Algorithms for Non-Linear Least Squares

Pramook Khungurn

November 26, 2024

This note is written as I study algorithms for the non-linear least squares problem, which will culminate in the Levenberg–Marquardt algorithm. Source materials I use for this note includes Madsen et al. [5], Frandsen et al. [2], and Norcedal and Wright [8].

1 Notations

- Scalars are denoted by regular (i.e., non-bold, non-italic) small latters: a, b, c, x, y, and z.
- We denote vectors with bold, small letters: $\mathbf{a}$, $\mathbf{b}$ $\mathbf{c}$, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$.
- Vector components are denoted with regular, small letters with a subscript. For example,

$$\mathbf{x} = (x_1, x_2, \ldots, x_n).$$

- Matrices, on the other hand, are denoted by regular, capital letters: A, B, and C.
- Scalar functions use the same type face as scalars, and vector functions use the same type face as vectors.
- Component functions of vector functions are denoted in the same was a vector components. That is, if $f : \mathbb{R}^n \to \mathbb{R}^m$, then

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix}.$$

- For derivatives, we use the notations I developed in a previous note [3].

2 Preliminary

- We are interested in solving the **non-linear least squares** problem. That is, we are given a non-linear vector function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, and we want to find $\mathbf{x}^*$ such that $\|\mathbf{f}(\mathbf{x}^*)\|^2$ is minimized. In other words, we want to compute

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \left\{ \|\mathbf{f}(\mathbf{x})\|^2 \right\}.$$

- Of course, non-linear least square is a special case of the general **function minimization** problem. Here, we are given a **loss function**, aka an **objective function**, $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$. We want to find

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \left\{ \mathcal{L}(\mathbf{x}) \right\}.$$

- Obviously, for the non-linear least square problem, we have that $\mathcal{L}(\mathbf{x}) = \|\mathbf{f}(\mathbf{x})\|^2$.
- Finding $\arg\min_{\mathbf{x}}\{\mathcal{L}(\mathbf{x})\}$ (i.e., the **global minimizer**) is very hard in general, especially with non-linear $\mathbf{f}$. So, we settle to find a local minimizer instead.

  Definition 1. *A point $\mathbf{x} \in \mathbb{R}^n$ is said to be a* **local minimizer** *of $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ if there exists $\delta > 0$ such that $\mathcal{L}(\mathbf{x}') \leq \mathcal{L}(\mathbf{x})$ for all $\|\mathbf{x} - \mathbf{x}'\| < \delta$. In other words, there exists a neighborhood of $\mathbf{x}$ such that $\mathcal{L}(\mathbf{x})$ is minimal.*

- We shall assume that $\mathcal{L}$ is differentiable to arbitrary order. As a result, we have that

$$\mathcal{L}(\mathbf{x} + \mathbf{h}) = \mathcal{L}(\mathbf{x}) + \nabla\mathcal{L}(\mathbf{x})\mathbf{h} + \frac{1}{2}\mathbf{h}^T H_{\mathcal{L}}(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^3)$$

  where $H_{\mathcal{L}}(\mathbf{x})$ denotes the Hessian matrix of $\mathcal{L}$:

$$H_{\mathcal{L}}(\mathbf{x}) = \begin{bmatrix} \nabla_{1,1}\mathcal{L}(\mathbf{x}) & \nabla_{1,2}\mathcal{L}(\mathbf{x}) & \cdots & \nabla_{1,n}\mathcal{L}(\mathbf{x}) \\ \nabla_{2,1}\mathcal{L}(\mathbf{x}) & \nabla_{2,2}\mathcal{L}(\mathbf{x}) & \cdots & \nabla_{2,n}\mathcal{L}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \nabla_{n,1}\mathcal{L}(\mathbf{x}) & \nabla_{n,2}\mathcal{L}(\mathbf{x}) & \cdots & \nabla_{n,n}\mathcal{L}(\mathbf{x}) \end{bmatrix} = \nabla((\nabla\mathcal{L}(\mathbf{x}))^T).$$

- **Definition 2.** *A point $\mathbf{x} \in \mathbb{R}^n$ is said to be a* **stationary point** *of $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ if $\nabla L(\mathbf{x}) = \mathbf{0}^T$.*
- **Theorem 3.** *A local minimizer of $\mathcal{L}$ is a stationary point.*

  In other words, being a staionary point is a necessary condition for being a local minimizer. It is not a sufficient condition because a stationary point can be a *local maximizer* or a *saddle point*
- The sufficient condition for a local minimizer is given below.

  Theorem 4. *If $\mathbf{x}$ is a stationary point of $\mathcal{L}$, and $H_{\mathcal{L}}(\mathbf{x})$ is positive definite, then $\mathbf{x}$ is a local minimizer.*

3 Descent Methods

- In this section, we present methods for the general function minimzation problem. We will deal with methods specific to non-linear least squares in the sections after this one.
- The methods in this section are iterative.
  - Start with a starting point $\mathbf{x}^{(0)}$.
  - We produce a series of points $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ...

  Then, we pray that the series of points would converge to a local minimizer $\mathbf{x}^*$.
- Let $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*$. The optimization process converges if $\lim_{k\to\infty} \|\mathbf{e}^{(k)}\| = 0$.
- We are interested in quantifying how the optimization process converges. The speed at which it converges can be measured by how much $\|\mathbf{e}^{(k+1)}\|$ becomes smaller than $\|\mathbf{e}^{(k)}\|$. There are multiple types of convergence
  - **Linear convergence** is when $\|\mathbf{e}^{(k+1)}\| \leq a\|\mathbf{e}^{(k)}\|$ for some $0 < a < 1$ for all k large enough.
  - **Quadratic convergence** is when $\|\mathbf{e}^{(k+1)}\| = O(\|\mathbf{e}^{(k)}\|^2)$ for all k large enough that $\|\mathbf{e}^{(k)}\|$ is small.
  - **Superlinear convergence** is when $\|\mathbf{e}^{(k+1)}\|/\|\mathbf{e}^{(k)}\| \to 0$ as $k \to \infty$.

Algorithm 1 Descent method

$\mathbf{x} \leftarrow \mathbf{x}^{(0)}$
while true do
  $\mathbf{h} \leftarrow$ FIND-DIRECTION($\mathbf{x}$)
  if (no such $\mathbf{h}$ exists) **then**
    return α
  else
    $\alpha \leftarrow$ COMPUTE-STEP-LENGTH($\mathbf{x}, \mathbf{h}$)
    $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{h}$
  end if
end while

- Most methods try to ensure the *descending condition*

$$\mathcal{L}(\mathbf{x}^{(k+1)}) < \mathcal{L}(\mathbf{x}^{(k)})$$

  for all $k \geq 0$. This prevents convergence to a local maximizer. However, we might still end up at a saddle point, but it is quite unlikely because a saddle point has directions that would increase the loss function's value. A **descent method** is one that tries to maintain the descending condition.

