Chapter 6

NUMERICAL OPTIMIZATION METHODS AND THEIR CONVERGENCE

CONTENTS

6.1 Gradient Descent Method

Gradient descent, and its stochastic approximation, in light of the special design of the overall loss function as an average of individual losses and Law of Large Number; stochastic gradient descent (or Mini-batch version) are two most frequently used iterative optimization algorithms for searching the local minimum of a loss function especially when the optimization criterion is smooth (*e.g.* differentiable) enough. The general steps for Gradient Descent method are:

1. starting at a random seed; then
2. taking a step with a modification of the approximation of the parameter estimates from the previous ones by an amount proportional to the negative of the gradient ∇ of the loss function (the overall) evaluated at the current point.

The following illustration explains the principle behind: suppose that the predicting model, $\hat{y}_n := f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)$, is parametric in $\boldsymbol{\theta}$ and we aim to minimize the mean square error (MSE) with the actual labels:

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2\,. \tag{6.1.1}$$

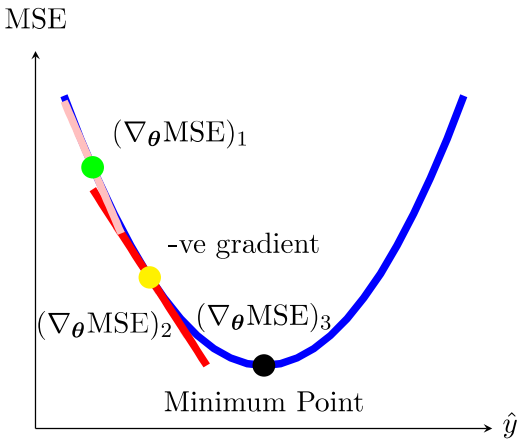


Figure 6.1.1: Gradient Descent for an MSE

Certainly, no matter what the loss functions are, any local minimum points should be also the stationary points. For example, Figure 6.1.1 shows the loss function MSE, and we observe that:

1. the magnitudes of the slopes of consecutive approximations are getting smaller and smaller when $\hat{y}$ approaches to the minimum point:
$$\|(\nabla_{\boldsymbol{\theta}}\text{MSE})_1\|_2 > \|(\nabla_{\boldsymbol{\theta}}\text{MSE})_2\|_2 > \|(\nabla_{\boldsymbol{\theta}}\text{MSE})_3\|_2 = 0\,;$$
2. the sign of the slope is in the opposite direction away from the minimum point.

Very usually, loss functions may possess multiple local minima, especially for various deep neural networks, the choice of initial guess is crucial for which the ultimate local minimum is being sought.

6.1.1 Learning Rate

In the step 2 in the last paragraph, the negative sign of the gradient is pointing the direction towards the next guess, while the step size for the next modification will be determined by a hyperparameter, called the *learning rate* η. With a learning rate η and loss function $L(\boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$ with the parameter $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, then:

$$\begin{gathered}\text{Step Size (Signed)} = \eta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\,; \\ \text{Next Move} = \text{Last Move} - \text{Step Size}\,.\end{gathered} \tag{6.1.2}$$

Alternatively, we have

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}\,. \tag{6.1.3}$$

Again, the negative step in calculating the next move is taken since the minimum point is in the opposite direction to the gradient at the current point. If the learning rate η is set too high, it is often not possible to reach the targeted nearby optimal point, but it may jump to one far away from the initial seed (See Figure 6.1.2); on the other hand, if the learning rate is set too low, more iteration times are needed to get the optimal point (See Figure 6.1.3). In practice, the learning rate η of 0.001 (`1e-3`) is used.

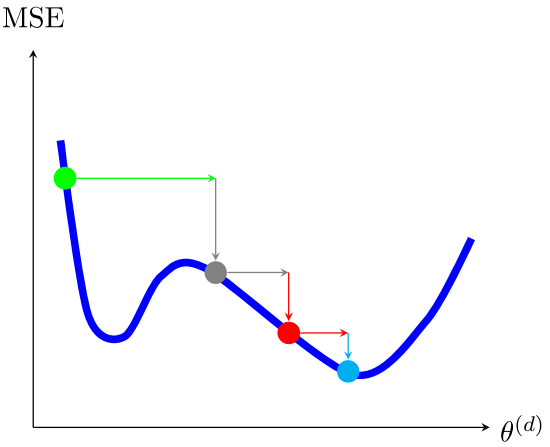


Figure 6.1.2: Learning rate is too large.

