THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

COLLEGE OF ARTS AND SCIENCES
Department of
Statistics & Operations Research

Optimization Strategies for Gauss-Newton Method in solving Nonlinear Regression

Duc Thanh Vinh Nguyen

STOR 512: Optimization for Machine Learning and Neural Networks

Spring 2024

Abstract

Non-linear least squares optimization is a fundamental technique in various fields such as computer vision, machine learning, and engineering. Unlike linear least squares, where the model parameters can be directly computed using closed-form solutions, non-linear least squares require iterative optimization methods due to the non-linearity of the model. Among these techniques, the Gauss-Newton method stands out as a widely adopted iterative optimization algorithm. This paper focuses on the optimization of the Gauss-Newton method by integrating line search strategies aimed at improving convergence rates, especially when initialized from arbitrary starting points. Additionally, it proposes an implementation of the method leveraging automatic differentiation, thereby streamlining the computational process when working with complex models. We discuss theoretical foundations and practical implementations in Python. Additionally, we present experimental results demonstrating the effectiveness of the improved Gauss-Newton method in solving nonlinear least squares optimization problems.

Contents

1. Introduction....... 4
2. Methodologies....... 5
2.1 Gauss-Newton Method....... 5
2.2 Line Search....... 5
2.2.1 Damped-step Gauss-Newton Method....... 5
2.2.2 Backtracking Line Search....... 6
2.2.3 Armijo-Goldstein Condition....... 7
2.2.4 Wolfe Conditions....... 8
2.3 Automatic Differentiation....... 9
3. Experiment....... 10
3.1 Model....... 10
3.2 Dataset....... 10
3.3 Testing....... 11
4. Results and findings....... 11

Student’s Contribution

Here is a summary of the contribution of member Duc Thanh Vinh Nguyen:

- Conducted further research on Gauss-Newton method.
- Conducted further research on line search.
- Conducted research on Automatic Differentiation.
- Developed Python code for algorithms based on the implementation of Gauss-Newton method given in the course STOR512: Optimization for Machine Learning and Neural Networks.
- Developed and implemented the testing environment to compare performance of different algorithms.

1. Introduction

We start by deriving the nonlinear least squares optimization problem from its application of nonlinear regression. Nonlinear regression is a powerful statistical technique used to model the relationship between a dependent variable and one or more independent variables when the relationship is not linear. Consider a nonlinear function $f: R^d x\, R^p \rightarrow R$, the nonlinear relation can be denoted as:

$$y = f(x; \beta) + \epsilon, \tag{1}$$

where ϵ is a random noise with zero mean, and we have $\beta \in R^p$ and a dataset $D = \{x_i, y_i\}_{i=1}^{n}$ of n data points with $x_i \in R^d$, $y_i \in R^d$. By assuming the noise ϵ follows a normal distribution of zero mean, we want to find the parameter β that minimize the sum of squares of noises on the given dataset. Thus, the optimization can be derived from the nonlinear regression model as follows:

$$min_{\beta \in R^p}\{L(\beta) := \frac{1}{2}\sum_{i=1}^{n}(f(x_i; \beta) - y_i)^2\}. \tag{2}$$

In general, if the function $f(.)$ is linear, the minimization problem can be solved through direct approach (e.g: Linear least squares). However, if the function becomes nonlinear, we often have to use iterative techniques to solve the problem. An iterative technique starts from an initial point β_0 and generate a better approximation after each iteration converging to the underlying truth value β_{truth} of the given dataset while following some predefined condition. The Gauss-Newton method is an example of such an approach. The main idea is to linearize the gradient mapping of the objective function L to obtain a linear equation, which can be directly solved. An advantage of Gauss-Newton method is that it only require an approximation of Hessian matrix of L, significantly reducing the overall computational cost. However, 2 problem may occur with this approach: (i) Depending on the initial point β_0, the method may not converge or converge to a different stationary point, meaning that the final result may not be the minimizer for our objective optimization problem. (ii) The objective function may be complex and it can be hard to evaluate its Jacobian matrix. This paper investigates solutions to both of these disadvantages, including line search for the convergence problem and auto differentiation for complex derivatives evaluation. We start by formally establishing the Gauss-Newton method. Subsequently, we will explore how to apply line search and automatic differentiation techniques, building upon the foundational Gauss-Newton method. We also present and discuss experimental scenarios in order to validate the effectiveness of these optimization techniques in the general Gauss-Newton method.

