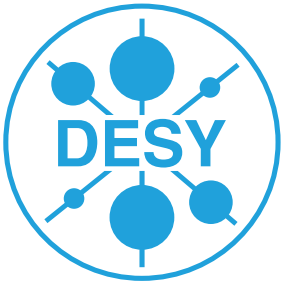
DESY Summer Student Program 2011



Python software for evaluation of silicon drift detector spectra

Maria Naumova, Chemistry department of Lomonosov Moscow State University

September 10, 2011

Contents

1 Introduction

A large variety of methods used at synchrotron radiation facilities are based on spectroscopic analysis of scattered radiation. The technical implementation requires either analyser cryslalls or energy resolving detectors. Recently a new generation of solid state energy resolving detectors, the so called silicon drift detectors (SDD) has become available for general use. For registeration of fluorescence spectra solid-state detectors with energy resolution of the input signal (intensity as function of photon energy) are used.

The Silicon Drift Detector (SDD) is derived from the principle of sideward depletion. A volume of high-resistivity n-type silicon is fully depleted by reverse biased p+ junctions covering both surfaces. Electrons generated within the depleted volume by thermal processes or by the absorption of ionizing radiation are forced to drift to a small-sized n+ anode by an electric field with a strong component parallel to the wafer surface. In the original SDD design the electric field is achieved by segmentation of the p+ regions on both surfaces to patterns of parallel strips and superposition of a voltage gradient. The direction of the voltage gradient is such that the n+ readout anode has the most positive potential and collects all electrons released within the depleted volume by the absorption of ionizing radiation or thermal generation [1].

Silicon Drift Detectors (SDDs) combine a large sensitive area with a small value of the output capacitance and are therefore well suited for high resolution, high count rate X-ray spectroscopy. The low leakage current level obtained by the elaborated processing technology makes it possible to operate them at room temperature or with moderate cooling [2]. The detectors function in single photon counting mode: every incoming photon is transformed into an electric pulse with an amplitude proportional to the energy of the incoming photon. Usually the pulses of incoming photons are collected over some time interval and saved in a histogramic memory. The resulting spectra are smooth curves with multiple Gaussian-like maxima. For quantitative evaluation of spectroscopic data one requires software capable of fitting this function to experimental data.

For analysis of the resulting spectra the software used should take into consideration form of maximum output of the detector. Peculiarity about SDD detectors is that peaks can't be completely approximated by Gaussians. Another function was proposed [3]. It is a sum of Gaussian and step-function, and it contains 7 parameters.

The problem is that nowadays their is no available software for approximation of such function.

Moreover, it is so far not clear how to solve this problem automatically (because spectrum with 10 peaks means variation of 70 parameters if we don't use simplifications).

The aim of this work was to create a software module in Python language for fitting of SDD-data obtained in synchrotron radiation experiments from substances with complex

chemical composition. The final goal of the project is to expand the widely used PyMca software package and make the quantitative SDD data analysis available for general public.
In more detail my task was:

1. to get acquainted with Linux
2. to learn principles of programming on Python (final step - approximation of functions)
3. find out dependance of parameters on the form of step-function
4. write a program that will approximate fluorescence spectra for SDDs