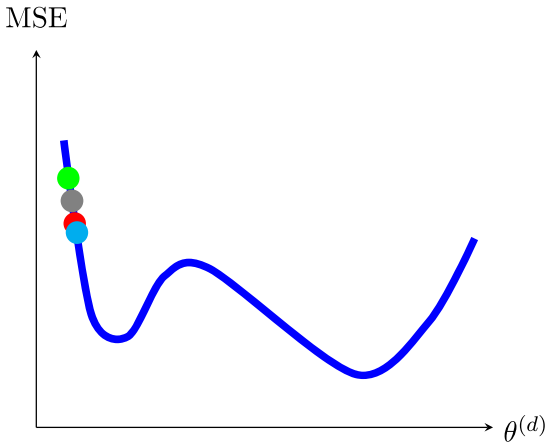


Figure 6.1.3: Learning rate is too small.

In theory, in order to guarantee convergence to the optimal point, the learning rate η cannot be constant so that a decreasing sequence of learning rate $\{\eta_t\}$ converging to zero is usually adopted. However, a small learning rate leads to a small step size, in which the loss function inevitably gets stuck in a local region as η_t converges to zero, and there is an obvious drawback of many iterations being required for slow convergence. On the other hand, with a large value of η_t, we might overshoot the updates to nowhere near the optimal solution. We must choose $\{\eta_t\}$ that balances the pace of convergence and accuracy. In practice, we change η once a great number of epoches, and for sake of simplicity; we first adopt a constant learning rate in the following discussions.

6.1.2 An Example with Gradient Descent Method

As an illustration, consider a simple linear regression model with one feature variable, and the dataset $\mathcal{S} = \{(x_n, y_n)\}_{n=1}^N$ and the predicting model is given by:

$$f_{\omega,b}(x) = \omega x + b\,, \qquad \text{for} \quad \boldsymbol{\theta} = \{\omega, b\} \in \mathbb{R}^2\,.$$

Adopting MSE $L(\omega, b)$ as the loss function and multiplying with $1/2$,

$$L(\omega, b) := \frac{1}{2N}\sum_{n=1}^{N}\Big(y_n - f_{\omega,b}(x_n)\Big)^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n - \omega x_n - b)^2\,.$$

Note that the gradient is the vector of partial derivatives with respect to ω and b, respectively:

$$\begin{cases} \dfrac{\partial L}{\partial \omega} = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} -x_n(y_n - \omega x_n - b)\,; \\ \dfrac{\partial L}{\partial b} = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} -(y_n - \omega x_n - b)\,. \end{cases}$$

The iteration scheme of (batch) gradient decent is composed with recursive epochs, each of which consists of using the whole training dataset to update both parameters ω and b. The number of epochs required in gradient decent can be approximated by the learning rate η. In Subsection 6.2.1, we introduced the Newton-Raphson Method, which can be seen as a special case of the gradient descent with $\eta = 1$, and it has a quadratic convergence rate such that the total number of steps required for convergence is usually around 10. With this in mind and a learning rate $\eta = 0.001$, the number of epoch required is then $10 \cdot 1/\eta = 10 \cdot 10^3 = 10,000$. As discussed at the beginning of this chapter, the steps are:

1. initializing the parameters $\omega^{(0)}$ and $b^{(0)}$ randomly, let say at:

$$\omega^{(0)} = 0 \qquad \text{and} \qquad b^{(0)} = 0\,;$$

2. updating the parameters $\omega^{(t)}$ and $b^{(t)}$ for each epoch $t = 0, \cdots, 10,000 - 1$ according to (6.1.2):

$$\begin{cases} \omega^{(t+1)} \leftarrow \omega^{(t)} - \eta \times \dfrac{\partial L}{\partial \omega^{(t)}}\,; \\ b^{(t+1)} \leftarrow b^{(t)} - \eta \times \dfrac{\partial L}{\partial b^{(t)}}\,. \end{cases} \tag{6.1.4}$$