- All methods in this note is a descent method with a particular structure. In each iteration of such a method, we do the following.
  - Find a direction $\mathbf{h}$ along which to move $\mathbf{x}^{(k)}$. (Here, k is the index of the iteration.)
  - Find the step length to move $\mathbf{x}^{(k)}$.
- The outline of the descent method is given in Algorithm 1.
- Consider how the value of $\mathcal{L}$ changes after the update.

$$\mathcal{L}(\mathbf{x} + \alpha\mathbf{h}) = \mathcal{L}(\mathbf{x}) + \alpha\nabla\mathcal{L}(\mathbf{x})\mathbf{h} + O(\alpha^2) \approx \mathcal{L}(\mathbf{x}) + \alpha\nabla\mathcal{L}(\mathbf{x})\mathbf{h}$$

  when α is small enough.

- **Definition 5.** *A direction* $\mathbf{h}$ *is a* **descent direction** *if* $\nabla\mathcal{L}(\mathbf{x})\mathbf{h} < 0$.
- If we are at $\mathbf{x}$ where there exists no descent direction, it must be that $\nabla\mathcal{L}(\mathbf{x}) = \mathbf{0}$, so $\mathbf{x}$ is a stationary point.
- If there is a descent direction, we then have to find the step length α such that $\mathcal{L}(\mathbf{x} + \alpha\mathbf{h}) < \mathcal{L}(\mathbf{x})$.

3.1 Gradient Descent

- One of the most popular descent method is **gradient descent**. It choose $\mathbf{h} = -(\nabla\mathcal{L}(\mathbf{x}))^T$ and α to be a fixed small constant.
- The choice of the descent direction is the best choice locally at $\mathbf{x}^{(k)}$.
- However, the convergence of gradient descent is linear and often very slow.
- Nevertheless, it offers good performance when $\mathbf{x}$ is far away the converged solution $\mathbf{x}^*$. So, it is often used as the initial phase of the iterative optimization process.

3.2 Newton's Method

- If $\mathbf{x}^*$ is a stationary point then, $\nabla \mathcal{L}(\mathbf{x}^*) = \mathbf{0}$.
- We also have that

$$\begin{aligned}\nabla \mathcal{L}(\mathbf{x}+\mathbf{h})^T &= \nabla \mathcal{L}(\mathbf{x})^T + \nabla(\nabla \mathcal{L}(\mathbf{x})^T)\mathbf{h} + O(\|\mathbf{h}\|^2) \\ &= \nabla \mathcal{L}(\mathbf{x})^T + H_{\mathcal{L}}(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2) \\ &\approx \nabla \mathcal{L}(\mathbf{x})^T + H_{\mathcal{L}}(\mathbf{x})\mathbf{h}\end{aligned}$$

  when $\|\mathbf{h}\|$ is small enough.

- Assuming that we can get to the stationary point in one step, it must be the case that $\nabla \mathcal{L}(\mathbf{x}+\mathbf{h})^T = \mathbf{0}$. So,

$$\mathbf{0} \approx \nabla \mathcal{L}(\mathbf{x})^T + H_{\mathcal{L}}(\mathbf{x})\mathbf{h}.$$

  We go one step further and assume that the above inequality is an equality.

$$\mathbf{0} = \nabla \mathcal{L}(\mathbf{x})^T + H_{\mathcal{L}}(\mathbf{x})\mathbf{h}.$$

  This gives

$$\mathbf{h} = -(H_{\mathcal{L}}(\mathbf{x}))^{-1}\nabla \mathcal{L}(\mathbf{x})^T.$$

- **Newton's method** chooses $\mathbf{h} = -(H_{\mathcal{L}}(\mathbf{x}))^{-1}\nabla \mathcal{L}(\mathbf{x})^T$ and $\alpha = 1$.
- If $H_{\mathcal{L}}(\mathbf{x})$ is positive definite, we have that

$$0 < \mathbf{h}^T H_{\mathcal{L}}(\mathbf{x})\mathbf{h}.$$

  Because of our choice of $\mathbf{h}$, we have that $H_{\mathcal{L}}(\mathbf{x})\mathbf{h} = -\mathcal{L}(\mathbf{x})^T$. As a result,

$$\begin{aligned}0 &< -\mathbf{h}^T \mathcal{L}(\mathbf{x})^T \\ \mathcal{L}(\mathbf{x})^T\mathbf{h} &< 0.\end{aligned}$$

  As a result, $\mathbf{h}$ is a descent direction if $H_{\mathcal{L}}(\mathbf{x})$ is positive definite.

- Newton's method is good at a late stage of the optimization process. That is, whne $\mathbf{x}$ is close to $\mathbf{x}^*$. If $\mathbf{x}^*$ is a local minimizer, then $H_{\mathcal{L}}(\mathbf{x}^*)$ is positive definite. As a result, there is a neighborhood around $\mathbf{x}^*$ such that, if $\mathbf{x}$ is in that neighborhood, then $H_{\mathcal{L}}(\mathbf{x})$ is also positive definite. As a result, we have that $\mathbf{h}$ is a descent direction. Moreover, the convergence in this neighborhood is quadratic.
- On the other hand, if the $H_{\mathcal{L}}(\mathbf{x}^*)$ is negative definite, then $\mathbf{h}$ will be an *ascent* direction, and the algorithm would converge to a local maximizer at a qudratic rate as well. To prevent this, we should always ensure that each update results in a decrease of the loss function.
- We can build a hybrid method between gradient descent and Newton's method with the following simple rule. If $H_{\mathcal{L}}(\mathbf{x})$ is positive definite, then we use Newton's method for the current optimization step. Otherwise, we use gradient descent. The pseudocode is given in Algorithm 2.
- We can check whether a matrix is positive definite by performing the Cholesky factorization, which, if successful, would imply that the matrix is positive definite and also give us a factorization to compute $-(H_{\mathcal{L}}(\mathbf{x}))^{-1}\nabla \mathcal{L}(\mathbf{x})^T$. Factorization takes $O(n^3)$ where n is the number dimension of $\mathbf{x}$.

