OADnotesday3

Rajin Ramphul

September 04, 2013

1 Day3

1.1 Fitting with scipy

Fitting using in built packages

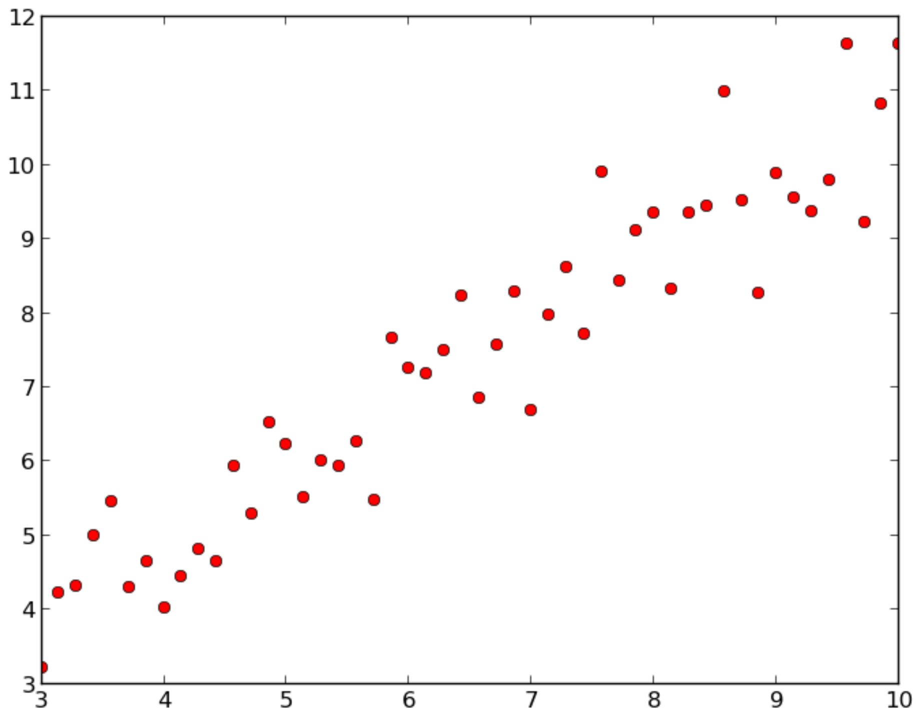
If you go through the method of least square you will know that it gets really tedious to derive the equations involved with 2 parameters (linear equation with gradient and y-intercept). Now in the case there are more parameters involve, or that the equation is not linear or as simple as that of a straight line? Then deriving those solutions will take ages and the coding might take long as well. Fortunately, people around the world have been developing techniques/ algorithm that you can use to do your least-square fitting in most cases that you will encounter. Also, those algorithms are in-built in some of the packages that you have in python, which makes life easy. All you then need to do is call those functions and they fit the data for you!

Straight line fitting

You have been provided ('lstsq1.txt') with some data. It is a noisy set of data for a straight line. First thing to do when you get some data, visualise how it looks – plot it using matplotlib.

```
In [11]: from IPython.core.display import Image
         Image(filename='1stplot.png',width = 500,height=500)
```

```
Out [11]:
```



