2018-09-01

Fitting Parameter Uncertainties in Least Squares Fitting

R. Steven Turley
Brigham Young University, turley@byu.edu

Fitting Parameter Uncertainties

Version 1.0

R. Steven Turley

September 1, 2018

Contents

1 Introduction

This article derives the formulas for estimating parameter uncertainties in least square fitting to data. It relies heavily on Bevington[1], the first edition of which served as my introduction to the subject as an undergraduate. There is additional helpful background on Wikipedia[2] and MathWorld[3]. I will use that same notation as in my article on linear least squares fitting[4] which is an inter-related companion to this one.

The goal in least squares is to find the best fit to a function of the form $f(x;\vec{b})$ to a set of data points (x_i, y_i). It is called "least squares" because by "best fit" I mean the function which finds the set of p parameters b_k which minimizes χ^2, the sum of the squares of the differences between $f(x_i;\vec{b})$ and y_i.

$$\chi^2 = \sum_{i=1}^{n} [f(x_i;\vec{b}) - y_i)]^2 \tag{1}$$

If the data is heteroscedactic (i.e. if the extent of the deviations of y_i from $f(x_i,\vec{b})$ varies across the range of x_i, the appropriate function to minimize is a weighted sum of the squares. This can be written in terms of the weights w_i or in terms of the relative uncertainties at each data point σ_i.

$$\chi^2 \ \sum_{i} [w_i(x_i;\vec{b}) - y_i)]^2 \tag{2}$$

Note that it defined w_i differently here than in the linear least squares paper[4] to match a convention in the Julia library code `LsqFit.jl` which I modified to use for these studies.

2 Estimating Uncertainties

The estimated uncertainty in the measured data s can be calculated from χ^2.

$$s = \sqrt{\frac{\chi^2}{n-p}} \tag{3}$$

$$= \sqrt{\frac{1}{n-p}\sum_{i=1}^{n}[f(x_i;\vec{b}) - y_i)]^2} \tag{4}$$

If we expand $f(y;\vec{b})$ in a Taylor series about the mean value $\bar{y}$,

$$y_i = f(x_i;\vec{b}) \tag{5}$$

$$\approx \bar{y} + \sum_k (b_k - \bar{b}_k)\left(\frac{\partial f}{\partial b_k}\right) + \cdots \tag{6}$$

$$y_i - \bar{y} = \sum_k (b_k - \bar{b}_k)\left(\frac{\partial f}{\partial b_k}\right) + \cdots \tag{7}$$

Substituting Eq. 7 into Eq. 4 and keeping only the first order terms,

$$s_y^2 = \frac{1}{n-p}\sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{8}$$

$$\approx \frac{1}{n-p}\left[\sum_{i,k}(b_k - \bar{b}_k)\left(\frac{\partial f}{\partial f}\partial b_k\right)\right]^2. \tag{9}$$

If we define C_{ij} to be the elements of the covariance matrix with

$$C_{ij} \equiv \frac{1}{n-p}\sum_i (b_i - \bar{b}_i)^2, \tag{10}$$

we note the the uncertainty in the fit parameter b_i is s_i and

$$s_i^2 = C_{ii}. \tag{11}$$

Thus the diagonal elements of the covariance matrix are the uncertainties we are seeking. The off-diagonal elements show the statistical correlations between the fit parameters. The Jacobian J of f is defined to be

$$J = \begin{pmatrix} \partial f(x_1;\vec{b})/\partial b_1 & \cdots & \partial f(x_1;\vec{b})/\partial b_p \\ \vdots & \vdots & \vdots \\ \partial f(x_n;\vec{b})/\partial b_1 & \cdots & \partial f(x_n;\vec{b})/\partial b_p \end{pmatrix}. \tag{12}$$

