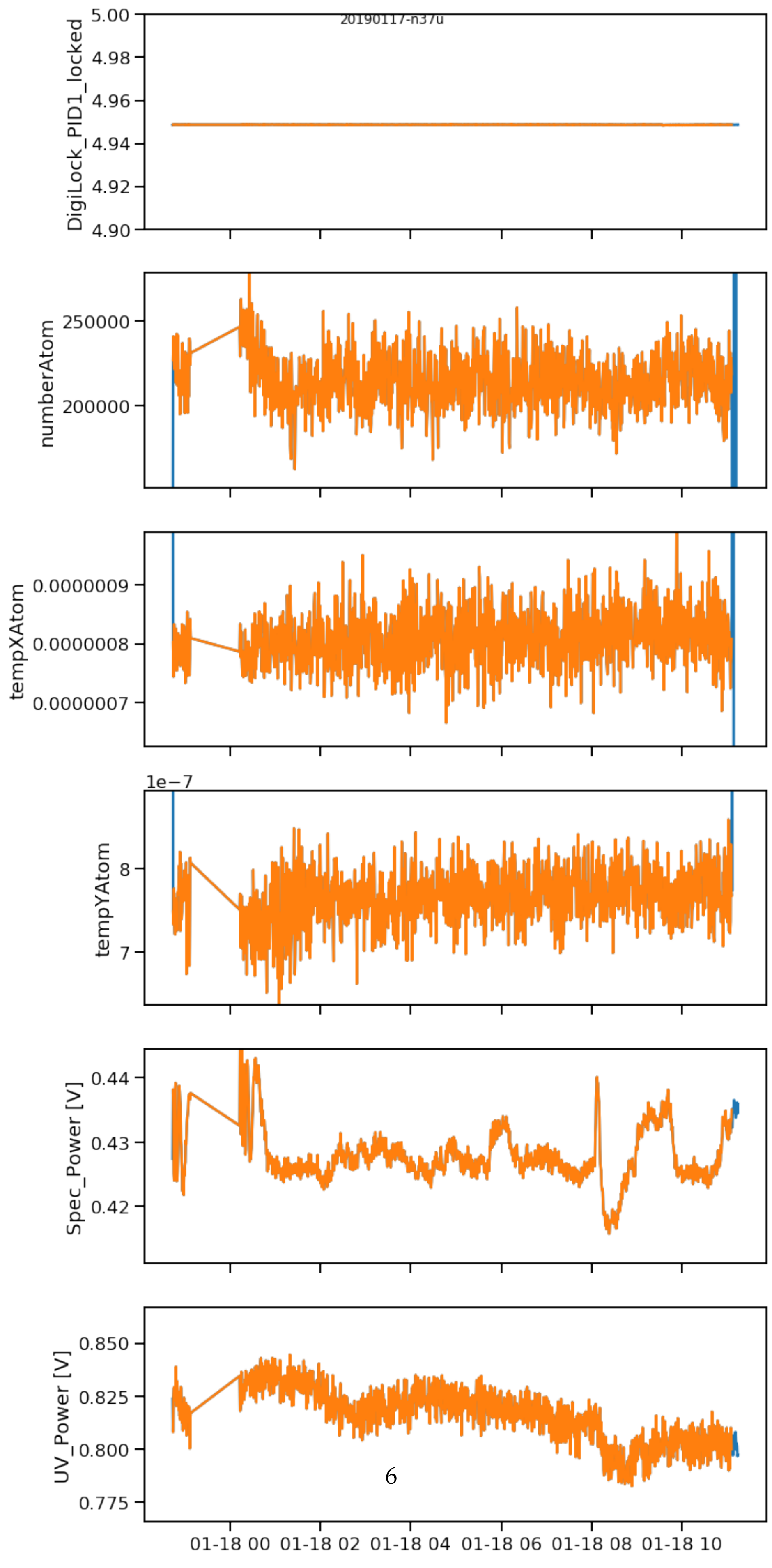
2.0.1 Retrieve all the data files and put in to a single DataFrame

```
In [4]: # Last modified 2019/04/05
        # Goes through all listed .csv files and returns data in a single DataFrame
        def gather_data(file_list):

            # Read in .csv files in to single DataFrame
            out = pd.DataFrame()
            for file_name in file_list:
                df = pd.read_csv(file_name)
                df['Scan'] = os.path.basename(file_name)
                df['Timestamp'] = pd.to_datetime(df['Timestamp'])

                out = pd.concat([out, df], join='outer', ignore_index=True)

            return out

        pol_data = gather_data(POL_LIST)
        unpol_data = gather_data(UNPOL_LIST)
```

2.0.2 Reduce the data by removing bad points

```
In [5]: # First pass at removing bad data points
        def reduce_data(df_data, name, show_plots):

            # Remove data where UV laser was not locked based on 'DigiLock_PID1_locked' column
            df_reduced = df_data[df_data['DigiLock_PID1_locked'] >= 4.9]

            # Remove data where atom number was above/below cutoff
            df_reduced = df_reduced[(100e3 <= df_reduced['numberAtom']) & (df_reduced['numberA

            # Remove data where tempYAtom was above/below cutoff
            df_reduced = df_reduced[(500E-9 <= df_reduced['tempYAtom']) & (df_reduced['tempYAt

            # Plotting original and reduced data sets
            df_orig = df_data.sort_values(by=['Timestamp'])
            df_redu = df_reduced.sort_values(by=['Timestamp'])

            if show_plots:
                [fig, ax]= plt.subplots(nrows=6, ncols=1, figsize=(10,20), sharex=True, sharey=
                fig.suptitle(name)

                ax[0].set_ylabel('DigiLock_PID1_locked')
                ax[0].plot(df_orig['Timestamp'], df_orig['DigiLock_PID1_locked'])
                ax[0].plot(df_redu['Timestamp'], df_redu['DigiLock_PID1_locked'])
                ax[0].set_ylim((4.9, 5))
                ax[1].set_ylabel('numberAtom')
                ax[1].plot(df_orig['Timestamp'], df_orig['numberAtom'])
                ax[1].plot(df_redu['Timestamp'], df_redu['numberAtom'])
                avg = np.mean(df_redu['numberAtom'])
                std = np.std(df_redu['numberAtom'])
                ax[1].set_ylim((avg-4*std, avg+4*std))

                ax[2].set_ylabel('tempXAtom')
                ax[2].plot(df_orig['Timestamp'], df_orig['tempXAtom'])
                ax[2].plot(df_redu['Timestamp'], df_redu['tempXAtom'])
                avg = np.mean(df_redu['tempXAtom'])
                std = np.std(df_redu['tempXAtom'])
                ax[2].set_ylim((avg-4*std, avg+4*std))

                ax[3].set_ylabel('tempYAtom')
                ax[3].plot(df_orig['Timestamp'], df_orig['tempYAtom'])
                ax[3].plot(df_redu['Timestamp'], df_redu['tempYAtom'])
                avg = np.mean(df_redu['tempYAtom'])
                std = np.std(df_redu['tempYAtom'])
                ax[3].set_ylim((avg-4*std, avg+4*std))

                ax[4].set_ylabel('Spec_Power [V]')
                ax[4].plot(df_orig['Timestamp'], df_orig['Spec_Power'])
                ax[4].plot(df_redu['Timestamp'], df_redu['Spec_Power'])
                avg = np.mean(df_redu['Spec_Power'])
                std = np.std(df_redu['Spec_Power'])
                ax[4].set_ylim((avg-4*std, avg+4*std))

                ax[5].set_ylabel('UV_Power [V]')
                ax[5].plot(df_orig['Timestamp'], df_orig['UV_Power'])
                ax[5].plot(df_redu['Timestamp'], df_redu['UV_Power'])
                avg = np.mean(df_redu['UV_Power'])
                std = np.std(df_redu['UV_Power'])
                ax[5].set_ylim((avg-4*std, avg+4*std))

                plt.tight_layout()
                plt.show()

            # Drop unused data columns
            df_reduced.drop(columns=['AI3', 'AI4', 'AI5', 'AI6', 'AI7'], inplace=True)

            return df_reduced

        pol_reduced = reduce_data(pol_data, POL_NAME, True)
        unpol_reduced = reduce_data(unpol_data, UNPOL_NAME, True)
```


