Numerical Optimization
Lecture Notes #24
Nonlinear Least Squares — Orthogonal Distance Regression

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Fall 2018
1 Summary
   - Linear Least Squares
   - Nonlinear Least Squares

2 Orthogonal Distance Regression
   - Error Models
   - Weighted Least Squares / Orthogonal Distance Regression
   - ODR = Nonlinear Least Squares, Exploiting Structure
Our study of non-linear least squares problems started with a look at linear least squares, where each residual $r_j(\bar{x})$ is linear, and the Jacobian therefore is constant. The objective of interest is

$$f(\bar{x}) = \frac{1}{2} \| J\bar{x} + \bar{r}_0 \|_2^2, \quad \bar{r}_0 = \bar{r}(0),$$

solving for the stationary point $\nabla f(\bar{x}^*) = 0$ gives the normal equations

$$J^T J\bar{x}^* = -J^T \bar{r}_0.$$

We have three approaches to solving the normal equations for $\bar{x}^*$ — in increasing order of computational complexity and stability:

(i) Cholesky factorization of $J^T J$,
(ii) QR-factorization of $J$, and
(iii) Singular Value Decomposition of $J$. 

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩
Problem: Nonlinear Least Squares

\[
\bar{x}^* = \text{arg min}_{\bar{x} \in \mathbb{R}^n} \left[ f(\bar{x}) \right] = \text{arg min}_{\bar{x} \in \mathbb{R}^n} \left[ \frac{1}{2} \sum_{j=1}^{m} r_j(\bar{x})^2 \right], \quad m \geq n,
\]

where the residuals \( r_j(\bar{x}) \) are of the form \( r_j(\bar{x}) = y_j - \Phi(\bar{x}; t_j) \). Here, \( y_j \) are the measurements taken at the locations/times \( t_j \), and \( \Phi(\bar{x}; t_j) \) is our model.

The key approximation for the Hessian

\[
\nabla^2 f(\bar{x}) = J(\bar{x})^T J(\bar{x}) + \sum_{j=1}^{m} r_j(\bar{x}) \nabla^2 r_j(\bar{x}) \approx J(\bar{x})^T J(\bar{x}).
\]
Line-search algorithm: **Gauss-Newton**, with the subproblem:

\[
J(\bar{x}_k)^T J(\bar{x}_k) \bar{p}^\text{GN}_k = -\nabla f(\bar{x}_k).
\]

Guaranteed descent direction, fast convergence (as long as the Hessian approximation holds up) **equivalence** to a linear least squares problem (used for efficient, stable solution).

---

Peter Blomgren, ⟨blomgren.peter@gmail.com⟩
**Trust-region algorithm:** Levenberg-Marquardt, with the subproblem:

\[
\tilde{p}_{LM}^k = \arg \min_{\tilde{p} \in \mathbb{R}^n} \frac{1}{2} \| J(\tilde{x}_k)\tilde{p} + \tilde{r}_k \|_2^2, \quad \text{subject to } \| \tilde{p} \| \leq \Delta_k.
\]

Slight advantage over Gauss-Newton (global convergence), same local convergence properties; also (locally) equivalent to a linear least squares problem.
Hybrid Algorithms:

• When implementing Gauss-Newton or Levenberg-Marquardt, we should implement a **safe-guard** for the large residual case, where the Hessian approximation fails.

• If, after some reasonable number of iterations, we realize that the residuals are **not** going to zero, then we are better off switching to a general-purpose algorithm for non-linear optimization, such as a quasi-Newton (BFGS), or Newton method.
So far we have assumed that there are no errors in the variables describing where / when the measurements are made, i.e. in the data set \{t_j, y_j\} where \(t_j\) denote times of measurement, and \(y_j\) the measured value, we have assumed that \(t_j\) are exact, and the measurement errors are in \(y_j\).

Under this assumption, the discrepancies between the model and the measured data are

\[
\epsilon_j = y_j - \Phi(\bar{x}; t_j), \quad i = 1, 2, \ldots, m.
\]

Next, we will take a look at the situation where we take errors in \(t_j\) into account — these models are known as errors-in-variables models, and their solutions in the linear case are referred to as total least squares optimization, or in the non-linear case as orthogonal distance regression.
Figure: (left) An illustration of how the error is measured in standard (fixed regressor) least squares optimization. (right) An example of **orthogonal distance regression**, where we measure the shortest distance to the model curve. *The right figure is actually not correct, why?*
For the mathematical formulation of orthogonal distance regression we introduce perturbations (errors) $\delta_j$ for the variables $t_j$, in addition to the errors $\epsilon_j$ for the $y_j$’s.

