Practical Optimization Algorithms and Applications

Chapter VIII: Least-Squares Problems

Lingfeng NIU

Research Center on Fictitious Economy & Data Science,
University of Chinese Academy of Sciences

niulf@ucas.ac.cn

Outline

Outline

Introduction

In least-squares problems, the objective function f of the least-squares problems has the following special form:

$$f(x) = \frac{1}{2} \sum_{j=1}^{m} r_j^2(x), \tag{1}$$

where each r_j is a smooth function from $\Re^n$ to $\Re$. We refer to each r_j as a *residual*, and we assume that $m \geq n$ in this chapter.

Introduction

In least-squares problems, the objective function f of the least-squares problems has the following special form:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x), \tag{1}$$

where each r_j is a smooth function from $\Re^n$ to $\Re$. We refer to each r_j as a *residual*, and we assume that $m \geq n$ in this chapter.

Assemble the individual components r_j from (1) into a residual vector $r : \Re^n \to \Re$ defined by

$$r(x) = (r_1(x), r_2(x), \cdots, r_m(x))^T. \tag{2}$$

Using this notation, we can rewrite f as $f(x) = \frac{1}{2}\|r(x)\|_2^2$.

An Example

Least-square problems arise in many areas of applications, and may in fact the largest source of unconstrained optimization problems. Almost anyone who formulates a parametrized model for a chemical, physical, financial, or economic application uses a function of the form (1) to measure the discrepancy between the model and the output of the system at various observation points. By minimizing this function, they select values for the parameters that best match the model to the data.

We would like to study the effect of a certain medication on a patient. We draw blood samples at certain times after the patient takes a dose, and measure the concentration of the medication in each sample, tabulating the time t_j and concentration y_j for each sample.

Based on our previous experience in such experiments, we find that the following model function

$$\phi(x;t) = x_1 + tx_2 + t^2 x_3 + x_4 e^{-x_5 t} \tag{3}$$

provides a good prediction on the concentration at time t, for appropriate value of the five-dimensional parameter vector $x = (x_1, \cdots, x_5)$.

An Example

A good way to measure the difference between the predicted model values and the observations is the following least-squares function:

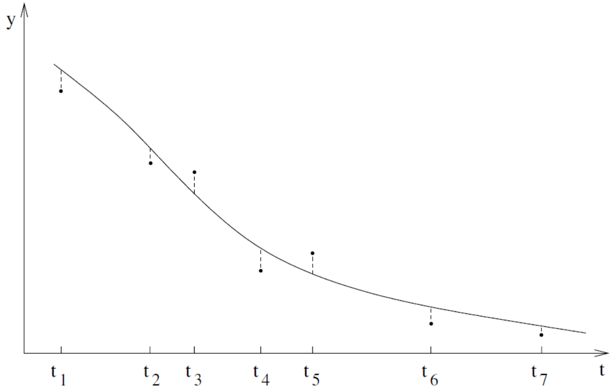
$$\frac{1}{2}\sum_{j=1}^{m}[\phi(x;t_j)-y_j]^2, \tag{4}$$

which sums the squares of the discrepancies between predictions and observations at each t_j. This function has precisely the form (1) if we define

$$r_j(x)=\phi(x;t_j)-y_j.$$

An Example

Graphically, each term in (4) represents the square of the vertical distance between the curve $\phi(x; t)$ (plotted as a function of t) and the point (t_j, y_j).



An Example

We choose the minimizer x^* of the least-squares problem as the best estimate of the parameters, and use $\phi(x^*; t)$ to estimate the concentration remaining in the patient's bloodstream at any time t.

An Example

We choose the minimizer x^* of the least-squares problem as the best estimate of the parameters, and use $\phi(x^*; t)$ to estimate the concentration remaining in the patient's bloodstream at any time t.

- The ordinate t in the model $\phi(x; t)$ could be a vector instead of a scalar.
- The sum-of-squares function (4) is not the only way of measuring the discrepancy between the model and the observations. Instead, we could use a sum of absolute values

$$\frac{1}{2}\sum_{j=1}^{m} |\phi(x; t_j) - y_j|, \tag{5}$$

or even some more complicated function.