20190117-n37p
DigiLock_PID1_locked
5.00
4.98
4.96
4.94
4.92
4.90
numberAtom
275000
250000
225000
200000
175000
tempXAtom
0.0000012
0.0000010
0.0000008
0.0000006
tempYAtom
0.0000009
0.0000008
0.0000007
Spec_Power [V]
0.44
0.43
0.42
UV_Power [V]
0.86
0.84
0.82
0.80
0.78
01-18 00
01-18 02
01-18 04
01-18 06
01-18 08
01-18 10



20190117-n37u
DigiLock_PID1_locked
5.00
4.98
4.96
4.94
4.92
4.90
numberAtom
250000
200000
tempXAtom
0.0000009
0.0000008
0.0000007
1e−7
tempYAtom
8
7
Spec_Power [V]
0.44
0.43
0.42
UV_Power [V]
0.850
0.825
0.800
0.775
01-18 00
01-18 02
01-18 04
01-18 06
01-18 08
01-18 10

2.0.3 Aggregrate data

```
In [6]: def aggregrate_data(df_data, indVar):
            df_grouped = df_data.groupby(by=indVar, as_index=False, sort=True)

            #df_out = df_grouped.agg([np.mean, np.std])
            df_out = df_grouped.agg({'sfiIntegral': [np.mean, np.std],
                                     'UV_Power': [np.mean, np.std],
                                     'Spec_Power': [np.mean, np.std],
                                     'numberAtom': [np.mean, np.std],
                                     'tempXAtom':  [np.mean, np.std],
                                     'tempYAtom':  [np.mean, np.std],})

            return df_out

        indVar = ['imagevcoAtom']

        pol_out = aggregrate_data(pol_reduced, indVar)
        unpol_out = aggregrate_data(unpol_reduced, indVar)
```

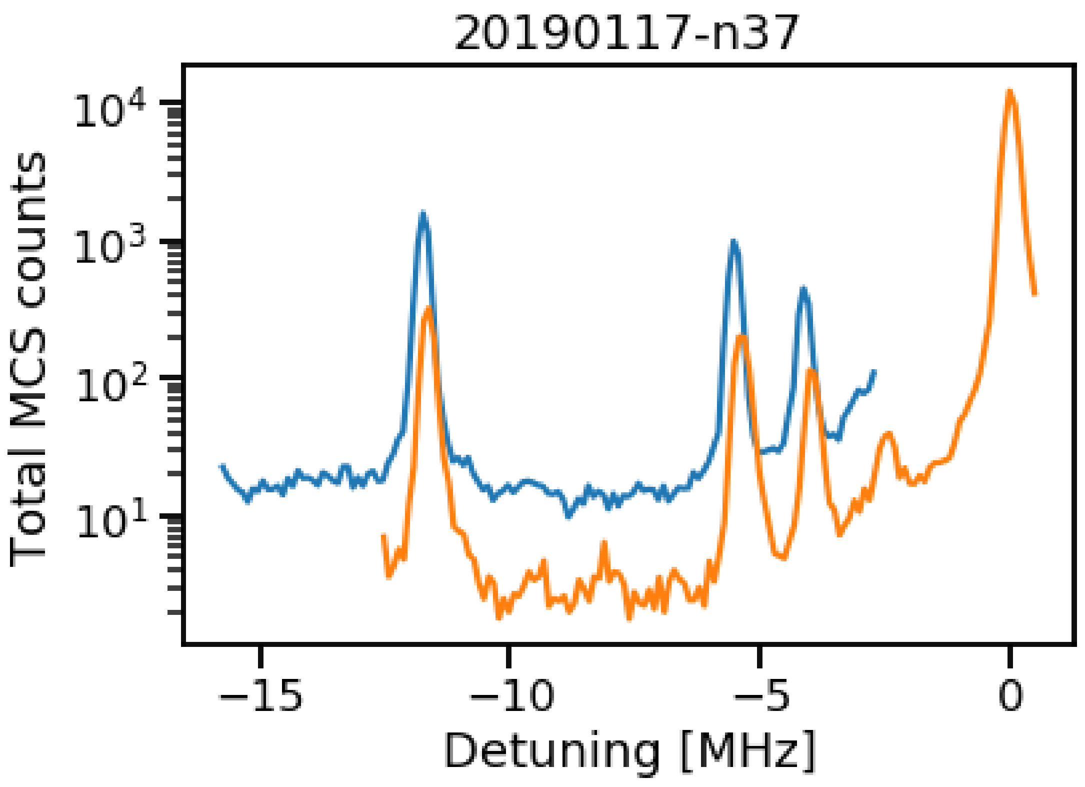
2.0.4 Adjust counts due to MCP saturation

```
In [7]: # Using the MCP saturation curve taken on 2019/03/27 by varying the UV power, we can a

        def mcp_saturation(df):
            df_temp = df
            observed_counts = unumpy.uarray(df_temp['sfiIntegral','mean'], df_temp['sfiIntegra
            observed_counts = observed_counts/NUM_EXPOSURES

            corrected_counts = -SATURATION_A*unumpy.log(1-observed_counts/SATURATION_A)
            corrected_counts = NUM_EXPOSURES*corrected_counts

            df_temp['sfiIntegral_linearized','mean'] = unumpy.nominal_values(corrected_counts)
            df_temp['sfiIntegral_linearized', 'std'] = unumpy.std_devs(corrected_counts)

            return df_temp

        pol_out = mcp_saturation(pol_out)
        unpol_out = mcp_saturation(unpol_out)
```

2.0.5 Calculate detunings

```
In [8]: def calc_detuning(df, atomic, zeeman, order):
            df['detuning'] = order*2*(df['imagevcoAtom'] - (atomic + order*zeeman))
            return df

        pol_out = calc_detuning(pol_out, ATOMIC, ZEEMAN, ORDER)
        unpol_out = calc_detuning(unpol_out, ATOMIC, 0, ORDER)
```

```
In [9]: plt.figure()
        plt.plot(pol_out['detuning'], pol_out['sfiIntegral_linearized']['mean'])
        plt.plot(unpol_out['detuning'], unpol_out['sfiIntegral_linearized']['mean'])
        plt.yscale('log')
        plt.title(SET_NAME)
        plt.xlabel('Detuning [MHz]')
        plt.ylabel('Total MCS counts')
        plt.show()
```



3 Analyzing the data

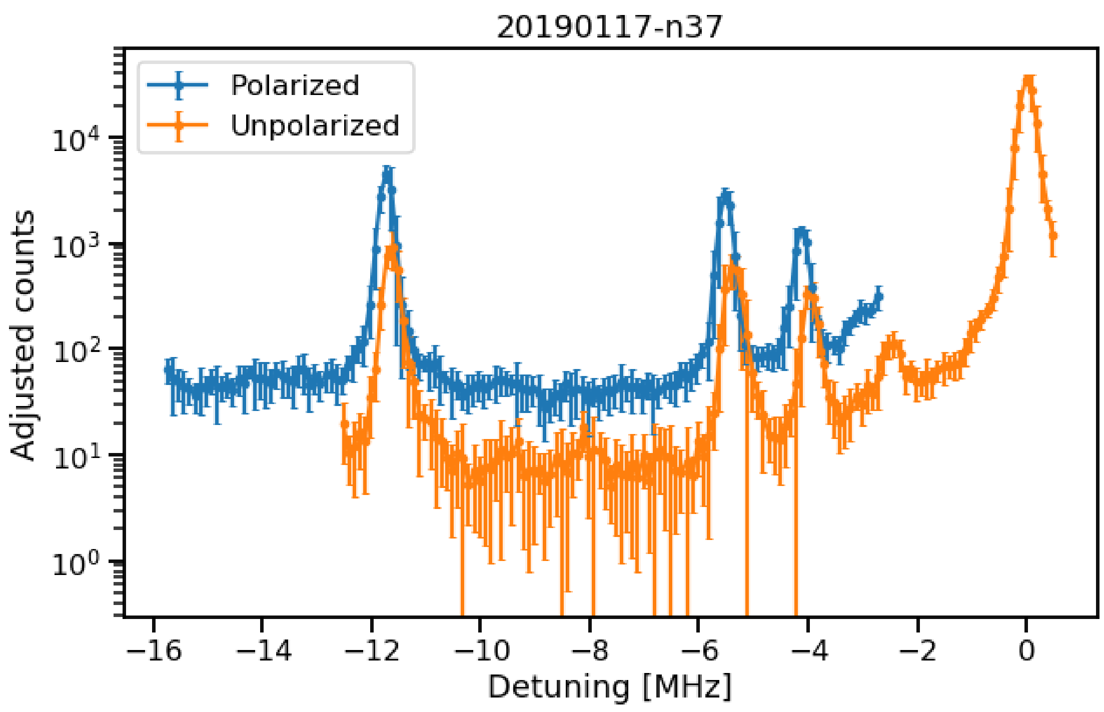
```
In [10]: pol_data = pol_out
         unpol_data = unpol_out
```