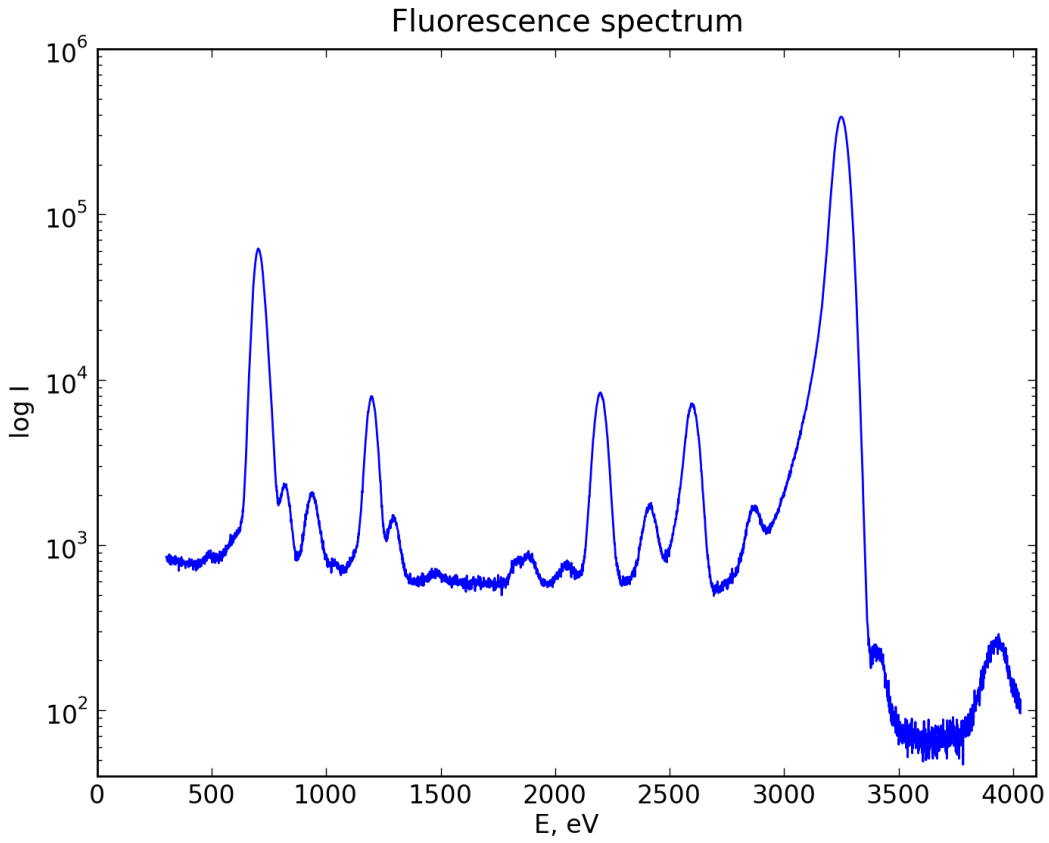


Figure 1: Multielement fluorescence spectrum

2 SDD spectra

For this work we chose a typical fluorescence spectrum of a multielement sample 1

It is known that sample includes a substrate composed of Si and W with possible trace impurities and a salt solution in water. Preliminary chemical composition analysis with PyMca shows presence of Fe, S, P, Cr, I, Ca, K. There is also a clear Ar line coming from the air.

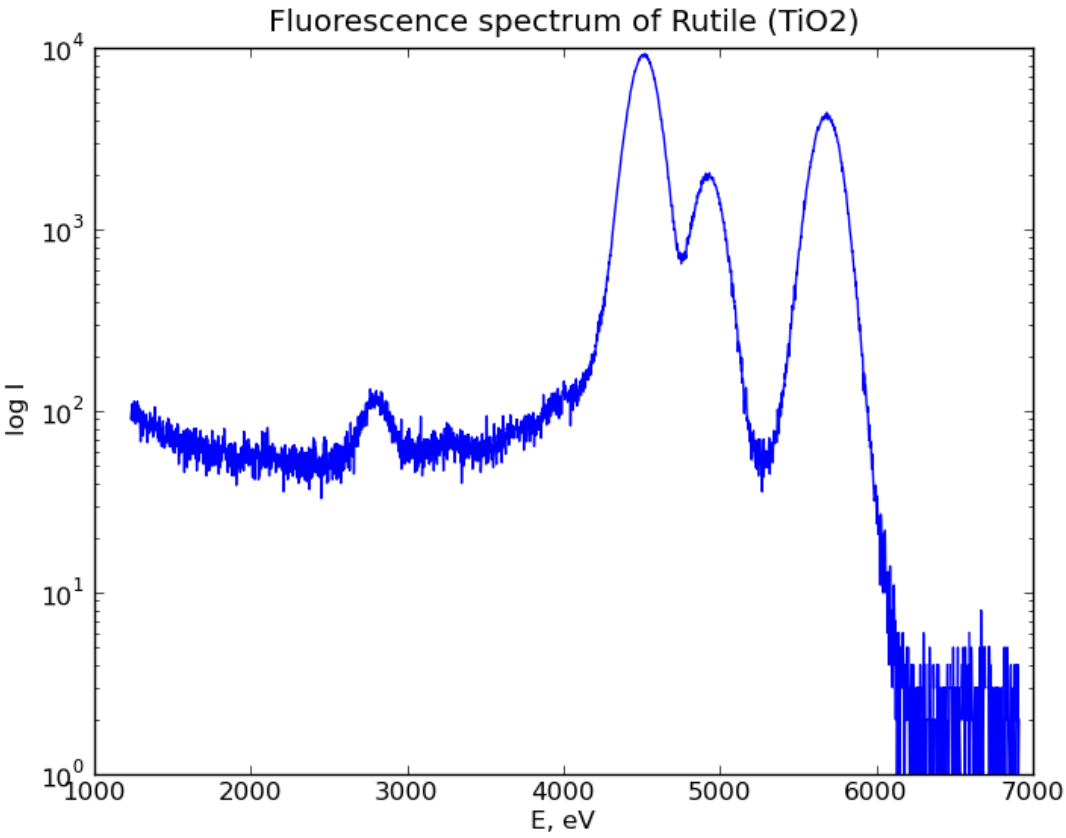


Figure 2: Fluorescence spectrum of rutile

It is important to mention that K series fluorescence lines are typically detected as two maxima (the energy resolution of the detector is not sufficient to separate $K\alpha_1$ and $K\alpha_2$ lines). The L-lines, in contrast, can have several detected peaks with interdependent intensities. PyMca is useful because if you choose an element, PyMca automatically finds all fluorescence peaks of this element, including escape peaks caused by detector material.
For test purposes I was using experimental curves from a more simple DAFS (diffraction

anomalous fine structure) experiment carried out on rutile (TiO2) sample 2.

Along with Ti $K\alpha$, $K\beta$ fluorescence lines the spectrum contains elastic scattering maximum coming from incident radiation and Ar K lines from the air.

