Lecture 9

χ^2 Analysis

Part of data analysis is determining the values of parameters in a theory that will best fit data. You can see the importance of finding a good fit to data. If there does not exist any values that allow agreement with the data, then the theory (or model) is not valid. If an acceptable fit is found, then one obtains values for the parameters in the theory and thier uncertainties.

A common method of comparing theory and experiment is to perform a χ^2 analysis. In the analysis, a function, χ^2 is defined in such a way that the smaller the χ^2 function is, then the better is the fit to the data. For example, suppose there are N data values which we label as $D_i \pm e_i$, where i is an integer from one to N. We have labeled e_i as the measurement error for measurement D_i.

For the sake of discussion, suppose there is a theory that describes the outcome D_i of the experiment, and that the theory has three free parameters, x_1, x_2, and x_3. We label $T_i(x_1, x_2, x_3)$ as the theory value for data point i. Note that the theory values for the data points depend on the three free parameters x_1, x_2, and x_3.

The χ^2 function is defined as:

$$\chi^2(x_1, x_2, x_3) = \sum_{i=1}^{N} \left(\frac{D_i - T_i(x_1, x_2, x_3)}{e_i} \right)^2 \tag{1}$$

Note that each term in the sum is positive, since each term is squared. Also, the size of each term is equal to the difference between the experimental data and what the model (theory) would give, divided by the error. So, the smaller the value of the χ^2 function, the better the theory is in fitting the data. The "best fit" values of x_1, x_2, and x_3 are those that minimize the χ^2 function. A good measure of the quality of the fit is the value of the χ^2 function divided by the number of data points, χ^2/N. Acceptible fits will have the χ^2/N (χ^2 per data point) between one and two.

Lorentzian Peak

$$\sigma(E) = \sigma_{max} \frac{(\Gamma/2)^2}{(E - E_0)^2 + (\Gamma/2)^2} \tag{2}$$

where E_0 is the peak center and Γ is the width (FWHM).

Searching the parameter space for the best fit

I'll describe two ways to determine the parameters that minimize the χ^2 function: using the root class Tminuit, and a grid search method.

Using Tminuit

```
#include "TMinuit.h"

Double_t z[5],x[5],errorz[5];

Double_t func(Double_t x,Double_t *par)
{
Double_t value=par[0]*(par[2]*par[2]/4.0)/ ((x-par[1])*(x-par[1])+par[2]*par[2]/4.0)
+ par[3];
return value;
}

void fcn(Int_t &npar, Double_t *gin, Double_t &f, Double_t *par, Int_t iflag)
{
const Int_t nbins = 5;
Int_t i;

//calculate chisquare
Double_t chisq = 0;
Double_t delta;
for (i=0;i<nbins; i++) {
delta = (z[i]-func(x[i],par))/errorz[i];
chisq += delta*delta;
}
f = chisq;
}
```

```
void Lfit()
{
// The z values
z[0]=15;
z[1]=30;
z[2]=90;
z[3]=35;
z[4]=18;
// The errors on z values
Float_t error = 5.0;
errorz[0]=error;
errorz[1]=error;
errorz[2]=error;
errorz[3]=error;
errorz[4]=error;
// the x values
x[0]=100;
x[1]=110;
x[2]=120;
x[3]=130;
x[4]=140;
```

```
TMinuit *gMinuit = new TMinuit(4); //initialize TMinuit with 4 params max
gMinuit→SetFCN(fcn);

Double_t arglist[10];
Int_t ierflg = 0;

arglist[0] = 1;
gMinuit→mnexcm("SET ERR", arglist ,1,ierflg);

// Set starting values and step sizes for parameters
static Double_t vstart[4] = 80.0, 120.0 , 10.0 , 5.0;
static Double_t step[4] = 1.0 , 1.0 , 1.0 , 1.0;
gMinuit→mnparm(0, "a1", vstart[0], step[0], 0,0,ierflg);
gMinuit→mnparm(1, "a2", vstart[1], step[1], 0,0,ierflg);
gMinuit→mnparm(2, "a3", vstart[2], step[2], 0,0,ierflg);
gMinuit→mnparm(3, "a4", vstart[3], step[3], 0,0,ierflg);

// Now ready for minimization step
arglist[0] = 500;
arglist[1] = 1.;
gMinuit→mnexcm("MIGRAD", arglist ,2,ierflg);

// Print results
Double_t amin,edm,errdef;
Int_t nvpar,nparx,icstat;
gMinuit→mnstat(amin,edm,errdef,nvpar,nparx,icstat);
gMinuit→mnprin(3,amin);
}
```

Writing the code in c

If you don't have a library with a minimizing routine, you will have to write one yourself. We will solve our problem using a simple grid search procedure. I'll describe the method for one parameter, which is similar to our method for finding the zeros of a function. For our four parameter search, we will just fit each parameter separately, one after the other.

