LEVENBERG-MARQUARDT ALGORITHM

YOU XUAN THUNG[*]

Abstract. Many data-fitting problems involve the use of a non-linear fit function. Solving the corresponding non-linear least squares problem requires a good algorithm. The Levenberg-Marquardt algorithm is a common method used to tackle this problem. In this paper, we introduce the algorithm, explain its relation with gradient descent and Gauss-Newton, and compare these methods using a set of 10 test problems. We find that the Levenberg-Marquardt algorithm is indeed better than gradient descent and Gauss-Newton for solving non-linear least squares problems. We also review an alternate damping parameter update rule introduced by Nielsen, and verify that it achieves smoother convergence.

Key words. Levenberg-Marquardt algorithm, gradient descent, Gauss-Newton, non-linear least squares

AMS subject classifications. 65K10

1. Introduction.

The Levenberg-Marquardt algorithm [2, 4] is a method for solving the non-linear least squares problem i.e. finding parameters $\boldsymbol{\beta} \in \mathbb{R}^n$ of a non-linear function $f(\mathbf{x}; \boldsymbol{\beta})$ to minimize the sum of squared deviations for a set of empirical m data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$:

$$\boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} F(\mathbf{x}; \boldsymbol{\beta}) = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{m} (y_i - f(\mathbf{x}_i; \boldsymbol{\beta}))^2 \tag{1.1}$$

While linear least squares problems can be tackled by solving a linear system of first order conditions, solutions to non-linear least squares problems require iterative algorithms. There are two main numerical algorithms to tackle non-linear least squares problems. Gradient descent updates parameters in the opposite direction of the gradient since this is the direction of steepest descent, but this method tends to exhibit slow convergence after the first few iterations. The Gauss-Newton method assumes that the fit function can be modelled as a first order Taylor series locally and finds the minimum of the consequently quadratic objective function. This method is effective insofar as the assumption holds, and successive iterates tend to diverge.

The Levenberg-Marquardt algorithm seeks to combine the best of both algorithms: functioning like the gradient descent method when parameters are still far from the optimum but functioning more like the Gauss-Newton method once parameters are closer to the optimum [1]. This is done by introducing a damping parameter λ that controls how the algorithm functions—larger λ brings the algorithm closer to gradient descent and smaller λ brings the algorithm closer to Gauss-Newton. λ is updated iteratively, and decreased if the solution is improving.

The paper is organized as follows. We discuss comparable methods in section 2, then introduce the Levenberg-Marquardt algorithm in section 3. We discuss experimental results in section 4, and the conclusions follow in section 5.

2. Comparable Methods.

The gradient descent method updates parameters iteratively in the direction of steepest descent. In the context of solving a least squares

[*]Center of Computational Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA (thungyx@mit.edu).

problem, the gradient of the objective function with respect to $\boldsymbol{\beta}$ is given as

$$-2(\mathbf{y} - \mathbf{f}(\mathbf{x};\boldsymbol{\beta}))^\top \frac{\partial}{\partial \boldsymbol{\beta}}(\mathbf{y} - \mathbf{f}(\mathbf{x};\boldsymbol{\beta})) = -2(\mathbf{y} - \mathbf{f}(\mathbf{x};\boldsymbol{\beta}))^\top \mathbf{J} \tag{2.1}$$

$\mathbf{y}$ is an m-dimensional vector of the empirical data, while $\mathbf{f}(\cdot)$ applies $f(\cdot)$ to each of the m empirical data points. $\mathbf{J}$ is the $(m \times n)$ Jacobian matrix $(\partial \mathbf{f}/\partial \boldsymbol{\beta})$.

Therefore, the search direction $\mathbf{h}_{gd}$ is given as

$$\mathbf{h}_{gd} = \mathbf{J}^\top(\mathbf{y} - \mathbf{f}(\mathbf{x};\boldsymbol{\beta})) \tag{2.2}$$

To avoid excessive computation arising from doing an exact line search, we stick to a fixed step size in our implementation.