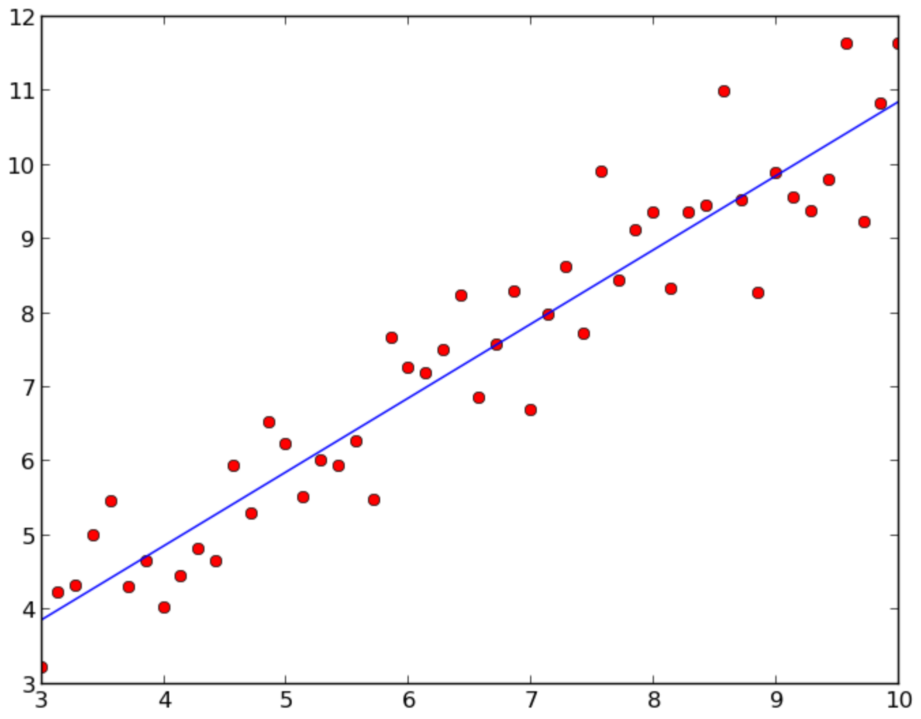
Below you are given the part of your code which does the fitting of a straight line using scipy module optimize.

```
# Fit the first set
fitfunc = lambda p, x: p[0] + p[1]*x # Target function where p[0] and p[1] are the paramete
errfunc = lambda p, x, y: fitfunc(p, x) - y # Distance to the target function for each data
p0 = [1.,1.] # Initial guess for the parameters, p
p1, success = optimize.leastsq(errfunc, p0[:], args=(x1, y1)) #x1 and y1 is your data and p
#optimize.leastsq gives back 2 variables (or arrays) and therefore it is better to store th
#here p1 and success.
```

try to understand it 1st. Remember that you are fitting a straight line with equation $y = a_0 + a_1 x$. Now your job is to write a code that will read the data file, do the fitting and then do the plotting of the answer as shown below

```
In [12]: Image(filename='2ndplot.png',width = 500,height=500)
```

```
Out [12]:
```



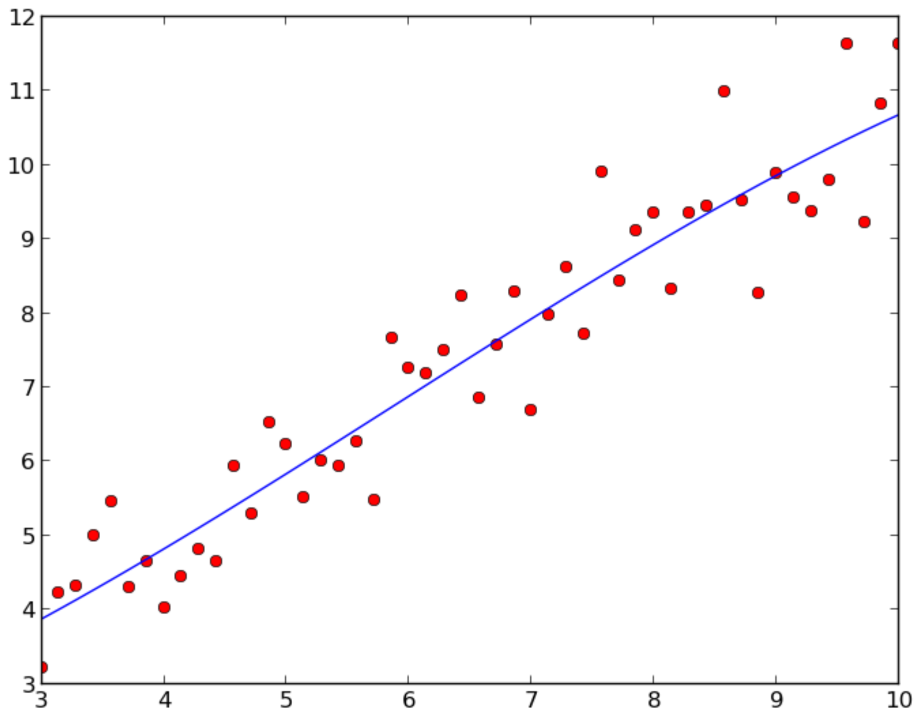
Just as a hint, when you are importing scipy do it in the following way

from scipy import optimize

Once you are done, try changing the code to fit a 3rd order (up to x3) polynomial to the same data.

