DataAnalysis_GammaSpectroscopy

November 4, 2017

1 Data analysis - Introduction for Gamma Spectroscopy

1.1 Table of Content

1.2 About this Notebook

The purpose of this *jupyter* notebook is to introduce data analysis in the frame of gamma spectroscopy. The example programming language is *Python3*, but of course most coding languages can do the job properly. If you have never programmed before there are so many great tutorials available across the web. There even exist plenty *Open Online Courses*, e.g. https://www.coursera.org/learn/python. Please have a look around for the one that you like the best. However, note that you do not need to be an expert in Python to pass the lab.

The data analysis can roughly be divided into four steps: 1. Read experimental data from file. 2. Fit Gaussians to peaks. 3. Calibrate the detector response. 4. Perform a statistical analysis (e.g. error propagation) and present results.

A dedicated python library, i.e. a folder with already written code, located in `HelpCode`, have been implemented for the data analysis connected to the labs in FYSC12 Nuclear Physics. The folder comprises functions that support 1-3 of the above-mentioned steps.

Full Python3 coding examples of how to perform the different steps of the data analysis is given below. Every example is finished with a template of how the `HelpCode`-folder can be used to perform the same calculations.

NOTE: It is strongly recommended that you program your own functions instead of using the framework directly out of the book. As you will find out, there will come a point where the

framework functionalities are not to your satisfaction, and then you need to code yourself. So, better get used to it right away :)

1.3 Read experimental data from file

The following code segment exemplifies how to read in an experimental data file into a list container. For an introduction on how to read and write files see e.g. http://www.pythonforbeginners.com/files/reading-and-writing-files-in- python.

```
In [1]: array_of_data = list()
        read_lines = 0
        total_data = 0
        start_read = False
        with open("test_data.Spe") as file:
            for j, line in enumerate(file):
                #if j < 100:
                    #print(line.split())
                if line.split()[0] == '$DATA:':
                    #print(line.split())
                    start_read = True
                elif start_read and read_lines == 0:
                    #print(line.split())
                    total_data = int(line.split()[1])+1
                    #print(total_data)
                    read_lines = read_lines + 1
                    continue
                elif start_read and line.split()[0] != '$ROI:':
                    #print(line.split())
                    array_of_data.append(int(line.split()[0]))
                elif start_read:
                    break

        print("Should read:", total_data, "lines. And read:", len(array_of_data))
```

```
Should read: 8192 lines. And read: 8192
```

1.3.1 HelpCode

With the `help_code` it is possible to perform conceptually the same operations through:

```
In [2]: from HelpCode.MCA import *
        #data = load_spectrum("test_data.Spe")
```

1.3.2 Read the calibrated background spectrum

The background spectrum that is to be analysed as a part of the lab is named `background_analysis.csv` and can be found in the current folder. This spectrum has been mea-

sured with another detector and is already calibrated. Read this spectrum with the `help_code` with:

```
In [3]: background_data = load_calibrated_spectrum("Background.txt")
```

1.3.3 Plotting the data

It is always good to visualise your data. This is how you can plot and visualise it in Python3.

```
In [4]: import matplotlib
        # choose a backend for web applications; remove for stand-alone applications:
        matplotlib.use('Agg')
        # enable interactive notebook plots (
        # alternative: use 'inline' instead of 'notebook' for static images)
        %matplotlib notebook

        #The following line is the ONLY one needed in stand-alone applications!
        import matplotlib.pyplot as plt

In [5]: plt.figure()
        # with the data read in with the first routine
        plt.plot(array_of_data)
        # or with the help_code variable "data"
        #plt.plot(background_data.x, background_data.y)

        plt.savefig("test_spectrum.png") #This is how you save the figure
        #axis = plt.gca()
        #axis.plot(array_of_data)

        ## Could be useful to see this in log scale..?
        #plt.yscale('log')
        #plt.ylim(ymin=1)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.4 Fit of data

Fitting functions can be made simply with the `scipy.optimize` module. The function `curve_fit` does the job for you and the documentation contains all the valuable information on how to use the function. It uses a method called least squares which you can read about in most course literature on statistics and for instance on Wolfram Alpha.

1.4.1 Fitting Gaussian

The following code shows how to use the function `curve_fit` to fit a peak in the data that was read in above (i.e. you will need to execute the above code section before this section will work).

