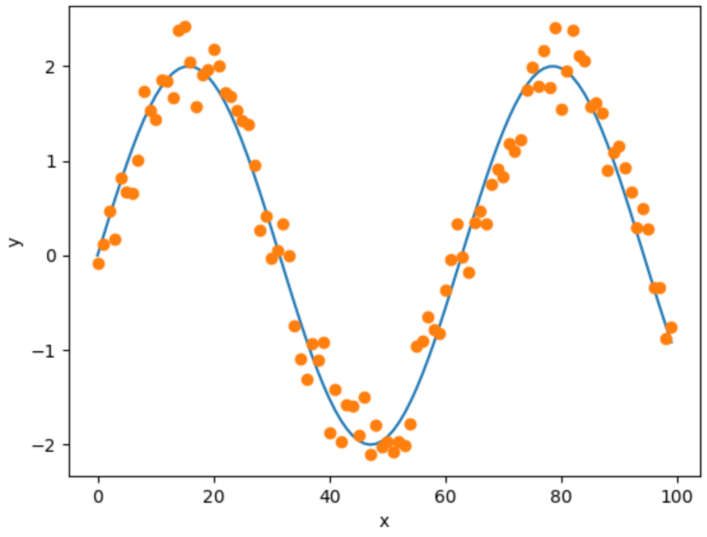
Function fitting using optimization/minimization in Python

Xiaoshan Xu

2018/12/06

Fitting experimental data

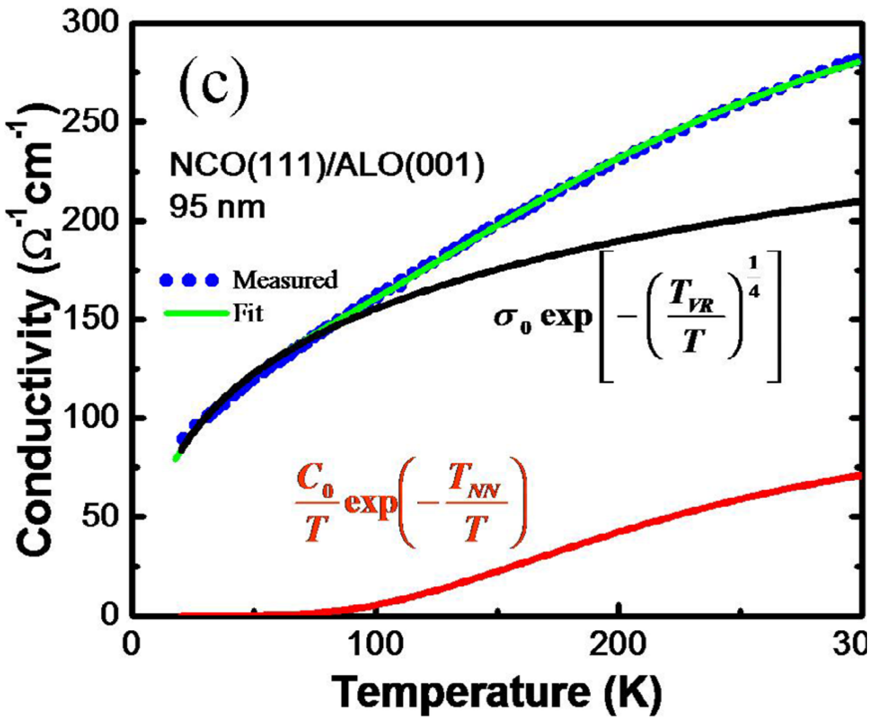
- Known:
  - Formula: $Y = f(x, a_1, a_2, \ldots)$, where x is the independent variable, y is the dependent variable, $a_1, a_2, \ldots$ are parameters that are independent on the variable x.
  - Experimental data: $\{x_i\}$ $\{y_i\}$
- If number of data points == number of parameters
  - $a_1, a_2\ldots$ can be directly solved from $y_1=f(x, a_1, a_2, \ldots)$, $y_2=f(x, a_1, a_2, \ldots)$
- If number of data points > number of parameters
  - Least square fit
    - $\chi_2(a_1, a_2, \ldots) = \Sigma\, [f(x_i)-y_i]^2$
    - Minimize chi2 by varying $a_1, a_2, \ldots$
    - Minimization/optimization algorithm: $\partial\chi_2/\partial a_j = 0$, $\partial^2\chi_2/\partial a^2_j > 0$
      - Python function:
        - from **scipy.optimize** import fmin
        - paras = **fmin**(chi2,paras0,args=(xs,ys));
        - Paras = $[a_1, a_2, \ldots]$
        - Chi2: $\chi_2(a_1, a_2, \ldots)$
        - xs, ys: experimental values