Suppose we want to find the mininum of a function $f(x)$. We start at some value x_{start}, choose a step size, Δ. If the function keeps decreasing as we increase x by steps of Δ, we are heading in the right direction. If not, then we step the other way. Once $f(x)$ starts increasing, we stop. For the final three values of x, labeled x_1, x_2, and x_3, we have:

$$f(x_1) > f(x_2) \; and \; f(x_3) > f(x_2) \tag{3}$$

x_2 is not the absolute minimum. One could decrease the step size and repeat the process, etc. However, since we also need to search on the other parameters as well, we will have to redo the search anyway. So a good way is to assume that the minimum is close to a parabola and find the value of x at the parabola's minimum point. The general form for a parabola is

$$y = ax^2 + bx + c \tag{4}$$

If we know a, and b, then the minimum is found when $dy/dx = 0$ or

$$x_{min} = -\frac{b}{2a} \tag{5}$$

Let $y_1 = f(x_1)$, $y_2 = f(x_2)$, and $y^3 = f(x_3)$. Then a, b, and c can be found by solving the three equations:

$$\begin{aligned} y_1 &= ax_1^2 + bx_1 + c \\ y_2 &= ax_2^2 + bx_2 + c \\ y_3 &= ax_3^2 + bx_3 + c \end{aligned}$$

Once a and b are known, then $x_{min} = -b/(2a)$. Solving these equations yields for x_{min}:

$$x_{min} = x_3 - \Delta \left(\frac{y_3 - y_2}{y_3 - 2y_2 + y_1} + \frac{1}{2} \right) \tag{6}$$

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double z[5],x[5],y[5],errorz[5],par[5];
double vstart[4], step[4],error, chisq, delta, theory;
int i,j,iloop, nbins;
double test1, test2, test3;

void fcn();
void fcn()
{
chisq = 0;
for (i=0;i¡nbins; i++)
{
theory=par[0]*(par[2]*par[2]/4.0)/((x[i]-par[1])*(x[i]-par[1])+par[2]*par[2]/4.0)+par[3];
delta = (z[i]-theory)/errorz[i];
chisq =chisq + delta*delta;
}
}
```

```
void main()
{
nbins=5;
// The z values
z[0]=15.0;
z[1]=30.0;
z[2]=90.0;
z[3]=35.0;
z[4]=18.0;
// The errors on z values
error = 5.0;
errorz[0]=error;
errorz[1]=error;
errorz[2]=error;
errorz[3]=error;
errorz[4]=error;
// the x values
x[0]=100;
x[1]=110;
x[2]=120;
x[3]=130;
x[4]=140;
// the starting values
par[0] = 90;
par[1] = 120.0;
par[2] = 20.0;
par[3] = 10.0;

step[0]=1.0;
step[1]=1.0;
step[2]=1.0;
step[3]=1.0;
```

```
for (iloop=1; iloop<100;iloop++)
{
for (j = 0; j < 4; j++)
{
par[j] = par[j] - step[j];
fcn();
test1 = chisq;
par[j] = par[j] + step[j];
fcn();
test2 = chisq;
if (test2 > test1)
{
step[j] = -1.0 * step[j];
test2 = test1;
test1 = chisq;
}
test3 = test2;
while (test3 <= test2)
{
par[j] = par[j] + step[j];
fcn();
test3 = chisq;
if (test3 < test2)
{
test1 = test2;
test2 = test3;
}
}
par[j] = par[j] - step[j] * ((test3 - test2) / (test3 - 2.0 * test2 + test1) + 0.5);
printf("chi2 = %lf \n",chisq);
}
printf("p1= %lf p2= %lf p3= %lf p4= %lf \n", par[0],par[1],par[2],par[3]);
}
}
```