This culminates in the following algorithm

Algorithm 2.1 Gradient descent

```
$k \leftarrow 1$; $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_0$; $\alpha \leftarrow \alpha_0$
$\mathbf{h}_{gd} \leftarrow \mathbf{J}^\top(\mathbf{y} - \mathbf{f}(\mathbf{x};\boldsymbol{\beta}))$
while $k < k_{max}$ and $||\mathbf{h}_{gd}||_\infty > \epsilon_{tol}$ do
    Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha\mathbf{h}_{gd}$
    $\mathbf{h}_{gd} \leftarrow \mathbf{J}^\top(\mathbf{y} - \mathbf{f}(\mathbf{x};\boldsymbol{\beta}))$
    $k \leftarrow k + 1$
end while
```

The Gauss-Newton method assumes that $f(\cdot)$ is well-approximated by its first order Taylor series expansion i.e.

$$\mathbf{f}(\mathbf{x};\boldsymbol{\beta} + \mathbf{h}_{gn}) \approx \mathbf{f}(\mathbf{x};\boldsymbol{\beta}) + \mathbf{J}\mathbf{h}_{gn} \tag{2.3}$$

This means that the objective function can be written as

$$F(\mathbf{x};\boldsymbol{\beta} + \mathbf{h}_{gn}) \approx (\mathbf{y} - (\mathbf{f} + \mathbf{J}\mathbf{h}_{gn}))^\top(\mathbf{y} - (\mathbf{f} + \mathbf{J}\mathbf{h}_{gn})) \tag{2.4}$$

$$= \mathbf{y}^\top\mathbf{y} + \mathbf{f}^\top\mathbf{f} - 2\mathbf{y}^\top\mathbf{f} - 2(\mathbf{y} - \mathbf{f})^\top\mathbf{J}\mathbf{h}_{gn} + \mathbf{h}_{gn}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h}_{gn} \tag{2.5}$$

where we drop the arguments for $\mathbf{f}$ to simplify notation. This results in an objective function which is quadratic with respect to $\mathbf{h}_{gn}$. Setting the gradient of this approximation to zero

$$\frac{\partial F}{\partial \mathbf{h}_{gn}} \approx -2(\mathbf{y} - \mathbf{f})^\top\mathbf{J} + 2\mathbf{h}_{gn}^\top\mathbf{J}^\top\mathbf{J} \tag{2.6}$$

yields the following analytical expression for the update

$$\mathbf{h}_{gn} = (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) \tag{2.7}$$

The corresponding algorithm is outlined in Algorithm 2.2.

3. The Levenberg-Marquardt Algorithm.

The Levenberg-Marquardt algorithm combines gradient descent and Gauss-Newton with the following parameter update:

$$\mathbf{h}_{lm} = (\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) \tag{3.1}$$

Algorithm 2.2 Gauss-Newton method

```
$k \leftarrow 1$; $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_0$
$\mathbf{g} \leftarrow \mathbf{J}^\top(\mathbf{y} - \mathbf{f})$
while $k < k_{max}$ and $||\mathbf{h}_{gd}||_\infty > \epsilon_{tol}$ do
    $\mathbf{h}_{gn} \leftarrow (\mathbf{J}^\top \mathbf{J})^{-1}\mathbf{g}$
    Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \mathbf{h}_{gn}$
    $\mathbf{g} \leftarrow \mathbf{J}^\top(\mathbf{y} - \mathbf{f})$
    $k \leftarrow k + 1$
end while
```

where λ functions as a damping parameter. A large λ gives us

$$\mathbf{h}_{lm} \approx \frac{1}{\lambda}\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) \tag{3.2}$$

which is akin to a short step in the steepest descent.

A small λ gives us $\mathbf{h}_{lm} \approx (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top(\mathbf{y} - \mathbf{f})$ which brings the update closer to a Gauss-Newton update.

Typically, λ is initialized to be large so that the algorithm functions more like a gradient descent at the start and more like Gauss-Newton after the solution improves [1].

The algorithm relies on an update rule for the damping parameter λ to adapt to the performance of the solution in the current iteration. To do so, we define a measure of relative improvement

$$\rho(\mathbf{h}_{lm}) = \frac{F(\mathbf{x}; \boldsymbol{\beta}) - F(\mathbf{x}; \boldsymbol{\beta} + \mathbf{h}_{lm})}{L(\mathbf{x}; 0) - L(\mathbf{x}; \mathbf{h}_{lm})} \tag{3.3}$$

The numerator captures the performance gain as predicted by our model f while the denominator captures the performance gain as predicted by the linear approximation of f.

