Data_fitting

January 14, 2016

1 Data fitting

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of eperimental data points.

In this exercise we will calculate the sum of the "medipix.edf" image along the vertical and horizontal axis and fit a gaussian function, either using scipy or the module from PyMca.

1.1 Initialization of the scientific scripting environment

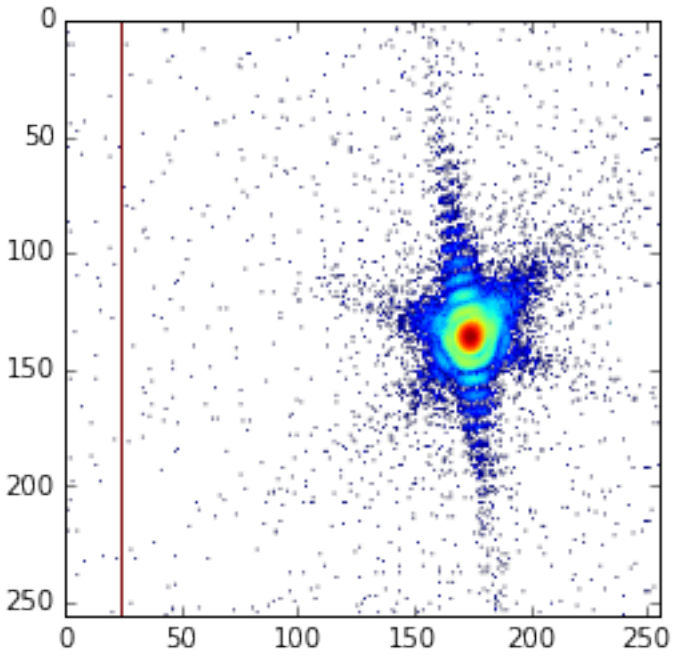
```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: import fabio
        img = fabio.open("medipix.edf").data
        imshow(np.log(img))
```

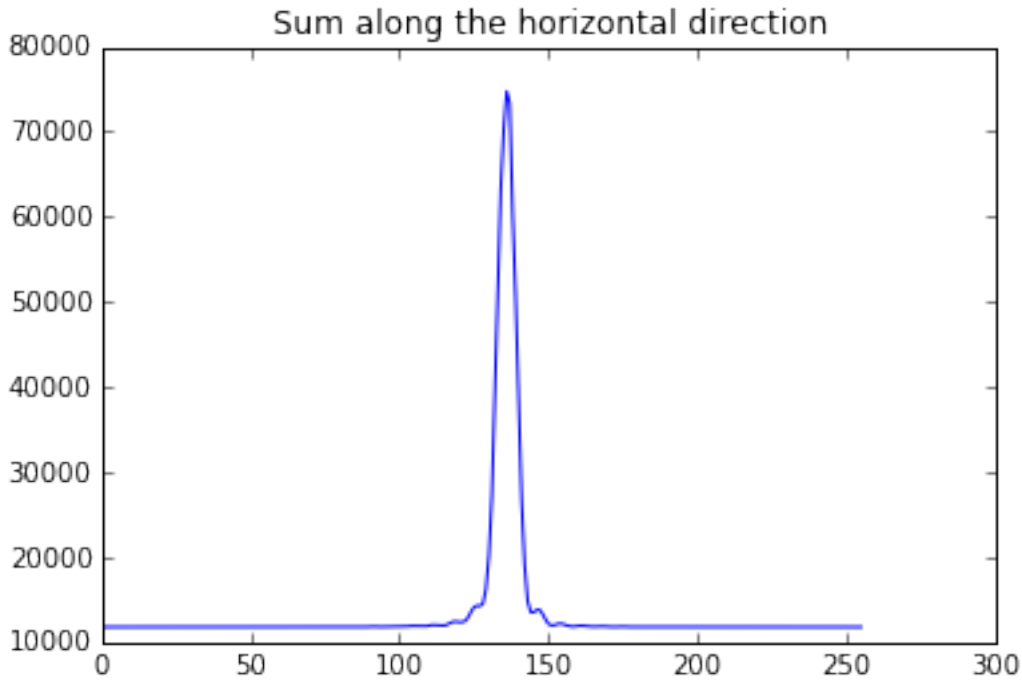
```
-c:3: RuntimeWarning: divide by zero encountered in log
```

```
Out[2]: <matplotlib.image.AxesImage at 0x7fe93341ccd0>
```



```
In [3]: vert = img.sum(axis=1)
        plot(vert)
        title("Sum along the horizontal direction")

Out[3]: <matplotlib.text.Text at 0x7fe9333559d0>
```