In [13]:
```
Image(filename='3rdplot.png',width = 500,height=500)
```

Out [13]:



Exercise

A chemical engineer is investigating the viscosity (y) of a polymer. Two process variables are of interest: temperature ,x1, and catalyst feed rate ,x2, (lb/hr). Suppose that engineer decides to fit a linear regression model:

Equation: $y = \beta_0 + \beta_1 x1 + \beta_2 x2$

The table of values is saved into the file ‘table.txt’, evaluate the fitted regression model

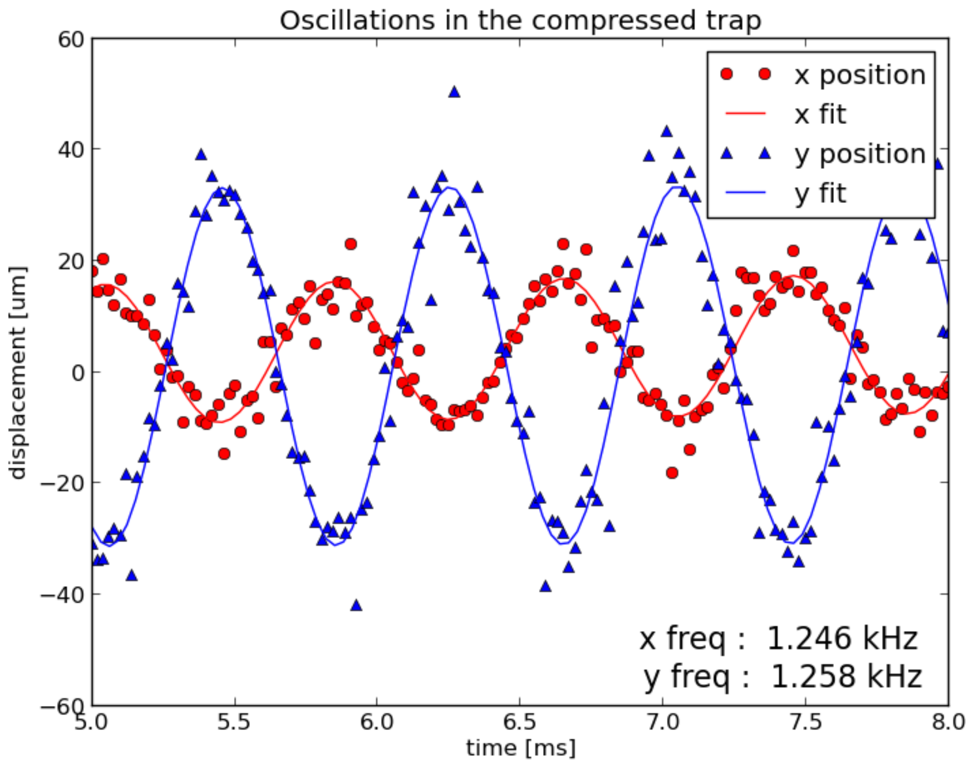
Note units of viscosity: centistokes (cSt) equilavent to mPa.s

Trigonometric fitting

Have a look at the data from ‘lstsq2.txt’ – This file contains 2 datasets. The file has 1 set of x values (1 column) and then 2 sets of y values with different periods. Try fitting trigonometric functions to both datasets so as to get a result which resembles to the following graph:

In [14]:
```
Image(filename='4thplot.png',width = 500,height=500)
```

Out [14]:



Weighted fitting

Up to now we have fitted the data by giving the same importance to each of the points we are fitting. To a first approximation, this is good, but if you have errorbars on your data points then you need to weigh each point by size of the errorbar associated with it. This is pure logic; The bigger the errorbar on a point, the less importance it should have, as we know that point to less precision. Points that are very precise – with small errorbars, you can give them, more weight when fitting.