Notations

The derivatives of $f(x)$ can be expressed in terms of the Jacobian of r, which is the $m \times n$ matrix of first partial derivatives defined by

$$J(x) = [\frac{\partial r_i}{\partial x_j}]_{i\in\{1,\cdots,m\},j\in\{1,\cdots,n\}} = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}. \tag{6}$$

where each $\nabla r_j(x), j = 1, 2, \cdots, m$ is the gradient of r_j. The gradient and Hessian of f can then be expressed as follows:

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x)\nabla r_j(x) = J(x)^T r(x), \tag{7a}$$

$$\begin{aligned} \nabla^2 f(x) &= \sum_{j=1}^{m} \nabla r_j(x)\nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x) \\ &= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x). \end{aligned} \tag{7b}$$

Outline

Linear Least-Squares Problems

Many models in data-fitting problems are linear functions of x. In this case, the residuals $r_j(x)$ also are linear and the problem of minimizing (1) is called a *linear least-squares problem*. We can write the residual vector as $r(x) = Jx - y$ for some matrix J and vector y, both independent of x, so that the objective is

$$f(x) = \frac{1}{2}\|Jx - y\|_2^2 \tag{8}$$

where $y = -r(0)$. We also have

$$\nabla f(x) = J^T(Jx - y), \qquad \nabla^2 f(x) = J^T J.$$

(Note that the second term in $\nabla^2 f(x)$ disappears, because $\nabla^2 r_i = 0$ for all i.) The function $f(x)$ is convex - a property that does not necessarily hold for the nonlinear problem (1). By setting $\nabla f(x^*) = 0$, we see that any solution x^* of (8) must satisfy

$$J^T J x^* = J^T y. \tag{9}$$

This equation are called the *normal equations* for (8). Algorithms for linear least-squares problems fall under the aegis of numerical linear algebra rather than optimization. We assume for the present that $m \geq n$ and that J has full column rank.

Linear Least-Squares Problems

The first and most obvious algorithm is simply to form and solve the system (9) by the following three-step procedure:

- compute the coefficient matrix $J^T J$ and the right-hand-side $J^T y$;
- compute the Cholesky factorization of the symmetric matrix $J^T J$;
- perform two triangular substitutions with the Cholesky factors to recover the solution x^*.

Linear Least-Squares Problems

The first and most obvious algorithm is simply to form and solve the system (9) by the following three-step procedure:

- compute the coefficient matrix $J^T J$ and the right-hand-side $J^T y$;
- compute the Cholesky factorization of the symmetric matrix $J^T J$;
- perform two triangular substitutions with the Cholesky factors to recover the solution x^*.

This method is frequently used in practice and is often effective, but it has one significant disadvantage, namely, that the condition number of $J^T J$ is the square of the condition number of J. Since the relative error in the computed solution of any problem is usually proportional to the condition number, the solution obtained by the Cholesky-based method may be much less accurate than solutions obtained from algorithms that work directly with the formulation (8). When J is ill conditioned, the Cholesky factorization process may even break down, since roundoff errors may cause small negative elements to appear on the diagonal.

Linear Least-Squares Problems

A second approach is based on a QR factorization of the matrix J. Since

$$\|Jx - y\|_2 = \|Q^T(Jx - y)\|_2 \tag{10}$$

for any $m \times m$ orthogonal matrix Q. Suppose we perform a QR factorization with column pivoting on the matrix J to obtain

$$J\Pi = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1 \;\; Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R, \tag{11}$$

where Π is an $n \times n$ permutation matrix, Q is $m \times m$ is the first n columns of Q, while Q_2 contains the last $m - n$ columns; R is $n \times n$ upper triangular. By combining (10) and (11), we obtain

$$\begin{aligned} \|Jx - y\|_2^2 &= \left\| \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} (J\Pi\Pi^T x - y) \right\|_2^2 \\ &= \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} (\Pi^T x) - \begin{bmatrix} Q_1^T y \\ Q_2^T y \end{bmatrix} \right\|_2^2 \\ &= \|R(\Pi^T x) - Q_1^T y\|_2^2 + \|Q_2^T y\|_2^2. \end{aligned} \tag{12}$$

