Least Squares Fit to a Linear Model

Computational Physics

Least Squares Fit
to a Linear Model

Outline

- Weighted Least Squares
- Linear Least Squares
- Example

Weighted Least Squares

Consider N measurements with DIFFERENT error values. How do we solve For the most probable mean value if our errors are not all the same??

Here is the probability of observing a particular result, given mean and variance:

$$P(x,\mu') = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} e^{-\frac{(x-\mu')^2}{2\sigma^2}}$$

So the probability of observing a set of N observations is the PRODUCT of these Individual probabilities:

$$P_{set}(\mu') = \prod_{i=1}^{N} P(x_i, \mu')$$

Expand this result.

$$P_{set}(\mu') = \left(\frac{1}{2\pi\sigma_1^2}\frac{1}{2\pi\sigma_2^2}\cdots\frac{1}{2\pi\sigma_N^2}\right) \exp\left(\sum_{i=1}^{N}\frac{(x_i-\mu')^2}{2\sigma_i^2}\right)$$

We will get the MAXIMUM probability when we MINIMIZE the quantity

$$S = \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{\sigma_i^2}$$

Minimizing "S" corresponds to weighting each data point by the square of its measurement error. Points with large errors do not contribute as much as points with small errors.

The quantity to minimize in a least squares sense is:

$$S = \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma_i^2}$$

Find the minimum

$$\frac{dS}{d\mu} = \sum_{i=1}^{N} -2\frac{(x_i - \mu)}{\sigma_i^2} = 0$$

to obtain the least squares result

$$\mu = \frac{\sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}}{\sum_{i=1}^{N} \frac{1}{\sigma_i^2}}$$

Linear Least Squares Fit

Linear Least Squares

- Linear Models

$$y = c_1 + c_2 x + c_3 x^2$$

$$y = c_1 + c_2 cos(x) + c_3 sin(x)$$

- Non-Linear Models

$$y = c_1 e^{c_2 t} + c_3$$

$$y = c_1 + c_2 cos(x + c_3)$$

$$\frac{dy}{dc_3} = -c_2 sin(x + c_3)$$

C3 is NON-LINEAR PARAMETER

Fit linear model to N data points:

Linear model with parameters: c1 c2 c3

$$y = c_1 + c_2 x + c_3 x^2$$

Write as linear system of equations for N points

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_i \\ \dots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \dots & \dots & \dots \\ 1 & x_i & x_1^2 \\ \dots & \dots & \dots \\ 1 & x_N & x_N^2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Parameters

Data

Model

Linear System of Equations

Parameters

$$\vec{y} = M \, \vec{c}$$

Data

Model

Least Squares Solution

$$\vec{c} = (M^T M)^{-1} M^T \, \vec{y}$$

Covariance Matrix

$$\begin{pmatrix} \sigma_{c1}^2 & \sigma_{c_1 c_2}^2 & \sigma_{c_1 c_3}^2 \\ \sigma_{c_2 c_1}^2 & \sigma_{c2}^2 & \sigma_{c_2 c_3}^2 \\ \sigma_{c_3 c_1}^2 & \sigma_{c_3 c_2}^2 & \sigma_{c3}^2 \end{pmatrix} = (M^T M)^{-1} \, \sigma_y^2$$

Python Example

```
# example of linear fit to polynomial - assume normal imports!

# make an "anonymous function" f
f = lambda x: 5.+3.*x+2.*x**2
npar = 3          # number of parameters

# X positions of data
X = np.linspace(0., 20., 21)
nobs = len(X)     # number of observations

# Y positions of data follow function f with random error
r = RandomState()
Y = f(X) + r.randn(nobs)

# create M - ones in first col, X in second col., X**2 in 3rd.
M = np.column_stack( (np.ones(nobs),X,X**2) )

# solve
MTM = np.dot(M.transpose(),M)
MTMINV = np.linalg.inv(MTM)
MTY = np.dot(M.transpose(),Y)

P = np.dot(MTMINV,MTY) # SOLUTION
```

Example of fit to a quadratic function.

Create some fake data with "noise" here.

This section is just a calculation of the least squares fit.

```
# compute ChiSq, RMS and print it
Residuals = Y - np.dot(M,P)
ChiSq = np.dot(Residuals.transpose(),Residuals)
RMS = math.sqrt(ChiSq/nobs)
print 'RMS = %f'%(RMS)

C = MTMINV * ChiSq/(nobs-npar)  # COVARIANCE MATRIX

# print solution
print 'Constant = %f +/- %f ' % (P[0],math.sqrt(C[0,0]))
print 'Linear    = %f +/- %f ' % (P[1],math.sqrt(C[1,1]))
print 'Quadratic= %f +/- %f ' % (P[2],math.sqrt(C[2,2]))
```

Now we compute the residuals and chi-square for the fit

We use ChiSq to estimate the variance of the data here.

OUTPUT:

```
RMS = 1.025832
Constant = 4.121529 +/- 0.661386
Linear    = 3.317490 +/- 0.153252
Quadratic= 1.984196 +/- 0.007398
```

Note that RMS is close to standard deviation of noise added to the function (1).

Correlation Coefficients

```
In [6]: C

Out[6]:

array([[  4.37431565e-01,  -8.52679597e-02,   3.46617722e-03],
       [ -8.52679597e-02,   2.34860871e-02,  -1.09458228e-03],
       [  3.46617722e-03,  -1.09458228e-03,   5.47291140e-05]])

In [7]: np.corrcoef(C)

Out[7]:

array([[ 1.        , -0.99832399,  0.99635529],
       [-0.99832399,  1.        , -0.99962192],
       [ 0.99635529, -0.99962192,  1.        ]])
```

Examine covariance matrix C

Use numpy corrcoef method to compute correlation coefficients between parameters.

NOTE: correlation coefficients are very high for this sort of function fit.

Errors on parameters are correspondingly high.