Since

$$\begin{aligned} L(\mathbf{x}; \mathbf{h}_{lm}) &\equiv (\mathbf{y} - (\mathbf{f} + \mathbf{J}\mathbf{h}))^\top(\mathbf{y} - (\mathbf{f} + \mathbf{J}\mathbf{h})) && (3.4)\\ &= \mathbf{y}^\top\mathbf{y} + \mathbf{f}^\top\mathbf{f} - 2\mathbf{y}^\top\mathbf{f} - 2\mathbf{h}^\top\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) + \mathbf{h}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h} && (3.5) \end{aligned}$$

the denominator of ρ is given as

$$\begin{aligned} L(\mathbf{x}; 0) - L(\mathbf{x}; \mathbf{h}_{lm}) &= 2\mathbf{h}_{lm}^\top\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) - \mathbf{h}_{lm}^\top\mathbf{J}^\top\mathbf{J}\mathbf{h}_{lm} && (3.6)\\ &= \mathbf{h}_{lm}^\top(2\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) - \mathbf{J}^\top\mathbf{J}\mathbf{h}_{lm}) && (3.7)\\ &= \mathbf{h}_{lm}^\top(2\mathbf{J}^\top(\mathbf{y} - \mathbf{f}) - (\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I} - \lambda\mathbf{I})\mathbf{h}_{lm}) && (3.8)\\ &= \mathbf{h}_{lm}^\top(\lambda\mathbf{h}_{lm} + \mathbf{J}^\top(\mathbf{y} - \mathbf{f})) && (3.9) \end{aligned}$$

If $\rho > 0$, then there is some performance gain and we accept the parameter update.

If there is strong relative improvement (large ρ), then we can adjust the damping parameter λ downwards, bringing the Levenberg-Marquardt update closer to a Gauss-Newton update. If relative improvement is poor, then we would adjust λ upwards, bringing the Levenberg-Marquardt update closer to a gradient descent update, while decreasing the step-size.

In the original implementation by Marquardt [4], the damping parameter is updated as such:

$$\texttt{if } \rho < \rho_1 \texttt{ then } \lambda \leftarrow \lambda * L_{up} \tag{3.10}$$

$$\texttt{if } \rho > \rho_2 \texttt{ then } \lambda \leftarrow \lambda / L_{down} \tag{3.11}$$

where $0 < \rho_1 < \rho_2 < 1$ and $L_{up}, L_{down} > 1$. Nielsen [6] notes that $\rho_1 = 0.25,\ \rho_2 = 0.75,\ L_{up} = 2,\ L_{down} = 3$ is a popular set of parameters which works well in general, but in their experiments with a range of non-linear least squares problems, they find that $(\rho_1, \rho_2) = (0.2, 0.8)$ leads to faster convergence (as measured by the number of iterations). In our reconstruction of the Levenberg-Marquardt algorithm, we use $(\rho_1, \rho_2) = (0.2, 0.8)$.

The full algorithm is outlined in Algorithm 3.1.

Algorithm 3.1 Levenberg-Marquardt algorithm

```
$k \leftarrow 1$; $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_0$; $\tau \leftarrow \tau_0$
$\mathbf{A} \leftarrow \mathbf{J}^\top \mathbf{J}$
$\mathbf{g} \leftarrow \mathbf{J}^\top (\mathbf{y} - \mathbf{f})$
$\lambda \leftarrow \tau_0 * \max\,[\mathbf{A}]_{ii}$
while $k < k_{max}$ and $||\mathbf{h}_{lm}||_\infty > \epsilon_{tol}$ and $||\mathbf{g}||_\infty > \epsilon_{tol}$ do
  $\mathbf{h}_{lm} \leftarrow (\mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{g}$
  $\boldsymbol{\beta}_{new} \leftarrow \boldsymbol{\beta} + \mathbf{h}_{lm}$
  $\rho \leftarrow \frac{F(\mathbf{x};\boldsymbol{\beta}) - F(\mathbf{x};\boldsymbol{\beta}+\mathbf{h}_{lm})}{L(\mathbf{x};0) - L(\mathbf{x};\mathbf{h}_{lm})}$
  if $\rho > 0$ then
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_{new}$
    $\mathbf{A} \leftarrow \mathbf{J}^\top \mathbf{J}$
    $\mathbf{g} \leftarrow \mathbf{J}^\top (\mathbf{y} - \mathbf{f})$
  end if
  if $\rho > 0.8$ then
    $\lambda \leftarrow \lambda / 3$
  end if
  if $\rho < 0.2$ then
    $\lambda \leftarrow \lambda * 2$
  end if
  $k \leftarrow k + 1$
end while
```

