Nonlinear Least Squares

(Com S 477/577 Notes)

Yan-Bin Jia

Nov 27, 2018

1 Introduction

In least-squares fitting, we fit a model function $F(\boldsymbol{x}; \boldsymbol{c})$, controlled by the parameter vector $\boldsymbol{c} = (c_1, \ldots, c_k)^\top$, to n data points $(\boldsymbol{x}_1, f_1), \ldots, (\boldsymbol{x}_n, f_n)$ of some unknow function f. This is achieved by minimizing the error function

$$\begin{aligned} \omega(\boldsymbol{c}) &= \sum_{i=1}^{n} \Big(f_i - F(\boldsymbol{x}_i; \boldsymbol{c}) \Big)^2 \\ &= (\boldsymbol{f} - \boldsymbol{\alpha})^\top (\boldsymbol{f} - \boldsymbol{\alpha}), \end{aligned} \tag{1}$$

where

$$\boldsymbol{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \quad \text{and} \quad \boldsymbol{\alpha}(\boldsymbol{c}) = \begin{pmatrix} F(\boldsymbol{x}_1; \boldsymbol{c}) \\ \vdots \\ F(\boldsymbol{x}_n; \boldsymbol{c}) \end{pmatrix}.$$

Linear least squares use a model function in the form of $F = c_1\phi_1(\boldsymbol{x}) + \cdots + c_k\phi_k(\boldsymbol{x})$, where $\phi_1(\boldsymbol{x}), \ldots, \phi_k(\boldsymbol{x})$ are basis functions.

It is more often that the model function assumes a nonlinear form. For example, to estimate the location of a target point $\boldsymbol{x}$, we take range measurements at n known locations $\boldsymbol{p}_i$, $1 \le i \le n$, to obtain its distances from these locations:

$$d_i = \|\boldsymbol{x} - \boldsymbol{p}_i\| + \nu_i,$$

where ν_is are the measurement noises. Then, we use least squares to estimate $\boldsymbol{x}$ as the location which minimizes the error

$$\sum_{i=1}^{n} \Big(\|\boldsymbol{x} - \boldsymbol{p}_i\| - d_i \Big)^2.$$

GPS works in the above manner.

When the model function is nonlinear in the parameters, we need to find their values numerically, starting with an initial guess $\boldsymbol{c}^{(0)}$. At each step, the parameter vector $\boldsymbol{c}^{(i)}$ is replaced by a new estimate $\boldsymbol{c}^{(i+1)} = \boldsymbol{c}^{(i)} + \boldsymbol{\delta}$, where $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_k)^\top$. If we use the gradient descent method, then $\boldsymbol{\delta}$ should be along the negative gradient $\nabla\omega$, equivalently, along

$$\boldsymbol{\delta}_g \equiv J^\top (\boldsymbol{f} - \boldsymbol{\alpha}). \tag{2}$$

where J is the Jacobian of $\boldsymbol{\alpha}$ as an $n \times k$ matrix given below:

$$J = \frac{\partial \boldsymbol{\alpha}}{\partial \boldsymbol{c}} = \begin{pmatrix} \frac{\partial F}{\partial c_1}(\boldsymbol{x}_1, \boldsymbol{c}) & \cdots & \frac{\partial F}{\partial c_k}(\boldsymbol{x}_1, \boldsymbol{c}) \\ \vdots & \ddots & \vdots \\ \frac{\partial F}{\partial c_1}(\boldsymbol{x}_n, \boldsymbol{c}) & \cdots & \frac{\partial F}{\partial c_k}(\boldsymbol{x}_n, \boldsymbol{c}) \end{pmatrix}. \tag{3}$$

Gradient descent has slow convergence since every two adjacent moving directions are perpendicular to each other, which results in the second movement undoing part of the progress made by the first movement. The step size also has to be carefully controlled, otherwise failure to converge may happen frequently.

