Chapter 8: Least-Squares Problems

Outline

Outline

Formulation

- Least-squares problem:

$$\min \quad f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x), \tag{1.1}$$

  where each r_j is a smooth function from $\mathbb{R}^n$ to $\mathbb{R}$. We refer to each r_j as a *residual*, and we assume that $m \geq n$ in the following.
- Arise in many applications, such as data fitting problems.

Jacobian

- By exploiting the special structure in f and its derivatives, efficient, robust minimization algorithms have been devised.
- Introduce the residual vector $r : \mathbb{R}^n \to \mathbb{R}$ defined by

$$r(x) = (r_1(x), r_2(x), \cdots, r_m(x))^T. \tag{1.2}$$

  Then rewrite f as $f(x) = \frac{1}{2}\|r(x)\|_2^2$.

- Jacobian of r: which is the $m \times n$ matrix defined by

$$J(x) = \left[\frac{\partial r_j}{\partial x_i}\right] = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix} \tag{1.3}$$

Derivatives of f

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

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x)\nabla r_j(x) = J(x)^T r(x), \tag{1.4}$$

$$\begin{aligned} \nabla^2 f(x) &= \sum_{j=1}^{m} \nabla r_j(x)\nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x) \\ &= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x)\nabla^2 r_j(x). \end{aligned} \tag{1.5}$$

- Knowing the Jacobian we can compute the first part of the Hessian $\nabla^2 f(x)$ for free.
- In many cases, the first part of the Hessian $\nabla^2 f(x)$ is more important, either because of near-linearity of the model near the solution (that is, $\nabla^2 r_j$ small) or because of small residuals (that is, r_j small).

Outline

Linear Least-Squares Problems

- Each function r_i is linear, so the Jacobian J is constant. We can write
$$f(x) = \frac{1}{2}\|Jx - y\|_2^2 \tag{2.1}$$
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.)
- $f(x)$ is convex, which does not necessarily hold for the nonlinear problem (1.1).
- By setting $\nabla f(x^*) = 0$, we see that any solution x^* of (2.1) must satisfy the so-called *normal equations*:
$$J^T J x^* = J^T y. \tag{2.2}$$
- Use direct methods or iterative methods to solve (2.2).

Linear Least-Squares Problems

- Algorithms for linear least-squares problems fall under the aegis of numerical linear algebra rather than optimization.
- We outline briefly three major algorithms for the unconstrained linear least-squares problem.
- 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 the Cholesky-based algorithm, namely solving $J^T J x^* = J^T y$ by the following three-step procedure:
  - compute $J^T J$ and $J^T y$;
  - compute the Cholesky factorization of $J^T J$;
  - perform two triangular substitutions with the Cholesky factors to recover the solution x^*.
- However, the solution may be much less accurate, because $\mathrm{cond}(J^T J) = \mathrm{cond}(J)^2$.

Linear Least-Squares Problems

- A second approach is based on a QR factorization of the matrix J.
- Note that
$$\|Jx - y\|_2 = \|Q^T(Jx - y)\|_2 \tag{2.3}$$
for any $m \times m$ orthogonal matrix Q.
- Suppose we perform a QR factorization with column pivoting on the matrix J to obtain
$$J\Pi = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1 \ \ Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R, \tag{2.4}$$
where
  - Π is an $n \times n$ permutation matrix;
  - Q is $m \times m$ orthogonal;
  - Q_1 contains the first n columns of Q, while Q_2 contains the last $m - n$ columns;
  - R is $n \times n$ upper triangular.

Linear Least-Squares Problems

- By combining (2.3) and (2.4), we obtain

$$\begin{aligned}
\|Jx - y\|_2^2 &= \left\| \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} (J\Pi\Pi^T x - y) \right\|_2^2 \\
&= \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} (\Pi^T x) - \begin{bmatrix} Q_1^T y \\ Q_2^T y \end{bmatrix} \right\|_2^2 \\
&= \|R(\Pi^T x) - Q_1^T y\|_2^2 + \|Q_2^T y\|_2^2. \qquad (2.5)
\end{aligned}$$

- Minimizing the first term yields

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

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

Linear Least-Squares Problems

- The SVD of J is given by

$$J = U \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = [U_1 \ \ U_2] \begin{bmatrix} S \\ 0 \end{bmatrix} V^T = U_1 S V^T, \tag{2.6}$$

  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 $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$.