3 Python software module

To write my program in Python I had to import pylab module with numerical functions and corresponding environment (contains core parts of numpy, scipy, and matplotlib):

- scipy - in particular, I had to import additionally 'special' module (for error-function) and 'optimize' module (for least square method).
- numpy - for work with arrays of data
- matplotlib - for making plots

The function we are going to use for peak approximation [3]:

$A(g) \cdot \exp\left(-\frac{(x-E)^2}{2\cdot\sigma^2}\right) + A(s) \cdot \mathrm{erf}\left(\frac{(x-X)}{B}\right) \cdot \exp\left(-\frac{b}{B}x\right)$

- amplitude of Gaussian $A(g)$,
- energy of fluorescence line E,
- standard deviation σ,
- parameters of exponent and error-function in step-function b, B,
- shift in error-function X,
- amplitude of the step-function $A(s)$.

3.1 Program description and usage

The program consists of the following parts:

1. Creating array of data
   a) Reading data from file (our file contains only intensities, we read data to 1D array).

x	array containing channel numbers (energy after recalibration); horisontal axes
y	array containing intensity data; vertical axes
time	horizontal axes coordinate (as well as x). used for making fit plots
fitfunc	a "template" for Gaussian function: $A(g) \cdot \exp\left(-\frac{(x-E)^2}{W}\right)$, amplitude A, energy shift E, $W = 2\sigma^2$, σ - standard deviation
fitfuncs	sum of Gaussian functions with the same parameters we use for the first part of approximation **(only Gaussians, without step-function)**
errfunc	error-function for least square method (for Gaussians without step function)
fiterf	a "template" for Gaussian+step-function approximation: $A(g) \cdot \exp\left(-\frac{(x-E)^2}{2\cdot\sigma^2}\right) + A(s) \cdot \mathrm{erf}\left(\frac{(x-X)}{B}\right) \cdot \exp\left(-\frac{b}{B}\right)$
fiterfs	function we use for the second step of approximation **(Gaussians + step-functions)**. We take fitted parameters of Gaussians after first iteration and use the same initial parameters for all step-functions
errfunc_erf	error-function for least square method (Gaussians + step functions)
l	array with the list of energies (coordinates of peaks)
num	number of peaks (length of the array l)
p (p0, p1)	array of initial fitting parameters for least square method. For the first iteration with **only Gaussians**.
pg	intermediate array we use to make a p0 array (for approximation by Gaussians)
c (c0, c1)	array of initial fitting parameters for least square method. For the second iteration with **Gaussians + step-functions**.
ce	intermediate array we use to make a c0 array (for approximation by Gaussians+step-functions)
ind1 (i in cycle)	index of elements in l array
ind2 (j in cycle)	index of elements in p (and c) array
ret, retu	internal cycle variables used for combining functions from "templates"
y_a(y_s)	variable used in least square module for giving an absolute value for errfunc (errfunc_erf) (it can be important with logarithmic coordinates)

Table 1: List of variables used in the program

b) Creating array of Natural numbers (as channel numbers) and rescaling from channels to energy.

2. Approximation by Gaussians

a) Creating array of initial values of the parameters: we take parameters of Gaussian (A - amplitude, W - $width = 2\sigma^2$) and make an array of type: [A_1, W_1, A_2, W_2, ...] with the same initial parameters of Gaussians.

b) Module for least square method: we have template block for Gaussian $A \cdot \exp\left(-\frac{(x-E)^2}{2\cdot\sigma^2}\right)$... and make a sum which contains as many blocks as many peaks we have. Then we carry out an approximation.

3. Approximation by Gaussian + step-function
   a) Now we have parameters of Gaussians and can add step-function.
   b) We put 4 parameters of the step-function [b, B, X, I] at the beginning of initial parameters array (c[0, 1, 2, 3]). $I = \frac{A(step)}{A(Gauss)}$.
   c) Then we stack whole array of Gaussian parameters to [b, B, X, I].
   Finally, our array of initial parameters looks like this: [b(s), B(s), X(s), I(s)] + num*[I(g), W(g)] (num = number of peaks).
   d) Module for least square method: the same procedure as with Gaussians - we take a template (Gaussian + step-function), construct our function (fiterfs) from it and held approximation of this function.
4. We receive the list of fit parameters and plot our data and fit. Results are saved in file 'fit_param.dat'.

Before approximating spectrum with use of step-function we can have a look at PyMca results, which provides fits using pure Gaussians for peak approximation.
I cut a part of spectrum 1 containing Ar peak. The results of PyMca evaluation are shown in Figure 3 :

