Nonlinear Optimization

Ying Xiong
School of Engineering and Applied Sciences
Harvard University
yxiong@seas.harvard.edu

Created as NonlinearLeastSquares.pdf: January 19th, 2014.
Updated: January 30th, 2014 (v0.2).
Migrated to NonlinearOptimization.pdf: April 28, 2014.

1 Nonlinear Minimization

1.1 Problem Statement

The nonlinear minimization problem is to find a (local) minimizer for an objective function $f(\cdot)$, which takes in a vector $\mathbf{x} \in \mathbb{R}^n$ as input and a scalar $f(\mathbf{x})$ as output. We denote the gradient of the function f as $\mathbf{f}' : \mathbb{R}^n \mapsto \mathbb{R}^n$ and the Hessian as $\boldsymbol{f}'' : \mathbb{R}^n \mapsto \mathbb{R}^{n \times n}$.

1.2 Descent Algorithms [1]

A broad and important class of algorithms take an iterative form

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{h}^k, \qquad k = 0, 1, 2, \ldots, \tag{1}$$

where if $\mathbf{f}'\left(\mathbf{x}^k\right) \neq 0$, we choose direction $\mathbf{h}^k$ so that $\mathbf{f}'\left(\mathbf{x}^k\right)^\top \mathbf{h}^k < 0$ and the step size α^k so that $f\left(\mathbf{x}^k + \alpha^k \mathbf{h}^k\right) < f\left(\mathbf{x}^k\right)$.

1.3 Descent Algorithms — Descent Directions

1.3.1 Newton-Type Methods [2]

At current ponit $\mathbf{x}$, consider the second order Taylor expansion of f

$$f(\mathbf{x} + \mathbf{h}) \approx q(\mathbf{h}) = f(\mathbf{x}) + \mathbf{h}^\top \mathbf{f}'(\mathbf{x}) + \frac{1}{2}\mathbf{h}^\top \boldsymbol{f}''(\mathbf{x})\mathbf{h}. \tag{2}$$

Newton's Method In its pure form, Newton's method minimize $q(\mathbf{h})$ with respect to $\mathbf{h}$, getting

$$\mathbf{h}_{\mathrm{n}} = -\left(\boldsymbol{f}''(\mathbf{x})\right)^{-1} \mathbf{f}'(\mathbf{x}). \tag{3}$$

It also chooses $\alpha^k = 1$ at every step. The pure Newton's algorithm converges quadratically when $\mathbf{x}$ is sufficiently close to a local minimizer, but it has some significant draw back: (1) $\boldsymbol{f}''(\mathbf{x})$ might not be invertible; (2) $\boldsymbol{f}''(\mathbf{x})$ might be expensive to compute; (3) $\mathbf{h}_n$ is not necessarily a descent direction.

Quasi-Newton Methods In Quasi-Newton methods, the Hessian matrix $\boldsymbol{f}''\left(\mathbf{x}^k\right)$ is approximated and updated at each iteration with $\boldsymbol{B}^k$. The descent direction is therefore

$$\mathbf{h}_{\text{qn}}^k = -\left(\boldsymbol{B}^k\right)^{-1} \mathbf{f}'\left(\mathbf{x}^k\right), \tag{4}$$

and usually a line search is performed to find optimal α^k, although $\alpha^k = 1$ is a good guess when $\mathbf{x}^k$ is close to a local minimum. Also note for $\mathbf{h}_{\text{qn}}^k$ to be a descent direction, the $\boldsymbol{B}^k$ is required to be positive definite.

One of the most popular approximation formula is the **BFGS formulae [3, 2]**

$$\boldsymbol{B}^{\text{new}} = \boldsymbol{B} + \frac{\mathbf{y}\mathbf{y}^\top}{\alpha\mathbf{h}^\top\mathbf{y}} - \frac{\mathbf{u}\mathbf{u}^\top}{\alpha\mathbf{h}^\top\mathbf{u}}, \tag{5}$$

where

$$\mathbf{y} = \mathbf{f}'(\mathbf{x}+\alpha\mathbf{h}) - \mathbf{f}'(\mathbf{x}), \quad \mathbf{u} = \alpha\boldsymbol{B}\mathbf{h}. \tag{6}$$