If we expand the quadratic in Eq. 9 and write the matrix C as

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1,p-1} & c_{1,p} \\ c_{21} & c_{22} & \cdots & c_{2,p-1} & c_{2,p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{p-1,1} & c_{p-1,2} & \cdots & c_{p-1,p-1} & c_{p-1,p} \\ c_{p1} & c_{p2} & \cdots & c_{p,p-1} & c_{p,p} \end{pmatrix} \tag{13}$$

and represent the identity matrix as I, then we can write Eq. 9 as

$$s_y I = C(J^T J) \tag{14}$$

$$C = s_y (J^T J)^{-1} \tag{15}$$

$$\sigma_i \approx \sqrt{C_{ii}}. \tag{16}$$

Note that if the f is linear in the parameters b_k, then f does not have any derivatives of higher order than 1. In this case, Eq. 7 is exact. However the computed uncertainties in the fit parameters are still an estimate since s_y only equals σ_y and s_i only equals σ_i in the limit as $n \to \infty$.

3 Numerical Examples

I will consider two numerical examples which demonstrate how well this formula works.

3.1 Linear Fit

Consider the function

$$f(x; b_1.b_2) = b_1 + b_2 x. \tag{17}$$

In other words,

$$g_1(x) = 1 \tag{18}$$

$$g_2(x) = x. \tag{19}$$

The Jacobian J has the elements

$$J_{i,1} = 1 \tag{20}$$

$$J_{i,2} = x_i \tag{21}$$

$$J = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_{n-2} \\ 1 & x_{n-1} \end{pmatrix} \tag{22}$$

$$J^T J = \begin{pmatrix} \sum x_i & \sum x_i^2 \\ N & \sum x_i \end{pmatrix} \tag{23}$$

Using the result from my linear least squares article[4] or polynomial fitting article[5], the solution for the fit parameters is the solution to

$$Ab = y \tag{24}$$

where

$$A = \begin{pmatrix} N & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix} \tag{25}$$

$$y = \begin{pmatrix} \sum y_i \\ \sum y_i x_i \end{pmatrix}. \tag{26}$$

To get good approximations, I'll use an array with 1,000 points from 0 to 1 for x and $m = 1.5$, $b = 3.0$. The random noise will have an amplitude of 0.2. Here is the Julia code for the implementation.

The function `lfit` fits 1,000 points to the line described in the previous paragraph. It returns the fit parameters in column 1 of the returned matrix and the estimated uncertainties of the parameters in column 2.

```
using LinearAlgebra
using Printf

function lfit()
    npts = 10000
    xpts = [(i-1.0)/(npts-1) for i=1:npts]
    m = 1.5
    b = 3.0
    ypts = b .+ m.*xpts .+ randn(npts)*0.2
    A = Matrix{Union{Missing, Float64}}(missing, 2, 2)
    y = Array{Union{Missing, Float64}}(missing, 2)
    A[1,1] = npts
    A[1,2] = sum(xpts)
    A[2,1] = A[1,2]
    A[2,2] = sum(xpts.*xpts)
    y[1] = sum(ypts)
    y[2] = sum(ypts.*xpts)
    b = A\y
    J = hcat(ones(npts),xpts)
    yfit = b[1] .+ xpts.*b[2]
    res = ypts .- yfit
    sigmay = sqrt.(sum(abs2,res)/(npts-2))
    cov = inv(J'*J)
    sigmai = sigmay.*sqrt.(diag(cov))
    [b sigmai]
end
```

The following code calls `lfit` 1,000 times to compute the fit parameters with successive sets of random noise. With n measurements of x

$$\sigma_x = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}. \tag{27}$$

The sums for the means of the fit parameters are accumulated in `psum`. The sums for the means of the squares of the fit parameters are accumulated in `sumsq`.

```
# repeat 1000 times for averages
let
    psum = zeros(2,2)
    sumsq = zeros(2,2)
    trials = 1000
    for i=1:trials
        ft = lfit()
        psum += ft
        sumsq += ft.*ft
    end
    pbar = psum./trials
    pbarsq = sumsq./trials
    sigma = sqrt.(pbarsq[:,1] .- pbar[:,1].^2)
    @printf("average intercept = %.3f +/- %.4f\n",
        pbar[1,1], pbar[1,2])
    @printf("average slope = %.3f +/- %.4f\n",
        pbar[2,1], pbar[2,2])
    @printf("computed uncertainty in intercept: %.2e\n",
        sigma[1])
    @printf("computed uncertainty in slope: %.2e\n",
        sigma[2])
end
```

This code produced the following output:

```
average intercept = 3.000 +/- 0.0040
average slope = 1.500 +/- 0.0069
computed uncertainty in intercept: 3.94e-03
computed uncertainty in slope: 6.89e-03
```

As you can see, the average of the fit parameters agreed very well with the model. The variance in the fit parameters agreed well with the estimated variances using the formulas in this article.