1.2 Definition of the fitting function

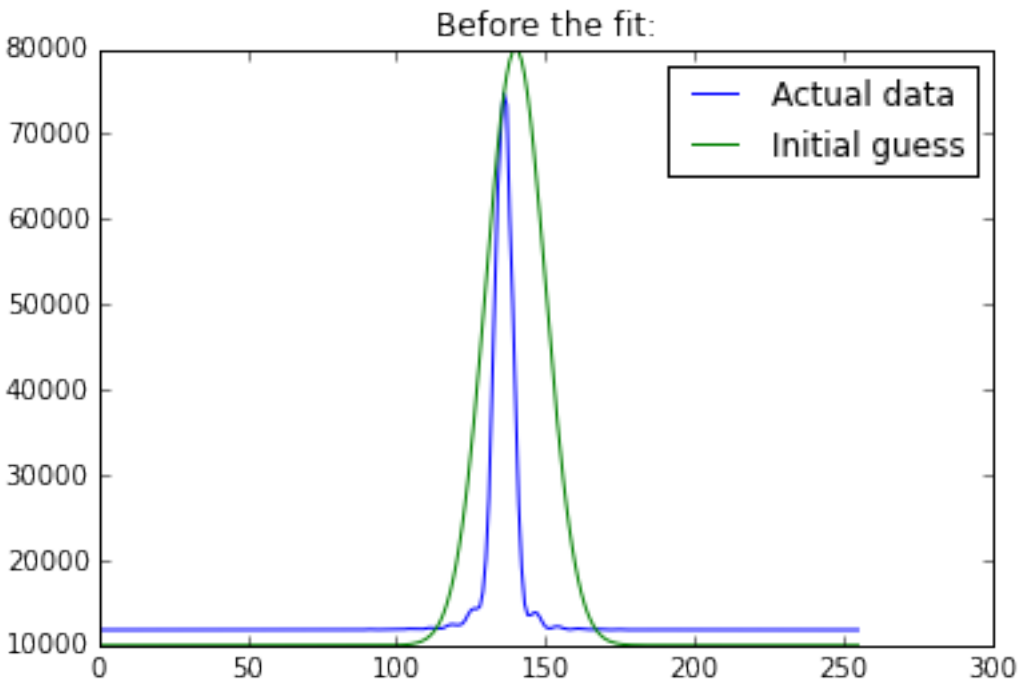
The fitting function is a function of the data-point (xdata) and of a sert of parameters.
For a gaussian function defined as: gaussian

```
In [4]: def gaussian(x, height, center, sigma, offset):
            return height*exp(-(x-center)**2/(2*sigma**2)) + offset
```

An essential part is the initial set of parameters p0:

```
In [5]: p0 = [70000, 140, 10,10000]
        xdata = numpy.arange(len(img),dtype="float")
        ydata = vert.astype("float")
        plot(vert, label="Actual data")
        plot(gaussian(xdata, *p0), label="Initial guess")
        legend()
        title("Before the fit:")

Out[5]: <matplotlib.text.Text at 0x7fe9332e22d0>
```



1.3 Fitting using *scipy* optimizers:

```
In [6]: from scipy.optimize import curve_fit
        p,cov = curve_fit(gaussian, xdata, ydata, p0)
        print(p)
```

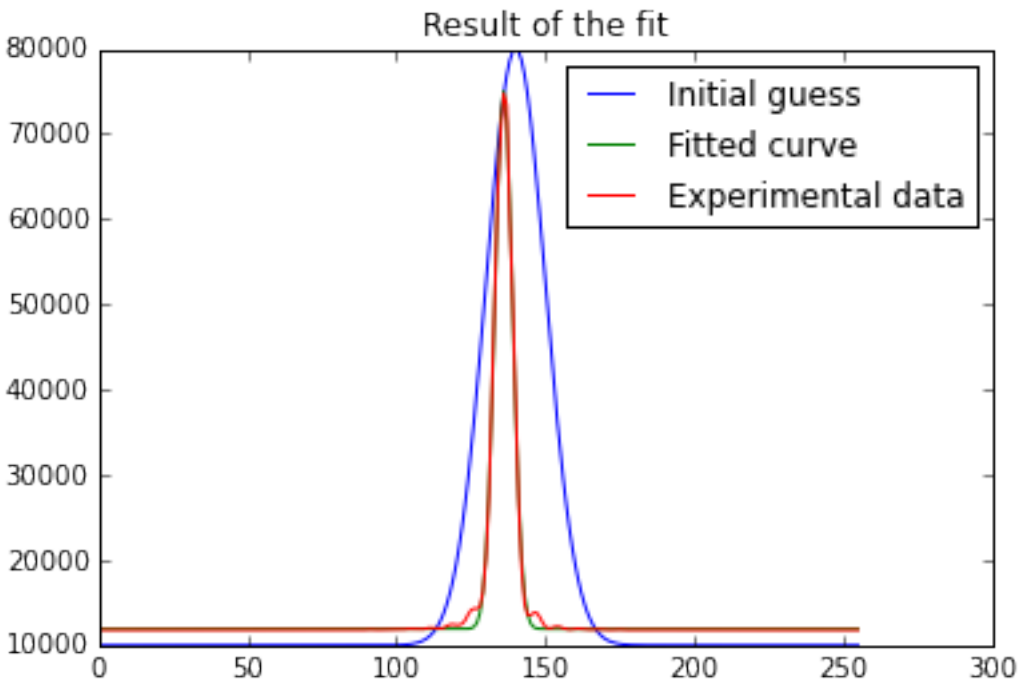
```
[  6.31031426e+04   1.35955545e+02   2.99885986e+00   1.19186952e+04]
```

```
In [7]: %timeit p,cov = curve_fit(gaussian, xdata, ydata, p0)
```

```
1000 loops, best of 3: 1.05 ms per loop
```

```
In [8]: #Display the result of the fit
        plot(gaussian(xdata, *p0), label="Initial guess")
        plot(gaussian(xdata, *p), label="Fitted curve")
        plot(vert,label="Experimental data")
        legend()
        title("Result of the fit")
```

```
Out[8]: <matplotlib.text.Text at 0x7fe92f6f93d0>
```