2 The Gauss-Newton Method

The Gauss-Newton method [3] linearizes the model function $F(\boldsymbol{x}; \boldsymbol{c})$ using the Taylor series, for $i = 1, \ldots, n$,

$$\begin{aligned} F(\boldsymbol{x}_i; \boldsymbol{c} + \boldsymbol{\delta}) &\approx F(\boldsymbol{x}_i; \boldsymbol{c}) + \sum_{j=1}^{k} \frac{\partial F}{\partial c_j}(\boldsymbol{x}_i; \boldsymbol{c}) \cdot \delta_j \\ &= F(\boldsymbol{x}_i; \boldsymbol{c}) + \left(\frac{\partial F}{\partial c_1}, \ldots, \frac{\partial F}{\partial c_k} \right) \Big|_{(\boldsymbol{x}_i, \boldsymbol{c})} \boldsymbol{\delta}. \end{aligned}$$

Let us gather all k equations into one:

$$\begin{aligned} \boldsymbol{\alpha}(\boldsymbol{c} + \boldsymbol{\delta}) &= \begin{pmatrix} F(\boldsymbol{x}_1; \boldsymbol{c}) \\ \vdots \\ F(\boldsymbol{x}_n; \boldsymbol{c}) \end{pmatrix} + J\boldsymbol{\delta} \\ &= \boldsymbol{\alpha}(\boldsymbol{c}) + J\boldsymbol{\delta}. \end{aligned} \tag{4}$$

Now we approximate the least-squares error (1) as the parameter vector changes its value from $\boldsymbol{c}$ to $\boldsymbol{c} + \boldsymbol{\delta}$:

$$\begin{aligned} \omega(\boldsymbol{c} + \boldsymbol{\delta}) &= \Big(\boldsymbol{f} - \boldsymbol{\alpha}(\boldsymbol{c} + \boldsymbol{\delta}) \Big)^\top \Big(\boldsymbol{f} - \boldsymbol{\alpha}(\boldsymbol{c} + \boldsymbol{\delta}) \Big) \\ &\approx (\boldsymbol{f} - \boldsymbol{\alpha}(\boldsymbol{c}) - J\boldsymbol{\delta})^\top (\boldsymbol{f} - \boldsymbol{\alpha}(\boldsymbol{c}) - J\boldsymbol{\delta}). \end{aligned}$$

The increment $\boldsymbol{\delta}$ is found by setting the partial derivative of the above with respect to $\boldsymbol{\delta}$ to zero, namely,

$$-2J^\top (\boldsymbol{f} - \boldsymbol{\alpha} - J\boldsymbol{\delta}) = \mathbf{0}.$$

Thus, we obtain

$$\boldsymbol{\delta} = \boldsymbol{\delta}_t \equiv (J^\top J)^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}), \tag{5}$$

where in the above the positive semidefinite $k \times k$ matrix $J^\top J$ is invertible provided that all the columns of J are linearly independent. For later use we rewrite the above equation as

$$J^\top J \boldsymbol{\delta}_t = J^\top (\boldsymbol{f} - \boldsymbol{\alpha}). \tag{6}$$

In practice, $\boldsymbol{c}$ is corrected by a fraction of $\boldsymbol{\delta}_t$ so as not to step out of the neighborhood within which the approximation (4) is good enough not to cause divergence. Failure to converge has been found to be not common with a step size $s\boldsymbol{\delta}_t$, $0 < s \leq 1$.

3 The Levenberg-Marquardt Method

Marquardt observed that $\boldsymbol{\delta}_t$ given in (5) is almost $\frac{\pi}{2}$ away from $\boldsymbol{\delta}_g$ given in (2) in most problems because of the severe elongation of the surface $\omega(\boldsymbol{c})$. This could make the Gauss-Newton method still very slow. He realized that any improved method would in some sense interpolate between $\boldsymbol{\delta}_t$ and $\boldsymbol{\delta}_g$. At a suggestion from Levenberg [6] to minimize ω locally, Marquardt [7] first modified the correction $\boldsymbol{\delta}_t$ and use instead

$$\boldsymbol{\delta}_l = \left(J^\top J + \lambda I_k\right)^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}), \tag{7}$$

where I_k is the $k \times k$ identity matrix. Note that the matrix $J^\top J + \lambda I_k$ is positive definite for a positive λ, which is a damping factor. The iteration formula is thus

$$\boldsymbol{c}^{(i)} = \boldsymbol{c}^{(i-1)} + \left(J^\top J + \lambda I_k\right)^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}), \tag{8}$$

where J and $\boldsymbol{\alpha}$ are evaluated at $\boldsymbol{c}^{(i-1)}$.