3.0.1 Adjust counts based on 689 nm and UV intensities

```
In [11]: def adjust_for_intensities(df):
             df_temp = df
```

```
        raw_sfiIntegral = unumpy.uarray(df_temp['sfiIntegral_linearized', 'mean'], df_temp
        raw_UV_Power = unumpy.uarray(df_temp['UV_Power', 'mean'], df_temp['UV_Power', 'st
        raw_Spec_Power = unumpy.uarray(df_temp['Spec_Power', 'mean'], df_temp['Spec_Power

        adjusted_sfiIntegral = raw_sfiIntegral/raw_UV_Power/raw_Spec_Power

        df_temp['sfiIntegral_adjusted', 'mean'] = unumpy.nominal_values(adjusted_sfiInteg
        df_temp['sfiIntegral_adjusted', 'std'] = unumpy.std_devs(adjusted_sfiIntegral)

        return df_temp

    # Correcting for 320 nm and 689 nm intensities point-by-point

    pol_data = adjust_for_intensities(pol_data)
    unpol_data = adjust_for_intensities(unpol_data)
```

```
In [12]: # Plotting results after adjusting for 320 nm and 689 nm powers

         plt.figure(figsize=(10,6))
         plt.errorbar(pol_data['detuning'], pol_data['sfiIntegral_adjusted', 'mean'],
                      yerr=pol_data['sfiIntegral_adjusted', 'std'],
                      fmt='.-', capsize=2, label='Polarized')
         plt.errorbar(unpol_data['detuning'], unpol_data['sfiIntegral_adjusted', 'mean'],
                      yerr=unpol_data['sfiIntegral_adjusted', 'std'],
                      fmt='.-', capsize=2, label='Unpolarized')
         plt.gca().set_yscale('log')
         plt.title(SET_NAME)
         plt.xlabel('Detuning [MHz]')
         plt.ylabel('Adjusted counts')
         plt.legend()
         plt.show()
```