When computing $\mathbf{h}_{\text{qn}}^k$, we need $\left(\boldsymbol{B}^k\right)^{-1}$ instead of $\boldsymbol{B}^k$ itself, which can be directly updated without performing an inverse

$$\left(\boldsymbol{B}^{\text{new}}\right)^{-1} = \boldsymbol{B}^{-1} + \frac{\left(\alpha\mathbf{h}^\top\mathbf{y} + \mathbf{y}^\top\mathbf{v}\right)\left(\alpha^2\mathbf{h}\mathbf{h}^\top\right)}{\left(\alpha\mathbf{h}^\top\mathbf{y}\right)^2} - \frac{\mathbf{v}\mathbf{h}^\top + \mathbf{h}\mathbf{v}^\top}{\mathbf{h}^\top\mathbf{y}}, \quad \text{with } \mathbf{v} = \boldsymbol{B}^{-1}\mathbf{y}. \tag{7}$$

One key property is that if $\boldsymbol{B}$ is positive definite and $\mathbf{h}^\top\mathbf{y} > 0$, then $\boldsymbol{B}^{\text{new}}$ is also positive definite [1]. Note that $\mathbf{h}^\top\mathbf{y} > 0$ is satisfied if α is chosen to satisfy the Wolfe's conditions (more specifically, the curvature condition).

The initial $\boldsymbol{B}^0$ is usually set as an identity matrix.

1.4 Descent Algorithms — Line Search [1]

At current point $\mathbf{x}$ and given a search direction $\mathbf{h}$, define a univariate function

$$\varphi(\alpha) = f(\mathbf{x}+\alpha\mathbf{h}), \tag{8}$$

whose derivative can be calculated as

$$\varphi'(\alpha) = \mathbf{h}^\top\mathbf{f}'(\mathbf{x}+\alpha\mathbf{h}). \tag{9}$$

The line search algorithms seek to (approximately) minimize $\varphi(\alpha)$ for $\alpha \geq 0$.

1.4.1 Exact Line Search

In the exact sense, line search minimizes $\varphi(\alpha)$ for $\alpha \geq 0$ or $\alpha \in [0, s]$. But this minimization can be expensive, and its accuracy is usually not critical. Therefore people usually perform a "soft line search", *i.e.* find an $\alpha \geq 0$ such that certain condition on $\varphi(\alpha), \varphi'(\alpha)$ is satisfied.

1.4.2 Wolfe Conditions [4]

A step length α is said to satisfy the Wolfe's condition if the following inequalities hold

$$f\left(\mathbf{x}+\alpha\mathbf{h}\right) \leq f\left(\mathbf{x}\right) + c_1\alpha\mathbf{h}^\top\mathbf{f}'\left(\mathbf{x}\right), \qquad \text{or equivalently,} \quad \varphi(\alpha) \leq \varphi(0) + c_1\alpha\varphi'(0), \tag{10}$$

$$\mathbf{h}^\top\mathbf{f}'\left(\mathbf{x}+\alpha\mathbf{h}\right) \geq c_2\mathbf{h}^\top\mathbf{f}'(\mathbf{x}), \qquad \text{or equivalently,} \quad \varphi'(\alpha) \geq c_2\varphi'(0). \tag{11}$$

with $0 < c_1 < c_2 < 1$. The first condition is also known as Armijo rule, and the second curvature condition. Typical values are $c_1 = 10^{-4}$ and $c_2 = 0.9$ for Newton or quasi-Newton methods. The geometric meaning for Wolfe conditions are shown below.

We note that the curvature condition (11) is not trivial if $\varphi(\alpha)$ is concave at $\alpha = 0$ (right plot of the following figure), in which case the left limit of the acceptable region does not goes to 0^+ even when $c_2 \to 1^-$. The curvature condition is also crucially necessary for positive-definite properties in BFGS-like algorithm.