Algorithm 2 A hybrid method between Netwon's and gradient descent

if $H_{\mathcal{L}}(\mathbf{x})$ is postiive definite **then**
  $\mathbf{h} \leftarrow -(H_{\mathcal{L}}(\mathbf{x}))^{-1}\nabla\mathcal{L}(\mathbf{x})^T$
  $\alpha \leftarrow 1$
else
  $\mathbf{h} \leftarrow -\nabla\mathcal{L}(\mathbf{x})^T$
  $\alpha \leftarrow$ small positive constant
end if

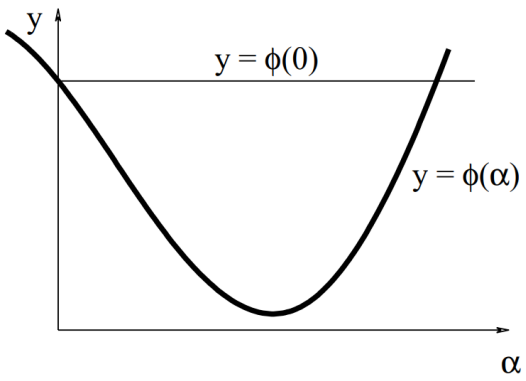

Figure 1: Loss value as a function $\varphi(\alpha)$ of the step length α when $\mathbf{h}$ is a descent direction [5].

3.3 Line Search

- We see that, in the last two sections, the choice of the step length α does not always work.
  - For gradient descent, a constant α does not guarantee the descending condition in any way possible.
  - For Newton's method, $\alpha = 1$ only works when the Hessian is positive definite.
- A **line search** is an algorithm to find an α such the descending condition is true.
- For a fixed $\mathbf{x}$ and a descent direction $\mathbf{h}$ and $\alpha \geq 0$, let

$$\varphi(\alpha) = \mathcal{L}(\mathbf{x} + \alpha\mathbf{h}).$$

- We have that

$$\varphi'(\alpha) = \frac{\partial\mathcal{L}(\mathbf{x}+\alpha\mathbf{h})}{\partial(\mathbf{x}+\alpha\mathbf{h})}\frac{\partial(\mathbf{x}+\alpha\mathbf{h})}{\alpha} = \nabla\mathcal{L}(\mathbf{x}+\alpha\mathbf{h})\mathbf{h}.$$

  As a result,

$$\varphi'(0) = \nabla\mathcal{L}(\mathbf{x})\mathbf{h}.$$

  Because $\mathbf{h}$ is a descent direction, we have that $\varphi'(0) < 0$. So, φ decreases first before potentially increases later. See Figure 1. As a result, there is $\alpha > 0$ such that $\varphi(\alpha) < \varphi(0)$. This means that the line search problem always has an answer if $\mathbf{h}$ is a descent direction.

- It is tempting to find $\alpha^* = \arg\min_{\alpha>0}\{\varphi(\alpha)$. This results in an algorithm called **exact line search**. However, this is still a hard optimization problem. Even if we allow ourselves to find a local minimizer instead of the global minimizer, literature still considers it not cost effective.

Algorithm 3 Backtracking line search

$\alpha \leftarrow \alpha^{(0)}$
while α does not satisfy some conditions **do**
    Decrease α.
end while

Algorithm 4 Interval binary search

Find an interval $[a, b]$ where α should be sampled from.
Sample α from $[a, b]$.
while (α does not satisfy some condition **do**
    Update $[a, b]$ to either $[a, \alpha]$ or $[\alpha, b]$.
    Sample α from $[a, b]$.
end while
return α

- Instead, what people do is **inexact line search** where we produce an α value that is not too short and not too long.
  - If α is too short, then $\varphi(0) - \varphi(\alpha)$ might be too small to call it an improvement, implying slow convergence.
  - If α is too large, then $\varphi(\alpha)$ might be greater than $\varphi(0)$.

3.3.1 Forms of Line Search Algorithm

- Inexact line search is generally an iterative algorithm.
  - Starts with an initial guess $\alpha \leftarrow \alpha^{(0)}$.
  - Keeps updating the value of α until it satisfies some conditions.
- In **backtracking line search** [8] we start with a large initial guess, say $\alpha^{(0)} = 1$. We keep decreasing the value of α until it satisfies some condition. See Algorithm 3 for the pseudocode.
- In **interval binary search** [2], we find an interval $[a, b]$ where an α value would be sampled from. After we sample α, we would update $[a, b]$ like what we do in binary search: either turning it into $[a, \alpha]$ or $[\alpha, b]$. We keep doing this until our sampled α satisifed some conditions. The pseudocode of the algorithm is given in Algorithm 4.

3.3.2 Backtracking Armijo Line Search

- The first concrete example of a line search algorithm uses a condition called the "Armijo condition" [8] with the backtracking line search template.
- The **Armijo condition** is as follow:

$$\varphi(\alpha) \leq \varphi(0) + c_1 \cdot \alpha \cdot \varphi'(0)$$

  where $0 < c_1 < 1$, and c_1 is often chosen to be around 10^{-4} [8]. It is depicted in Figure 2.
- We note that the Armijo condition always hold at $\alpha = 0$.
- If we graph φ as a function of α, the Armijo condition requires that we find α such that $(\alpha, \varphi(\alpha))$ is below the line of slope $\gamma\varphi'(0)$ with vertical intercept $\varphi(0)$. In other words, it requires that the loss function decreases by some sufficient amount, determined by γ.

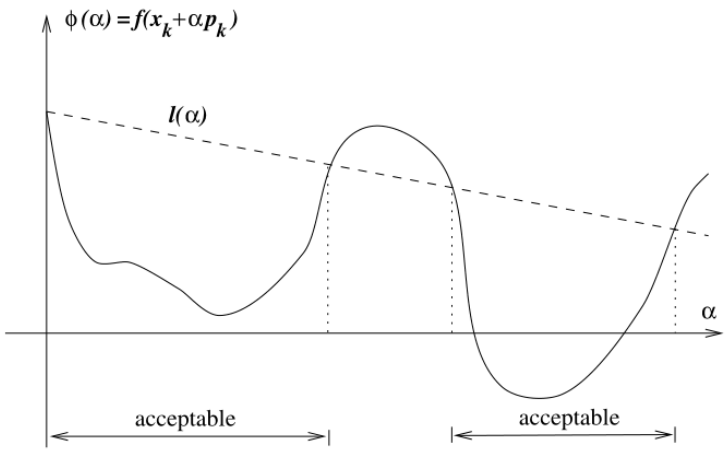

Figure 2: The Armijo condition.