3.1 Fitting the data

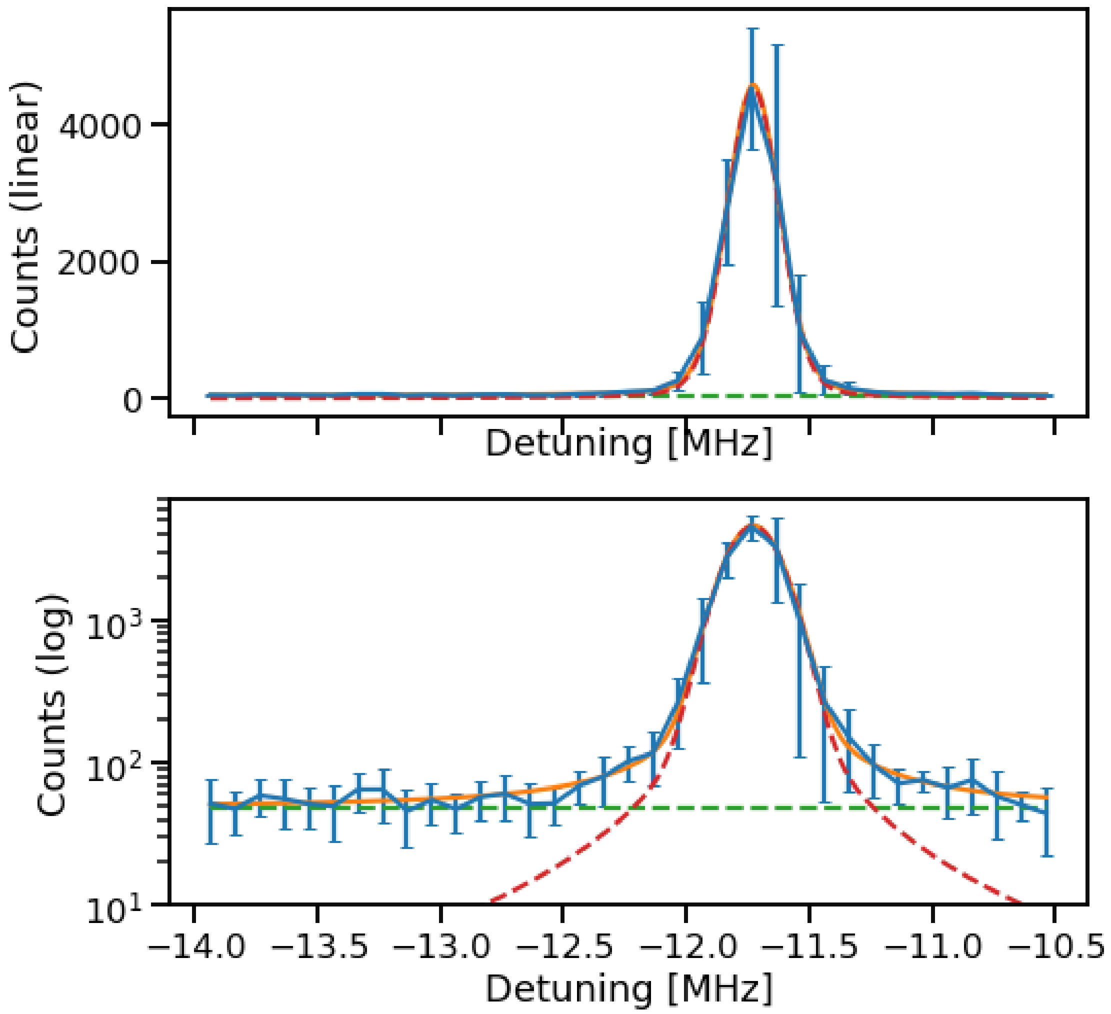
3.1.1 n=37, polarized sample

n=37, polarized, dimer, ground state

In [13]:
```
data = pol_data
plot_components = True
plot_initial_guess = False

# Choosing a range to fit over
xmin = -14
xmax = -10.5

xdata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['detuning']
ydata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegral_ad
ydata_unc = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegra

# Specify fit model
bg = PolynomialModel(degree=0, prefix='bg_')
d0 = PseudoVoigtModel(prefix='d0_')

fit_model = bg + d0
fit_params = fit_model.make_params()
```

```
fit_params['bg_c0'].set(min(ydata), vary=True)
#fit_params['bg_c1'].set(0, vary=True)
#fit_params['bg_c2'].set(0, vary=True)

fit_params['d0_center'].set(-11.73, vary=True)
fit_params['d0_amplitude'].set(1110, min=0, vary=True)
fit_params['d0_sigma'].set(0.1, min=0, vary=True)

# Initial guess
xfit = np.linspace(min(xdata), max(xdata), num=1000, endpoint=True)
yinit = fit_model.eval(fit_params, x=xfit)

# Fitting
fit_result = fit_model.fit(ydata, fit_params, x=xdata)
dely = fit_result.eval_uncertainty(x=xdata)
yfit = fit_model.eval(fit_result.params, x=xfit)

print(fit_result.fit_report())

# Plotting output
[fig, axs] = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=False, figsize=(8,8))

axs[0].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[0].plot(xfit, yfit, label='fit')
axs[0].set_xlabel('Detuning [MHz]')
axs[0].set_ylabel('Counts (linear)')

axs[1].set_yscale('log')
axs[1].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[1].plot(xfit, yfit, label='fit')
axs[1].set_xlabel('Detuning [MHz]')
axs[1].set_ylabel('Counts (log)')
axs[1].set_ylim(bottom=10)

if plot_components:
    # generate components
    fit_comps = fit_result.eval_components(x=xfit)

    for key in fit_comps.keys():
        for ax in axs:
            ax.plot(xfit, fit_comps[key], '--')

if plot_initial_guess:
    for ax in axs:
        ax.plot(xfit, yinit, '-.', label='init')

plt.show()
```

```
[[Model]]
    (Model(polynomial, prefix='bg_') + Model(pvoigt, prefix='d0_'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 31
    # data points      = 35
    # variables        = 5
    chi-square         = 22988.4853
    reduced chi-square = 766.282843
    Akaike info crit   = 237.059023
    Bayesian info crit = 244.835764
[[Variables]]
    bg_c0:         48.5367539 +/- 6.46553452 (13.32%) (init = 44.17786)
    d0_fraction:   0.22958763 +/- 0.03653210 (15.91%) (init = 0.5)
    d0_sigma:      0.12703631 +/- 9.6731e-04 (0.76%) (init = 0.1)
    d0_center:    -11.7278682 +/- 6.9608e-04 (0.01%) (init = -11.73)
    d0_amplitude:  1323.24082 +/- 17.9169808 (1.35%) (init = 1110)
    d0_fwhm:       0.25407262 +/- 0.00193463 (0.76%) == '2.0000000*d0_sigma'
    d0_height:     4530.61762 +/- 27.9680452 (0.62%) == '(((1-d0_fraction)*d0_amplitude)/(d0_s
[[Correlations]] (unreported correlations are < 0.100)
    C(d0_fraction, d0_amplitude) =  0.892
    C(bg_c0, d0_amplitude)       = -0.689
    C(bg_c0, d0_fraction)        = -0.594
    C(d0_fraction, d0_sigma)     = -0.440
    C(d0_sigma, d0_amplitude)    = -0.133
```



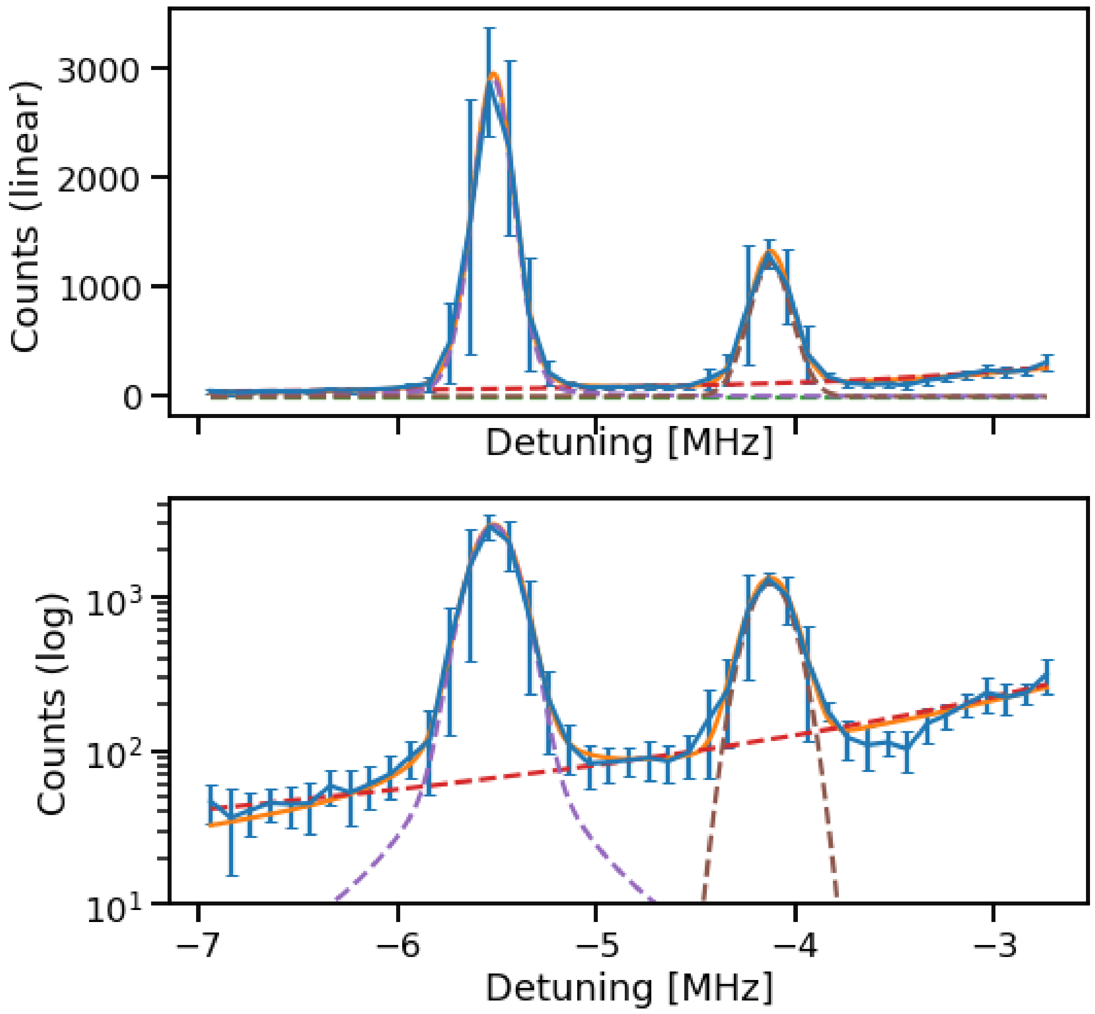
```
In [14]: # Extracting the fitted parameters

         labview_params = [('numberAtom', 'mean'), ('tempXAtom', 'mean'), ('tempYAtom', 'mean')

         print('Xmin [MHz]:\t' + str(min(xdata)))
         print('Xmax [MHz]:\t' + str(max(xdata)))
         for p in labview_params:
             print(p[0] + '(mean):\t' + str(np.mean(data[(xmin <= data['detuning']) & (data['de
             print(p[0] + '(std):\t' + str(np.std(data[(xmin <= data['detuning']) & (data['detu

         # Saving to variables
         d0_pol_int = ufloat(fit_result.params['d0_amplitude'].value, fit_result.params['d0_am
         d0_pol_N = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax
                       np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)
         d0_pol_Ty = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xma
                        np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax
Xmin [MHz]:        -13.936779999999999
Xmax [MHz]:        -10.536780000000363
numberAtom(mean):        221406.6121269841
numberAtom(std):        4996.366336551975
tempXAtom(mean):        8.650717184761904e-07
tempXAtom(std):        1.8130747192595513e-08
tempYAtom(mean):        7.699863187301588e-07
tempYAtom(std):        1.1030080251084386e-08
UV_Power(mean):        0.8167016252790178
UV_Power(std):        0.0016085922719757363
Spec_Power(mean):        0.42880945986793156
Spec_Power(std):        0.0009978833261078603
```