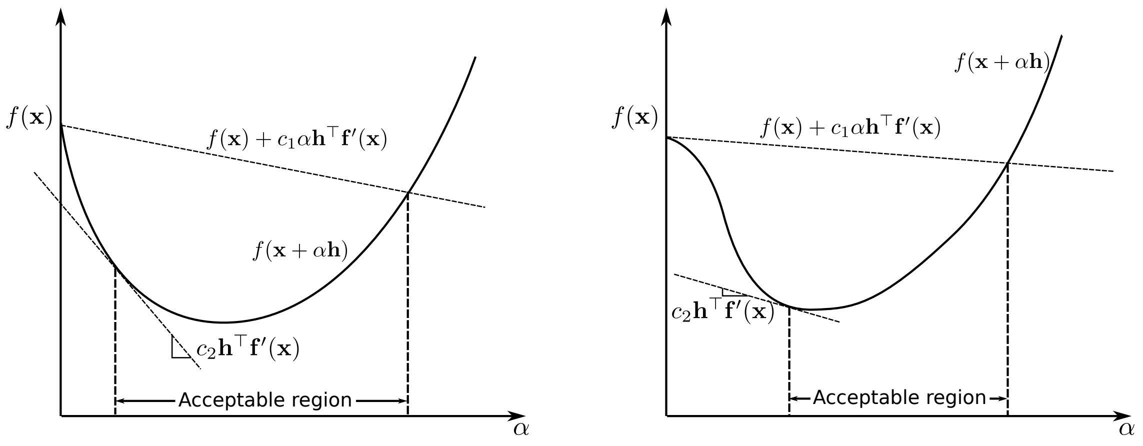
$f(\mathbf{x})$
$f(\mathbf{x}) + c_1 \alpha \mathbf{h}^\top \mathbf{f}'(\mathbf{x})$
$f(\mathbf{x} + \alpha\mathbf{h})$
$c_2 \mathbf{h}^\top \mathbf{f}'(\mathbf{x})$
Acceptable region
α
$f(\mathbf{x})$
$f(\mathbf{x} + \alpha\mathbf{h})$
$f(\mathbf{x}) + c_1 \alpha \mathbf{h}^\top \mathbf{f}'(\mathbf{x})$
$c_2 \mathbf{h}^\top \mathbf{f}'(\mathbf{x})$
Acceptable region
α

Figure 1: Wolfe conditions.

There is also a strong Wolfe condition on curvature, which replaces (11) with

$$\left|\mathbf{h}^\top \mathbf{f}'(\mathbf{x} + \alpha\mathbf{h})\right| \le \left|c_2 \mathbf{h}^\top \mathbf{f}'(\mathbf{x})\right|, \qquad \text{or equivalently,} \quad |\varphi'(\alpha)| \ge c_2 |\varphi'(0)|. \tag{12}$$

1.4.3 Backtracking Algorithm

The backtracking line search algorithm starts from an initial step size, say 1.0 for most quadi-Newton methods, and keep decreasing the step size by a factor of β until the Armijo rule (10) is satisfied. The β is usually chosen from $1/2$ to $1/10$ depending on the confidence on the quality of the initial step size (use big β if the confidence is high).

However, the result of backtracking algorithm does not necessarily satisfy the curvature condition (11), and therefore is not directly suitable for BFGS algorithm.

1.4.4 Bracketing Algorithm [2]

Algorithm 1: Bracketing line search algorithm

Input : The univarite function $\varphi(\cdot)$ such that $\varphi'(0) < 0$.
Input : Initial interval $[a, b]$ for result α, use $[0, 1]$ as default.
Output: A step size α that satisfies Wolfe conditions (10) and (11). Note that α does not necessarily lie in the original $[a, b]$ interval.

```
1  while (10) holds and (11) does not hold for $\alpha = b$ do
2      $[a, b] \leftarrow [b, 2b]$.
3  end
4  $\alpha \leftarrow b$.
5  while (10) or (11) does not hold do
6      $c \leftarrow (\varphi(b) - \varphi(a) - (b-a)\varphi'(a)) / (b-a)^2$
7      if $c > 0$ then $\alpha \leftarrow \min\{0.9a + 0.1b,\ \max\{0.1a + 0.9b,\ a - \varphi'(a)/2c\}\}$;
8      else $\alpha \leftarrow (a+b)/2$;
9      if (10) holds then $a \leftarrow \alpha$;
10     else $b \leftarrow \alpha$;
11 end
```

The first loop makes sure that (1) a satisfies sufficient descent condition and (2) b satisfies curvature condition or b does not satisfy sufficient descent condition. This makes sure $[a, b]$ includes at least one point in the acceptable region.