Linear Least-Squares Problems

- Then we obtain

$$\begin{aligned} \|Jx - y\|^2 &= \|U^T(Jx - y)\|^2 \\ &= \left\| \begin{bmatrix} S \\ 0 \end{bmatrix} (V^T x) - \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} y \right\|^2 \\ &= \|S(V^T x) - U_1^T y\|^2 + \|U_2^T y\|^2 \end{aligned}$$

- At last, we can obtain

$$x^* = VS^{-1} U_1^T y = \sum_{i=1}^{n} \frac{u_i^T y}{\sigma_i} v_i. \tag{2.7}$$

- 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. Such information 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 $n \ll m$ 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 CG method, to solve the normal equations (2.2).

Outline

The Gauss-Newton Method

- Nonlinear least-squares problem

$$\min \quad f(x) = \frac{1}{2}\|r(x)\|^2$$

- $\nabla f(x) = J(x)^T r(x)$
- $\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^m r_j(x) \nabla^2 r_j(x)$
- Instead of solving the standard Newton equations $\nabla^2 f(x_k) p = -\nabla f(x_k)$, we obtain the search direction p_k^{GN} by solving

$$J_k^T J_k p_k^{GN} = -J_k^T r_k. \tag{3.1}$$

This simple modification gives a surprising number of advantages over the plain Newton's method.

The Gauss-Newton Method

- Save 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 (1.5) 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 column rank and the gradient $\nabla f_k \neq 0$, p_k^{GN} is a descent direction for f, and therefore a suitable direction for a line search.
- p_k^{GN} solves the linear least-squares subproblem

$$\min_p \frac{1}{2} \| J_k p + r_k \|^2. \tag{3.2}$$

Hence, we can find the search direction by applying the linear least-squares algorithms. If the QR or SVD algorithms are used, there is no need to calculate the Hessian approximation $J_k^T J_k$ explicitly.

The Gauss-Newton Method

- The subproblem (3.2) suggests another motivation for the Gauss-Newton step. Rather than forming a quadratic model of the function $f(x)$, we are forming a linear model of the vector function $r(x+p) \approx r(x) + J(x)p$, substituted into the function $\frac{1}{2}\|\cdot\|^2$.
- In other words, we use the approximation

$$f(x_k + p) = \frac{1}{2}\|r(x_k + p)\|^2 \approx \|J_k p + r_k\|^2,$$

and choose p_k^{GN} to be the minimizer of this approximation.

Convergence of the Gauss-Newton Method

Theorem 1

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 $\|J(x)z\| \geq \gamma \|z\|$ 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

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

Proof Sketch. Since

$$p_k^{GN} = -(J_k^T J_k)^{-1} J_k^T r_k = -(J_k^T J_k)^{-1} \nabla f_k,$$

there exists $\eta > 0$ such that

$$\cos \theta_k = -\frac{(\nabla f_k)^T p_k^{GN}}{\|p_k^{GN}\| \|\nabla f_k\|} = \frac{\|J_k p_k^{GN}\|^2}{\|p_k^{GN}\| \|J_k^T J_k p_k^{GN}\|} \geq \eta.$$

Then according to Zoutendijk condition we have $\nabla f_k \to 0$.

The Levenberg-Marquardt Method

- Recall that the Gauss-Newton method is like Newton's method with line search, except that we use approximation $\nabla^2 f_k \approx J_k^T J_k$.
- The Levenberg-Marquardt method can be derived by replacing the line search strategy with a trust-region strategy.
- 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.
- The second-order Hessian component in (1.4) is still ignored, however, so the local convergence properties of the two methods are similar.

The Levenberg-Marquardt Method

- The Levenberg-Marquardt method can be described and analyzed using the trust-region framework.
- For a spherical trust region, the subproblem to be solved at each iteration is

$$\min_p \frac{1}{2}\|J_k p + r_k\|^2, \text{subject to} \|p\| \le \Delta_k, \tag{3.3}$$

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

$$m_k(p) = \frac{1}{2}\|r_k\|^2 + p^T J_k^T r_k + \frac{1}{2} p^T J_k^T J_k p. \tag{3.4}$$

  We drop the iteration counter k during the rest of this section.