Nielsen [6] notes that the original damping parameter strategy leads to rugged convergence behaviour and suggests an alternative updating strategy that could lead to faster convergence.

Algorithm 3.2 Nielsen updating strategy for LM algorithm

```
if $\rho > 0$ then
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_{new}$
  $\lambda \leftarrow \lambda * \max\{1/3, 1 - (2\rho - 1)^3\}$
  $\nu \leftarrow 2$
else
  $\lambda \leftarrow \lambda * \nu; \nu \leftarrow 2 * \nu$
end if
```

4. Experiments.

4.1. Implementation Details. We develop algorithms for gradient descent, Gauss-Newton, Levenberg-Marquardt and Levenberg-Marquardt with the Nielsen updating rule in Julia. To ensure comparability, the implementations are developed almost from scratch[1] and any step that is equivalent across the four algorithms is computed in the same manner. We set the maximum number of iterations as 10000 and the tolerable error as 10^{-12}. For gradient descent, we use a step size of 0.01.

4.2. Test Problems. To compare the performance of the different algorithms and updating rule, we choose 10 problems from [5] and [6]. The problems from [5] have also been used in [6] and [3].

The problems below are listed using the following format: (a) dimension, (b) function definition, (c) data, (d) starting point $\boldsymbol{\beta}_0$ and τ_0, (e) solution. $\mathbf{e}$ ($\mathbf{E}$) refers to the vector (matrix) of ones and $\mathbf{I}$ is the identity matrix.

1. *Linear function, full rank*
   a) n variable, $m \geq n$
   b) $\mathbf{f}(\mathbf{x};\boldsymbol{\beta}) = \mathbf{x}\boldsymbol{\beta} - \mathbf{e}$
   c) $\mathbf{x} = \begin{bmatrix} \mathbf{I} - \frac{2}{m}\mathbf{E} \\ -\frac{2}{m}\mathbf{E} \end{bmatrix}$, $\mathbf{y} = \mathbf{0}$
   d) $\boldsymbol{\beta}_0 = \mathbf{e}$, $\tau_0 = 10^{-8}$
   e) $\boldsymbol{\beta}^* = -\mathbf{e}$
2. *Linear function, rank 1*
   a) n variable, $m \geq n$
   b) $\mathbf{f}(\mathbf{x};\boldsymbol{\beta}) = \mathbf{x}\boldsymbol{\beta} - \mathbf{e}$
   c) $\mathbf{x} = \begin{bmatrix} 1 \\ \vdots \\ m \end{bmatrix} \begin{bmatrix} 1 & \cdots & n \end{bmatrix}$, $\mathbf{y} = \mathbf{0}$
   d) $\boldsymbol{\beta}_0 = \mathbf{e}$, $\tau_0 = 10^{-8}$
   e) Any $\boldsymbol{\beta}^*$ such that $\begin{bmatrix} 1 & \cdots & n \end{bmatrix} \boldsymbol{\beta}^* = \frac{3}{2m+1}$
3. *Rosenbrock function*
   a) $n = m = 2$
   b) $\mathbf{f}(\boldsymbol{\beta}) = \begin{bmatrix} 10(\beta_2 - \beta_1^2) \\ 1 - \beta_1 \end{bmatrix}$
   c) $\mathbf{y} = \mathbf{0}$
   d) $\boldsymbol{\beta}_0 = \begin{bmatrix} -1.2 & 1 \end{bmatrix}^\top$, $\tau_0 = 1$
   e) $\boldsymbol{\beta}^* = \mathbf{e}$
4. *Powell singular function*
   a) $n = m = 4$
   b) $\mathbf{f}(\boldsymbol{\beta}) = \begin{bmatrix} \beta_1 + 10\beta_2) \\ \sqrt{5}(\beta_3 + \beta_4) \\ (\beta_2 - 2\beta_3)^2 \\ \sqrt{10}(\beta_1 - \beta_4)^2 \end{bmatrix}$
   c) $\mathbf{y} = \mathbf{0}$
   d) $\boldsymbol{\beta}_0 = \begin{bmatrix} 3 & -1 & 0 & 1 \end{bmatrix}^\top$, $\tau_0 = 10^{-8}$
   e) $\boldsymbol{\beta}^* = \mathbf{0}$
5. *Freudenstein and Roth function*
   a) $n = m = 2$