In practice, we avoid infinite loop by requesting $b < b_{\max}$ in the first loop and apply a maximum number of iterations to the second loop. If this fails to make progress on $\varphi(\alpha)$, we set $\alpha = 0$. Thus the line search algorithm is guaranteed to finish and in finite time and not increase the objective, even if input with a non-descending direction.

2 Nonlinear Least Squares

2.1 Problem Statement

The nonlinear least squares problem is to find a (local) minimizer for cost function

$$F(\mathbf{x}) = \sum_{i=1}^{m} (f_i(\mathbf{x}))^2 = \|\mathbf{f}(\mathbf{x})\|^2 = \mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}), \tag{13}$$

where $f_i : \mathbb{R}^n \mapsto \mathbb{R},\ i = 1, \ldots, m$ are given nonlinear functions.

A nonlinear least squares problem is a special variant of the more general nonlinear optimization problem, and the special form provides useful structure that we can exploit. Define

$$(\boldsymbol{J}(\mathbf{x}))_{i,j} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}) \tag{14}$$

the Jacobian matrix of $\mathbf{f}(\mathbf{x})$, then we have

$$\mathbf{F}'(\mathbf{x}) = 2\boldsymbol{J}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}), \tag{15}$$

$$\boldsymbol{F}''(\mathbf{x}) = 2\boldsymbol{J}(\mathbf{x})^\top \boldsymbol{J}(\mathbf{x}) + 2\sum_{i=1}^{m} f_i(\mathbf{x})\, \boldsymbol{f}_i''(\mathbf{x}), \tag{16}$$

which means even when we do not have second-order information of $\mathbf{f}(\mathbf{x})$, we still know *something* about $\boldsymbol{F}''(\mathbf{x})$ from $\boldsymbol{J}(\mathbf{x})$ alone. Furthermore, if $\mathbf{f}(\mathbf{x}) \approx \mathbf{0}$ near the minima, then $2\boldsymbol{J}(\mathbf{x})^\top \boldsymbol{J}(\mathbf{x})$ is a good approximation of $\boldsymbol{F}''(\mathbf{x})$.

2.2 Algorithms

We make a linear approximation on $\mathbf{f}(\mathbf{x})$ near a given $\mathbf{x}$ as

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) \approx \boldsymbol{\ell}(\mathbf{h}) = \mathbf{f}(\mathbf{x}) + \boldsymbol{J}(\mathbf{x})\,\mathbf{h}, \tag{17}$$

which yields

$$F(\mathbf{x} + \mathbf{h}) \approx L(\mathbf{h}) = \boldsymbol{\ell}(\mathbf{h})^\top \boldsymbol{\ell}(\mathbf{h}) \tag{18}$$

$$= \mathbf{f}^\top \mathbf{f} + 2\mathbf{h}^\top \boldsymbol{J}^\top \mathbf{f} + \mathbf{h}^\top \boldsymbol{J}^\top \boldsymbol{J}\mathbf{h} \tag{19}$$

$$= F(\mathbf{x}) + 2\mathbf{h}^\top \boldsymbol{J}^\top \mathbf{f} + \mathbf{h}^\top \boldsymbol{J}^\top \boldsymbol{J}\mathbf{h}. \tag{20}$$

Note that this is equivalent to perform a second order Taylor expansion on $F(\mathbf{x})$ and approximate $\boldsymbol{F}''$ as $2\boldsymbol{J}^\top \boldsymbol{J}$.

2.2.1 Gauss-Newton Algorithm

The Gauss-Newton algorithm minimize (20) directly, with

$$\mathbf{h}_{\text{gn}} = -\left(\boldsymbol{J}^\top \boldsymbol{J}\right)^{-1} \boldsymbol{J}^\top \mathbf{f}. \tag{21}$$

The algorithm has at least two short-comings: (1) $\left(\boldsymbol{J}^\top \boldsymbol{J}\right)$ might be singular and (2) $\mathbf{h}_{\text{gn}}$ might not be a descending direction.