Linear Least-Squares Problems

No choice of x has any effect on the second term of this last expression, but we can minimize $\|Jx - y\|$ by driving the first term to zero, that is, by setting

$$x^* = \Pi R^{-1} Q_1^T y.$$

In practice, we perform a triangular substitution to solve $Rz = Q_1^T y$, then permute the components of z to get $x^* = \Pi z$.

This QR-based approach does not degrade the conditioning of the problem unnecessarily. The relative error in the final computed solution x^* is usually proportional to the condition number of J, not its square, and this method is usually reliable.

Linear Least-Squares Problems

A third approach, based on the singular-value decomposition (SVD) of J, can be used in the circumstances which call for greater robustness or more information about the sensitivity of the solution to perturbations in the data (J or y).

The SVD of J is given by

$$J = U \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = [U_1 U_2] \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = U_1 S V^T, \tag{13}$$

where U is $m \times m$ orthogonal; U_1 contains the first n columns of U, U_2 the last $m - n$ columns; V is $n \times n$ orthogonal; S is $n \times n$ diagonal, with diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$. At last, we can obtain

$$x^* = \sum_{i=1}^{n} \frac{u_i^T y}{\sigma_i} v_i. \tag{14}$$

This formula yields useful information about the sensitivity of x^*. When σ_i is small, x^* is particularly sensitive to perturbations in y that affect $u_i^T y$, and also to perturbations in J that affect this same quantity. Information such as this is particularly useful when J is nearly rank-deficient, that is, when $\sigma_n/\sigma_1 \ll 1$.

Linear Least-Squares Problems

All three approaches above have their place.

- The Cholesky-based algorithm is particularly useful when $m \gg n$ and it is practical to store $J^T J$ but not J itself.
- The QR approach avoids squaring of the condition number and hence may be more numerically robust.
- The SVD approach is the most robust and reliable one for ill-conditioned problems, while the most expensive.

When the problem is very large scale, it may be efficient to use iterative techniques, such as the conjugate gradient method, to solve the normal equations (9).

Outline

The Gauss-Newton Method

The simplest method for nonlinear squares problem is Gauss-Newton method, which can be viewed as a modified Newton's method with line search.

Instead of solving the standard Newton equations $\nabla^2 f(x_k)p = -\nabla f(x_k)$, we exclude the second-order term from $\nabla^2 f(x_k)$ and obtain p_k^{GN} by solving

$$J_k^T J_k p_k^{GN} = -J_k^T r_k. \tag{15}$$

P_k^{GN} is in fact the solution of the linear least-squares problem

$$\min_p \frac{1}{2}\|J_k p + r_k\|^2. \tag{16}$$

Hence, we can find the search direction by applying the linear least-squares algorithms.

The Advantage of the Gauss-Newton Method

The simple modification in Gauss-Newton method gives a surprising number of advantages over the plain Newton's method.

- The use of the approximation $\nabla^2 f_k \approx J_k^T J_k$ saves us the trouble of computing individual Hessians $\nabla^2 r_i$, $i = 1, \cdots, m$ of the residuals.

The Advantage of the Gauss-Newton Method

The simple modification in Gauss-Newton method gives a surprising number of advantages over the plain Newton's method.

- The use of the approximation $\nabla^2 f_k \approx J_k^T J_k$ saves us the trouble of computing individual Hessians $\nabla^2 r_i$, $i = 1, \cdots, m$ of the residuals.
- There are many interesting situations in which the first term $J^T J$ in (7b) dominant the second term(at least close to the solution x^*), so that the convergence rate of Gauss-Newton is similar to that of Newton's method.