Algorithm 5 Backtracking Armijo line search

```
procedure BACKTRACKING-ARMIJO-LINE-SEARCH
    α ← α^(0)
    while φ(α) > φ(0) + c1 α φ'(0) do
        α ← ρα
    end while
    return α
end procedure
```

  - As a result, the Armijo condition is sometimes referred to as the **sufficient decrease condition**.
- The way we decrease α is to exponentially decay it. We pick a constant $0 < \rho < 1$ and update $\alpha \leftarrow \rho\alpha$.
- The pseudocode of the full algorithm is given in Algorithm 5.
- The algorithm is gauranteed to terminate because the Armijo condition is always satisfied when α is low enough.
- A drawback of the algorithm is that there is no conditions that prevent α from being too small.

3.3.3 Line Search with Wolfe Conditions

- The **Wolfe conditions** [8] consists of two conditions. The first condition ensures that α is small enough, and the second conditions ensures that α is not too small.
  - The first condition is just the Armijo condition with the extra requirement that $c_2 < 0.5$ [5].
  - The second condition is called the **curvature condition**:

    $$\varphi'(\alpha) \geq c_2 \cdot \varphi'(0)$$

    where $c_1 < c_2 < 1$, and c_2 is often picked to be much greater than c_1, typically in the order of 0.1 [1]. See Figure 3.
  - The curvature condition requires that the gradient increases by some amount, making sure that the picked α yields $\mathbf{x}$ that is close to a stationary point.

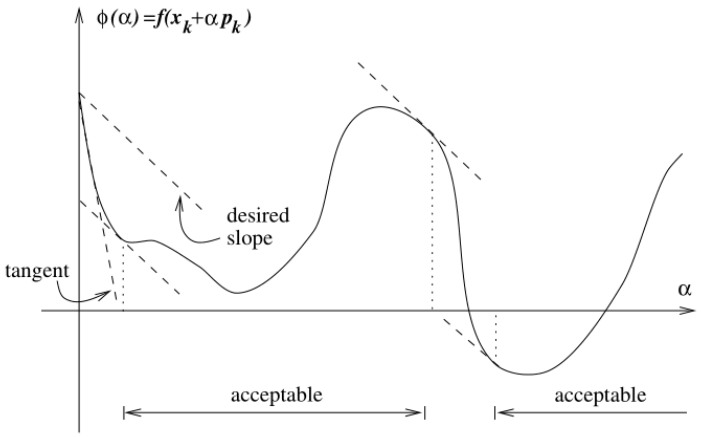

Figure 3: Curvature condition [1, 8].

- There is a stronger version of the Wolfe conditions, and they are known as the **strong Wolfe conditions**. The strong version differs from the regular one by changing the curvature condition to the following **strong curvature condition**:

$$|\varphi'(\alpha)| \leq c_2 \cdot |\varphi'(0)|.$$

  In other words, while the curvature condition does not have limits on how positive $\varphi'(\alpha)$ can be, the strong curvature condition requires that $\varphi'(\alpha)$ cannot be too positive. It should be clear that the strong curvature condition implies the regular curvature condition.

- We note that *both versions the curvature condition do not hold at* $\alpha = 0$.
  - If we substitute $\alpha = 0$, the curvature condition reads $\varphi'(0) \geq \beta \cdot \varphi'(0)$. In other words, $(1 - \beta)\varphi'(0) \geq 0$. This is false because $1 - \beta > 0$, but $\varphi'(0) < 0$.
  - On the other hand, the strong curvature condition reads $|\varphi'(0)| \leq c_2 \cdot |\varphi'(0)|$, which does not make sense because $|\varphi'(0)| > 0$ and $c_2 < 1$.

- The following lemma shows that there always exist a step length that satisfy both versions of the Wolfe conditions given that the loss function $\mathcal{L}$ is smooth and bounded below.

  Lemma 6. *Suppose that* $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ *is continuously differentiable. Let* $\mathbf{h}$ *be a descent direction at* $\mathbf{x}$. *Assume that* $\varphi(\alpha) = \mathcal{L}(\mathbf{x} + \alpha\mathbf{h})$ *is bounded below along ray* $0 \leq \alpha < \infty$. *Then, if* $0 < c_1 < c_2 < 1$, *there exists intervals of* α *values satisfying the Wolfe conditions and the strong Wolfe conditions.*

  The proof is in Norcedal and Wright [8].

- The line search algorithm in this section also from Norcedal and Wright. Being a binary-search type algorithm, has two phases. We now discuss the first phase where an interval $[a, b]$ containing an α value that satisfies the *strong* Wolfe condition is identified. The second phase is abstracted as a subroutine called Zoom$(\cdot, \cdot)$. The implementation of this subroutine will be discussed later.
  - The "zoom" name comes from Norcedal and Wright. I think this is not a good name.

- The pseudocode of the first phase is given in Algorithm 6.

Algorithm 6 Line search with strong Wolfe conditions

$a \leftarrow 0$
$b \leftarrow \min\{1, \alpha_{\max}\}$
while true do
  if $\varphi(b) > \varphi(0) + c_1 b \varphi'(0)$ **or** $\varphi(b) \geq \varphi(a)$ **then**
    return ZOOM(a, b)
  end if
  if $|\varphi'(b)| \leq c_2 |\varphi'(0)|$ **then**
    return b
  end if
  if $\varphi'(b) \geq 0$ **then**
    return ZOOM(b, a)
  end if
  $a \leftarrow b$
  if $a = \alpha_{\max}$ **then**
    return $\alpha_{\max}$
  end if
  $b \leftarrow \min\{2b, \alpha_{\max}\}$
end while