3.2 Nonlinear Fit

The second fit I tried was a fit to the nonlinear function

$$f(x) = \frac{b_1}{b_2 + cos(2\pi x / b_3)} \tag{28}$$

with $b_1 = 0.5$, $b_2 = 1.35$, and $b_3 = 0.3$. I won't go into the details of the fitting since that's implemented in the LsqFit module which is documented in the linear least squares article[4]. The routine curve_fit returns a structure with the degrees of freedom, the Jacobian, and the mean square error which I will use to estimate the fit error.

Here is the `nlfit` function which did a single nonlinear fit. Because the function was more difficult to fit and nonlinear fitting is an iterative process, this function took a lot longer to run than in the linear case.

```
using LsqFit
using Printf
using LinearAlgebra

function nlfit()
    npts = 10000
    f(x,p) = p[1]./(p[2] .+ cos.(2*pi .* x ./p[3]))
    xpts = [(i-1.0)/(npts-1) for i=1:npts]
    p0 = [0.5,1.35,0.3]
    ypts = f(xpts,p0)+randn(npts)/0.2
    wt = ones(npts)
    cf = curve_fit(f, xpts, ypts, wt, p0)
    sigmay = sqrt.(sum(abs2,cf.resid)/(cf.dof))
    J = cf.jacobian
    cov = inv(J'*J)
    sigmai = sigmay.*sqrt.(diag(cov))
    [cf.param sigmai]
end
```

This function was called in much the same way that `lfit` was called for linear fits. The major difference is that we are fitting three parameters this time instead of two.

```
let
    psum = zeros(3,2)
    sumsq = zeros(3,2)
    trials = 1000
    for i=1:trials
        ft = nlfit()
        psum += ft
        sumsq += ft.*ft
    end
    pbar = psum./trials
    pbarsq = sumsq./trials
    sigma = sqrt.(pbarsq[:,1] .- pbar[:,1].^2)
    @printf("average b1 = %.3f +/- %.4f\n",
        pbar[1,1], pbar[1,2])
    @printf("average b2 = %.3f +/- %.4f\n",
        pbar[2,1], pbar[2,2])
```

```
    @printf("average b3 = %.3f +/- %.4f\n",
            pbar[3,1], pbar[3,2])
    @printf("computed uncertainty in b1: %.2e\n",
        sigma[1])
    @printf("computed uncertainty in b2: %.2e\n",
        sigma[2])
        @printf("computed uncertainty in b3: %.2e\n",
            sigma[3])
end
```

The computed uncertainties were not as close to the average uncertainties in this case as they were with a linear fit using fewer parameters, but they are reasonable.

```
average b1 = 0.508 +/- 0.1154
average b2 = 1.362 +/- 0.1080
average b3 = 0.297 +/- 0.0027
computed uncertainty in b1: 1.24e-01
computed uncertainty in b2: 1.19e-01
computed uncertainty in b3: 4.20e-02
```

References

[1] Philip R. Bevington, D. Keith Robinson, "Data Reduction and Error Analysis for the Physical Sciences," Third Edition, McGraw Hill, 2003.

[2] Wikipedia, "Monotonic least squares," `https://en.wikipedia.org/wiki/Linear_least_squares` (accessed 31 Aug 2018).

[3] Eric W. Weisstein, "Least Squares Fitting," From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/LeastSquaresFitting.html` (accessed 31 Aug 2018).

[4] R. Steven Turley, "Linear Least Squares," BYU, 2018.

[5] R. Steven Turley, "Polynomial Fitting," BYU, 2018.