n=37, polarized, dimer, 1st and 2nd excited states

```
In [15]: data = pol_data
         plot_components = True
         plot_initial_guess = False

         # Choosing a range to fit over
         xmin = -7
         xmax = 0

         xdata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['detuning']
         ydata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegral_ad
         ydata_unc = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegra

         # Specify fit model
         bg = PolynomialModel(degree=0, prefix='bg_')
         a = PseudoVoigtModel(prefix='a_')
         d1 = PseudoVoigtModel(prefix='d1_')
         d2 = PseudoVoigtModel(prefix='d2_')
         #d3 = PseudoVoigtModel(prefix='d3_')

         #fit_model = bg + a + d1 + d2 + d3
         fit_model = bg + a + d1 + d2
         fit_params = fit_model.make_params()

         fit_params['bg_c0'].set(min(ydata), vary=True)
         #fit_params['bg_c1'].set(2, vary=True)
         #fit_params['bg_c2'].set(0, vary=True)

         fit_params['a_center'].set(0, vary=False)
         fit_params['a_amplitude'].set(112863, vary=True)
         fit_params['a_sigma'].set(0.1, min=0, vary=True)

         fit_params['d1_center'].set(-5.52, min=-6, max=-5, vary=True)
```

```
fit_params['d1_amplitude'].set(828, min=0, vary=True)
fit_params['d1_sigma'].set(0.1, min=0, vary=True)

fit_params['d2_center'].set(-4.12, min=-4.5, max=-3.75, vary=True)
fit_params['d2_amplitude'].set(332, min=0, vary=True)
fit_params['d2_sigma'].set(0.1, min=0, vary=True)

#fit_params['d3_center'].set(-2.75, min=-3.5, max=-2.75, vary=True)
#fit_params['d3_amplitude'].set(100, min=0, vary=True)
#fit_params['d3_sigma'].set(0.1, min=0, vary=True)

# Initial guess
xfit = np.linspace(min(xdata), max(xdata), num=1000, endpoint=True)
yinit = fit_model.eval(fit_params, x=xfit)

# Fitting
fit_result = fit_model.fit(ydata, fit_params, x=xdata)
dely = fit_result.eval_uncertainty(x=xdata)
yfit = fit_model.eval(fit_result.params, x=xfit)

print(fit_result.fit_report())

# Plotting output
[fig, axs] = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=False, figsize=(8,8))

axs[0].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[0].plot(xfit, yfit, label='fit')
axs[0].set_xlabel('Detuning [MHz]')
axs[0].set_ylabel('Counts (linear)')

axs[1].set_yscale('log')
axs[1].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[1].plot(xfit, yfit, label='fit')
axs[1].set_xlabel('Detuning [MHz]')
axs[1].set_ylabel('Counts (log)')
axs[1].set_ylim(bottom=10)

if plot_components:
    # generate components
    fit_comps = fit_result.eval_components(x=xfit)

    for key in fit_comps.keys():
        for ax in axs:
            ax.plot(xfit, fit_comps[key], '--')

if plot_initial_guess:
    for ax in axs:
        ax.plot(xfit, yinit, '-.', label='init')
        plt.show()

C:\Users\reyalp\Anaconda3\lib\site-packages\lmfit\lineshapes.py:31: RuntimeWarning: divide by z
  return (amplitude/(s2pi*sigma)) * exp(-(1.0*x-center)**2 / (2*sigma**2))


[[Model]]
    (((Model(polynomial, prefix='bg_') + Model(pvoigt, prefix='a_')) + Model(pvoigt, prefix='d1
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 13650
    # data points      = 43
    # variables        = 12
    chi-square         = 22724.0560
    reduced chi-square = 733.034066
    Akaike info crit   = 293.609108
    Bayesian info crit = 314.743510
[[Variables]]
    bg_c0:        -12.5381278 +/- 22.9569319 (183.10%) (init = 36.39904)
    a_fraction:    0.75786470 +/- 657425.932 (86747137.20%) (init = 0.5)
    a_sigma:       0.04722322 +/- 34.2400381 (72506.78%) (init = 0.1)
    a_center:      0 (fixed)
    a_amplitude:   176563.150 +/- 1.5317e+11 (86751735.03%) (init = 112863)
    d1_fraction:   0.20756915 +/- 0.06423479 (30.95%) (init = 0.5)
    d1_sigma:      0.12529480 +/- 0.00150342 (1.20%) (init = 0.1)
    d1_center:    -5.51545845 +/- 0.00105542 (0.02%) (init = -5.52)
    d1_amplitude:  828.389321 +/- 20.1807118 (2.44%) (init = 828)
    d2_fraction:   1.3008e-09 +/- 0.15881478 (12209429410.50%) (init = 0.5)
    d2_sigma:      0.12806451 +/- 0.00332776 (2.60%) (init = 0.1)
    d2_center:    -4.12396256 +/- 0.00295449 (0.07%) (init = -4.12)
    d2_amplitude:  332.015501 +/- 25.5157988 (7.69%) (init = 332)
    a_fwhm:        0.09444644 +/- 68.4833226 (72510.22%) == '2.0000000*a_sigma'
    a_height:      1327204.05 +/- 1.5235e+12 (114790578.80%) == '(((1-a_fraction)*a_amplitude)/
    d1_fwhm:       0.25058961 +/- 0.00300684 (1.20%) == '2.0000000*d1_sigma'
    d1_height:     2897.76949 +/- 29.6035081 (1.02%) == '(((1-d1_fraction)*d1_amplitude)/(d1_si
    d2_fwhm:       0.25612903 +/- 0.00665552 (2.60%) == '2.0000000*d2_sigma'
    d2_height:     1217.77582 +/- 1031.04338 (84.67%) == '(((1-d2_fraction)*d2_amplitude)/(d2_s
[[Correlations]] (unreported correlations are < 0.100)
    C(a_fraction, a_amplitude)    = -1.000
    C(d2_fraction, d2_amplitude)  = -0.948
    C(d1_fraction, d1_amplitude)  =  0.914
    C(bg_c0, a_sigma)             = -0.843
    C(a_sigma, d2_amplitude)      = -0.715
    C(a_sigma, d2_fraction)       =  0.640
    C(a_amplitude, d2_center)     =  0.548
    C(a_fraction, d2_center)      = -0.548
    C(bg_c0, d2_amplitude)        =  0.468
```

```
C(d1_fraction, d1_sigma)      = -0.432
C(bg_c0, d2_fraction)         = -0.424
C(d2_fraction, d2_sigma)      =  0.332
C(d1_amplitude, d2_amplitude) =  0.323
C(d1_amplitude, d2_fraction)  = -0.267
C(d1_fraction, d2_amplitude)  =  0.253
C(bg_c0, d1_amplitude)        = -0.215
C(d1_fraction, d2_fraction)   = -0.204
C(bg_c0, d1_fraction)         = -0.204
C(a_sigma, d1_amplitude)      = -0.194
C(d1_sigma, d1_amplitude)     = -0.158
C(a_sigma, d1_fraction)       = -0.153
C(d2_sigma, d2_amplitude)     = -0.119
```



```
In [16]: # Extracting the fitted parameters

         labview_params = [('numberAtom', 'mean'), ('tempXAtom', 'mean'), ('tempYAtom', 'mean'

         print('Xmin [MHz]:\t' + str(min(xdata)))
         print('Xmax [MHz]:\t' + str(max(xdata)))
         for p in labview_params:
             print(p[0] + '(mean):\t' + str(np.mean(data[(xmin <= data['detuning']) & (data['de
             print(p[0] + '(std):\t' + str(np.std(data[(xmin <= data['detuning']) & (data['detu

         # Saving to variables
         d1_pol_int = ufloat(fit_result.params['d1_amplitude'].value, fit_result.params['d1_amp
         d1_pol_N = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)
                           np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)
         d1_pol_Ty = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xma
                            np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax

         d2_pol_int = ufloat(fit_result.params['d2_amplitude'].value, fit_result.params['d2_amp
         d2_pol_N = d1_pol_N
         d2_pol_Ty = d1_pol_Ty
```

```
Xmin [MHz]:        -6.936779999999999
Xmax [MHz]:        -2.7367800000001807
numberAtom(mean):        215163.27733850133
numberAtom(std):        4754.8353260496015
tempXAtom(mean):        8.684678762532301e-07
tempXAtom(std):        4.297956220106409e-08
tempYAtom(mean):        7.685880064082688e-07
tempYAtom(std):        1.005433720912225e-08
UV_Power(mean):        0.8151492175513767
UV_Power(std):        0.002686155341955794
Spec_Power(mean):        0.42689396348110453
Spec_Power(std):        0.0005859764145383775
```