[1]We only use the `LinearAlgebra` and `ForwardDiff` packages.

b) $\mathbf{f}(\boldsymbol{\beta}) = \begin{bmatrix} \beta_1 - \beta_2(2 - \beta_2(5 - \beta_2)) - 13 \\ \beta_1 - \beta_2(14 - \beta_2(1 + \beta_2)) - 29 \end{bmatrix}$
c) $\mathbf{y} = \mathbf{0}$
d) $\boldsymbol{\beta}_0 = \begin{bmatrix} 0.5 & 2 \end{bmatrix}^\top, \ \tau_0 = 1$
e) $\boldsymbol{\beta}^* \approx \begin{bmatrix} 11.4128 & -0.896805 \end{bmatrix}^\top$

6. *Bard function*
a) $n = 3, \ m = 15$
b) $\mathbf{f}_i(\boldsymbol{\beta}) = y_i - \left(\beta_1 + \frac{u_i}{\beta_2 v_i + \beta_3 w_i}\right)$
c) $u_i = i, \ v_i = 16 - i, \ w_i = \min\{u_i, v_i\}$

TABLE 1
Data for Problem 6 (Bard)

i	y_i	i	y_i	i	y_i
1	0.14	6	0.32	11	0.73
2	0.18	7	0.35	12	0.96
3	0.22	8	0.39	13	1.34
4	0.25	9	0.37	14	2.10
5	0.29	10	0.58	15	4.39

d) $\boldsymbol{\beta}_0 = \mathbf{e}, \ \tau_0 = 10^{-8}$
e) $\boldsymbol{\beta}^* \approx \begin{bmatrix} 0.82411 & 1.133036 & 2.343695 \end{bmatrix}^\top$

7. *Box 3-dimensional function*
a) $n = 3, \ m \geq 3$ variable
b) $\mathbf{f}_i(\mathbf{x}; \boldsymbol{\beta}) = e^{-\beta_1 x_i} - e^{-\beta_2 x_i} - \beta_3(e^{-x_i} - e^{-10 x_i})$
c) $x_i = i/10, \ \mathbf{y} = \mathbf{0}$
d) $\boldsymbol{\beta}_0 = \begin{bmatrix} 0 & 10 & 20 \end{bmatrix}^\top, \ \tau_0 = 10^{-8}$
e) $\boldsymbol{\beta}^* = \begin{bmatrix} 1 & 10 & 1 \end{bmatrix}^\top$ or $\begin{bmatrix} 10 & 1 & -1 \end{bmatrix}^\top$ or $\begin{bmatrix} a & a & 0 \end{bmatrix}^\top$ for $a \in \mathbb{R}$

8. *Jennrich and Sampson function*
a) $n = 2, \ m \geq 2$ variable
b) $\mathbf{f}_i(\boldsymbol{\beta}) = 2 + 2i - (e^{\beta_1 i} + e^{\beta_2 i})$
c) $\mathbf{y} = \mathbf{0}$
d) $\boldsymbol{\beta}_0 = \begin{bmatrix} 0.3 & 0.4 \end{bmatrix}^\top, \ \tau_0 = 1$
e) For $m = 5, \ \boldsymbol{\beta}^* \approx \begin{bmatrix} 0.378468 & 0.378468 \end{bmatrix}^\top$
For $m = 10, \ \boldsymbol{\beta}^* \approx \begin{bmatrix} 0.257825 & 0.257825 \end{bmatrix}^\top$
For $m = 20, \ \boldsymbol{\beta}^* \approx \begin{bmatrix} 0.165191 & 0.165191 \end{bmatrix}^\top$