We next illustrate an application of (6.1.4) with **R** and Python with the "`Advertising.csv`"[1] dataset, which originates from the book: *An Introduction to Statistical Learning* written by James et al. (2013). The dataset consists of information from 200 markets about their budgets on *Television* (TV) advertisement, budgets on *Radio* advertisement, budgets on *Newspaper* advertisement, all these amounts are in thousands of US dollars, and their annual *Sales* of their product in thousands of units sold. However, in this example, we shall only consider the budgets on *Television* (TV) advertisement and their annual *Sales* of their product.

[1]The dataset can be found in `https://www.statlearning.com/resources-first-edition`

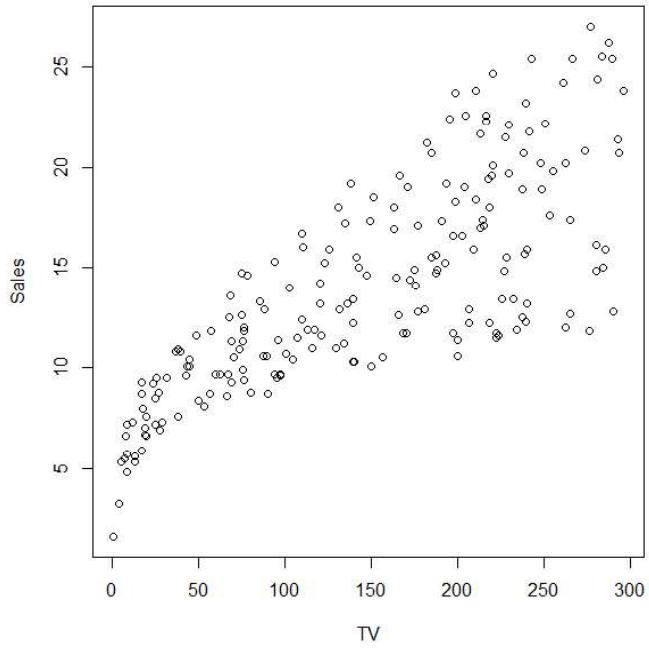


Figure 6.1.4: Sales against the budget on Television advertisement, in 1,000 units.

We aim to build a simple linear regression model that predicts the thousands units to be sold of a given market based on its budget on television adverising, where the linear model is given by

$$\widehat{\text{Sales}} = f_{\omega,b}(x) = \omega x + b = \omega \cdot \text{Television} + b\,. \tag{6.1.5}$$

We next implement the gradient descent method with (6.1.4) in **R** first, and then in Python in Programme 6.1.3:

```
> Gradient_Descent <- function(x, y, eta=1e-3, epochs=1e4){
+     omega_t <- 0
+     b_t <- 0
+     for (epoch in 1:epochs){
+         # calculate gradient for omega_t and b_t
+         dw <- mean(-x * (y - omega_t * x - b_t))
+         db <- mean(-(y - omega_t * x - b_t))
+
+         # update omega_t and b_t
+         omega_t <- omega_t - dw * eta
+         b_t <- b_t - db * eta
+     }
+     return(list(omega=omega_t, b=b_t))
+ }
>
> data <- read.csv("Advertising.csv")
> x <- as.matrix(data["TV"])
> y <- as.matrix(data["sales"])
>
```

```
> Gradient_Descent(x, y, eta=1e-3, epochs=1e5)
$omega
[1] NaN

$b
[1] NaN
```

Programme 6.1.1: Gradient descent for linear regression model with `eta=1e-3` in **R**.

Here the default learning rate $\eta = 0.001$ and the default number of epochs is 10,000. In the first attempt where the learning rate η is set at 0.001, the gradient descent algorithm does not converge. It is because the learning rate of $\eta = 0.001$ is still too large for the "`Advertising.csv`" dataset, it exaggerates both parameters ω and b such that they deviate further away from their respective optimal values.