We relate the measurements and the model in the following way

$$\epsilon_j = y_j - \Phi(\bar{x}; t_j + \delta_j),$$

and define the minimization problem:

$$(\bar{x}^*, \bar{\delta}^*) = \arg \min_{\bar{x}, \bar{\delta}} \frac{1}{2} \sum_{j=1}^{m} \left[ w_j^2 \left( y_j - \Phi(\bar{x}; t_j + \delta_j) \right)^2 + d_j^2 \delta_j^2 \right],$$

where $\bar{d}$ and $\bar{w}$ are two vectors of weights which denote the relative significance of the error terms.
Orthogonal Distance Regression: The Weights

The weight-vectors $\bar{d}$ and $\bar{w}$ must either be supplied by the modeler, or estimated in some clever way.

If all the weights are the same $w_j = d_j = C$, then each term in the sum is simply the shortest distance between the point $(t_j, y_j)$ and curve $\Phi(\bar{x}; t)$ (as illustrated in the previous figure).

In order to get the orthogonal-looking figure, I set $w_j = 1/0.5$ and $d_j = 1/4$, thus adjusting for the different scales in the $t$- and $y$-directions.

The shortest path between the point and the curve will be normal (orthogonal) to the curve at the point of intersection.

We can think of the scaling (weighting) as adjusting for measuring time in fortnights, seconds, milli-seconds, micro-seconds, or nano-seconds...
Orthogonal Distance Regression: In Terms of Residuals $r_j$

By identifying the $2m$ residuals

$$r_j(\bar{x}, \delta) = \begin{cases} w_j \left[ y_j - \Phi(\bar{x}; t_j + \delta_j) \right] & j = 1, 2, \ldots, m \\ d_{j-m}\delta_{j-m} & j = (m + 1), (m + 2), \ldots, 2m \end{cases}$$

we can rewrite the optimization problem

$$(\bar{x}^*, \delta^*) = \arg\min_{\bar{x}, \delta} \frac{1}{2} \sum_{j=1}^{m} w_j^2 \left[ y_j - \Phi(\bar{x}; t_j + \delta_j) \right]^2 + d_j^2\delta_j^2,$$

in terms of the $2m$-vector $\bar{r}(\bar{x}, \delta)$

$$(\bar{x}^*, \delta^*) = \arg\min_{\bar{x}, \delta} \frac{1}{2} \sum_{j=1}^{2m} r_j(\bar{x}, \delta)^2 = \arg\min_{\bar{x}, \delta} \frac{1}{2} \left\| \bar{r}(\bar{x}, \delta) \right\|_2^2.$$
If we take a cold hard stare at the expression

\[
(\bar{x}^*, \bar{\delta}^*) = \arg \min_{\bar{x}, \bar{\delta}} \frac{1}{2} \sum_{i=1}^{2m} r_j(\bar{x}, \bar{\delta})^2 = \arg \min_{\bar{x}, \bar{\delta}} \frac{1}{2} \| \bar{r}(\bar{x}, \bar{\delta}) \|_2^2.
\]

We realize that this is now a standard (nonlinear) least squares problem with \(2m\) residuals and \((n + m)\) unknowns — \(\{\bar{x}, \bar{\delta}\}\).

We can use any of the techniques we have previously explored for the solution of the nonlinear least squares problem.

However, a straight-forward implementation of these strategies may prove to be quite expensive, since the number of parameters have doubled to \(2m\) and the number of independent variables have grown from \(n\) to \((n + m)\). Recall that usually \(m \gg n\), so this is a drastic growth of the problem.
**Figure:** We recast ODR as a much larger standard nonlinear least squares problem.

Standard LSQ-solution via QR/SVD $\sim O(mn^2)$, for $m \gg n$; slows down by a factor of $2(1 + m/n)^2$. 
Fortunately we can save a lot of work by exploiting the structure of the Jacobian of the Least Squares problem originating from the orthogonal distance regression — many entries are zero!

\[
\frac{\partial r_j}{\partial \delta_i} = w_j \frac{\partial [y_j - \Phi(\bar{x}; t_j + \delta_j)]}{\partial \delta_i} = 0, \quad \forall i, j \leq m, \ i \neq j
\]

\[
\frac{\partial r_j}{\partial \delta_i} = \frac{\partial [d_{j-m} \delta_{j-m}]}{\partial \delta_i} = \begin{cases} 
0 & i \neq (j - m), j > m \\
 d_{j-m} & i = (j - m), \ j > m 
\end{cases}
\]

\[
\frac{\partial r_j}{\partial x_i} = \frac{\partial [d_{j-m} \delta_{j-m}]}{\partial x_i} = 0, \quad i = 1, 2, \ldots, n, \ j > m
\]

Let \( v_j = w_j \frac{\partial [y_j - \Phi(\bar{x}; t_j + \delta_j)]}{\partial \delta_j} \), and let \( D = \text{diag}(\bar{d}) \), and \( V = \text{diag}(\bar{v}) \), then we can write the Jacobian of the residual function in matrix form...
We now have

\[ J(\bar{x}, \bar{\delta}) = \begin{bmatrix} \hat{J} & V \\ 0 & D \end{bmatrix}, \]

where \( D \) and \( V \) are \( m \times m \) diagonal matrices, \( D = \text{diag}(\bar{d}) \), and \( V = \text{diag}(\bar{v}) \), and \( \hat{J} \) is the \( m \times n \) matrix defined by

\[ \hat{J} = \begin{bmatrix} \frac{\partial [w_j(y_j - \Phi(\bar{x}; t_j + \delta_j))]}{\partial x_i} \\ \end{bmatrix}_{j = 1, 2, \ldots, m \atop i = 1, 2, \ldots, n} \]