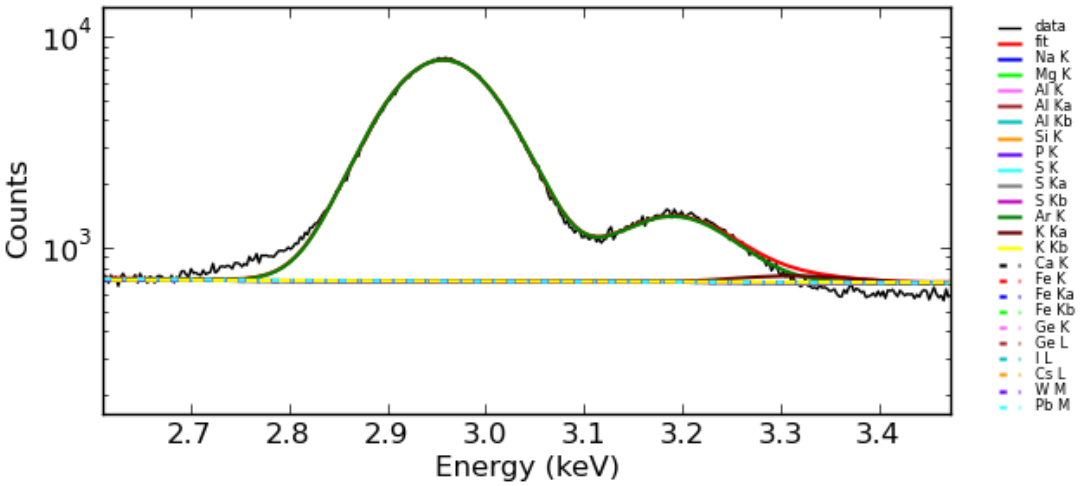


Figure 3: Approximation of Ar fluorescence peaks in PyMca

With PyMca using Gaussian curves, we can clearly see that our peaks are asymmetric: left side has a 'bigger' tail. My aim was exactly to take this 'tail' into account in the

fitting function.

The first experience shows that in most cases one can not correctly fit the data of that type leaving out high-energy component of the spectrum, because the tail of the step function to the left of the Gaussian peak can decrease very slowly and act as a linear background for the maxima on the low energy side. Therefore, in many cases the correct tactics can include a preliminary step, at which the elastic line is fitted alone and the result is subtracted from the data set. This is especially effective if the distance between elastic and fluorescence lines is large. Otherwise, one will need to fit all the maxima simultaneously, taking into account that the parameters for the elastic line can differ from those of fluorescence spectrum.

To make approximation for many peaks (with a lot of parameters) we have to specify these parameters somehow. Thereby, activity progress:

1. Initial idea about restrictions on the parameters:

   - not to take shift of Gaussian E as a free parameter (because it is the energy of fluorescence). We normally know the chemical composition and can take fluorescence energy data from standard tables. In this case correct calibration plays crucial role.
   - to take b, B, X and $I = \frac{A(step)}{A(Gauss)}$ of error function the same for all peaks. (We take X relative to Gaussian coordinate!)
   - with these guesses we have (2*num + 4) parameters: A and W of Gaussians (2*num) and b, B, X, I of step-functions (4).

2. When we have to approximate function with big number of independent parameters, initial values can play a crucial role.
   That's why I decided to divide approximation into 2 steps: the first - approximation by sum of Gaussians, then (with new initial parameters for Gaussians) addition of step-function and final approximation.

3. First, I worked in linear coordinates. In this case program automatically makes the shift value X positive, and the step-function is shifted to the right (instead being shifted to the left as supposed). To solve this problem I take absolute magnitude of the shift value, and the program doesn't have opportunity to make right shift. However, the program makes shift close to zero, and step-function curve still looks shifted to the right. I didn't find a way to get rid of this effect in linear coordinates.

4. When we analyse real spectra we often use semilogarithmic coordinates (as small peaks can be of big interest). Moreover, amplitude of step-function is significantly lower than amplitude of Gaussian, and for having adequate approximation we need logarithmic coordinates.

I found out that use of logarithmic coordinates for both steps of approximation can lead to wrong results (see appendix for explanation).

5. That's why we decided to use linear coordinates on the first step of approximation and logarithmic coordinates for the second step of approximation.

6. This algorithm worked for the simple spectrum (rutile) if we consider 2 peaks (K_α and K_β of Ti) 4. In this fit, step-functions really look like steps.