Fitting experimental data

- Known:
  - Formula: Y = f(x,a,b,c), where x is the independent variable, y is the dependent variable, a,b,c are parameters that are independent on the variable x.
  - Experimental data: $\{x_i\}$ $\{y_i\}$
- Simple example:
  - $\sigma(T) = \sigma_0 \exp\left[-\left(\frac{T_{VR}}{T}\right)^{\frac{1}{4}}\right] + \frac{C_0}{T}\exp(-\frac{T_{NN}}{T})$
  - Independent variable T
  - Dependent variable σ
  - Parameters: $\sigma_0, T_{VR}, C_0, T_{NN}$
  - Experimental data:

σ_1	T_1
σ_2	T_2
σ_3	T_3
…	…



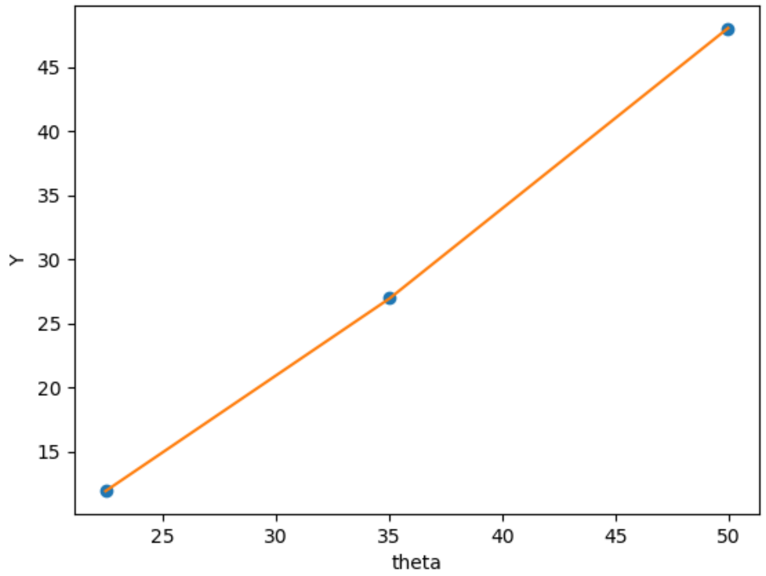
X-ray diffraction Cohen's Method

- True lattice parameter a_0 will satisfy Bragg's law:
- $\sin^2(\theta)_{true} = \frac{\lambda^2}{4a_0}(h^2 + k^2 + l^2)$. Use to rewrite $\Delta \sin^2(\theta)$:
- $\Delta \sin^2(\theta) = \sin^2(\theta)_{obs} - \sin^2(\theta)_{true} \Longrightarrow$
  $\sin^2(\theta) - \frac{\lambda^2}{4a_0}(h^2 + k^2 + l^2) = D sin^2(2\theta)$. This is an equation of form:
- $\sin^2(\theta) = C\alpha + A\delta; C \equiv \frac{\lambda^2}{4a_0}, \alpha \equiv (h^2 + k^2 + l^2), A = \frac{D}{10}, \delta = 10\sin^2(2\theta)$
- Enter experimental values for α, $\sin^2(\theta)$, and δ, solve for A and C (which has a_0!)

Complex function: identifying f(x)

- Known:
  - Formula: $\sin^2(\theta) - \frac{\lambda^2}{4a_0^2}(h^2 + k^2 + l^2) = Dsin^2(2\theta)$.
  - Experimental data: $\{h_i, k_i, l_i\}$ $\{theta_i\}$
- Redefine:
  - y = $h^2 + k^2 + l^2$;
  - x = θ
  - y=f(x) = $\frac{4a_0^2}{\lambda^2}$ $[\sin^2(x) - Dsin^2(2x)]$
  - Fitting parameters: a_0, D
  - Experimental data $\{h_i^2 + k_i^2 + l_i^2\}$ $\{theta_i\}$
- After fitting:
  - a= 8.09970337
  - D=9.75018311e-09
  - Chi2 = 0.005530