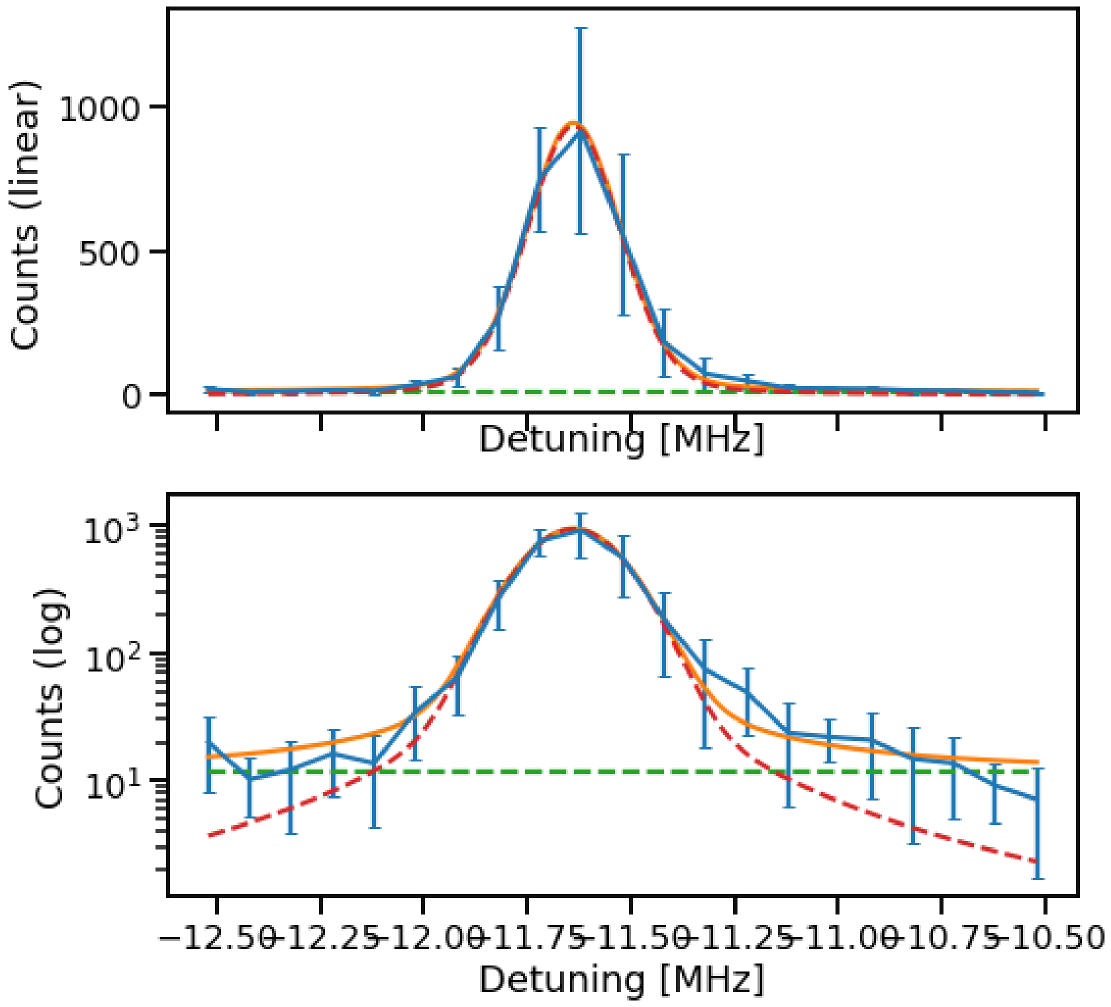
3.1.2 n=37, unpolarized sample

n=37, unpolarized, dimer, ground state

```
In [17]: data = unpol_data
         plot_components = True
         plot_initial_guess = False

         # Choosing a range to fit over
         xmin = -14
         xmax = -10.5

         xdata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['detuning']
         ydata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegral_ad
```

```python
ydata_unc = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegra

# Specify fit model
bg = PolynomialModel(degree=0, prefix='bg_')
d0 = PseudoVoigtModel(prefix='d0_')

fit_model = bg + d0
fit_params = fit_model.make_params()

fit_params['bg_c0'].set(min(ydata), vary=True)
#fit_params['bg_c1'].set(0, vary=True)
#fit_params['bg_c2'].set(0, vary=True)

fit_params['d0_center'].set(-11.73, vary=True)
fit_params['d0_amplitude'].set(276.46, min=0, vary=True)
fit_params['d0_sigma'].set(0.1, min=0, vary=True)

# Initial guess
xfit = np.linspace(min(xdata), max(xdata), num=1000, endpoint=True)
yinit = fit_model.eval(fit_params, x=xfit)

# Fitting
fit_result = fit_model.fit(ydata, fit_params, x=xdata)
dely = fit_result.eval_uncertainty(x=xdata)
yfit = fit_model.eval(fit_result.params, x=xfit)

print(fit_result.fit_report())

# Plotting output
[fig, axs] = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=False, figsize=(8,8))

axs[0].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[0].plot(xfit, yfit, label='fit')
axs[0].set_xlabel('Detuning [MHz]')
axs[0].set_ylabel('Counts (linear)')

axs[1].set_yscale('log')
axs[1].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[1].plot(xfit, yfit, label='fit')
axs[1].set_xlabel('Detuning [MHz]')
axs[1].set_ylabel('Counts (log)')

if plot_components:
    # generate components
    fit_comps = fit_result.eval_components(x=xfit)

    for key in fit_comps.keys():
        for ax in axs:
            ax.plot(xfit, fit_comps[key], '--')

if plot_initial_guess:
    for ax in axs:
        ax.plot(xfit, yinit, '-.', label='init')

plt.show()
```

```
[[Model]]
    (Model(polynomial, prefix='bg_') + Model(pvoigt, prefix='d0_'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 43
    # data points      = 21
    # variables        = 5
    chi-square         = 2736.99952
    reduced chi-square = 171.062470
    Akaike info crit   = 112.271997
    Bayesian info crit = 117.494609
[[Variables]]
    bg_c0:         11.6266671 +/- 5.31156360 (45.68%) (init = 7.137951)
    d0_fraction:   0.23780292 +/- 0.10221696 (42.98%) (init = 0.5)
    d0_sigma:      0.13433107 +/- 0.00234649 (1.75%) (init = 0.1)
    d0_center:    -11.6384107 +/- 0.00163487 (0.01%) (init = -11.73)
    d0_amplitude:  289.665430 +/- 11.8381280 (4.09%) (init = 276.46)
    d0_fwhm:       0.26866214 +/- 0.00469298 (1.75%) == '2.0000000*d0_sigma'
    d0_height:     935.239462 +/- 14.6807283 (1.57%) == '(((1-d0_fraction)*d0_amplitude)/(d0_s
[[Correlations]] (unreported correlations are < 0.100)
    C(d0_fraction, d0_amplitude) =  0.930
    C(bg_c0, d0_amplitude)       = -0.840
    C(bg_c0, d0_fraction)        = -0.754
    C(d0_fraction, d0_sigma)     = -0.407
    C(d0_sigma, d0_amplitude)    = -0.152
```



```
In [18]: # Extracting the fitted parameters

         labview_params = [('numberAtom', 'mean'), ('tempXAtom', 'mean'), ('tempYAtom', 'mean')

         print('Xmin [MHz]:\t' + str(min(xdata)))
         print('Xmax [MHz]:\t' + str(max(xdata)))
         for p in labview_params:
             print(p[0] + '(mean):\t' + str(np.mean(data[(xmin <= data['detuning']) & (data['de
             print(p[0] + '(std):\t' + str(np.std(data[(xmin <= data['detuning']) & (data['detu

         # Saving to variables
         d0_unpol_int = ufloat(fit_result.params['d0_amplitude'].value, fit_result.params['d0_a
         d0_unpol_N = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xma
                             np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xma
         d0_unpol_Ty = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xm
                              np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xma
Xmin [MHz]:        -12.519999999999982
Xmax [MHz]:        -10.519999999999982
numberAtom(mean):        220029.32671957675
numberAtom(std):        4126.165465996539
tempXAtom(mean):        8.011494470370371e-07
tempXAtom(std):        1.1586069757952399e-08
tempYAtom(mean):        7.597133553968256e-07
tempYAtom(std):        7.3766579436708695e-09
UV_Power(mean):        0.8162416262349124
UV_Power(std):        0.0015926980814092865
Spec_Power(mean):        0.4300394403366816
Spec_Power(std):        0.0009235036170427374
```