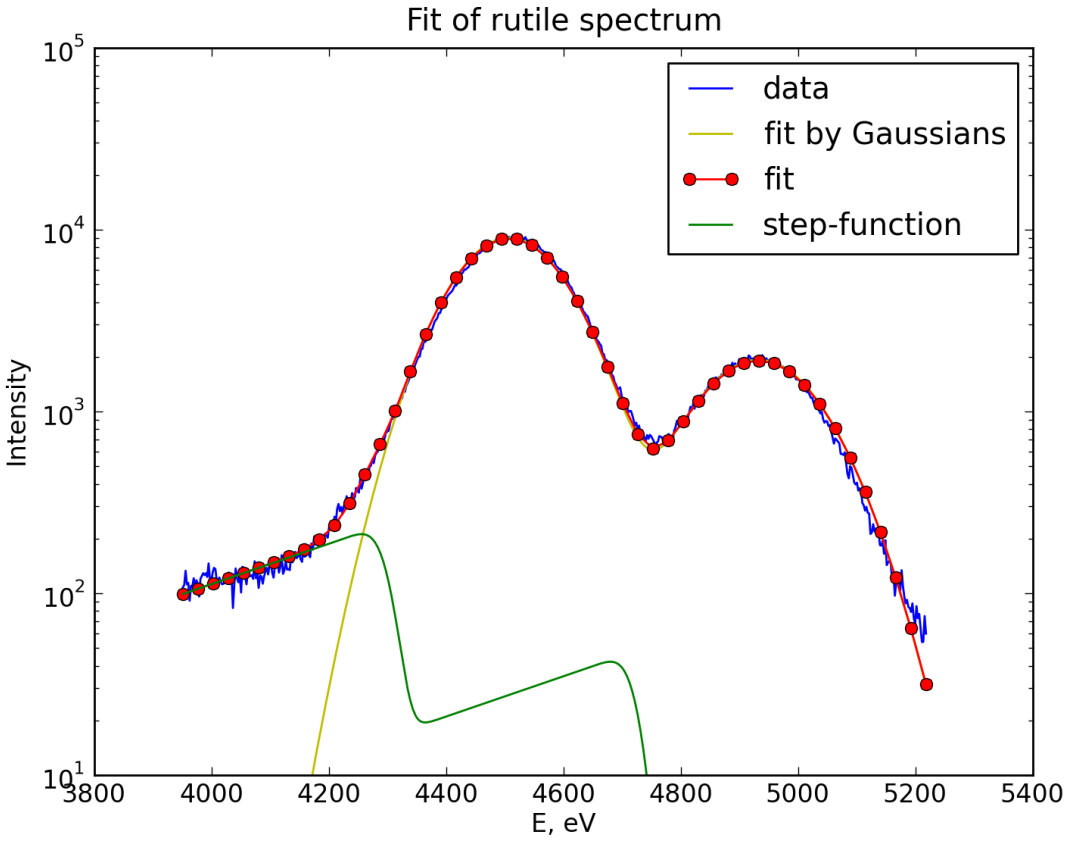


Figure 4: Approximation of Ti K_α and K_β fluorescence lines

7. However, when we add the third peak and take parameters of 2 peaks as the initial, plot of initial parameters looks very well, but program doesn't really work. To be continued. Results can be seen on figure 5.

A problem that also hasn't been solved is that program doesn't recognize Ar peak as Gaussian at the first step of approximation (because its amplitude is small comparing to other peaks even in logarithmic coordinates.)

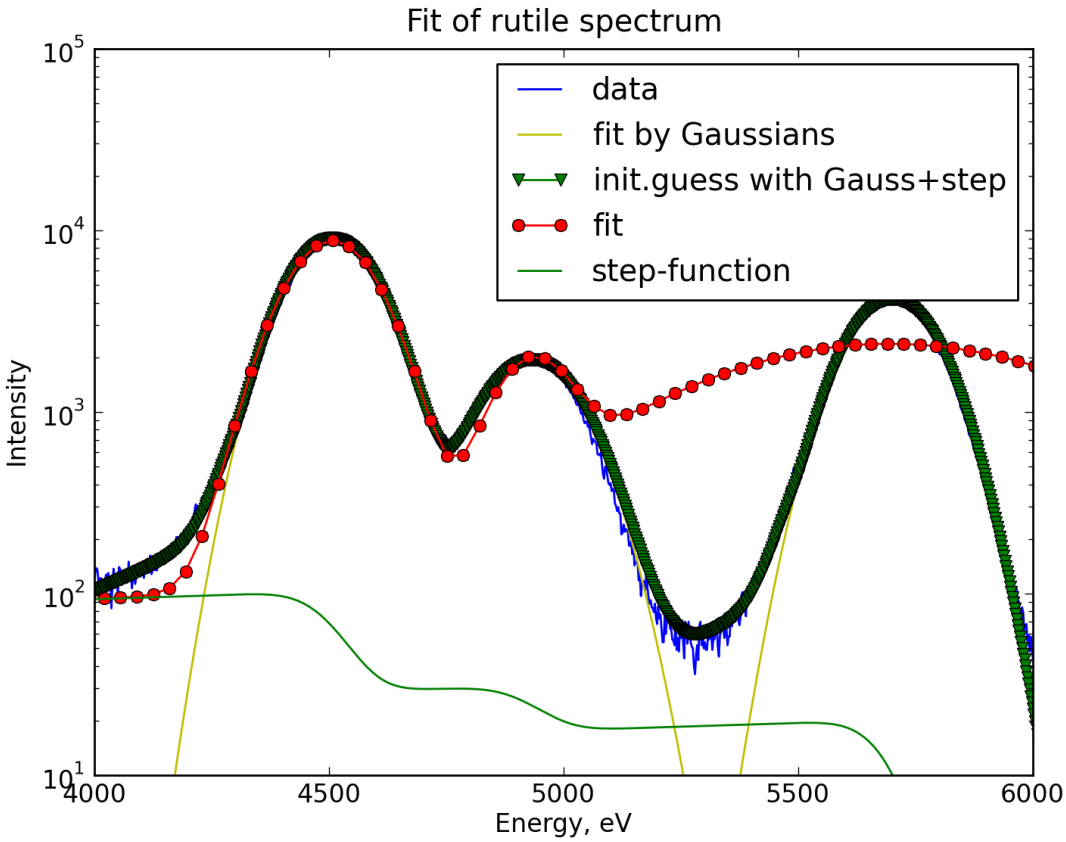


Figure 5: Approximation of Ti K_α and K_β fluorescence lines + elastic scattering peak

4 Conclusion and future development

In the frame of the project I have implemented a software module for quantitative evaluation of energy dispersive silicon drift detector data. The program was successfully applied to experimental data from rutile samples. The software is yet not capable to evaluate data sets with multiple overlaping peaks. This happens most probably due to the presence of weak maxima not described in the initial parameter list. Even weak maxima not incorporated into the fit will lead to instability of the fit procedure. It is therefore advised to first use the existing PyMca package to obtain a full list of chemical elements with corresponding fluorescence lines.