2. Methodologies

2.1 Gauss-Newton Method

Considering the nonlinear least squares problem (2), given a point $\beta_t \in R^p$, where t is the current iteration, $\nabla L = F'(\beta_t)^T F(\beta_t)$, and approximate $\nabla^2 L(\beta_t) \approx F'(\beta_t)^T F'(\beta_t)$, the linearization of the gradient mapping of objective function L around β_t gives a linear equation:

$$F'(\beta_t)^T F'(\beta_t)(\beta_{t+1} - \beta_t) + F'(\beta_t)^T F(\beta_t) = 0. \tag{3}$$

We then solve the linear equation (3) for β_{t+1}:

$$\beta_{t+1} = \beta_t - (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t). \tag{4}$$

The algorithm can be defined as follows:

Algorithm 1: Gauss-Newton Method

```
$\beta_t = \beta_0$
$Loop\ until\ converge\ condition\ is\ met$:
    $\beta_t = \beta_t - (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t)$
$End\ loop$
```

2.2 Line Search

2.2.1 Damped-step Gauss-Newton Method

Let's denote $\Delta(\beta_t) = (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t)$, the function (4) becomes:

$$\beta_{t+1} = \beta_t - \Delta(\beta_t). \tag{5}$$

The value $\Delta(\beta_t)$ can be considered as a search direction similar to the search direction in a descent method. If we add a step-size into the search direction:

$$\beta_{t+1} = \beta_t - \eta_t \Delta(\beta_t),\ where\ \eta_t \in (0,\ 1]. \tag{6}$$

The algorithm now becomes:

Algorithm 2: Damped-step Gauss-Newton Method

```
$\beta_t = \beta_0; \eta_t \in (0,\ 1]$
$\Delta(\beta_t) = (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t)$
$Loop\ until\ converge\ condition\ is\ met$:
    $\beta_{t+1} = \beta_t - \eta_t \Delta(\beta_t)$
    $\beta_t = \beta_{t+1}$
$End\ loop$
```

This results in a slower but smoother convergence of the algorithm, especially when the step size is small. While the approach somewhat mitigates the limitations of the Gauss-Newton method with arbitrary initial points, a more effective strategy involves dynamically adjusting the parameter in each iteration based on the outcomes of a line search. We will delve into this method in the subsequent section.

2.2.2 Backtracking Line Search

A line search technique is employed to determine a sufficient distance to travel along a specified search direction. The method starts with a relatively large estimate of the step size (i.e., full step size) for movement along the line search direction, and progressively scales down the step size by a factor. This iterative operation will keep going until satisfying a predefined stopping criterion, which is based on the step size and the local gradient of the objective function. A straightforward approach for determining the step-size involves conducting a back-tracking line search during each iteration. Given an objective function L and the iterative scheme (6), this search aims to find an appropriate step size $\eta_t \in (0,\ 1]$ satisfying a stopping condition:

$$L(\beta_t - \eta_t \Delta(\beta_t)) \le L(\beta_t). \tag{7}$$

This is simply mean we want to ensure that the residual at the next value β_{t+1} is always smaller than the current residual at β_t. The algorithm can be derived as follows:

Algorithm 3: Gauss-Newton Method with Back-tracking Line Search

```
$\beta_t = \beta_0$;
$Loop\ until\ converge\ condition\ is\ met$:
    $\eta_t = 1$
    $\Delta(\beta_t) = (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t)$
    $Loop\ until\ L(\beta_t - \eta_t \Delta(\beta_t)) \le L(\beta_t)$:
        $\eta_t = reduce\_stepsize(\eta_t)$
    $End\ loop$
    $\beta_{t+1} = \beta_t - \eta_t \Delta(\beta_t)$
    $\beta_t = \beta_{t+1}$
$End\ loop$
```

In the beginning of each outer iteration, we initialize $\eta_t = 1$, this is equivalent to the original Gauss-Newton method with a step size of 1. If the residual fails to decrease, we reduce the step size using a predefined function. The reduction should be based on the nature of the input data, a default implementation could involve halving η_t at each iteration of the inner loop: $\eta_t = \frac{\eta_t}{2}$.

2.2.3 Armijo-Goldstein Condition

We will now investigate some widely-adopted stopping criterions, including Armijo-Goldstein condition and Wolfe Conditions. We start with a formal definition of the Armijo-Goldstein condition:

$$L(\beta_t - \eta_t \Delta(\beta_t)) \leq L(\beta_t) + \eta_t * c * (\nabla L(\beta_t)^T \Delta(\beta_t)), \tag{8}$$

Where $L(.)$ is the objective function, η_t is the step size, $c \in (0, 1)$ is a relatively small scaling factor (i.e., 10^{-4}), $\nabla L(\beta_t)$ is the local gradient at β_t, and $\Delta(\beta_t)$ is the search direction. This condition guarantees that the decrease in the objective function at each iteration remains sufficient. Nevertheless, it does not ensure the optimal reduction value, as any η_t which is small enough will be able to satisfy the condition. We can apply this condition into the Gauss-Newton method as follows:

Algorithm 4: Gauss-Newton Method with Armijo-Goldstein Line Search

```
$\beta_t = \beta_0; c = 10^{-4}$
$Loop\ until\ converge\ condition\ is\ met$:
    $\Delta(\beta_t) = (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t)$
    $\eta_t = 1$
    $Loop\ until\ L(\beta_t - \eta_t \Delta(\beta_t)) \leq L(\beta_t) + \eta_t * c * (\nabla L(\beta_t)^T \Delta(\beta_t))$:
        $\eta_t = reduce\_stepsize(\eta_t)$
    $End\ loop$
```

```
        $\beta_{t+1} = \beta_t - \eta_t\Delta(\beta_t)$
        $\beta_t = \beta_{t+1}$
$End\ loop$
```

2.2.4 Wolfe Conditions

As previously noted, a drawback of the Armijo-Goldstein condition is the potential for excessively small step sizes, leading to slower convergence and increased computational demands. The Wolfe conditions address this problem by imposing an additional constraint, forming a pair of conditions. Given an objective function $L(.)$, search direction $\Delta(\beta_t)$, a current step size η_t, and two scaling factors $c1$, $c2 \in (0, 1)$, the Wolfe conditions can be defined as:

$$L(\beta_t - \eta_t\Delta(\beta_t)) \leq L(\beta_t) + \eta_t * c_1 * (\nabla L(\beta_t)^T\Delta(\beta_t)), \tag{9}$$

$$\nabla L(\beta_t - \eta_t\Delta(\beta_t))^T\Delta(\beta_t) \geq c_2 * (\nabla L(\beta_t)^T\Delta(\beta_t)). \tag{10}$$

The inequation (9) is directly derived from the Armijo-Goldstein condition, while the inequation (10) is known as the curvature condition. The curvature condition ensures that the slope of the objective function along the search direction decreases sufficiently and the slope of the objective function at the new point $\beta_{t+1} = \beta_t - \eta_t\Delta(\beta_t)$ is not too small. In other words, the Armijo condition sets an upper limit and the curvature condition establishes a lower bound on the step size. Hence, the Wolfe conditions allow inexact line search methods to calculate the new step size closer to the optimal value. The two conditions depend on the two scaling factor $c1$ and $c2$, $c1$ is often chosen to be small (i.e., $c1 = 10^{-4}$) as in the original Armijo condition, on the other hand $c1$ is often bigger (i.e., $c2 = 0.1$).