Take a look at the data file – lstsq3.txt It is the same data file lstsq1.txt with the exception that in this one, we have a 3rd column where we have the errorbar for each point

Try a weighted least-square fitting on this new data file. To do that, modify the fitting function to:

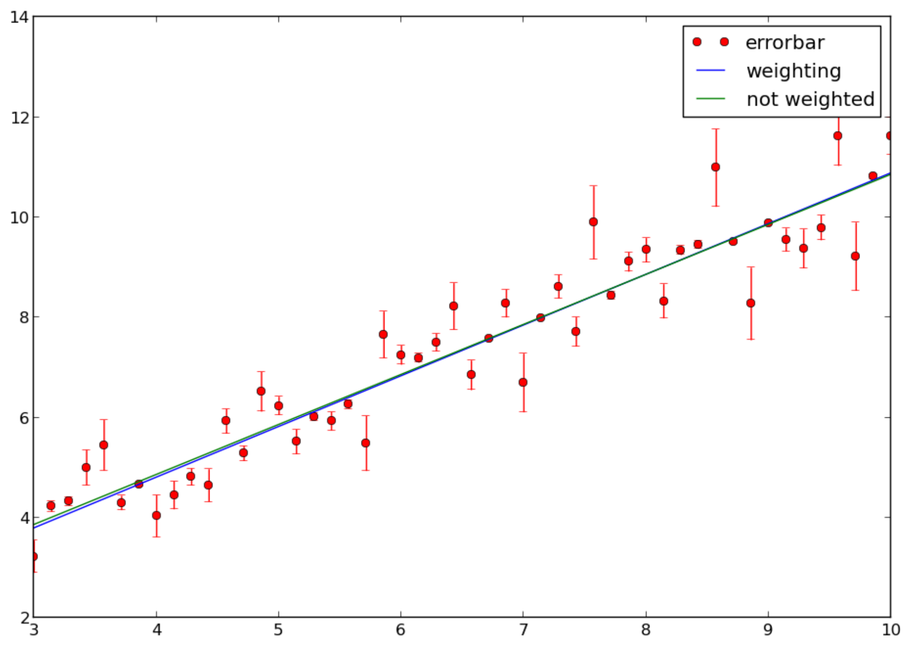
```
# Fit the first set
fitfunc = lambda p, x: p[0] + p[1]*x # Target function
errfunc = lambda p, x, y,yerr: (fitfunc(p, x) - y)/yerr # Distance to the target function
p0 = [1.,1.] # Initial guess for the parameters
p1, success = optimize.leastsq(errfunc, p0[:], args=(x1, y1,yerr)) #x1 and y1 is your data
```

You will notice that the only change that we did was to divide the residual (fitfunc(p, x) – y) by the errorbar. This is all that weighting requires. Therefore if yerr is big, [(fitfunc(p, x) – y)/yerr] will be small while with small errorbar, yerr will be small and [(fitfunc(p, x) – y)/yerr] will be big.

Create a plot to compare both fitting i.e. with and without weighting. You should get something like:

In [16]:
```
Image(filename='5thplot.png',width = 600,height=600)
```

Out [16]:



You will not notice a big difference in this particular case, but the fact that there is a difference in the fit, should convince you of the importance of weighting your data points when fitting.

Proper parameter estimation

One of the major problem that you will face when using optimize from scipy is that it output only the value of the parameter but does not give you any idea of the standard deviation associated with the fitted parameters. While in some cases (usually when doing quick evaluations) , it is fine to just have the value of the parameter, in research, this is not acceptable. Any scientific value has almost no meaning if you cannot give the uncertainty associated with it. What we require then, is a python package which, after the fitting, will give you the value of the parameter and its uncertainty. There are 2 main packages which does that; lmfit and kmpfit. Both are good, but kmpfit is a bit more advance and we will concentrate on using only kmpfit here to avoid confusion. Look at the example of a straight line fit using kmpfit below and run it.