The Levenberg-Marquardt Method

We can characterize the solution of (3.3) in the following theorem:

Theorem 2

The vector p^{LM} is a solution of the trust-region subproblem

$$\min_p \frac{1}{2}\|Jp + r\|^2, \text{subject to} \|p\| \le \Delta, \tag{3.5}$$

if and only if p^{LM} is feasible and there is a scalar $\lambda \ge 0$ such that

$$\begin{aligned} (J^T J + \lambda I)p^{LM} &= -J^T r, && (3.6a) \\ \lambda(\Delta - \|p^{LM}\|) &= 0. && (3.6b) \end{aligned}$$

Implementation of the Levenberg-Marquardt Method

Note that the equations (3.6a) are just the normal equations for the linear least-squares problem

$$\min_p \frac{1}{2} \left\| \begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix} p + \begin{bmatrix} r \\ 0 \end{bmatrix} \right\|^2. \tag{3.7}$$

Just as in the Gauss-Newton case, the equivalence between (3.6a) and (3.7) gives us a way of solving the subproblem without computing the matrix-matrix product $J^T J$ and its Cholesky factorization.

Implementation of the Levenberg-Marquardt Method

Least-squares problems are often poorly scaled. Some of the variables could have values of about 10^4, while other variables could be of order 10^{-6}. If such wide variations are ignored, the algorithms above may encounter numerical difficulties or produce solutions of poor quality. One way to reduce the effects of poor scaling is to use an *ellipsoidal trust region* in place of the spherical trust region defined above. The step is confined to an ellipse in which the lengths of the principal axes are related to the typical values of the corresponding variables.

Implementation of the Levenberg-Marquardt Method

Analytically, the trust-region subproblem becomes

$$\min_p \frac{1}{2}\|J_k p + r_k\|^2, \text{ subject to } \|D_k p\| \le \Delta_k, \tag{3.8}$$

where D_k is a diagonal matrix with positive diagonal entries. Instead of (3.3), the solution of (3.8) satisfies an equation of the form

$$(J_k^T J_k + \lambda D_k^2) p_k^{LM} = -J_k^T r_k, \tag{3.9}$$

and, equivalently, solves the linear least-squares problem

$$\min_p \frac{1}{2}\left\| \begin{bmatrix} J_k \\ \sqrt{\lambda} D_k \end{bmatrix} p + \begin{bmatrix} r_k \\ 0 \end{bmatrix} \right\|^2. \tag{3.10}$$

The diagonals of the scaling matrix D_k can change from iteration to iteration, as we gather information about the typical range of values for each component of x. If the variation in these elements is kept within certain bounds, then the convergence theory for the spherical case continues to hold.

Implementation of the Levenberg-Marquardt Method

For problems in which m and n are large and $J(x)$ is sparse, we may prefer to solve (3.3) or (3.8) approximately using the truncated CG algorithm with $J_k^T J_k$ replacing the exact Hessian $\nabla^2 f_k$. Positive semidefiniteness of the matrix $J_k^T J_k$ makes for some simplification of the trust-region method, because negative curvature cannot rise.

Convergence of the Levenberg-Marquardt Method

It is not necessary to solve the trust-region problem (3.3) exactly in order for the Levenberg-Marquardt method to enjoy glocal convergence properties.

Theorem 3

Let $\eta \in (0, \frac{1}{4})$ in the trust-region algorithm, and suppose that the level set $\mathcal{L} = \{x | f(x) \leq f(x_0)\}$ 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 (3.3) satisfies the inequality

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

for some constant $c_1 > 0$, and in addition $\|p_k\| \leq \gamma \Delta_k$ for some constant $\gamma \geq 1$. 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 (3.3) is an inadequate representation of the function f, because the second-order part of the Hessian $\nabla^2 f(x)$, $\sum_{j=1}^{m} r_j(x_k)\nabla^2 r_j(x_k)$, is too significant to be ignored.
- On large-residual problems, the asymptotic convergence rate of Gauss-Newton and Levenberg-Marquardt algorithms is only linear.
- 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.
- 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.

Methods for Large-Residual Problems

- 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.

Methods for Large-Residual Problems

- 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.
- 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,

$$\begin{aligned} (B_j)_{k+1}(x_{k+1} - x_k) &= \nabla_j(x_{k+1}) - \nabla_j(x_k) \\ &= (\text{row } j \text{ of } J(x_{k+1}))^T - (\text{row } j \text{ of } J(x_k))^T \end{aligned} \tag{3.11}$$

Methods for Large-Residual Problems

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}.

Methods for Large-Residual Problems

Dennis, Gay, and Welsch add requirements that S_{k+1} be symmetric and that the difference of S_{k+1} 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.

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!