9. *Osborne 1 function*
a) $n = 5, \ m = 33$
b) $\mathbf{f}_i(\mathbf{x}; \boldsymbol{\beta}) = \beta_1 + \beta_2 e^{-\beta_4 x_i} + \beta_3 e^{-\beta_5 x_i}$
c) $x_i = 0.02i$
d) $\boldsymbol{\beta}_0 = \begin{bmatrix} 0.5 & 1.5 & -1 & 0.01 & 0.02 \end{bmatrix}^\top, \ \tau_0 = 10^{-8}$
e) $\boldsymbol{\beta}^* \approx \begin{bmatrix} 0.37541 & 1.93585 & -1.46469 & 0.01287 & 0.02212 \end{bmatrix}^\top$

10. *Exponential fit*
a) $n = 4, \ m = 45$
b) $\mathbf{f}_i(\mathbf{x}; \boldsymbol{\beta}) = \beta_3 e^{-\beta_1 x_i} + \beta_4 e^{-\beta_2 x_i}$
c) $x_i = 0.02i$
d) $\boldsymbol{\beta}_0 = \begin{bmatrix} -1 & -2 & 1 & -1 \end{bmatrix}^\top, \ \tau_0 = 10^{-3}$

TABLE 2
Data for Problem 9 (Osborne 1)

i	y_i	i	y_i	i	y_i
1	0.844	12	0.718	23	0.478
2	0.908	13	0.685	24	0.467
3	0.932	14	0.658	25	0.457
4	0.936	15	0.628	26	0.448
5	0.925	16	0.603	27	0.438
6	0.908	17	0.580	28	0.431
7	0.881	18	0.558	29	0.424
8	0.850	19	0.528	30	0.420
9	0.818	20	0.522	31	0.414
10	0.784	21	0.506	32	0.411
11	0.751	22	0.490	33	0.406

TABLE 3
Data for Problem 10 (Exponential fit)

i	y_i	i	y_i	i	y_i
1	0.090542	16	0.288903	31	0.150874
2	0.124569	17	0.300820	32	0.126220
3	0.179367	18	0.303974	33	0.126266
4	0.195654	19	0.283987	34	0.106384
5	0.269707	20	0.262078	35	0.118923
6	0.286027	21	0.281593	36	0.091868
7	0.289892	22	0.267531	37	0.128926
8	0.317475	23	0.218926	38	0.119273
9	0.308191	24	0.225572	39	0.115997
10	0.336995	25	0.200594	40	0.105831
11	0.348371	26	0.197375	41	0.075261
12	0.321337	27	0.182440	42	0.068387
13	0.299423	28	0.183892	43	0.090823
14	0.338972	29	0.152285	44	0.085205
15	0.304763	30	0.174028	45	0.067203

e) $\boldsymbol{\beta}^* \approx \begin{bmatrix} -4 & -5 & 4 & -4 \end{bmatrix}^\top$

Problems 9 and 10 are common models used in real-life data fitting problems, e.g. the concentration of medicine in blood as a function of time [6], economic growth, virus spread etc.

4.3. Results and Discussion. We compare both the computation time and the number of iterations across the different algorithms for each of the 10 problems. The results are presented in Table 4 and Table 5.

Taken together, we find that for non-linear least squares problems, the Levenberg-Marquardt algorithm generally performs better than gradient descent and Gauss-Newton. In most cases, gradient descent and Gauss-Newton fail to converge but when Gauss-Newton converges, it is generally faster than Levenberg-Marquardt. This makes sense since Levenberg-Marquardt's improvement over Gauss-Newton is not so much speed but avoiding the problem of diverging iterates when the fit function cannot be approximated as a first order Taylor series locally.