The Advantage of the Gauss-Newton Method

The simple modification in Gauss-Newton method gives a surprising number of advantages over the plain Newton's method.

- The use of the approximation $\nabla^2 f_k \approx J_k^T J_k$ saves us the trouble of computing individual Hessians $\nabla^2 r_i$, $i = 1, \cdots, m$ of the residuals.
- There are many interesting situations in which the first term $J^T J$ in (7b) dominant the second term(at least close to the solution x^*), so that the convergence rate of Gauss-Newton is similar to that of Newton's method.
- Whenever J_k has full rank and the gradient ∇f_k is nonzero, the direction p_k^{GN} is a descent direction for f, and therefore a suitable direction for a line search.

The Advantage of the Gauss-Newton Method

The simple modification in Gauss-Newton method gives a surprising number of advantages over the plain Newton's method.

- The use of the approximation $\nabla^2 f_k \approx J_k^T J_k$ saves us the trouble of computing individual Hessians $\nabla^2 r_i$, $i = 1, \cdots, m$ of the residuals.
- There are many interesting situations in which the first term $J^T J$ in (7b) dominant the second term(at least close to the solution x^*), so that the convergence rate of Gauss-Newton is similar to that of Newton's method.
- Whenever J_k has full rank and the gradient ∇f_k is nonzero, the direction p_k^{GN} is a descent direction for f, and therefore a suitable direction for a line search.

Convergence of the Gauss-Newton Method

Theorem

Suppose that each residual function r_j is Lipschitz continuously differentiable in a neighborhood $\mathcal{N}$ of the level set $\mathcal{L} = \{x | f(x) \leq f(x_0)\}$, and that the Jacobians $J(x)$ satisfy the uniform full-rank condition, i.e. there is a constant $\gamma > 0$ such that $\|J(x)z\| \geq \|z\|$ on $\mathcal{N}$. Then if the iterates x_k are generated by the Gauss-Newton method with step lengths α_k that satisfy Wolfe conditions, we have

$$\lim_{k \to \infty} J_k^T r_k = 0.$$

The Levenberg-Marquardt Method

The Levenberg-Marquardt method, in fact, is the trust region method which use the approximation $\nabla^2 f_k \approx J_k^T J_k$ for the Hessian. In details. For a spherical trust region, the subproblem to be solved at each iteration is

$$\min_p \frac{1}{2}\|J_k p + r_k\|^2, \text{subject to} \|p\| \leq \Delta_k, \tag{17}$$

where $\Delta_k > 0$ is the trust-region radius. In effect, we are choosing the model function $m_k(\cdot)$ to be

$$m_k(p) = \frac{1}{2}\|r_k\|^2 + p^T J_k^T r_k + \frac{1}{2} p^T J_k^T J_k p. \tag{18}$$

- The second-order Hessian component in (7a) is still ignored, however, so the local convergence properties of the two methods are similar.
- The use of a trust region avoids one of the weaknesses of Gauss-Newton, namely, its behavior when the Jacobian $J(x)$ is rank-deficient, or nearly so.

Convergence of the Levenberg-Marquardt Method

It is not necessary to solve the trust-region problem (17) exactly in order for the Levenberg-Marquardt method to enjoy glocal convergence properties.

Theorem

Let $\eta \in (0, \frac{1}{4})$ in the trust-region algorithm, and suppose that the level set $\mathcal{L} = \{x | f(x) \le f(x_0)\}$ is bounded and that the residual functions $r(\cdot)$, $j = 1, \cdots, m$ are Lipschitz continuously differentiable in a neighborhood $\mathcal{N}$ of $\mathcal{L}$. Assume that for each k, the approximate solution p_k of (17) satisfies the inequality

$$m_k(0) - m_k(p_k) \ge c_1 \|J_k^T r_k\| \min(\Delta_k, \frac{\|J_k^T r_k\|}{\|J_k^T J_k\|}),$$

for some constant $c_1 > 0$, and in addition $\|p_k\| \le \gamma \Delta_k$ for some constant $\gamma \ge 1$. We then have that

$$\lim_{k\to\infty} \nabla f_k = \lim_{k\to\infty} J_k^T r_k = 0.$$

Methods for Large-Residual Problems

In large-residual problems, the quadratic model in (17) is an inadequate representation of the function f, because the second-order part of the Hessian $\nabla^2 f(x)$ is too significant to be ignored.