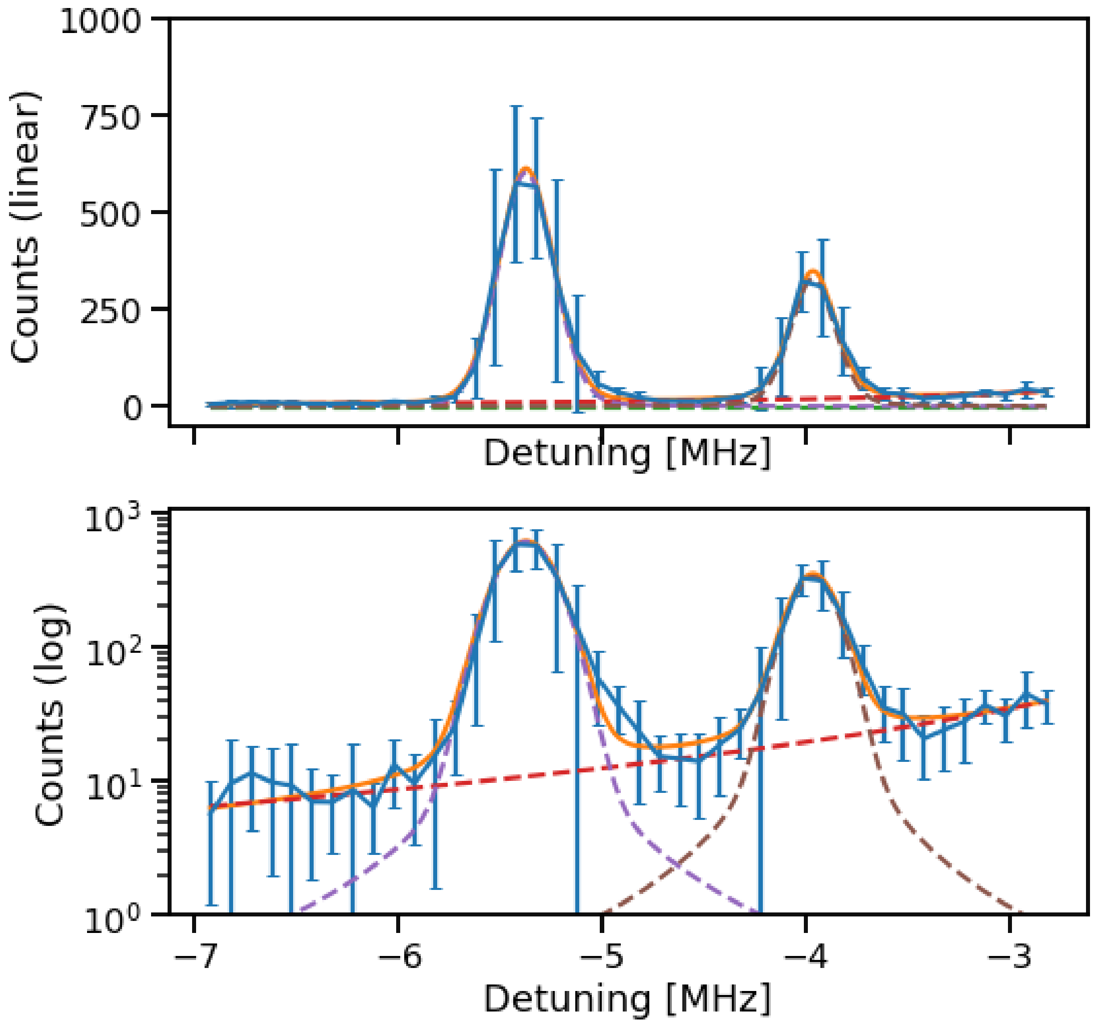
n=37, unpolarized, dimer, 1st and 2nd excited states

```
In [19]: data = unpol_data
         plot_components = True
         plot_initial_guess = False

         # Choosing a range to fit over
         xmin = -7
         xmax = -2.75

         xdata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['detuning']
         ydata = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegral_ad
         ydata_unc = data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)]['sfiIntegra

         # Specify fit model
         bg = PolynomialModel(degree=0, prefix='bg_')
         a = PseudoVoigtModel(prefix='a_')
         d1 = PseudoVoigtModel(prefix='d1_')
         d2 = PseudoVoigtModel(prefix='d2_')
         #d3 = PseudoVoigtModel(prefix='d3_')

         #fit_model = bg + a + d1 + d2 + d3
         fit_model = bg + a + d1 + d2
         fit_params = fit_model.make_params()

         fit_params['bg_c0'].set(min(ydata), vary=True)
         #fit_params['bg_c1'].set(0, vary=True)
         #fit_params['bg_c2'].set(0, vary=True)

         fit_params['a_center'].set(0, vary=False)
         fit_params['a_fraction'].set(.65, vary=True)
         fit_params['a_amplitude'].set(30000, vary=True)
         fit_params['a_sigma'].set(0.1, min=0, vary=True)
```

```
fit_params['d1_center'].set(-5.37, min=-6, max=-5, vary=True)
fit_params['d1_amplitude'].set(400, min=0, vary=True)
fit_params['d1_sigma'].set(0.1, min=0, vary=True)

fit_params['d2_center'].set(-3.98, min=-4.5, max=-3.75, vary=True)
fit_params['d2_amplitude'].set(100, min=0, vary=True)
fit_params['d2_sigma'].set(0.1, min=0, vary=True)

#fit_params['d3_center'].set(-2.5, min=-3, max=-1.5, vary=True)
#fit_params['d3_amplitude'].set(10, min=0, vary=True)
#fit_params['d3_sigma'].set(0.1, min=0, vary=True)
#fit_params['d3_fraction'].set(0.1, vary=True)

# Initial guess
xfit = np.linspace(min(xdata), max(xdata), num=1000, endpoint=True)
yinit = fit_model.eval(fit_params, x=xfit)

# Fitting
fit_result = fit_model.fit(ydata, fit_params, x=xdata)
dely = fit_result.eval_uncertainty(x=xdata)
yfit = fit_model.eval(fit_result.params, x=xfit)

print(fit_result.fit_report())

# Plotting output
[fig, axs] = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=False, figsize=(8,8))

axs[0].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[0].plot(xfit, yfit, label='fit')
axs[0].set_xlabel('Detuning [MHz]')
axs[0].set_ylabel('Counts (linear)')
axs[0].set_ylim(top=1000)

axs[1].set_yscale('log')
axs[1].errorbar(xdata, ydata, yerr=ydata_unc, capsize=3, label='data')
axs[1].plot(xfit, yfit, label='fit')
axs[1].set_xlabel('Detuning [MHz]')
axs[1].set_ylabel('Counts (log)')
axs[1].set_ylim(bottom=1)

if plot_components:
    # generate components
    fit_comps = fit_result.eval_components(x=xfit)

    for key in fit_comps.keys():
        for ax in axs:
            ax.plot(xfit, fit_comps[key], '--')
        if plot_initial_guess:
            for ax in axs:
                ax.plot(xfit, yinit, '-.', label='init')

        plt.show()
```

```
[[Model]]
    (((Model(polynomial, prefix='bg_') + Model(pvoigt, prefix='a_')) + Model(pvoigt, prefix='d
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 126
    # data points      = 42
    # variables        = 12
    chi-square         = 3500.31739
    reduced chi-square = 116.677246
    Akaike info crit   = 209.763451
    Bayesian info crit = 230.615486
[[Variables]]
    bg_c0:        -0.91314619 +/- 11.3970767 (1248.11%) (init = 5.696249)
    a_fraction:    0.97390718 +/- 1138041.22 (116853149.30%) (init = 0.65)
    a_sigma:       0.01142227 +/- 526.828101 (4612288.48%) (init = 0.1)
    a_center:      0 (fixed)
    a_amplitude:   87805.5698 +/- 1.0280e+11 (117075101.33%) (init = 30000)
    d1_fraction:   0.12438223 +/- 0.14306706 (115.02%) (init = 0.5)
    d1_sigma:      0.15995076 +/- 0.00490647 (3.07%) (init = 0.1)
    d1_center:    -5.37190225 +/- 0.00254711 (0.05%) (init = -5.37)
    d1_amplitude:  214.462985 +/- 12.9122104 (6.02%) (init = 400)
    d2_fraction:   0.26200485 +/- 0.32332992 (123.41%) (init = 0.5)
    d2_sigma:      0.13070444 +/- 0.00648991 (4.97%) (init = 0.1)
    d2_center:    -3.96417878 +/- 0.00389417 (0.10%) (init = -3.98)
    d2_amplitude:  100.264377 +/- 14.1062234 (14.07%) (init = 100)
    a_fwhm:        0.02284454 +/- 1053.65621 (4612288.54%) == '2.0000000*a_sigma'
    a_height:      2477289.51 +/- 4.2235e+12 (170490670.01%) == '(((1-a_fraction)*a_amplitude)
    d1_fwhm:       0.31990152 +/- 0.00981299 (3.07%) == '2.0000000*d1_sigma'
    d1_height:     604.550896 +/- 24.8764300 (4.11%) == '(((1-d1_fraction)*d1_amplitude)/(d1_s
    d2_fwhm:       0.26140888 +/- 0.01297983 (4.97%) == '2.0000000*d2_sigma'