5 Acknowlegments

I strongly appreciate my supervisor Dmitri Novikov, for interesting task and brave decision to confide me (a chemist) writing a program.

I'm very grateful to Carsten Richter, whose fresh ideas have often been the last step to make my program work.
And a lot of thanks to summer student Christian Wehrberger for fun, Ubuntu, Latex and debugging.

6 Appendix

The appendix contains several examples of possible problems with approximation; text of the fit program; text of the program for 3D data presentation.

6.1 Appendix 1: difficult cases

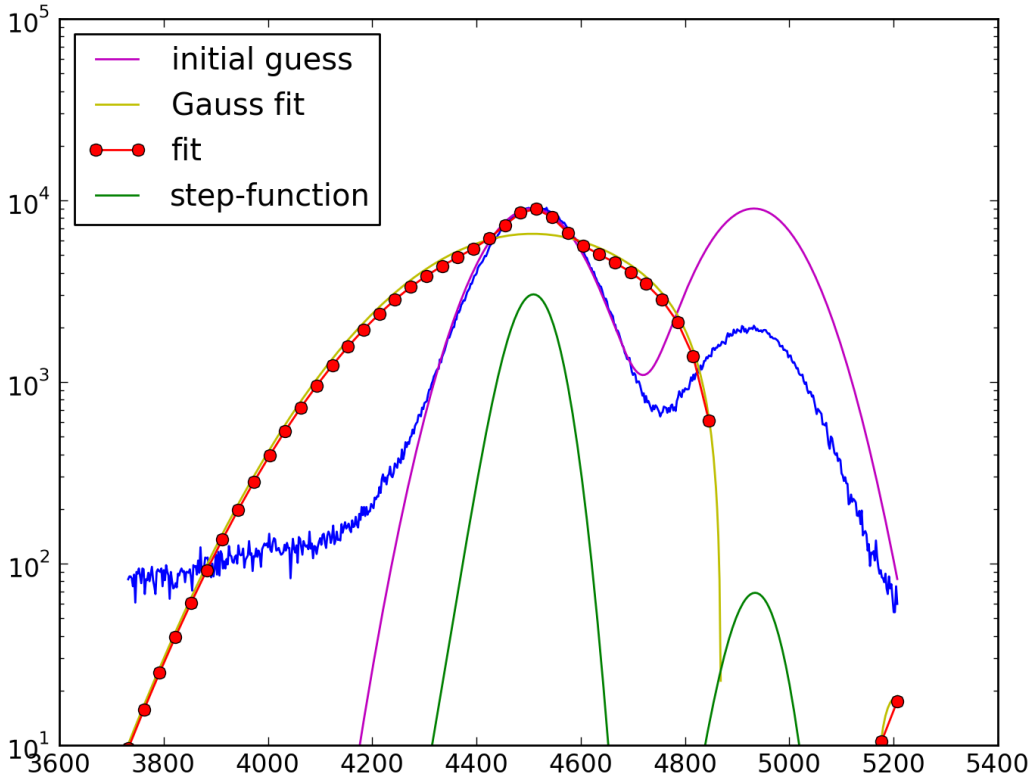


Figure 6: Fit with using semilogarithmic scale on both steps of approximation

This plot 6 shows an attempt to make approximation in two steps using logarithmic coordinates in both.

Reason for that was that long tales of step functions have much bigger weight in logarithmic coordinates than in linear coordinates, and approximation by Gaussians without step-function can't be held correctly (Gaussian tales approach to zero, and it doesn't answer real background).