Table 4

Number of iterations taken for each algorithm to solve each of the least squares problems. Maximum number of iterations in each case is 10000. We note cases where convergence is not achieved with an 'NA'. Each iteration involves one Jacobian computation unless otherwise noted by the number in parentheses. We bold the algorithm with the shortest number of Jacobian evaluations. If they are equal, we compare the number of iterations.

Pno	n	m	GD	GN	LM	LM(N)
1	4	100	10000	**1**	3	3
2	4	100	NA	NA	**4**	**4**
3	2	2	NA	**2**	28	29
4	4	4	NA	**10**	15	15
5	2	2	NA	**43**	101(72)	**57(41)**
6	3	15	10000	**6**	15(11)	17(10)
7	3	100	NA	NA	**10(9)**	15(10)
8	2	5	NA	NA	**37(18)**	37(19)
	2	10	NA	NA	43(21)	**38(21)**
	2	20	NA	NA	37(22)	**34(22)**
9	5	33	NA	**6**	18(15)	18(15)
10	4	45	NA	NA	212(182)	**183(178)**

Table 5

Time taken for each algorithm to solve each of the least squares problems, in seconds. We run each algorithm 10 times, after clearing cache and remove the top 2 and bottom 2 results to get rid of outliers before taking the average of the remaining 6 computation timings. We note cases where convergence is not achieved with an 'NA'. We bold the fastest time for each problem.

Pno	n	m	GD	GN	LM	LM(N)
1	4	100	2.503	2.772	1.321	**1.231**
2	4	100	NA	NA	1.279	**1.251**
3	2	2	NA	**1.120**	1.201	**1.120**
4	4	4	NA	**1.172**	1.194	1.212
5	2	2	NA	**1.183**	1.214	**1.208**
6	3	15	1.780	1.530	**1.527**	1.550
7	3	100	NA	NA	1.613	1.575
8	2	5	NA	NA	**1.705**	1.776
	2	10	NA	NA	0.268	**0.260**
	2	20	NA	NA	**0.260**	**0.260**
9	5	33	NA	**1.614**	1.671	1.717
10	4	45	NA	NA	1.765	**1.761**

In Problem 5, we highlight the results of both Gauss-Newton and Levenberg-Marquardt (Nielsen updating rule) since the former arrived at the global minimum while the latter arrived at the local minimum. To investigate why this is the case, we plotted the convergence path of $\boldsymbol{\beta}$ for all three algorithms (Figure 1), we find that in both Levenberg-Marquardt implementations, the estimates take a fairly straightforward path towards the local minimum whereas for Gauss-Newton, the estimate seems to go all over the $\mathbb{R}^2$ space until it eventually hits the global minimum. Even when we start at $\boldsymbol{\beta_0} = [11.4, -0.9]^\top$, Gauss-Newton still converges to the global minimum. Although it may then be tempting to suggest that Gauss-Newton is a better algorithm since it seems to guarantee convergence towards a global optimum, this is not be the case in general. Rather, it is this "uncontrolled" behavior that causes Gauss-Newton to not converge at all for several of the other non-linear least squares problems.

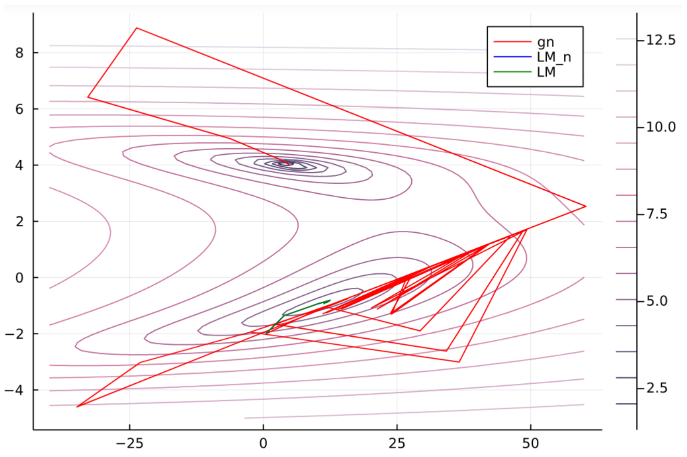

FIG. 1. *Convergence path of $\boldsymbol{\beta}$ for Gauss-Newton and Levenberg-Marquardt algorithms. Contour map is plotted for* $f(\boldsymbol{\beta}) = \log((\beta_1 - \beta_2(2 - \beta_2(5 - \beta_2)) - 13)^2 + (\beta_1 - \beta_2(14 - \beta_2(1 + \beta_2)) - 29)^2)$. *For both LM algorithms (green and blue), the estimates take a straightforward path towards the local minimum. For Gauss-Newton (red), the estimates are all over the place but eventually reach the global minimum.*