We can now use this matrix in e.g. the Levenberg-Marquardt algorithm...
Figure: If we exploit the structure of the Jacobian, the problem is still somewhat tractable.
If we partition the step vector $\bar{p}$, and the residual vector $\bar{r}$ into

$$
\bar{p} = \begin{bmatrix} \bar{p}_x \\ \bar{p}_\delta \end{bmatrix}, \quad \bar{r} = \begin{bmatrix} \tilde{r}_1 \\ \tilde{r}_2 \end{bmatrix}
$$

where $\bar{p}_x \in \mathbb{R}^n$, $\bar{p}_\delta \in \mathbb{R}^m$, and $\tilde{r}_1, \tilde{r}_2 \in \mathbb{R}^m$, then e.g. we can write the Levenberg-Marquardt subproblem in partitioned form

$$
\begin{bmatrix}
J^T \hat{J} + \lambda I_n \\
V \hat{J} \\
\hat{J}^T V \\
V^2 + D^2 + \lambda I_m
\end{bmatrix}
\begin{bmatrix}
\bar{p}_x \\
\bar{p}_\delta
\end{bmatrix}
= -
\begin{bmatrix}
\hat{J}^T \tilde{r}_1 \\
V \tilde{r}_1 + D \tilde{r}_2
\end{bmatrix}
$$

Since the (2, 2)-block $V^2 + D^2 + \lambda I_m$ is diagonal, we can eliminate the $\bar{p}_\delta$ variables from the system...
This leads to the $n \times n$-system $A\tilde{p}_x = \tilde{b}$, where

$$A = \begin{bmatrix} \hat{J}^T\hat{J} + \lambda I_n \hline \hat{J}^T V \end{bmatrix} \begin{bmatrix} \hat{J}^T \hline V^2 + D^2 + \lambda I_m \end{bmatrix}^{-1} \begin{bmatrix} \hat{V} \hline \hat{J} \end{bmatrix},$$

$$\tilde{b} = \begin{bmatrix} -\hat{J}^T \tilde{r}_1 + \hat{J}^T V \hline V \tilde{r}_1 + D \tilde{r}_2 \end{bmatrix} \begin{bmatrix} \hat{V} \hline \hat{J} \end{bmatrix}^{-1} \begin{bmatrix} \hat{V} \hline \hat{J} \end{bmatrix} \begin{bmatrix} \hat{J}^T \tilde{r}_1 \hline V \tilde{r}_1 + D \tilde{r}_2 \end{bmatrix}.$$

Hence, the total cost of finding the LM-step is only marginally more expensive than for the standard least squares problem.
The derived system is typically very ill-conditioned since we have formed a modified version of the normal equations $\hat{J}^T \hat{J} + \text{"stuff"} \ldots$ With some work we can recast is as an $m \times n$ linear least squares problem

$$\bar{p}_x = \arg\min_{\bar{p}} \| \bar{A} \bar{p} - \tilde{b} \|_2,$$

where

$$\tilde{A} = \left[ \hat{J} + \lambda [\hat{J}^T]^{+} - V \left[ V^2 + D^2 + \lambda I_m \right]^{-1} V \hat{J} \right]$$

$$\tilde{b} = \left[ -\tilde{r}_1 + V \left[ V^2 + D^2 + \lambda I_m \right]^{-1} \left[ V \tilde{r}_1 + D \tilde{r}_2 \right] \right]$$

Where the “mystery factor” $[\hat{J}^T]^{+}$ is the pseudo-inverse of $\hat{J}^T$. Expressed in terms of the QR-factorization $QR = \hat{J}$, we have

$$\hat{J}^T = R^T Q^T, \quad [\hat{J}^T]^{+} = QR^{-T},$$

Since $QR^{-T} R^T Q^T = I = R^T Q^T QR^{-T}$.
Software and References

**MINPACK**

**ODRPACK**

**Other**
The NAG (Numerical Algorithms Group) library and HSL (formerly the Harwell Subroutine Library), implement several robust nonlinear least squares implementations.

**GvL**
Golub and van Loan’s *Matrix Computations, 4th edition* (chapters 5–6) has a comprehensive discussion on orthogonalization and least squares; explaining in gory detail much of the linear algebra (*e.g.* the SVD and QR-factorization) we swept under the rug.
orthogonal distance regression
  Jacobian structure, 15
  Levenberg-Marquardt formulation, 19
  linear least squares formulation, 20
  minimization problem, 10
  residuals, 12