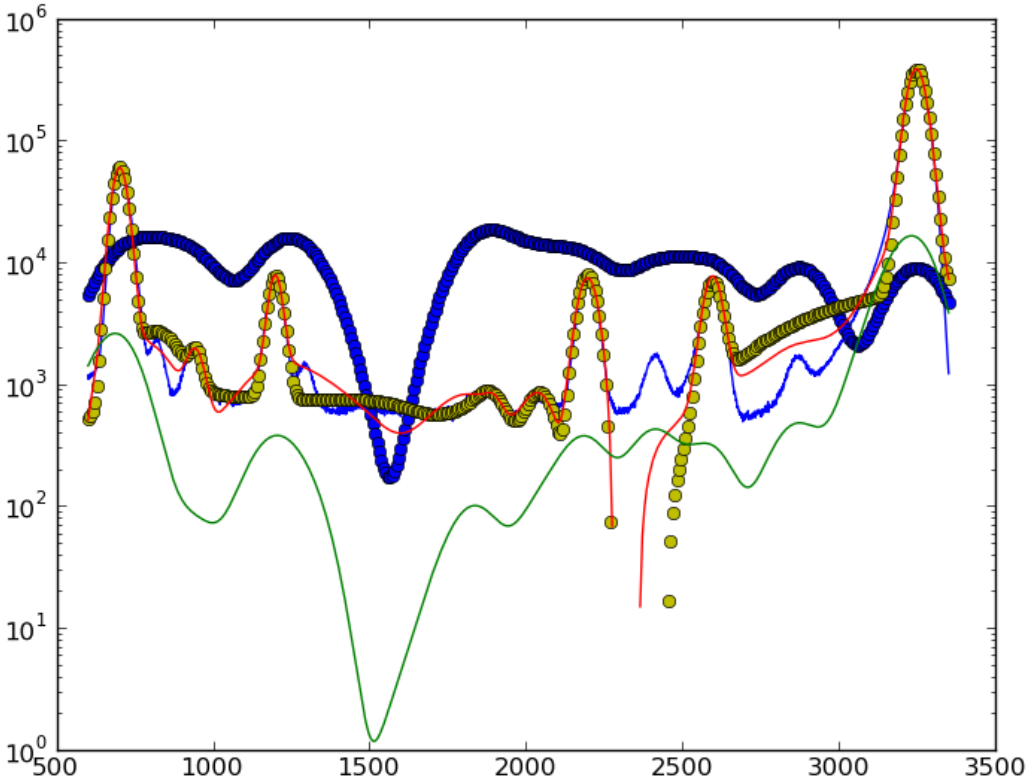


Figure 7: "Fit" of multielement spectrum

The plot 7 shows an attempt to use the program for multielement spectrum. It's clear that it doesn't work in this case.

6.2 Appendix 2: Fit program text

You can find the text of the program for approximation of Ti K_α and K_β fluorescence lines in additional file. In order to add some other peaks print their energies to 'l' array. Don't forget to change x and y values (lines 17, 18) and values of xmin, xmax, ymin, ymax values (line 114). Energy calibration is obligatory, otherwise these parameters don't fit.

```
from numpy import *
import matplotlib.pyplot as plt
from pylab import *
from scipy import *
from scipy import optimize
from scipy import special as sp
fig = plt.figure()

# we load all spectrum and cut 2 peaks from it
y = loadtxt ('/home/masha/Desktop/pythons/scan_mca_5700eV.fio')
x = arange (4096)+1

#we make rescaling: channels to energy. Here: calibration by Ka line of Ti
#and elastic scattering peak
a = (5700.-4507.)/(2459.-1942.)
x = 4507. + (x-1942.)*a
x = x[1700:2250]
y = y[1700:2250]+0.001 #we add small positive number to escape log(0) situations

plt.semilogy(x,y, label = 'data')

l = [4507., 4932.]

num = len(l) #number of peaks

# First part: approximations by Gaussians

# 1) creating p0 of necessary size:
#We take parameters of gaussian (Intensity, Width = 2sigma**2) and make
#an array of type: [I, W, I, W, ...] with the same initial parameters of gaussians.

pg = [9020., 16100.] # Initial guess for the parameters
p0 = []
for i in range(num):
p0 = hstack((pg,p0))

# 2) module for leastsquare method
```

```
# we have template for gaussian...
def fitfunc(p, x, l, ind1, ind2): return p[ind2]*exp(-(l[ind1]-x)**2/p[ind2+1])

#...and make a sum which contains as many blocks as many peaks we have
def fitfuncs(p, x, l):
ret = 0.
for i in arange(num):
j = i*2
ret += fitfunc(p, x, l, i, j)
return ret

#function for least square method
def errfunc(p, x, y): return fitfuncs(p, x, l) - y
 #least square method in work
p1, success = optimize.leastsq(errfunc, p0[:], args=(x, y))

# time is just coordinate instead of x to make plots.
time = linspace(x.min(), x.max(), 50)

#plt.semilogy(time, fitfuncs(p0, time, l), 'bo', label = 'initial guess')
plt.semilogy(x, fitfuncs(p1, x, l), 'y-', label = 'fit by Gaussians')

#Second part: approximation by Gaussian + step-function
#(in logarithmic coordinates)

# We take parameters of step-function (b, B, X, I) the same for all peaks
#and put them at the beginning of parameters-array c (c[0, 1, 2, 3]).
#I is  relative intensity I = I(step)/I(Gauss).
#Then we add the whole block of gaussian parameters.

#New array of initial parameters: [b(e), B(e), X(e), I(e)] + i*[I(g), W(g)]

c0 = p1
ce = [0.079, -30., -208., -0.04]
c0 = hstack((ce,c0))

# the same procedure as with gaussians: we make a template (fiterf) and use it
#for making function (fiterfs) from it.
def fiterf(c, x, l, ind1, ind2, num):
return c[ind2]*exp(-(l[ind1]-x)**2/c[ind2+1]) + 0.5*abs(c[3]*c[ind2])* \
exp(-(x-l[ind1])*c[0]/c[1]) * (sp.erf((x-l[ind1]+abs(c[2]))/c[1]) + 1)

def fiterfs(c, x, l):
```

```
ret = 0.
for i in arange(num):
j = i*2+4
ret += fiterf(c, x, l, i, j, num)
return ret

# We use special expression for errfunc for semilog cooordinates:
def errfunc_erf(c, x, y):
y_s = fiterfs(c,x,l)
y_s = abs(y_s)
#print (((log10((y_s)) - log10(y))*sqrt(y))**2).sum()
return (log10((y_s)) - log10(y))*sqrt(y)

c1, success = optimize.leastsq(errfunc_erf, c0[:], args=(x, y))

#plt.semilogy(x, fiterfs(c0, x, l), "v-", label = 'init.guess with Gauss+step')

plt.semilogy(time, fiterfs(c1, time, l), "ro-", label = 'fit')

#this block was created to plot a step-function". Principles are the same as earlier.
#We take array of new parameters 'c1' and make a plot with these data.
def ef(c, x, l, ind1, ind2, num): return 0.5*abs(c[3]*c[ind2])* \
exp(-(x-l[ind1])*c[0]/c[1]) * (sp.erf((x-l[ind1]+abs(c[2]))/c[1]) + 1)

def f(c, x, l):
retu = 0.
for i in arange(num):
j = i*2+4
retu += ef(c, x, l, i, j, num)
return retu

plt.semilogy(x, f(c1, x, l), 'g-', label = 'step-function')
plt.axis([3800, 5400, 10, 100000])
plt.xlabel('E, eV')
plt.ylabel('Intensity')
plt.title('Fit of rutile spectrum')
plt.legend(loc = 'best')

print 'b(e) = ', round(c1[0], 3)
print 'B(e) = ', round(c1[1],1)
print 'X(e) = ', int(c1[2])
print 'I(e) = ', round(c1[3],3)
print 'Gaussian parameters:'
print 'Energy, eV Intensity 2sigma**2'
```

```
for i in arange(num):
j = i*2+4
print int(l[i]),'', int(c1[j]), '', int(c1[j+1])

plt.savefig('/home/masha/Desktop/reportas/tex/5700_2peaks_lin+log.png', dpi = 200)

filename = 'fit_param.dat'
filehandle = open(filename,'w')
filehandle.write('b(e) = ' + str(round(c1[0], 3)) + '\n')
filehandle.write('B(e) = ' + str(round(c1[1],1)) + '\n')
filehandle.write('X(e) = ' + str(int(c1[2])) + '\n')
filehandle.write('I(e) = ' + str(round(c1[3],3)) + '\n')
filehandle.write('Gaussian parameters:\n')
filehandle.write('Energy [eV]\tIntensity\t2sigma**2\n')
for i in arange(num):
j = i*2+4
filehandle.write(str(int(l[i])) + '\t\t' + str(int(c1[j])) + '\t\t' + \
str(int(c1[j+1])) + '\n')
filehandle.close()

plt.show()
```