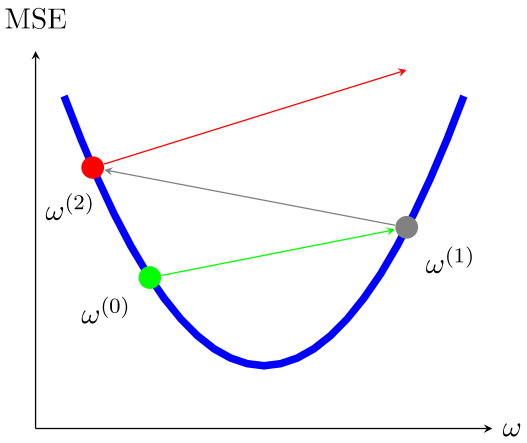


Figure 6.1.5: Overshooting with larger learning rate.

To remedy this overshooting issue, a smaller learning rate $\eta = 0.00006$ (`6e-5`) is adopted but with more epochs, for instance $10 \cdot 1/\eta = 0.333$M. To achieve a better approximation in this example, a larger epoch, let say 1M (`1e6`) epochs is used. Finally, we compare the estimated parameter values $\hat{\omega}$ and $\hat{b}$ against the coefficients' estimates by the **R** build-in linear regression model `lm()`. These 1M epochs are almost the minimal number of steps required for the gradient descent method to arrive at the similar coefficients estimate by the `lm()`, and so one can feel how tricky Newton method is in practice.

```
> Gradient_Descent(x, y, eta=6e-5, epochs=1e6)
$omega
[1] 0.04753665

$b
[1] 7.032592

> lm(y~x)

Call:
lm(formula = y ~ x)
```

```
12
13 Coefficients:
14 (Intercept)             x
15     7.03259       0.04754
```

Programme 6.1.2: Gradient descent for linear regression model with `eta=6e-5` in **R**.

The final model is:

$$\widehat{\text{Sales}} = 0.04754 \cdot \text{Television} + 7.03259\,.$$

Similarly in Python:

```python
import pandas as pd
import numpy as np

def Gradient_Descent(x, y, eta=1e-3, epochs=1e5):
    omega_t, b_t = 0, 0
    for epoch in range(int(epochs)):
        # calculate gradient for omega_t and b_t
        dw = np.mean(-x * (y - omega_t * x - b_t))
        db = np.mean(-(y - omega_t * x - b_t))

        # update omega_t and b_t
        omega_t = omega_t - dw * eta
        b_t = b_t - db * eta

    return({"omega": omega_t, "b": b_t})

dataset = pd.read_csv("Advertising.csv")
x, y = dataset["TV"].values, dataset["sales"].values

print(Gradient_Descent(x, y, eta=1e-3, epochs=1e5))
print(Gradient_Descent(x, y, eta=6e-5, epochs=1e6))

from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(x.reshape(-1, 1), y)
print({"omega": model.coef_[0], "b": model.intercept_})
```

```
{'omega': nan, 'b': nan}
{'omega': 0.04753664940393587, 'b': 7.032591782591389}
{'omega': 0.047536640433019764, 'b': 7.032593549127693}
```

Programme 6.1.3: Gradient descent for linear regression model in Python.