To provide theoretical basis, Marquardt established the following properties:

i. $\omega(\boldsymbol{c} + \boldsymbol{\delta}_l) \leq \omega(\boldsymbol{c} + \boldsymbol{\delta})$ for all $\boldsymbol{\delta}$ satisfying $\|\boldsymbol{\delta}\| = \|\boldsymbol{\delta}_l\|$;

ii. $\|\boldsymbol{\delta}_l(\lambda)\|$ decreases to zero monotonically as $\lambda \to \infty$;

iii. $\boldsymbol{\delta}_l$ rotates from $\boldsymbol{\delta}_t$ to $\boldsymbol{\delta}_g$ monotonically as $\lambda \to \infty$.

The second insight Marquardt had was to replace the identity matrix I_k in (7) with a diagonal matrix $D = \mathrm{diag}(d_1, \ldots, d_k)$, where $d_1, \ldots d_k$ are the diagonal elements of $J^\top J$. This scaling is adapted to the curvature of the fitting error $\omega(\boldsymbol{c})$, so there is larger movement along the direction if the gradient is smaller, and it also makes λ dimensionless. More specifically, we denote $E = \mathrm{diag}(\sqrt{d_1}, \ldots, \sqrt{d_k})$ and introduce

$$\begin{aligned} K &= E^{-1} J^\top J E^{-1}, \\ \boldsymbol{g} &= E^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}), \\ \boldsymbol{\sigma}_t &= E \boldsymbol{\delta}_t, \end{aligned}$$

and then transform (6) into

$$K \boldsymbol{\sigma}_t = \boldsymbol{g}.$$

In parallel to the change from (5) to (7), we modify the above equation to obtain a scaled correction:

$$\boldsymbol{\sigma}_m = (K + \lambda I_k)^{-1} \boldsymbol{g}.$$

Then, the increment to $\boldsymbol{c}$ is

$$\begin{aligned} \boldsymbol{\delta}_m &= E^{-1} \boldsymbol{\sigma}_m \\ &= E^{-1} \left(E^{-1} J^\top J E^{-1} + \lambda I_k\right)^{-1} E^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}) \\ &= \left(J^\top J + \lambda D\right)^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}). \end{aligned} \tag{9}$$

This gives rise to the following iteration formula:

$$\boldsymbol{c}^{(i)} = \boldsymbol{c}^{(i-1)} + \Big(J^\top J + \lambda D\Big)^{-1} J^\top (\boldsymbol{f} - \boldsymbol{\alpha}). \tag{10}$$

A sufficiently large λ will rotate $\boldsymbol{\delta}_m$ towards the gradient $\boldsymbol{\delta}_g$ enough to decrease the error ω. However, a value of λ too large would make the method work like steepest descent, thereby inherit its property of rapid initial progress followed by increasingly slower progress. Such a strategy would be a poor one globally. If λ is too big, $\boldsymbol{c}^{(i)}$ is too close to $\boldsymbol{c}^{(i-1)}$ to make fast progress. And if λ is too small, $\boldsymbol{c}^{(i)}$ is too far from $\boldsymbol{c}^{(i-1)}$, and linearization yields a poor approximation. The value of λ needs to be chosen small so minimization of ω is done in the maximum neighborhood over which the linearization (4) is adequate. Based on these considerations, Marquardt proposed an iterative algorithm below. Denote by $\lambda^{(i)}$, $\boldsymbol{c}^{(i)}$, and $\omega^{(i)}$ the values of λ, $\boldsymbol{c}$, and ω at the end of the ith iteration.