6.3 Appendix 3: 3D plots program

This program was written to make 3D plots when we have set of spectra and want to see them in one coordinate system. In this case I had 48 spectra and took natural numbers from 1 to 48 as y coordinate. For x coordinate I took numbers of channels (from 1 to 4096).

```
from numpy import *
import pylab as p
from matplotlib import cm
import mpl_toolkits.mplot3d.axes3d as p3
 #we define function that reads our file
a = 48                  # number of files
number0 = range(a)              #list of number of files, begins from 0

data = loadtxt('/home/masha/Desktop/data_files_from_bw1/f1.dat')
 # we need to count the size of array

column = data[:,0]             #for this we take one column from our array
column = column[200:3900]
clm = column.size              #and ask for its size
```

```
datalist = arange(clm)+1       #we create a list of Natural numbers.
#Later we will add new strings with data to it. It will become a 2D array.

 #we define function that reads our file
def read_from_file(lpath, lfile_name):
    ldata=loadtxt(lpath+'/'+ lfile_name, dtype = int)
    return ldata

path = '/home/masha/Desktop/data_files_from_bw1/'

names = range (a)             # we need a list, its size = number of files
for i in number0:
    names[i] = 'f' + str(i+1)+ '.dat'

x=column
y=arange(a)
z = column

for file_name in names:
    data = read_from_file(path, file_name)
    d = log(data[:,1]+1)
    d = d[200:3900]
    z = vstack((z, d))

[X,Y] = meshgrid(x,y)
fig=p.figure()
ax = p3.Axes3D(fig)

z = z[1:] # we don't need the first column (natural numbers) any more

surf = ax.plot_surface(X, Y, z, cmap=cm.jet)

ax.set_zlim3d(0, 14)

fig.colorbar(surf, shrink=0.5, aspect=5)

ax.set_xlabel('X')
ax.set_ylabel('Y')
#ax.set_zlabel('Z')
p.show()
```

7 Links

1. P. Lechner et al. “Silicon drift detectors for high resolution, high count rate X-ray spectroscopy at room temperature“ International Centre for Diffraction Data 2004, Advances in X-ray Analysis, V. 47, p.53 - 59.
2. P. Lechner et al. Silicon drift detectors for high count rate X-ray spectroscopy at room temperature. Nuclear Instruments and Methods in Physics Research: A. 2001, V. 458, I. 1-2, p. 281-287
3. Phd thesis of Bente Walz, DESY