```
In [6]: from scipy.optimize import curve_fit

        def GaussFunc(x, A, mu, sigma):
            return A*np.exp(-(x-mu)**2/(2.*sigma**2))

        #mu = np.asarray([3300, 3750])
        #E = np.asarray([1173.2, 1332.5])
        mu = 3300
        A = array_of_data[mu]
        sigma = 1
        guess = [A, mu, sigma]
        n = 50 #number of points on each side to include in fit

        x = np.asarray(range(len(array_of_data)))
        y = np.asarray(array_of_data)

        estimates, covar_matrix = curve_fit(GaussFunc,
                                            x[mu-n:mu+n],
                                            y[mu-n:mu+n],
                                            p0=guess)

        print("Estimates of (A mu sigma) = (", estimates[0], estimates[1], estimates[2], ")\n")

        print("Covariance matrix = \n", covar_matrix, "\n")

        print("Uncertainties in the estimated parameters: \n[ sigma^2(A) sigma^2(mu), sigma^2(si

        #plt.figure()
        #plt.plot(x[mu-n:mu+n],y[mu-n:mu+n], linestyle="", marker="*")
        #plt.plot(x[mu-n:mu+n], GaussFunc(x[mu-n:mu+n], estimates[0], estimates[1], estimates[2]
```

```
Estimates of (A mu sigma) = ( 1701.43796992 3300.1739634 2.69078263554 )

Covariance matrix =
 [[  2.22927047e+02   4.29594398e-06  -2.35037636e-01]
 [  4.29594398e-06   7.43418433e-04  -6.79408923e-09]
 [ -2.35037636e-01  -6.79408923e-09   7.43418426e-04]]

Uncertainties in the estimated parameters:
[ sigma^2(A) sigma^2(mu), sigma^2(sigma) ] =
[ 222.927046993 0.000743418433343 0.00074341842635 ]


<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x7f3c791ab278>]
```

1.4.2 HelpCode

With the `HelpCode`:

```
In [7]: from HelpCode.fithelpers import *

        gauss = fit_gaussian_at_idx(x, y, mu, npoints=n)
        print("Estimates of (A mu sigma) = (", gauss.A, gauss.mu, gauss.sigma, ")\n")

        print("Covariance matrix = \n", gauss.covar_matrix, "\n")

        print("Uncertainties in the estimated parameters: \n[ sigma^2(A) sigma^2(mu), sigma^2(si

        #plt.figure()
        #plt.plot(data.x[mu-n:mu+n], data.y[mu-n:mu+n], linestyle="", marker="*")
        #plt.plot(data.x[mu-n:mu+n], gauss.value(data.x)[mu-n:mu+n])
```

```
Estimates of (A mu sigma) = ( 1701.43796992 3300.1739634 2.69078263554 )

Covariance matrix =
 [[  2.22927047e+02   4.29594398e-06  -2.35037636e-01]
 [  4.29594398e-06   7.43418433e-04  -6.79408923e-09]
 [ -2.35037636e-01  -6.79408923e-09   7.43418426e-04]]

Uncertainties in the estimated parameters:
[ sigma^2(A) sigma^2(mu), sigma^2(sigma) ] =
[ 222.927046993 0.000743418433343 0.00074341842635 ]


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-7-ad674a0ab41c> in <module>()
          9
```

```
     10 plt.figure()
---> 11 plt.plot(data.x[mu-n:mu+n], data.y[mu-n:mu+n], linestyle="", marker="*")
     12 plt.plot(data.x[mu-n:mu+n], gauss.value(data.x)[mu-n:mu+n])


NameError: name 'data' is not defined
```

1.4.3 Fit a line

```
In [ ]: x = np.asarray([1,3,5,7])
        y = np.asarray([1.3, 2.1, 2.9, 4.2])
        #If you are more or less uncertain about your y-values this can be used in the fit by in
        sigmay = np.asarray([0.5, 0.3, 0.1, 0.2])

        guess = [2, 1]

        def LineFunc(x, k, m):
            return k*x+m

        estimates, covar_matrix = curve_fit(LineFunc,
                                            x,
                                            y,
                                            p0 = guess,
                                            sigma = sigmay)

        print("Estimates of (k m) = (", estimates[0], estimates[1], ")\n")

        #plt.figure()
        #plt.plot(x,y, linestyle="", marker="*")
        #plt.plot(x, LineFunc(x, estimates[0], estimates[1]))
```

1.5 Statistical analysis

Background theory and instructions on how to perform statistical analysis on experimental data, with error propagation, can be found in the document `error_analysis.pdf`), but of course also easily through a google search.