1. Let $\nu > 1$, and initialize $\lambda^{(0)} = 10^{-2}$.
2. Set λ to $\lambda^{(i-1)}/\nu$ and $\lambda^{(i-1)}$, respectively. Obtain two values of $\boldsymbol{\delta}_m$ according to (9), and compute $\omega(\boldsymbol{c}^{(i-1)} + \boldsymbol{\delta}_m)$.
3. Let ω_1 and ω_2 be the computed values of ω using $\lambda^{(i-1)}/\nu$ and $\lambda^{(i-1)}$, respectively.
   (a) If $\omega_1 \le \omega^{(i-1)}$, Set $\lambda^{(i)} \leftarrow \lambda^{(i-1)}/\nu$ and $\omega^{(i)} \leftarrow \omega_1$.
   (b) If $\omega_1 > \omega^{(i-1)}$ but $\omega_2 \le \omega^{(i-1)}$, set $\lambda^{(i)} \leftarrow \lambda^{(i-1)}$ and $\omega^{(i)} \leftarrow \omega_2$.
   (c) If $\omega_1 > \omega^{(i-1)}$ and $\omega_2 > \omega^{(i-1)}$, start with $\lambda \leftarrow \lambda^{(i-1)}$ and repeatedly set $\lambda \leftarrow \lambda\nu$ until $\omega(\boldsymbol{c}^{(i-1)} + \boldsymbol{\delta}_m) \le \omega^{(i-1)}$. Set $\lambda^{(i)}$ and $\omega^{(i)}$ to the ending values of λ and ω.
4. Set $\boldsymbol{c}^{(i)} \leftarrow \boldsymbol{c}^{(i-1)} + \boldsymbol{\delta}_m$, where $\boldsymbol{\delta}_m$ is evaluated from (9) with $\lambda = \lambda^{(i)}$.
5. Set $i \leftarrow i + 1$. Repeat steps 2 through 4.

The Levenberg-Marquardt method can be applied to other types of optimization and root finding problems. For example, suppose we want to solve a system of equations

$$f_i(\boldsymbol{x}) = 0, \qquad i = 1, \ldots, k.$$

We can turn it into a least-squares problem to minimize the error

$$\omega = \sum_{i=1}^{k} f_i^2(\boldsymbol{x}),$$

and apply the Levenberg-Marquardt method subsequently.[1]

[1] Mathematica uses the Levenberg-Marquardt method in its command `FindMinimum` when given the `Method -> LevenbergMarquardt` option.

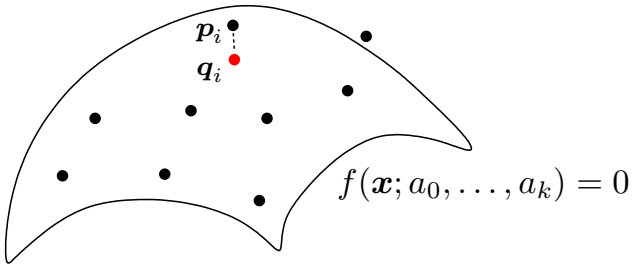

Figure 1: Fitting a surface over points.

4 Curve and Surface Fitting

In computer vision, robotics, and geometric modeling, it is often important to derive some implicit representation of an object from raw shape data obtained using an imaging or touch sensing device. Such an implicit representation often takes the form of the zero set of a polynomial of even degree.[2] In practice, quartics or superquartics [1] have often been the choices for describing surfaces of moderate geometric complexities.

An implicit representation in polynomial form has several advantages. First, it gives us a succinct shape description with few parameters so storage of a large amount of raw data is no longer necessary. Second, using the polynomial we can compute some algebraic invariants (which are expressions in terms of the polynomial's coefficients) to recognize the object from a set of known models. Third, we would be able to differentiate the shape to obtain local geometry indices such as curvature and torsion. Fourth, evaluating the polynomial at a point would easily tell us whether the point is inside, outside, or on the surface. Finally, a polynomial fit handles noisy data effectively in the sense that most shape irregularities due to noise are filtered out.