In [17]:
```
import numpy as np
from kapteyn import kmpfit

def residuals(p, data):      # Residuals function needed by kmpfit
x, y = data            # Data arrays is a tuple given by programmer
a, b = p                 # Parameters which are adjusted by kmpfit
return (y-(a+b*x))

d = np.array([3.14,4.71,5.14,6.14,6.85,7.71,8.71,8.85,9.85])
v = np.array([2.76,4.52,6.67,6.59,7.58,8.88,9.95,10.38,10.56])

paramsinitial = [0, 0.0]
fitobj = kmpfit.Fitter(residuals=residuals, data=(d,v)) #initialising the
fitobj.fit(params0=paramsinitial)      #performing the fit
```

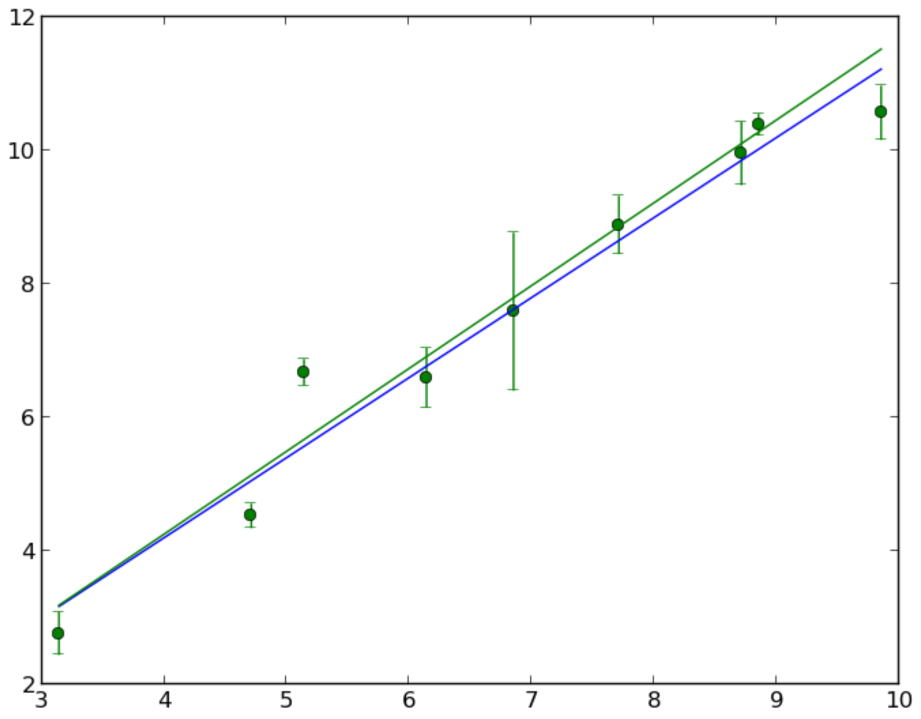
```
print "\nFit status kmpfit:"
print "===================="
print "Best-fit parameters:        ", fitobj.params
print "Asymptotic error:           ", fitobj.xerror
print "Error assuming red.chi^2=1: ", fitobj.stderr
print "Chi^2 min:                  ", fitobj.chi2_min
print "Reduced Chi^2:              ", fitobj.rchi2_min
print "Iterations:                 ", fitobj.niter
print "Number of free pars.:       ", fitobj.nfree
print "Degrees of freedom:         ", fitobj.dof
```

```
Fit status kmpfit:
====================
Best-fit parameters:         [-0.6051895609852O327,
1.2002734232669532]
Asymptotic error:            [ 1.137883    0.16025662]
Error assuming red.chi^2=1:  [ 0.65562223  0.09233621]
Chi^2 min:                   2.32386108888
Reduced Chi^2:               0.331980155555
Iterations:                  2
Number of free pars.:        2
Degrees of freedom:          7
```

Now if you had an array for the associated errors in the y value to be n = np.array([0.324,0.184,-0.197,0.447,-1.184,0.437,-0.475,-0.163,0.407]) Then modify the above code weight the fitting process. Plot both the not weighted and the weighted fit on a same graph as shown below

```
In [18]: Image(filename='6thplot.png',width = 600,height=600)
```

```
Out [18]:
```