2.2.2 Levenberg-Marquardt Algorithm [5, 6, 7]

Levenberg-Marquardt algorithm is a damped Gaussian-Newton method

$$\mathbf{h}_{\text{lm},1} = -\left(\boldsymbol{J}^\top \boldsymbol{J} + \mu \boldsymbol{I}\right)^{-1} \boldsymbol{J}^\top \mathbf{f}, \tag{22}$$

or, as suggested by Marquardt

$$\mathbf{h}_{\text{lm},2} = -\left(\boldsymbol{J}^\top \boldsymbol{J} + \mu \,\text{diag}\left(\boldsymbol{J}^\top \boldsymbol{J}\right)\right)^{-1} \boldsymbol{J}^\top \mathbf{f}. \tag{23}$$

We write the two forms together as

$$\mathbf{h}_{\text{lm}}^\top = -\left(\boldsymbol{J}^\top \boldsymbol{J} + \mu \boldsymbol{D}\right)^{-1} \boldsymbol{J}^\top \mathbf{f}, \tag{24}$$

where the "damping matrix" $\boldsymbol{D}$ can either be $\boldsymbol{I}$ or diag $\left(\boldsymbol{J}^\top \boldsymbol{J}\right)$.

Choice of Damping Factor [8] Define a *gain ratio*

$$\varrho = \frac{F\left(\mathbf{x}\right) - F\left(\mathbf{x} + \mathbf{h}_{\text{lm}}\right)}{L(\mathbf{0}) - L(\mathbf{h}_{\text{lm}})}, \tag{25}$$

where $L(\mathbf{h})$ is defined in (18), and the denominator can be calculated as

$$L(\mathbf{0}) - L(\mathbf{h}_{\text{lm}}) = \mathbf{h}_{\text{lm}}^\top \left(\mu \boldsymbol{D} \mathbf{h}_{\text{lm}} - \boldsymbol{J}^\top \mathbf{f}\right) \tag{26}$$

The update rule for μ will be

$$\mu_{k+1} = \begin{cases} \mu \cdot \max\left\{\frac{1}{3}, 1 - (2\varrho - 1)^3\right\}; \quad \nu = 2 & \text{if } \varrho > 0, \\ \mu \cdot \nu; \quad \nu = 2 \cdot \nu & \text{otherwise.} \end{cases} \tag{27}$$

The initial μ is usually set as $\tau \cdot \max_i \left\{\left(\boldsymbol{J}^\top \boldsymbol{J}\right)_{i,i}\right\}$, where τ is a user specified parameter, which should be a small value, *e.g.* $\tau = 10^{-6}$ if $\mathbf{x}_0$ is a good approximation to the final local minimum, and 10^{-3} or even 1 otherwise.

Algorithm Description The full Levenberg-Marquardt is described as below.

Algorithm 2: Levenberg-Marquardt method

```
Input : $\mathbf{f}(\mathbf{x}), \boldsymbol{J}(\mathbf{x})$: Input function and its Jacobian matrix.
Input : $\mathbf{x}_0$: Initial guess.
Input : $\tau$: A parameter specifying initial damping factor, default $10^{-3}$.
Input : A stopping criterion.
Output: $\mathbf{x}$: A local minimum.
1  $\mathbf{x} = \mathbf{x}_0,\ \mu = \tau \cdot \max_i \left\{\left(\boldsymbol{J}^\top \boldsymbol{J}\right)_{i,i}\right\},\ \nu = 2.$
2  while the stopping criterion is not met do
3      Calculate $\mathbf{h}_{\text{lm}}$ according to (24).
4      Calculate $\varrho$ according to (25).
5      if $\varrho > 0$ then
6          $\mathbf{x} = \mathbf{x} + \mathbf{h}_{\text{lm}},\ \mu = \mu \cdot \max\left\{\frac{1}{3}, 1 - (2\varrho - 1)^3\right\},\ \nu = 2.$
7      else
8          $\mu = \mu \cdot \nu,\ \nu = 2\nu.$
9      end
10 end
```

Implementation Notes A few remarks on implementation details.

1. When the step size $\mathbf{h}_{\text{lm}}$ is very small, the calculation of ϱ in (Step 4) can suffer from numerical underflow. One needs to check whether $L(\mathbf{0}) - L(\mathbf{h}_{\text{lm}}) < \varepsilon$, where ε is the machine's numerical percision, and if so, terminate the algorithm. When the algorithm terminates this way, we are usually very close to a local minimum.