In surface fitting, a family of quartic polynomials with stably bounded zero sets is often considered. Choice of the family takes into account factors such as the effectiveness of representation and the computational efficiency of fitting. Polynomial coefficients are the parameters to be determined by, say, a least-squares method. For example, we may minimize the sum of squares of the distances from individual data points to the polynomial surface.

Suppose we need to fit a curve or surface over n data points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ in $\mathbb{R}^2$ or $\mathbb{R}^3$. The family of polynomial curves or surfaces that we consider assumes the form

$$f(\boldsymbol{x}; a_0, \ldots, a_k) = 0, \tag{11}$$

where $\boldsymbol{x} = (x, y)$ or (x, y, z), and $a_0, \ldots, a_k$ are the coefficients. Denote by d_i the distance from $\boldsymbol{p}_i$ to the surface (11), for $i = 1, ..., n$. Then the fitting problem can be formulated in a least-squares fashion as

$$\min_{a_0,\ldots,a_k} \sum_{i=0}^{k} d_i^2. \tag{12}$$

The issue, of course, is that we cannot determine d_i, $1 \le i \le n$, until $a_0, \ldots, a_k$ are known, which conversely depends on the knowledge of d_i according to (12). To get out of this chicken-and-egg situation, let us approximate d_i as follows. Denote by $\boldsymbol{q}_i$ the closest point to $\boldsymbol{p}_i$ on the surface to

[2]It is known that the zero sets of polynomials of odd degrees always define unbounded curves and surfaces.

be determined. Obviously, $f(\boldsymbol{q}_i; a_0, \ldots, a_k) = 0$. The best fitting f will place $\boldsymbol{q}_i$ close enough to $\boldsymbol{p}$. So we can approximate the value of f at $\boldsymbol{q}_i$ using Taylor's series at $\boldsymbol{p}_i$, discarding all terms of the second order and above:

$$f(\boldsymbol{q}_i; a_0, \ldots, a_k) \approx f(\boldsymbol{p}_i; a_0, \ldots, a_k) + \nabla f(\boldsymbol{p}_i; a_0, \ldots, a_k)^\top (\boldsymbol{q}_i - \boldsymbol{p}_i).$$

Since the left hand side of the above is zero, we have

$$\nabla f(\boldsymbol{p}_i; a_0, \ldots, a_k)^\top (\boldsymbol{q}_i - \boldsymbol{p}_i) \approx -f(\boldsymbol{p}_i; a_0, \ldots, a_k).$$

Because $\boldsymbol{q}_i$ and $\boldsymbol{p}_i$ are very close to each other, the vector $\boldsymbol{q}_i - \boldsymbol{p}_i$ and the gradient $\nabla f(\boldsymbol{p}_i; a_0, \ldots, a_k)$ are nearly parallel. The above equation yields an approximation:

$$d_i = \|\boldsymbol{q}_i - \boldsymbol{p}_i\| \approx \frac{|f(\boldsymbol{p}_i; a_0, \ldots, a_k)|}{\|\nabla f(\boldsymbol{p}_i; a_0, \ldots, a_k)\|}.$$

With a substitution of the above approximation into (12), we reformulate the fitting problem as

$$\min_{a_0, \ldots, a_k} \frac{f^2(\boldsymbol{p}_i; a_0, \ldots, a_k)}{\|\nabla f(\boldsymbol{p}_i; a_0, \ldots, a_k)\|^2}.$$

The reader is referred to [2, 5, 8] for more on surface fitting techniques and experiments.

5 Surface Patch Reconstruction

A robotic hand can reconstruct an unknown surface patch by touch [4]. The idea is to track along three concurrent[3] curves on the surface, obtaining tactile data points (x_k, y_k, z_k), $1 \le k \le n$, in the meantime. Each curve is the intersection of the patch with a separate plane (called the sampling plane) within which the tracking motion is presently constrained. Denote by $\boldsymbol{p}$ the intersection point of the three curves. Through fitting a parabola to the data points along each curve, its curvature at $\boldsymbol{p}$ is estimated. The surface's normal curvature at $\boldsymbol{p}$ in the tangent direction of this curve is the product of the (estimated) curvature with the cosine of the angle between the sampling plane and the normal plane containing the tangent direction.