We will derive the implementation of Wolfe conditions in the Gauss-Newton method:

Algorithm 5: Gauss-Newton Method with Wolfe Conditions Line Search

```
$\beta_t = \beta_0; c_1 = 10^{-4}; c_2 = 0.1$
$Loop\ until\ converge\ condition\ is\ met:$
        $\eta_t = 1$
```

```
    $armijo = FALSE$
    $curvature = FALSE$
    $\Delta(\beta_t) = (F'(\beta_t)^T F'(\beta_t))^{-1} F'(\beta_t)^T F(\beta_t)$
    $Loop\ until\ armijo = TRUE\ and\ curvature = TRUE:$
        $armijo = L(\beta_t - \eta_t \Delta(\beta_t)) \le L(\beta_t) + \eta_t * c * (\nabla L(\beta_t)^T \Delta(\beta_t))$
        $curvature = \nabla L(\beta_t - \eta_t \Delta(\beta_t))^T \Delta(\beta_t) \ge c_2 * (\nabla L(\beta_t)^T \Delta(\beta_t))$
        $\eta_t = reduce\_stepsize(\eta_t)$
    $End\ loop$
    $\beta_{t+1} = \beta_t - \eta_t \Delta(\beta_t)$
    $\beta_t = \beta_{t+1}$
$End\ loop$
```

2.3 Automatic Differentiation

Automatic Differentiation (AD) is a computational technique widely used in optimization, machine learning, and scientific computing for efficiently and accurately evaluating derivatives of mathematical functions. Automatic differentiation exploits the chain rule of calculus to systematically compute derivatives of any single-variable and multivariable function with high precision. At its core, automatic differentiation breaks down a function into a sequence of elementary operations and their corresponding derivatives. This method of differentiation mirrors the way humans manually compute derivatives, but in a much more efficient and scalable manner with the computational power of computers. This makes AD particularly well-suited for scenarios where gradient information is crucial, such as gradient-based optimization algorithms like gradient descent or backpropagation in neural networks.

Naturally, implementing an automatic differentiation approach into the Gauss-Newton approach is straightforward. Instead of inputting data into the gradient function vector, we calculate the Jacobian matrix in each iteration of the iterative process. However, this adaptation increases the computational load per iteration, demanding more resources. Nonetheless, it offers researchers a quick proof of concept before fine-tuning the method for larger-scale applications.

3. Experiment

In this section, we present an experimental scenario aimed at concretely validating the effectiveness of the discussed optimization techniques, including line search and automatic differentiation, within the framework of the general Gauss-Newton method.

3.1 Model

We start by establishing the underlying nonlinear model that will serve as the core of our experiment. In particular, we adopted the classic Beverton-Holt model into a nonlinear least square optimization problem as follows:

$$min_{\beta \in R^2} \{ L(\beta) := \frac{1}{2} \sum_{i=1}^{n} (\beta_0 exp(\frac{x}{\beta_1}) - y_i)^2 \}, \quad (11)$$

where $\beta = (\beta_0, \beta_1)^T \in R^2; \beta_1 \neq 0$ and a dataset $D = \{x_i, y_i\}_{i=1}^{n}$ of n data points with $x_i, y_i \in R$. The Jacobian of the objective function $L(\beta)$ with regarding to β can be formulated as:

$$\frac{\partial L(\beta)}{\partial \beta_0} = exp(\frac{x}{\beta_1}); \ \frac{\partial L(\beta)}{\partial \beta_1} = - \frac{\beta_0 x}{\beta_0^2} exp(\frac{x}{\beta_1}). \quad (12)$$