1.4 Fitting using the PyMca library

PyMca changed recently its position so depending if you are using PyMca4 or PyMca5 the import path varies ...

```
In [9]: try:
            from PyMca5.PyMca import Gefit
        except ImportError:
            from PyMca import Gefit
```

PyMca's Levenberg-Marquardt fitting module needs the function to be layed out the other way around:

```
In [10]: def gaussian_ge(param, xdata):
             return gaussian(xdata, *param)

         p, chi2, err = Gefit.LeastSquaresFit(gaussian_ge, p0, xdata=xdata, ydata=ydata)
         print(p)
```

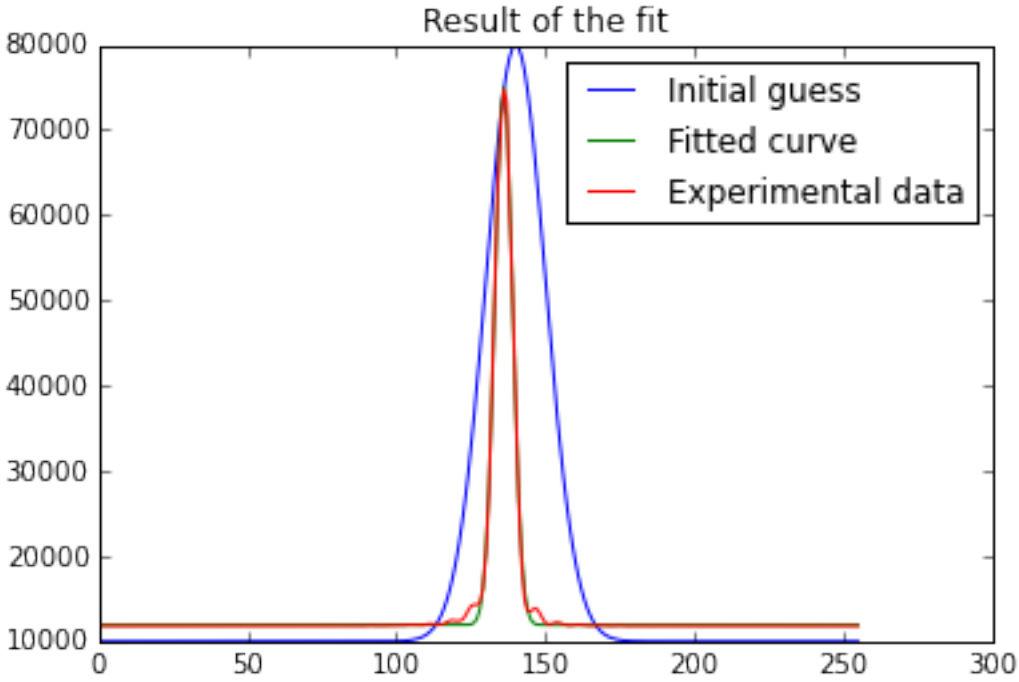
```
[63103.22036555503, 135.95551675017475, 2.998849240794874, 11918.81798431471]
```

```
In [11]: %timeit p, chi2, err = Gefit.LeastSquaresFit(gaussian_ge, p0, xdata=xdata, ydata=ydata)
```

```
100 loops, best of 3: 3.57 ms per loop
```

```
In [12]: #Display the result of the fit
         plot(gaussian(xdata, *p0), label="Initial guess")
         plot(gaussian(xdata, *p), label="Fitted curve")
         plot(vert,label="Experimental data")
         legend()
         title("Result of the fit")
```

Out[12]: <matplotlib.text.Text at 0x7fe92f699690>



2 Conclusion

Curve fitting goes alway via the following steps:

- get the data_point x_data and y_data
- define the fitting function as y_data = function (x_data, param)
- chose an initial guess for the set of parameters p0
- run the optimizer
- check the result.

Note that some optimizer don't fit the function but instead minimize the error_function: error(param) = y_data - function(x_data) It is often necessary to look at the documentation of the fitting function