On the other hand, Levenberg-Marquardt's improvement over gradient descent is speed and in the cases where gradient descent converges, we find that Levenberg-Marquardt is indeed faster. In fact, for Problem 6, the final iterate for gradient descent (at iteration 10000) has not necessarily converged as the gradient of the sum of squared errors is in the order of 10^{-5} which is still several orders larger than the tolerable error of 10^{-12}.

Between the original algorithm and Levenberg-Marquardt with Nielsen's updating rule, we find that the latter does lead to smoother convergence, as we see in Problem 10 (Figure 2). λ increases more sharply at the start with the Nielsen updating rule, and once it decreases, it takes a fairly smooth path until convergence. In the original algorithm, λ exhibits a rugged behavior where it has a step increase for every decrease and this contributes to the norm of the gradient exhibiting a rugged behavior too, causing convergence to be less smooth and take a larger number of iterations.

Across all 10 problems, we find that the Nielsen updating rule does lead to better performance, having the faster computation time amongst the two for 7 of 12 cases, shorter convergence as measured by the number of Jacobian evaluations for 5 of 12 cases, and tying in 4 cases.

5. Conclusions. In this paper, we review the Levenberg-Marquardt algorithm, an important method to solve non-linear least squares problem. We compare this method to two other methods that predate its inception—gradient descent and Gauss-Newton and explain how the Levenberg-Marquardt is really a mix of both algorithms and combines the best of both. We implement all three algorithms in Julia and compare them using a series of test problems. We also implement a different updating rule for comparison. We find that the Levenberg-Marquardt algorithm strictly outperforms gradient descent and although there are test problems for which Gauss-Newton

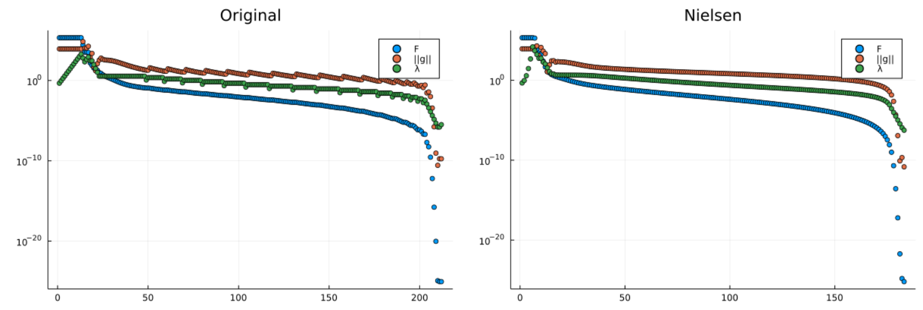

FIG. 2. *Plots of F (sum of squared errors), $||g||$ (norm of gradient of F), λ (damping parameter) against the number of iterations for both implementations of Levenberg-Marquardt in solving Problem 10 (exponential fit)*

achieves faster convergence, Levenberg-Marquardt is probably a better choice for general usage. Between the two updating rules tested, we find that the Nielsen updating rule achieves smoother and generally faster convergence.

REFERENCES

[1] H. P. Gavin, *The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems*, Department of Civil and Environmental Engineering, Duke University, 19 (2019).

[2] K. LEVENBERG, *A method for the solution of certain non-linear problems in least squares*, Quarterly of Applied Mathematics, 2 (1944), pp. 164–168.

[3] K. Madsen, *A combined Gauss-Newton and Quasi-Newton method for non-linear least squares*, 1988.

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

[5] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, 1978.

[6] H. B. Nielsen, *Damping parameter in Marquardt's method*, 1999.