- The first phase uses the knowledge (stated but not proven in Norcedal and Wright that) that $[a, b]$ contains an α that satisflys the strong Wolfe conditions if at leaset one of the following three conditions hold:
  - b violates the Armijo condition. In other words, $\varphi(b) > \varphi(0) + c_1 b \varphi'(0)$.
  - $\varphi(b) \geq \varphi(a)$.
  - $\varphi'(b) \geq 0$.
- We now turn to the second phase, which is embodied by the ZOOM$(\cdot, \cdot)$ function. The pseudocode is given in Algorithm 7.
- The ZOOM$(\cdot, \cdot)$ function uses another function GET-STEP-LENGTH$(\alpha_{\text{lo}}, \alpha_{\text{hi}})$ to sample an α from an interval bounded by α_{lo} and α_{hi}. We shall discuss this function later.
- Let us discuss the parameters α_{lo} and α_{hi}.
  - We can see that ZOOM$(\cdot, \cdot)$ takes two arguments: α_{lo} and α_{hi}.
  - As should be appearent in the first phase (Algorithm 6), we do not require that $\alpha_{\text{lo}} < \alpha_{\text{hi}}$.
  - Instead, we require that the following properties hold at all times.
    - The interval bounded by α_{lo} and α_{hi} contains a step length that satisfy the strong Wolfe conditions.
    - α_{lo} satisfies the Armijo condition. Moreover, among all α values that satisfied the Armijo condition that have been generated so far, $\varphi(\alpha_{\text{lo}})$ has the smallest value.
    - α_{hi} is chosen so that $\varphi'(\alpha_{\text{lo}})(\alpha_{\text{hi}} - \alpha_{\text{lo}}) < 0$.
- One can check that the updates during the while loop of Algorithm 7 always maintain the above invariance. However, this might be quite tedious, and I will not be going through all the cases.
- Let us now discuss GET-STEP-LENGTH$(\cdot, \cdot)$. Norcedal and Wright presents a sophisticated algorithm that requires approximating φ with a cubic polynomial, but the presentation was not very clear to me. On the other hand, Frandsen et al. presents an algorithm based on approximating φ with a cubic polynomial, and we shall present the algorithm in this note. The pseudocode is given in Algorithm 8.

Algorithm 7 The second phase of the line search with strong Wolfe conditions.

procedure ZOOM($\alpha_{\text{lo}}, \alpha_{\text{hi}}$)
  while true do
    $\alpha \leftarrow$ GET-STEP-LENGTH($\alpha_{\text{lo}}, \alpha_{\text{hi}}$)
    if $\varphi(\alpha) > \varphi(0) + c_1\alpha\varphi'(0)$ **or** $\varphi(\alpha) \geq \varphi(\alpha_{\text{lo}})$ **then**
      $\alpha_{\text{hi}} \leftarrow \alpha$
      continue
    end if
    if $|\varphi(\alpha)| \leq c_2|\varphi'(0)|$ **then**
      return α
    end if
    if $\varphi'(\alpha)(\alpha_{\text{hi}} - \alpha_{\text{lo}}) \geq 0$ **then**
      $\alpha_{\text{hi}} \leftarrow \alpha_{\text{lo}}$
    end if
    $\alpha_{\text{lo}} \leftarrow \alpha$
  end while
end procedure

Algorithm 8 The second phase of the line search with strong Wolfe conditions.

procedure GET-STEP-LENGTH(a, b)
  $D \leftarrow b - a$
  $c \leftarrow (\varphi(b) - \varphi(a) - D\varphi'(a))/D^2$
  if $c > 0$ **then**
    $\alpha \leftarrow a - \varphi'(a)/(2c)$
    $a \leftarrow a + 0.1D$
    $b \leftarrow b - 0.1D$
    $a, b \leftarrow \min\{a, b\}, \max\{a, b\}$
    return $\min\{\max\{\alpha, a\}, b\}$.
  else
    return $(a + b)/2$
  end if
end procedure

- To explain how the function works, first note that the polynomial

$$\psi(\alpha) = \varphi(a) + \varphi'(a)(\alpha - a) + c(\alpha - a)^2,$$

  where

$$c = \frac{\varphi(b) - \varphi(a) - \varphi'(a)(b - a)}{(b - a)^2},$$

  satisfies the following properties:

  - $\psi(a) = \varphi(a)$,
  - $\psi'(a) = \varphi'(a)$, and
  - $\psi(b) = \varphi(b)$.

  So, $\psi(t)$ is an approximation of $\varphi(\alpha)$ in the interval bounded by a and b.

- If $c > 0$, then the polynomial is a parabola that opens up, so it has a global minimum. The minimizer α is determined by

$$\begin{aligned} \psi'(\alpha) &= 0 \\ \varphi'(a) + 2c(\alpha - a) &= 0 \\ \alpha &= a - \frac{\varphi'(a)}{2c}. \end{aligned}$$

  As a result, we choose this value as the candicate. However, α might fall outside the interval bounded by a and b or too close to a and b to prevent meaningful interval shrinking. So, we bound α so that the interval shrinks by at least 10%.

- If $c \leq 0$, then the polynomial does not have a global minimum, and the minimum is at one of the endpoints. In this case, we just chooise α to be the middle point between a and b.

- One thing to be aware of is that line search can be expensive because one needs to evaluate $\varphi(\alpha)$ and $\varphi'(\alpha)$ many times. When implementing one of these algorithms, we need to be vigilant and now overcompute stuffs.

3.4 Trust Region and Damped Methods

- We assume that we have a function $\mathtt{L} : \mathbb{R}^n \to \mathbb{R}$ that models the behavior of $\mathcal{L}$ in the neighborhood of the current solution $\mathbf{x}$.

$$\mathcal{L}(x + \mathbf{h}) \approx \mathtt{L}(\mathbf{h}).$$

- Usually, the model $\mathtt{L}$ is a quadratic function of the form:

$$\mathtt{L}(\mathbf{h}) = c + \mathbf{b}^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T A\mathbf{h}.$$

  where $c \in \mathbb{R}$, $\mathbf{b} \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix.

- In general, we want $\mathtt{L}$ to with $\mathcal{L}$ up to first order. As a result, we typically choose $c = \mathcal{L}(\mathbf{x})$ and $\mathbf{b} = \nabla\mathcal{L}(\mathbf{x})^T$. The matrix A is either the Hessian of $\mathcal{L}$ at $\mathbf{x}$ or some approximation of it.

- In a **trust region method**, we assume that we know a positive constant Δ such that the model is sufficiently accurate inside a ball with radius Δ around $\mathbf{x}$. Then, we choose the update direction by computing

$$\mathbf{h} = \underset{\|\mathbf{h}\|\leq\Delta}{\arg\min}\{\mathtt{L}(\mathbf{h})\}. \tag{1}$$

- In a **damped method**, we use the following minimization problem instead:

$$\mathbf{h} = \underset{\|\mathbf{h}\|\leq\Delta}{\arg\min}\left\{\mathtt{L}(\mathbf{h}) + \frac{1}{2}\mu\mathbf{h}^T\mathbf{h}\right\}. \tag{2}$$

  The term $\frac{1}{2}\mu\mathbf{h}^T\mathbf{h}$ penalize large update direction.

- The template for a trust region and damped method is given in Algorithm 9.

- We see that the trust region method is quite different from line search.

  – In line search, we come up with an update direction first. Then, we choose a step length to update along that direction.