2. Due to possible ill-conditioning, the matrix $\left(\boldsymbol{J}^\top \boldsymbol{J}\right)$ can be singular, and when μ is very small — which happens after a number of consecutive success descent — the matrix $\left(\boldsymbol{J}^\top \boldsymbol{J} + \mu \boldsymbol{I}\right)$ can also be close to singular, which causes numerical issues. To circumvent this, we put a minimum on μ in (Step 6), changing it to $\mu = \max\left\{\mu_{\text{min}}, \mu \cdot \max\left\{\frac{1}{3}, 1 - (2\varrho - 1)^3\right\}\right\}$, in order to make sure $\left(\boldsymbol{J}^\top \boldsymbol{J} + \mu \boldsymbol{I}\right)$ is always well-conditioned. We choose $\mu_{\text{min}} = 10^{-12}$ in our implementation.

2.3 Bounded Constraints

One of the common variants of the unconstrained nonlinear least squares problem is to add bounded constraints

$$l_i \le x_i \le u_i, \tag{28}$$

with $-\infty \le l_i < u_i \le +\infty$ (infinity bound means not constraint). To incorporate such constraint, we define a mapping from the unconstrained space to constrained space

$$\mathbf{x}(\mathbf{y}) : \mathbb{R}^n \mapsto [\mathbf{l}, \mathbf{u}] = \prod_i [l_i, u_i], \tag{29}$$

and perform an unconstrained optimization on the function $f(\mathbf{x}(\mathbf{y}))$ with respect to $\mathbf{y}$.

Mapping Function We list the specific mapping function from the unconstrained space to constrained space. The general rule is that (1) the mapping is smooth and (2) the absolute value of derivative is smaller than 1 (but also close to 1 in most of the place). We also provide one possible inverse of the mapping $y_i^0(x_i)$, which is used for initialization.

- $l_i = -\infty$, $u_i = +\infty$, x_i unconstrained

$$x_i = y_i, \quad \frac{dx_i}{dy_i} = 1; \quad y_i^0 = x_i. \tag{30}$$

- $l_i = -\infty$, $x_i \le u_i < +\infty$

$$x_i = u_i + 1 - \sqrt{y_i^2 + 1}, \quad \frac{dx_i}{dy_i} = -\frac{y_i}{\sqrt{y_i^2 + 1}}; \quad y_i^0 = \sqrt{(u_i + 1 - x_i)^2 - 1}. \tag{31}$$

- $-\infty < l_i \le x_i$, $u_i = +\infty$

$$x_i = l_i - 1 + \sqrt{y_i^2 + 1}, \quad \frac{dx_i}{dy_i} = \frac{y_i}{\sqrt{y_i^2 + 1}}; \quad y_i^0 = \sqrt{(l_i - 1 - x_i)^2 - 1}. \tag{32}$$

- $-\infty < l_i \le x_i \le u_i < +\infty$

$$x_i = \frac{l_i + u_i}{2} + \frac{u_i - l_i}{2} \sin \frac{2y_i}{u_i - l_i}, \quad \frac{dx_i}{dy_i} = \cos \frac{2y_i}{u_i - l_i}; \quad y_i^0 = \frac{u_i - l_i}{2} \arcsin \frac{2x_i - (u_i + l_i)}{u_i - l_i}. \tag{33}$$

References

[1] D. P. Bertsekas, *Nonlinear programming*, 2nd ed. Athena Scientific, 1999.

[2] P. E. Frandsen, K. Jonasson, H. B. Nielsen, and O. Tingleff, *Unconstrained optimization*, 3rd ed., 2004.

[3] Wikipedia, “Plagiarism — Wikipedia, the free encyclopedia,” 2014, [Online; accessed 28-April-2014]. [Online]. Available: http://en.wikipedia.org/wiki/Broyden-Fletcher-Goldfarb-Shanno_algorithm

[4] ——, “Plagiarism — Wikipedia, the free encyclopedia,” 2014, [Online; accessed 28-April-2014]. [Online]. Available: http://en.wikipedia.org/wiki/Wolfe_conditions

[5] K. Levenberg, “A method for the solution of certain problems in least squares,” *Quarterly of applied mathematics*, vol. 2, pp. 164–168, 1944.

[6] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[7] Wikipedia, “Plagiarism — Wikipedia, the free encyclopedia,” 2014, [Online; accessed 20-January-2014]. [Online]. Available: http://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

[8] K. Madsen, H. B. Nielsen, and O. Tingleff, *Methods for non-linear least squares problems*, 2nd ed., 2004.