Gaussian Fitting

Gaussian profile is one which is very commonly use to fit data in science because it is a good approximation to nature in lots of cases. There no big difference between linear fitting and gaussian fitting – only the model/residual function needs to be updated. Modify the program you have use above to fit a gaussian to the data ('gausslsq.txt') by using the following equation as model

$$f(x) = Ae^{\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} + noise \tag{1}$$

To plot your results use the following plotting script

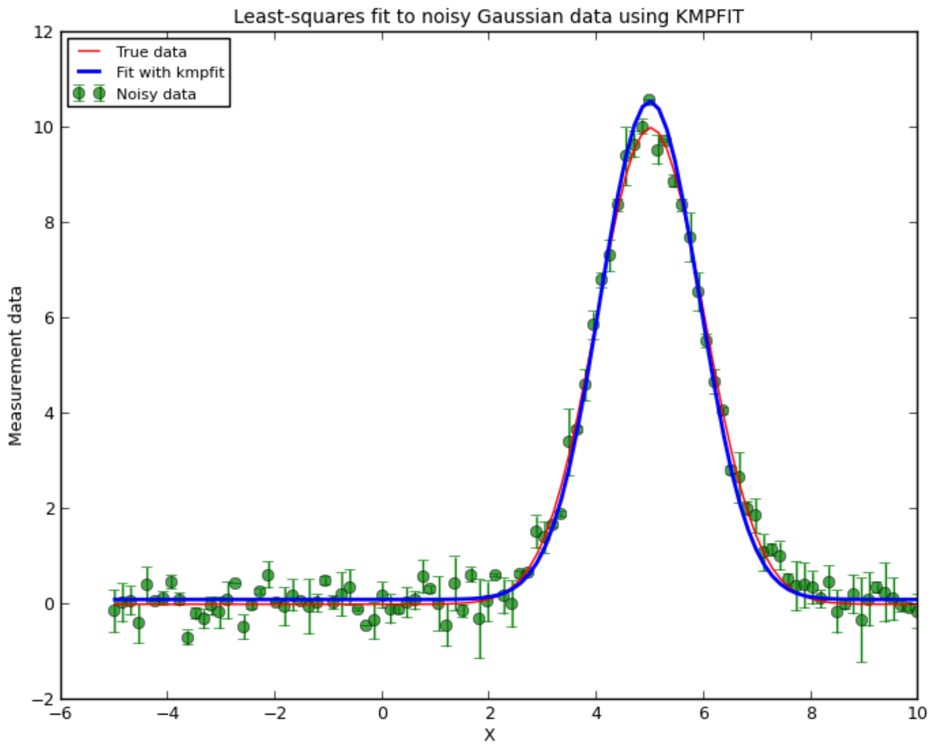
```
#import matplotlib in this way
from matplotlib.pyplot import figure, show, rc

# Plot the result (place at the end of your code)
rc('font', size=9)
rc('legend', fontsize=8)
fig = figure()
frame = fig.add_subplot(1,1,1)
frame.errorbar(x, y, yerr=err, fmt='go', alpha=0.7, label="Noisy data")
frame.plot(x, my_model(truepars,x), 'r', label="True data")
frame.plot(x, my_model(fitobj.params,x), 'b', lw=2, label="Fit with kmpfit")
frame.set_xlabel("X")
frame.set_ylabel("Measurement data")
frame.set_title("Least-squares fit to noisy Gaussian data using KMPFIT",
            fontsize=10)
leg = frame.legend(loc=2)
show()
```

where my_model is the gaussian function you need to write. Also dont forget that you still need to write a residual function.

In [19]:
```
Image(filename='7thplot.png',width = 600,height=600)
```

Out [19]:



Below is the link where the kmpfit package is found. It has an extensive tutorial, which I advise you to refer if you are doing fitting profiles or lines to data.

http://www.astro.rug.nl/software/kapteyn/kmpfittutorial.html