    d2_height:     329.893740 +/- 18.9019508 (5.73%) == '(((1-d2_fraction)*d2_amplitude)/(d2_s
[[Correlations]] (unreported correlations are < 0.100)
    C(a_fraction, a_amplitude)    = -0.999
    C(d2_fraction, d2_amplitude)  =  0.956
    C(d1_fraction, d1_amplitude)  =  0.938
    C(bg_c0, a_sigma)             = -0.883
    C(a_sigma, d2_amplitude)      = -0.813
    C(a_fraction, d1_sigma)       =  0.757
    C(a_amplitude, d1_sigma)      = -0.756
    C(a_sigma, d2_fraction)       = -0.739
    C(bg_c0, d2_amplitude)        =  0.674
    C(bg_c0, d2_fraction)         =  0.617
```

```
C(d1_fraction, d1_sigma)      = -0.526
C(a_amplitude, d1_amplitude)  =  0.521
C(a_fraction, d1_amplitude)   = -0.506
C(d1_sigma, d1_amplitude)     = -0.486
C(a_fraction, d1_center)      =  0.466
C(a_amplitude, d1_center)     = -0.462
C(a_fraction, d2_sigma)       = -0.442
C(a_amplitude, d2_sigma)      =  0.440
C(d2_fraction, d2_sigma)      = -0.420
C(a_sigma, d1_amplitude)      = -0.375
C(a_amplitude, d1_fraction)   =  0.374
C(a_fraction, d1_fraction)    = -0.360
C(d1_sigma, d1_center)        =  0.353
C(d1_sigma, d2_sigma)         = -0.345
C(a_sigma, d1_fraction)       = -0.339
C(d1_amplitude, d2_amplitude) =  0.318
C(d1_fraction, d2_amplitude)  =  0.294
C(d1_amplitude, d2_fraction)  =  0.282
C(a_fraction, d2_center)      =  0.254
C(d1_fraction, d2_fraction)   =  0.254
C(a_amplitude, d2_center)     = -0.253
C(d2_sigma, d2_amplitude)     = -0.246
C(d1_amplitude, d2_sigma)     =  0.232
C(d1_center, d1_amplitude)    = -0.215
C(d1_center, d2_sigma)        = -0.202
C(a_fraction, d2_amplitude)   =  0.191
C(d1_sigma, d2_center)        =  0.186
C(d1_fraction, d2_sigma)      =  0.176
C(d1_sigma, d2_amplitude)     =  0.159
C(a_amplitude, d2_amplitude)  = -0.159
C(a_fraction, d2_fraction)    =  0.156
C(bg_c0, d1_center)           =  0.152
C(d1_fraction, d1_center)     = -0.150
C(d1_center, d2_amplitude)    =  0.147
C(bg_c0, a_fraction)          =  0.140
C(d1_sigma, d2_fraction)      =  0.136
C(bg_c0, d1_sigma)            =  0.135
C(a_amplitude, d2_fraction)   = -0.127
C(d1_center, d2_center)       =  0.122
C(d1_center, d2_fraction)     =  0.122
C(d1_amplitude, d2_center)    = -0.119
C(bg_c0, d2_sigma)            = -0.111
C(d2_sigma, d2_center)        = -0.107
C(bg_c0, a_amplitude)         = -0.105
C(a_sigma, d1_center)         = -0.105
```



```
In [20]: # Extracting the fitted parameters

         labview_params = [('numberAtom', 'mean'), ('tempXAtom', 'mean'), ('tempYAtom', 'mean'

         print('Xmin [MHz]:\t' + str(min(xdata)))
         print('Xmax [MHz]:\t' + str(max(xdata)))
         for p in labview_params:
             print(p[0] + '(mean):\t' + str(np.mean(data[(xmin <= data['detuning']) & (data['de
             print(p[0] + '(std):\t' + str(np.std(data[(xmin <= data['detuning']) & (data['detu

         # Saving to variables
         d1_unpol_int = ufloat(fit_result.params['d1_amplitude'].value, fit_result.params['d1_a
         d1_unpol_N = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xma
                           np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax)
         d1_unpol_Ty = ufloat(np.mean(data[(xmin <= data['detuning']) & (data['detuning'] <= xm
                            np.std(data[(xmin <= data['detuning']) & (data['detuning'] <= xmax

         d2_unpol_int = ufloat(fit_result.params['d2_amplitude'].value, fit_result.params['d2_
         d2_unpol_N = d1_pol_N
         d2_unpol_Ty = d1_pol_Ty
```

```
Xmin [MHz]:         -6.920000000000073
Xmax [MHz]:         -2.8200000000001637
numberAtom(mean):         214390.20608465612
numberAtom(std):         5092.455246429882
tempXAtom(mean):         8.110325584656083e-07
tempXAtom(std):         1.291357899183854e-08
tempYAtom(mean):         7.663323687830689e-07
tempYAtom(std):         9.178686356571183e-09
UV_Power(mean):         0.8180317389271247
UV_Power(std):         0.0023161780393332588
Spec_Power(mean):         0.4273456714771412
Spec_Power(std):         0.0004863364319521899
```

4 Some uncertainty calculations

```
In [21]: d0_pol_int_corr = d0_pol_int/(d0_pol_N**2/d0_pol_Ty**(3/2))
         d1_pol_int_corr = d1_pol_int/(d1_pol_N**2/d1_pol_Ty**(3/2))
         d2_pol_int_corr = d2_pol_int/(d2_pol_N**2/d2_pol_Ty**(3/2))

         d0_unpol_int_corr = d0_unpol_int/(d0_unpol_N**2/d0_unpol_Ty**(3/2))
         d1_unpol_int_corr = d1_unpol_int/(d1_unpol_N**2/d1_unpol_Ty**(3/2))
         d2_unpol_int_corr = d2_unpol_int/(d2_unpol_N**2/d2_unpol_Ty**(3/2))
```

```
In [26]: d1_pol_int_corr/d0_pol_int_corr
         d2_pol_int_corr/d0_pol_int_corr

         d1_unpol_int_corr/d0_unpol_int_corr
         d2_unpol_int_corr/d0_unpol_int_corr
```

```
Out[26]: 0.6610835728241667+/-0.04953260990579538
```