Three normal curvatures are thus obtained. From them and the relative orientations of the corresponding tangent vectors at $\boldsymbol{p}$, we solve for the principal curvatures κ_1 and κ_2 and the Darboux frame at the point $\boldsymbol{p}$. Under this frame, the surface patch locally takes the form

$$z(x, y) = \frac{1}{2}\Big(\kappa_1 x^2 + \kappa_2 y^2\Big) + \sum_{3 \le i+j \le d} a_{ij} x^i y^j, \tag{13}$$

where the terms of degrees above two are added to describe a larger area of the surface. The coefficients of the polynomial, gathered into a vector $\boldsymbol{a}$, are determined in a least-squares sense

$$\min_{\boldsymbol{a}} f(\boldsymbol{a}) \tag{14}$$

where

$$f(\boldsymbol{a}) = \frac{1}{n} \sum_{k=1}^{n} (z(x_k, y_k) - z_k)^2. \tag{15}$$

[3] intersecting at one point

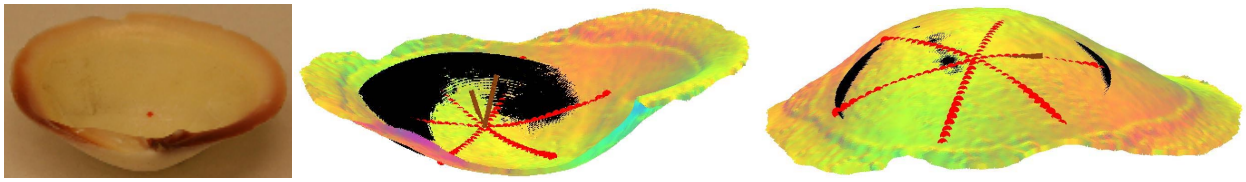

Figure 2: Reconstruction of the surface of a shell from three lines of data points

However, the resulting surface described by $z(x, y)$ may "fold" many times unrealistically, as indicated by a high total absolute Gaussian curvature:

$$g(\boldsymbol{a}) = \iint_D |K(x, y)| \cdot \sqrt{1 + z_x^2 + z_y^2}\, dxdy. \tag{16}$$

To get rid of unnecessary "folds", we minimize the total absolute Gaussian curvature subject to the constraint that the surface fit should "pass through" the sampled data points:

$$\min_{\boldsymbol{a}} g(\boldsymbol{a}) \qquad \text{subject to } f(\boldsymbol{a}) = \mathbf{0}.$$

The technique of Lagrange multipliers is then applied.

Figure 2 displays a shell and two views (inside-out and outside-in) of the reconstructed patch[4] overlaid onto its mesh model generated by NextEngine's desktop 3-D scanner (accuracy 0.127mm). The data points from tracking are shown in red while the estimated Darboux frame in brown. The average distance of the mesh vertices in the overlay area to the patch is 0.2283mm, indicating a close match.

References

[1] A. H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[2] M. M. Blane, Z. Lei, H. Civi, D. B. Cooper. The 3L algorithm for fitting implicit polynomial curves and surfaces to data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):298–313, 2000.

[3] H. O. Hartley. The modified G-N method for the fitting of nonlinear regression functions by least squares. *Technometrics*, 3(2):269–280, 1961.

[4] Y.-B. Jia and J. Tian. Surface patch reconstruction from "one-dimensional" tactile data. *IEEE Transactions on Automation Science and Engineering*, 7(2):400–407, 2010.

[5] D. Keren and D. Cooper. Describing complicated objects by implicit polynomials. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1):38–53, 1994.

[6] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quaterly of Applied Mathematics*, 2:164–168, 1944.

[4] In the reconstruction, d was set to be 4.

[7] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[8] G. Taubin, F. Cukierman, S. Sullivan, J. Ponce, and D. J. Kriegman. Parametrized families of polynomials for bounded algebraic curve and surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3):287–303, 1994.