3.2 Dataset

We then generate an experimental dataset with the size of 50 data points using the underlying model and a truth parameter $\beta_{truth} = (0.15,\ 3.75)^T$, the choice of the truth parameter is not important as long as it satisfies the condition of the Beverton-Holt model. We also add an additional noise $\varepsilon \sim N(0, 0.1)$ following the Normal distribution with a mean of 0 and a standard deviation of 0.1. The final dataset D can be formally presented as: $D = \{x_i, y_i\}_{i=1}^{50},\ where\ x_i \in [-\ 10,\ 10]\ and\ y_i = 0.15 * exp(\frac{x_i}{3.75}) + \varepsilon.$

3.3 Testing

We introduce a maximum number of iterations T (default to 100) and a error tolerance number $tolerance$ (default to 10^{-3}). The error of the objective function can be simply calculated to be the norm of the gradient of objective function $||\nabla L(\beta_t)||$ at the current β_t of iteration t. If $||\nabla L(\beta_t)|| \leq tolerance$, we say that the algorithm has converged, if not we keep running the iterative scheme for T times, meaning the algorithm does not converge after T iterations. Subsequently, we implement and execute the discussed algorithms utilizing Python to monitor both the convergence rate and the accuracy of the approximations. We conduct these assessments with different distinct arbitrary initial guess points, randomly generated to progressively move away from the true solution. The result of the experiment will be investigated in the next section.

4. Results and findings

In this section, we will analyze the outcomes derived from the crafted experiment. Our analysis focuses on 2 criteria: (i) The convergence rate, quantifying the number of iterations needed for convergence; (ii) The accuracy assessment, deciding whether the algorithm attains convergence at a sufficient approximation compared to the truth parameters.

When applying the damped-step Gauss-Newton method (algorithm 2) for arbitrary initial points, we observe improved convergence compared to the original method, particularly with smaller step sizes (e.g., 0.1), resulting in higher accuracy. However, achieving this level of accuracy demands a substantially increased number of iterations, often doubling or even tripling the original count. Consequently, the algorithm's efficiency is significantly compromised as expected.

By incorporating line search optimization, both the convergence rate and accuracy of the Gauss-Newton method experience enhancements. The choice of random starting points doesn't influence the algorithm's overall accuracy; rather, it affects the number of line search iterations required to determine an optimal step size for each iteration. Generally, employing Armijo-Goldstein's condition (algorithm 4) facilitates a faster discovery of a sufficient step size compared to a straightforward line search (algorithm 3). Wolfe's conditions (algorithm 5) surpasses Armijo's rule by additionally establishing a lower bound for the step size, preventing "jumping over" critical points.

When examining automatic differentiation optimization, it produces identical outcomes to the conventional method of computing the Jacobian matrix via vectors of functions. Nonetheless, the additional time taken per iteration is notably large. Consequently, this technique might not be well-suited for large-scale Gauss-Newton problems and should be reserved for smaller-scale applications.

In conclusion, our exploration into various optimization techniques for the Gauss-Newton method has shed light on their respective implementations, advantages, and limitations.

Bibliography

[1] S. Gratton, A. S. Lawless, and N. K. Nichols, “Approximate Gauss–Newton Methods for Nonlinear Least Squares Problems,” *SIAM Journal on Optimization*, vol. 18, no. 1, pp. 106–132, Jan. 2007, doi: https://doi.org/10.1137/050624935.

[2] P. Teunissen, “Nonlinear least-squares,” Mar. 1990. Accessed: May 08, 2024. [Online]. Available: https://www.researchgate.net/publication/281545607_Nonlinear_least-squares

[3] K. Madsen, H. Nielsen, and O. Tingleff, “Methods for nonlinear least squares problems,” 2004.

[5] Q. Tran-Dinh, Class Lecture, Topic: "AutoDiff and Back Propagation” STOR 512, College of Arts and Science, University of North Carolina at Chapel Hill, North Carolina, Apr., 10, 2024.

[6] Q. Tran-Dinh, Class Lecture, Topic: "Nonlinear Least Squares” STOR 512, College of Arts and Science, University of North Carolina at Chapel Hill, North Carolina, Feb., 21, 2024.