Methods for Large-Residual Problems

In large-residual problems, the quadratic model in (17) is an inadequate representation of the function f, because the second-order part of the Hessian $\nabla^2 f(x)$ is too significant to be ignored.

On large-residual problems, the asymptotic convergence rate of Gauss-Newton and Levenberg-Marquardt algorithms is only linear-slower than the superlinear convergence attained by algorithms for general unconstrained problems, such as Newton or quasi-Newton.

Methods for Large-Residual Problems

In large-residual problems, the quadratic model in (17) is an inadequate representation of the function f, because the second-order part of the Hessian $\nabla^2 f(x)$ is too significant to be ignored.

On large-residual problems, the asymptotic convergence rate of Gauss-Newton and Levenberg-Marquardt algorithms is only linear-slower than the superlinear convergence attained by algorithms for general unconstrained problems, such as Newton or quasi-Newton.

If the individual Hessians $\nabla^2 r_j$ are easy to calculate, it may be better to ignore the structure of the least-squares objective and apply Newton's method with trust region or line search to the problem of minimizing f. Quasi-Newton methods are another option.

Methods for Large-Residual Problems

In large-residual problems, the quadratic model in (17) is an inadequate representation of the function f, because the second-order part of the Hessian $\nabla^2 f(x)$ is too significant to be ignored.

On large-residual problems, the asymptotic convergence rate of Gauss-Newton and Levenberg-Marquardt algorithms is only linear-slower than the superlinear convergence attained by algorithms for general unconstrained problems, such as Newton or quasi-Newton.

If the individual Hessians $\nabla^2 r_j$ are easy to calculate, it may be better to ignore the structure of the least-squares objective and apply Newton's method with trust region or line search to the problem of minimizing f. Quasi-Newton methods are another option.

Methods for Large-Residual Problems

However, the behavior of both Newton and quasi-Newton on early iterations (before reaching a neighborhood of the solution) may be inferior to Gauss-Newton and Levenberg-Marquardt.

Methods for Large-Residual Problems

However, the behavior of both Newton and quasi-Newton on early iterations (before reaching a neighborhood of the solution) may be inferior to Gauss-Newton and Levenberg-Marquardt.

Of course, we often do not know before hand whether a problem will turn out to have small or large residuals at the solution. It seems reasonable, therefore, to consider *hybrid algorithms*, which would behave like Gauss-Newton or Levenberg-Marquardt if the residuals turn out to be small (and hence take advantage of the cost saving associated with these methods) but switch to Newton or quasi-Newton steps if the residuals at the solution appear to be large. There are a couple of ways to construct hybrid algorithms.

Fletcher and Xu's Methods for Large-Residual Problems

This approach maintains a sequence of positive definite Hessian approximation B_k.

- If the Gauss-Newton step from x_k reduces the function f by a certain fixed amount (say, a factor of 5), then this step is taken and B_k is overwritten by $J_k^T J_k$.
- Otherwise, a direction is computed using B_k, and the new point x_{k+1} is obtained by doing a line search.

In either case, a BFGS-like update is applied to B_k to obtain a new approximation B_{k+1}.

- In the zero-residual case, the method eventually always takes Gauss-Newton steps (giving quadratic convergence),
- while it eventually reduces to BFGS in the nonzero-residual case (giving superlinear convergence).

Numerical results show good results for this approach on small-, large-, and zero-residual problems.