Algorithm 9 A template for trusted region and damped methods.

while not satisifed **do**
  Compute $\mathbf{h}$ according to (1) or (2).
  if $\mathcal{L}(x+\mathbf{h}) < \mathcal{L}(\mathbf{x})$ **then**
    $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{h}$
  end if
  Update $\mathtt{L}$ and Δ or μ.
end while

  - In trust region method, we start with an initial guess of Δ and $\mathtt{L}$. We find an update direction $\mathbf{h}$. If $\mathbf{h}$ results in an improvement, we update along $\mathbf{h}$ with step length $\alpha = 1$. Otherwise, we do not move (i.e., $\alpha = 0$). In any case, we need to update Δ and $\mathtt{L}$.
    * If the previous update fails, we generally shrink Δ to make the model more accurate.
    * If the previous update success, we need to update $\mathcal{L}$ to take into account the new value of $\mathbf{x}$. We may then set Δ to a high value to make sure that we catch potential large updates at a new point.

- The improvement of the update step is determined by the **gain ratio** [5]:

$$\varrho = \frac{\mathcal{L}(\mathbf{x}) - \mathcal{L}(\mathbf{x}+\mathbf{h})}{\mathtt{L}(\mathbf{0}) - \mathtt{L}(\mathbf{h})}.$$

  The nominator is called the **actual decrease**, and the denominator is called the **predicted decrease** [8]. By construction, the predicted decrease is always positive.

- With a trust region method, the following strategy for updating Δ is widely used.

if $\varrho < 0.25$ **then**
  $\Delta \leftarrow \Delta/2$
else
  $\Delta \leftarrow \max\{\Delta, 3\|\mathbf{h}\|\}$
end if

- In a damped method, a small ϱ indicates that we should increase the damping factor μ. Otherwise, damping might be decreased because the model is a good approximation. A widely used strategy is proposed by Marquardt for the famed Levenberg–Marquardt algorithm [6].

if $\varrho < 0.25$ **then**
  $\mu \leftarrow 2\mu$
end if
if $\varrho > 0.75$ **then**
  $\mu \leftarrow \mu/3$
end if

- The method above is not sensitive to minor changes in the thresholds 0.25 and 0.75 or the numbers $p_1 = 2$ and $p_2 = 3$. However, p_1 and p_2 should be chosen so that the μ values would not oscillate, which would slow down convergence.

- It turns out that the threshold 0.25 and 0.75 are not that good. The following strategy by Nielsen performs better [7].

  $\nu \leftarrow 2$
  if $\varrho > 0$ **then**
    $\mu \leftarrow \mu \max\{1/3, 1 - (2\varrho - 1)^3\}$
    $\nu \leftarrow 2$
  else
    $\mu \leftarrow \mu\nu$
    $\nu \leftarrow 2\nu$
  end if

- In a damped method, the update $\mathbf{h}$ is computed by finding a statinary point of the function

$$\begin{aligned}\psi_\mu(\mathbf{h}) &= \mathcal{L}(\mathbf{h}) + \frac{1}{2}\mu\mathbf{h}^T\mathbf{h} \\ &= c + \mathbf{b}^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T A\mathbf{h} + \frac{1}{2}\mu\mathbf{h}^T\mathbf{h} \\ &= c + \mathbf{b}^T\mathbf{h} + \frac{1}{2}\mathbf{h}^T(A + \mu I)\mathbf{h}.\end{aligned}$$

  This requires that

$$(\nabla\psi_\mu(\mathbf{h}))^T = \mathbf{0},$$

  or

$$\begin{aligned}(A + \mu I)\mathbf{h} + \mathbf{b} &= 0 \\ \mathbf{h} &= -(A + \mu I)^{-1}\mathbf{b}.\end{aligned}$$

  If μ is sufficiently large, the symmetric matrix $A + \mu I$ is positive definite, and $\mathbf{h}$ would be a minimizer of ψ_u.

- In a trust region method, the step $\mathbf{h}$ is the solution to the *constrained* optimization problem

  minimize $\mathcal{L}(\mathbf{h})$
  subject to $\mathbf{h}^T\mathbf{h} \leq \Delta^2$

  However, we will not discuss how to solve this problem in this note.

3.5 Damped Newton's Method

- An illuminating example of damped methods is the **damped Newton's method**.
- The model $\mathtt{L}(\mathbf{h})$ is given by

$$\mathtt{L}(\mathbf{h}) = \mathcal{L}(\mathbf{x}) + \nabla\mathcal{L}(\mathbf{x})\mathbf{h} + \frac{1}{2}\mathbf{h}^T H_{\mathcal{L}}(\mathbf{x})\mathbf{h}.$$

- The update direction $\mathbf{h}_{\text{dn}}$ takes the form

$$(H_{\mathcal{L}}(\mathbf{x}) + \mu I)\mathbf{h}_{\text{dn}} = -\nabla\mathcal{L}(\mathbf{x})^T.$$

- When μ is small, the equation above becomes close to the equation of Newont's method, and the method's behavior becomes similar to that of Newton's method.
- However, when μ is large, we have that $\mathbf{h}_{\text{dm}} \approx -\frac{1}{\mu}\mathcal{L}(\mathbf{x})^T$, which is a short step in the direction of the gradient.
- As a result, we can think of the damped Newton's method has a hybrid between gradient descent and Newton's method.

4 Non-Linear Least Squares

- We are given a vector function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, and we want to find

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \left\{ \frac{1}{2} \|\mathbf{f}(\mathbf{x})\|^2 \right\} = \arg\min_{\mathbf{x}} \left\{ \frac{1}{2} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}) \right\}$$

- Provided that $\mathbf{f}$ has continuous second partial derivatives, we have that

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) = \mathbf{f}(\mathbf{x}) + \nabla \mathbf{f}(\mathbf{x})\mathbf{h} + O(\|\mathbf{h}\|^2)$$

  where $\nabla \mathbf{f}(\mathbf{x}) \in \mathbb{R}^{m \times n}$ is the derivative of $\mathbf{f}$ at $\mathbf{x}$. This is often called the **Jacobian matrix** and is usually abbreviate as just $J(\mathbf{x})$.