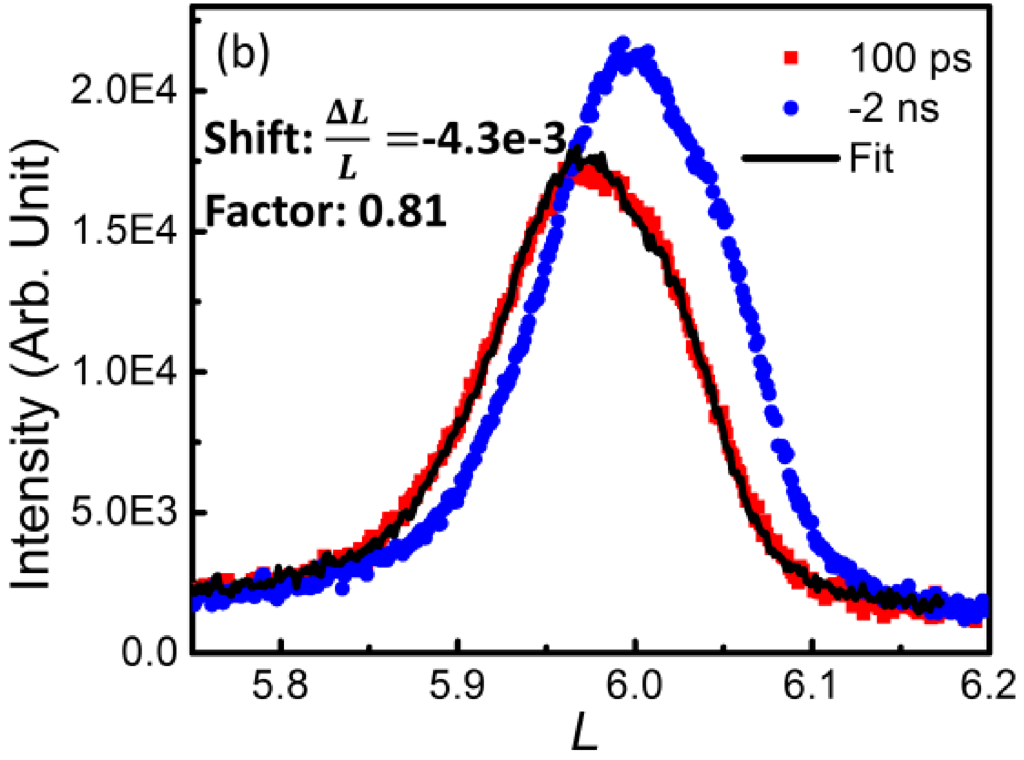
(h,k,l)	2theta/x	Y	a
222	44.9	**12**	**8.1188**
333	70.03	**27**	**8.1050**
444	99.96	**48**	**8.0968**



```
from pylab import *;
from scipy.optimize import fmin;

def f(theta,a,D):
y = 4*a**2/1.79**2*(sin(theta)**2-D*sin(2*theta));
return y;

def chi2(paras,thetas,ys):
a,D = paras;
yfit = f(thetas,a,D);
chi2 = ((yfit-ys)**2).sum();
return chi2;

thetas =array([44.9,70.03,99.96])/2/180*pi;
ys = array([12.,27.,48.]);
paras0 = [8.1,1e-8];
paras = fmin(chi2,paras0,args=(thetas,ys));

a,D= paras;
print "a=",a,"D=",D
```

More complex fit, beyond $y = f(x)$, $\{x_i\}$ $\{y_i\}$

- Known:
  - Formula: $Y1 = f1(x)$, $y2 = f2(x)$; $Y2 = a*f1(x-\text{delta})$
  - Experimental data $\{x_i\}$ $\{y1_i\}$ $\{y2_i\}$
- Redefine:
  - Cannot redefine an experimental $\{X_i\}$, $\{Y_i\}$ simply from $\{x_i\}$ $\{y1_i\}$ $\{y2_i\}$
  - Note that, what's really essential is $\chi_2(a_1,a_2,\ldots)$
  - As long as $\chi_2(a_1,a_2,\ldots)$ can be defined from $\{x_i\}$ $\{y1_i\}$ $\{y2_i\}$, the fitting can be done
  - Specifically
    - From $\{x_i\}$ $\{y1_i\}$, get $\{x_i\text{-}d\}$ $\{y1_i\}$
    - Find common range $\{x_i\text{-}d\}$ and $\{x_i\}$
    - Interpolate to get $\{y1_i'\}$
    - Calculate chi2 = $\Sigma\, [y1_i' - y2_i]^2$



Conclusion

- Least square fit can be used to find unknown parameters of a formula from the experimental data
- The python program can go beyond the simple $y = f(x)$, $\{x_i\}$ $\{y_i\}$ scenario.