Dennis, Gay, and Welsch's Method

A second way to combine Gauss-Newton and quasi-Newton ideas is to maintain approximations to just the second-order part of the Hessian. That is, we maintain a sequence of matrices S_k that approximate the summation term

$$\sum_{j=1}^{m} r_j(x_k)\nabla^2 r_j(x_k),$$

and then use the overall Hessian approximation

$$B_k = J_k^T J_k + S_k$$

in a trust-region or line search model for calculating the step p_k. Updates to S_k are devised so that the approximate Hessian B_k, or its constituent parts, mimics the behavior of the corresponding exact quantities over the step just taken.

We describe the algorithm of Dennis, Gay, and Welsch, which is probably the best-known algorithm in this class because of its implementation in the well-known NL2SOL package.

Dennis, Gay, and Welsch's Method

Ideally, S_{k+1} should be a close approximation to the exact second-order term at $x = x_{k+1}$, that is,

$$S_{k+1} \approx \sum_{j=1}^{m} r_j(x_{k+1}) \nabla^2 r_j(x_{k+1}).$$

Since we do not want to calculate the individual Hessians $\nabla^2 r_j$ in this formula, we could replace each of them with an approximation $(B_j)_{k+1}$ and impose the condition that $(B_j)_{k+1}$ should mimic the behavior of its exact counterpart $\nabla^2 r_j$ over the step just taken; that is,

$$\begin{aligned} (B_j)_{k+1}(x_{k+1} - x_k) &= \nabla r_j(x_{k+1}) - \nabla r_j(x_k) \\ &= (\text{row } j \text{ of } J(x_{k+1}))^T - (\text{row } j \text{ of } J(x_k))^T \end{aligned} \tag{19}$$

Dennis, Gay, and Welsch's Method

This condition leads to a secant equation on S_{k+1}, namely,

$$\begin{aligned}
S_{k+1}(x_{k+1} - x_k) &= \sum_{j=1}^{m} r_j(x_{k+1})(B_j)_{k+1}(x_{k+1} - x_k) \\
&= \sum_{j=1}^{m} r_j(x_{k+1})[(\text{row } j \text{ of } J(x_{k+1}))^T - (\text{row } j \text{ of } J(x_k))^T] \\
&= J_{k+1}^T r_{k+1} - J_k^T r_{k+1}
\end{aligned}$$

As usual, this condition does not completely specify the new approximation S_{k+1}.

Dennis, Gay, and Welsch's Method

Dennis, Gay, and Welsch add requirements that S_{k+1} be symmetric and that the difference $S_{k+1} - S_k$ from the previous estimate S_k be minimized in a certain sense, and derive the following update formula:

$$S_{k+1} = S_k + \frac{(y^\sharp - S_k s)y^T + y(y^\sharp - S_k s)^T}{y^T s} - \frac{(y^\sharp - S_k s)^T s}{(y^T s)^2} y y^T,$$

where

$$\begin{aligned} s &= x_{k+1} - x_k, \\ y &= J_{k+1}^T r_{k+1} - J_k^T r_k, \\ y^\sharp &= J_{k+1}^T r_{k+1} - J_k^T r_{k+1}. \end{aligned}$$

Note that above formulation is a slight variant on the DFP update for unconstrained minimization. It would be identical if $y^\sharp$ and y were the same.

Outline

Notes and References

Algorithms for linear least Squares are discussed comprehensively in

- A. Brörck, *Numerical Methods for Least Squares Problems*, SIAM Publications, Philadelphia, Penn., 1996.
- Chapter 5 of G.H. Golub and C.F. Van Loan, *Matrix Computations*, The johns Hopkins University Press, Baltimore, third ed., 1996.
- C. L. Lawson and R.J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

Notes and References

Nonlinear least-squares methods:

- Major numerical software libraries such as IMSL, HSL NAG, and SAS, as well as programming environments such as Mathematics and Matlab, contain robust nonlinear least-squares implementation.
- Other high quality implementations include DFNLP, MINPACK, NL2SOL and NLSSOL.
- The nonlinear programming packages LANCELOT, KNITRO, and SNOPT provide large-scale implementations of the Gauss-Newton and Levenberg-Marquardt methods.

Thanks for your attention!