- Our loss function is given by

$$\mathcal{L}(\mathbf{x}) = \frac{1}{2} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}).$$

  The gradient of the loss function is given by

$$\nabla \mathcal{L}(\mathbf{x}) = \mathbf{f}(\mathbf{x})^T J(\mathbf{x}).$$

  Moreover, the Hessian is given by

$$H_{\mathcal{L}}(\mathbf{x}) = J(\mathbf{x})^T J(\mathbf{x}) + \sum_{i=1}^{m} f_i(\mathbf{x}) H_{f_i}(\mathbf{x}).$$

4.1 The Gauss–Newton Method

- The method is based on a linear approximation to the components of $\mathbf{f}$. When $\|\mathbf{h}\|$ is small, we have that

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

  As a result,

$$\begin{aligned}
\mathcal{L}(\mathbf{x} + \mathbf{h}) &= \frac{1}{2} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}) \\
&\approx \frac{1}{2} \boldsymbol{\ell}(\mathbf{x})^T \boldsymbol{\ell}(\mathbf{x}) \\
&= \frac{1}{2} (\mathbf{f}(\mathbf{x}) + J(\mathbf{x})\mathbf{h})^T (\mathbf{f}(\mathbf{x}) + J(\mathbf{x})\mathbf{h}) \\
&= \frac{1}{2} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}) + \mathbf{h}^T J(\mathbf{x})^T \mathbf{f}(\mathbf{x}) + \frac{1}{2} \mathbf{h}^T J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h} \\
&= \mathcal{L}(\mathbf{x}) + \mathbf{h}^T J(\mathbf{x})^T \mathbf{f}(\mathbf{x}) + \frac{1}{2} \mathbf{h}^T J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h}
\end{aligned}$$

- Let

$$\mathtt{L}(\mathbf{h}) = \mathcal{L}(\mathbf{x}) + \mathbf{h}^T J(\mathbf{x})^T \mathbf{f}(\mathbf{x}) + \frac{1}{2} \mathbf{h}^T J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h}.$$

  The Gauss–Newton update $\mathbf{h}_{\text{gn}}$ minimizes $\mathtt{L}(\mathbf{h})$:

$$\mathbf{h}_{\text{gn}} = \arg\min_{\mathbf{h}} \left\{ \mathtt{L}(\mathbf{h}) \right\}$$

Algorithm 10 Gauss–Newton update step.

Compute $\mathbf{h}_{\text{gn}} = -\big(J(\mathbf{x})^T J(\mathbf{x})\big)^{-1} J(\mathbf{x})^T \mathbf{f}(\mathbf{x})$.
Find step length α with a line search.
$\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{h}_{\text{gn}}$

- We have that
  - $\nabla \mathtt{L}(\mathbf{h}) = \mathbf{f}(\mathbf{x})^T J(\mathbf{x}) + \mathbf{h}^T J(\mathbf{x})^T J(\mathbf{x})$.
  - $H_{\mathtt{L}}(\mathbf{h}) = J(\mathbf{x})^T J(\mathbf{x})$.
- To find $\mathbf{h}_{\text{gn}}$, we set $(\nabla \mathtt{L}(\mathbf{h}))^T$ to $\mathbf{0}$, which gives

$$\begin{aligned} J(\mathbf{x})^T \mathbf{f}(\mathbf{x}) + J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h} &= \mathbf{0} \\ \mathbf{h} &= -\big(J(\mathbf{x})^T J(\mathbf{x})\big)^{-1} J(\mathbf{x})^T \mathbf{f}(\mathbf{x}), \end{aligned}$$

  If $\mathbf{J}(\mathbf{x})$ has full rank, we have that $J(\mathbf{x})^T J(\mathbf{x})$ would be positive definite, and $\mathbf{h}_{\text{gn}}$ would be the unique minimizer of $\mathtt{L}(\mathbf{h})$.
- A typical implementation of the Gauss–Newton method is given in Algorithm 10.
- The method with line search has gauranteed convergence provided that $\{\mathbf{x} : \mathcal{L}(\mathbf{x}) \leq \mathcal{L}(\mathbf{x}^{(0)})\}$ is bounded, and the Jacobian $J(\mathbf{x})$ has full rank in all steps.
- While Newton's method have quadratic convergence, the convergence of Gauss–Newton method is generally linear.
  - The surprising thing is that the loss value at the local minimizer $\mathcal{L}(\mathbf{x}^*)$ actually controls the convergence speed. See Madset et al. [5].

4.2 The Levenberg–Marquardt Method

- Levenberg [4] and later Marquard [6] suggests the use of a damped Gauss–Newton method.
- The update direction $\mathbf{h}_{\text{lm}}$ is determined by solving the following equation:

$$(J(\mathbf{x})^T J(\mathbf{x}) + \mu I)\mathbf{h}_{\text{lm}} = -J(\mathbf{x})^T \mathbf{f}(\mathbf{x}).$$

- The damping parameter μ has several effects.
  - For $\mu > 0$, the matrix on the LHS is positive definite. This ensures that $\mathbf{h}_{\text{lm}}$ is a descent direction of the model $\mathtt{L}$.
  - For large values of μ, we get that $\mathbf{h}_{\text{lm}} \approx -J(\mathbf{x})^T \mathbf{f}(\mathbf{x})/\mu = -\nabla L(\mathbf{x})/\mu$, which is a short step in the direction of the gradient of the loss function. This is good if $\mathbf{x}$ is far from the solution.
  - If μ is very small, then $\mathbf{h}_{\text{lm}} \approx \mathbf{h}_{\text{gn}}$, which is a good step in the final stages of the iterations. If $\mathcal{L}(\mathbf{x}^*) = 0$, we can get quadratic final convergence.
- As a damped method, we do not need line search to determine the step length. However, we must come up with a way to update μ.
- The initial value of μ should be related to the size of the elements in $A^{(0)} = J(\mathbf{x}^{(0)})^T J(\mathbf{x}^{(0)})$. It is typically set to

$$\mu^{(0)} = \tau \cdot \max_{1 \leq i \leq n} \{a_{ii}^{(0)}\}$$

  where τ is a constant specified by the user.

- The algorithm is not very sensitive to the choice of τ.
- As a rule of thumb, one should use a small value. For example, $\tau = 10^{-6}$ if $\mathbf{x}^{(0)}$ is believed to be a good appriximation of $\mathbf{x}^*$.
- Otherise, we use $\tau = 10^{-3}$ or $\tau = 1$.

- Updating μ is controlled by the gain ratio

$$\varrho = \frac{\mathcal{L}(\mathbf{x}) - \mathcal{L}(\mathbf{x} + \mathbf{h}_{\text{lm}})}{\mathtt{L}(\mathbf{0}) - \mathtt{L}(\mathbf{h}_{\text{lm}})}.$$

  The denominator is given by

$$\begin{aligned}
\mathtt{L}(\mathbf{0}) - \mathtt{L}(\mathbf{h}_{\text{lm}}) &= -{\mathbf{h}_{\text{lm}}}^T J(\mathbf{x})^T \mathbf{f}(\mathbf{x}) - \frac{1}{2}\mathbf{h}_{\text{lm}}^T J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h}_{\text{lm}} \\
&= {\mathbf{h}_{\text{lm}}}^T (J(\mathbf{x})^T J(\mathbf{x}) + \mu I)\mathbf{h}_{\text{lm}} - \frac{1}{2}\mathbf{h}_{\text{lm}}^T J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h}_{\text{lm}} \\
&= \frac{1}{2}{\mathbf{h}_{\text{lm}}}^T (2J(\mathbf{x})^T J(\mathbf{x}) + 2\mu I)\mathbf{h}_{\text{lm}} - \frac{1}{2}\mathbf{h}_{\text{lm}}^T J(\mathbf{x})^T J(\mathbf{x}) \mathbf{h}_{\text{lm}} \\
&= \frac{1}{2}{\mathbf{h}_{\text{lm}}}^T (J(\mathbf{x})^T J(\mathbf{x}) + 2\mu I)\mathbf{h}_{\text{lm}} \\
&= \frac{1}{2}{\mathbf{h}_{\text{lm}}}^T \big((J(\mathbf{x})^T J(\mathbf{x}) + \mu I) + \mu I\big)\mathbf{h}_{\text{lm}} \\
&= \frac{1}{2}{\mathbf{h}_{\text{lm}}}^T \big(\mu \mathbf{h}_{\text{lm}} - J(\mathbf{x})^T \mathbf{f}(\mathbf{x})\big).
\end{aligned}$$

  Because $\mathbf{h}_{\text{lm}}$ is a descent direction, it must be that $-\mathbf{h}_{\text{lm}} J(\mathbf{x})^T \mathbf{f}(\mathbf{x})$ must be positive. As a result, the denominator is always positive.

- To update μ, we can use Nielsen's rule [7] in Section 3.4.

- Now, we need to decide when the stop the algorithm. There are three criteria.
  - First, we should stop when we reach a stationary point. This means that $(\nabla \mathcal{L}(\mathbf{x}))^T = J(\mathbf{x})^T \mathbf{f}(x) = \mathbf{0}$. As a result, we can use

$$\|J(\mathbf{x})^T \mathbf{f}(x)\|_\infty \leq \varepsilon_1$$

    where ε_1 is a small positive constant chosen by the user.
  - Another criteria is to stop when the change in $\mathbf{x}$ is small.

$$\|\mathbf{x}_{\text{new}} - \mathbf{x}\| \leq \varepsilon_2(\|\mathbf{x}\| + \varepsilon_2)$$

    where ε_2 is another small positive constant chosen by the user. This expression gives a gradual change from relative size $\varepsilon_2\|\mathbf{x}\|$ when $\|\mathbf{x}\|$ is large and to absolute size ε_2^2 when $\|\mathbf{x}\|$ is close to $\mathbf{0}$.
  - Lastly, the algorithm should terminate when a maximum number of iterations, chosen by the user, is reached.

- The pseudocode of the Levenberg–Marquardt algorithm is given in Algorithm 11.

References

[1] Cao, L., Sui, Z., Zhang, J., Yan, Y., and Gu, Y. Line search methods. `https://optimization.cbe.cornell.edu/index.php?title=Line_search_methods`, 2021. Accessed: 2024-11-24.

Algorithm 11 The Levenberg–Marquardt algorithm.

$k \leftarrow 0$
$\nu \leftarrow 2$
$\mathbf{x} \leftarrow \mathbf{x}^{(0)}$
$A = J(\mathbf{x})^T J(\mathbf{x})$
$\mathbf{g} \leftarrow J(\mathbf{x})^T \mathbf{f}(\mathbf{x})$
$\mu = \tau \cdot \max_{1 \leq i \leq n}\{a_{ii}\}$
while $\|\mathbf{g}\|_\infty > \varepsilon_1$ **and** $k \leq k_{\max}$ **do**
  Determint $\mathbf{h}_{\text{lm}}$ by solving $(A + \mu I)\mathbf{h}_{\text{lm}} = \mathbf{g}$.
  if $\|\mathbf{h}_{\text{lm}} \leq \varepsilon_2(\|\mathbf{x}\| + \varepsilon_2)$ **then**
    return x
  end if
  $\mathbf{x}_{\text{new}} \leftarrow \mathbf{x} + \mathbf{h}_{\text{lm}}$
  $\varrho \leftarrow (\mathcal{L}(\mathbf{x}) - \mathcal{L}(\mathbf{x}_{\text{new}}))/(\mathtt{L}(\mathbf{0}) - \mathtt{L}(\mathbf{h}_{\text{lm}}))$
  if $\varrho > 0$ **then**
    $\mathbf{x} \leftarrow \mathbf{x}_{\text{new}}$
    $A = J(\mathbf{x})^T J(\mathbf{x})$
    $\mathbf{g} \leftarrow J(\mathbf{x})^T \mathbf{f}(\mathbf{x})$
    $\mu \leftarrow \mu \cdot \max\{\frac{1}{3}, 1 - (2\varrho - 2)^3\}$
    $\nu \leftarrow 2$
  else
    $\mu \leftarrow \mu\nu$
    $\nu \leftarrow 2\nu$
  end if
  $k \leftarrow k + 1$
end while
return x

[2] Frandsen, P. E., Jonasson, K., Nielsen, H. B., and Tingleff, O. Unconstrained optimization. `http://www2.imm.dtu.dk/pubdb/edoc/imm3217.pdf`, 2004. Accessed: 2024-11-24.

[3] Khungurn, P. Notations for multivariable derivatives. `https://pkhungurn.github.io/notes/notes/math/multivar-deriv-notations/multivar-deriv-notations.pdf`, 2022. Accessed: 2024-11-23.

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

[5] Madsen, K., Nielsen, H., and Tingleff, O. *Methods for Non-Linear Least Square Problems*. April 2004.

[6] Marquardt, D. W. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics 11*, 2 (1963), 431–441.

[7] Nielsen, H. B. Damping parameters in marquardt's method. `https://www2.imm.dtu.dk/documents/ftp/tr99/tr05_99.pdf`, 1999. Accessed: 2024-11-26.

[8] Nocedal, J., and Wright, S. *Numerical optimization*, 2. ed. ed. Springer series in operations research and financial engineering. Springer, New York, NY, 2006.