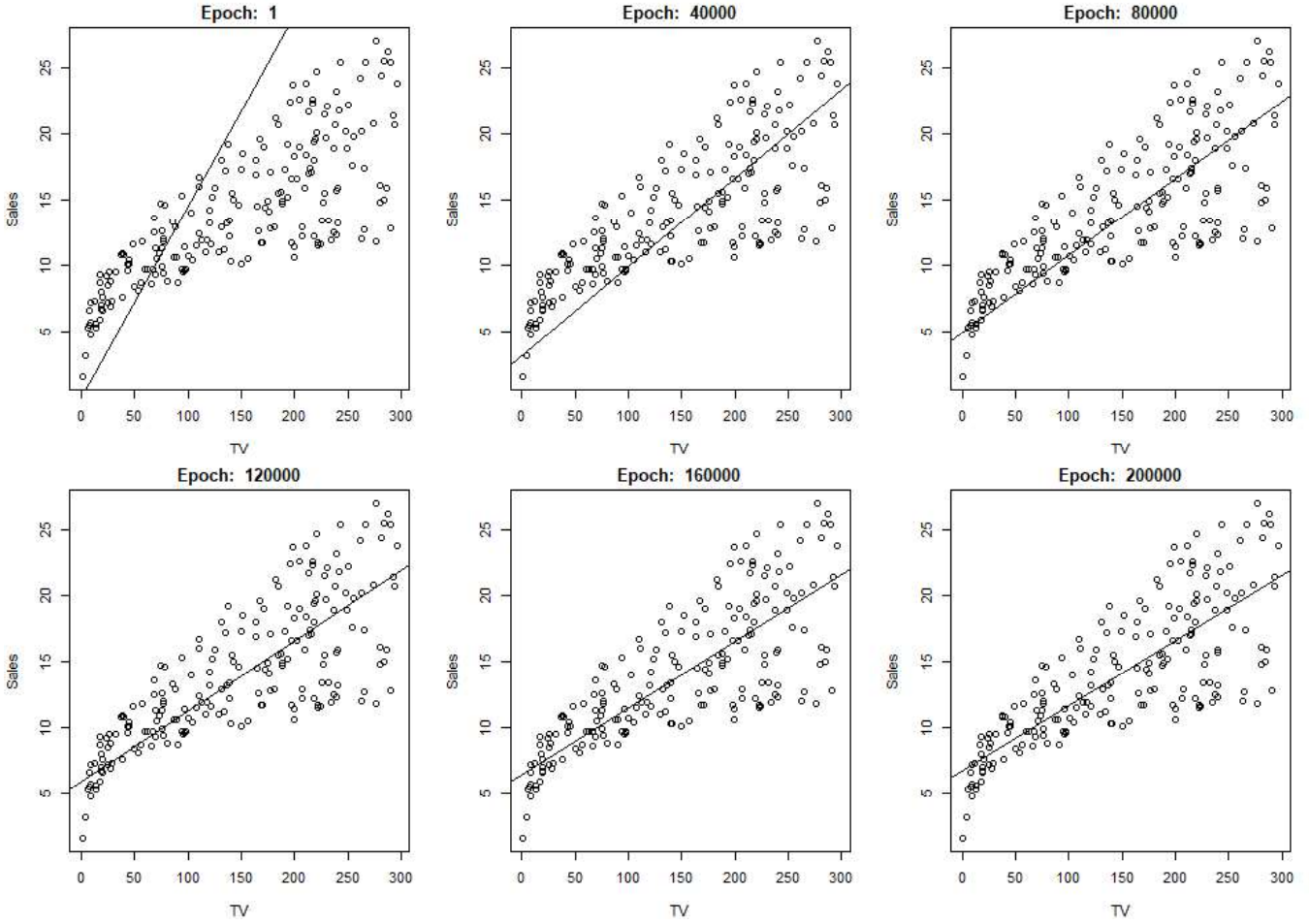


Figure 6.1.6: The progression of the estimated regression line through every 40,000 gradient descent epochs.

From Figure 6.1.6, at the first epoch 1, the regression line obviously does not fit with the data as the choice was randomly chosen. However, at the 120,000-th epoch, the regression line is getting pretty close to the ultimate best fitted line. After 160,000 epochs, the regression line stabilizes such that having more epoch would not yield a significant improvement in the fitting of the regression line.

6.1.3 Convergence of Gradient Descent

Definition 6.1.1. A function $L : \mathbb{R}^D \mapsto \mathbb{R}$ is said to be **Lipschitz continuous** on $\mathbb{R}^D$ if there exists a positive constant C such that

$$|L(\boldsymbol{\theta}) - L(\boldsymbol{\theta}')| \leq C\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2\,, \qquad \forall\, \boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^D\,,$$

where $\|\cdot\|_2$ is the usual Euclidean norm. L is also said to be C-Lipschitz continous.

Given the convex loss function L, with a C-Lipschitz continous $\nabla_{\boldsymbol{\theta}} L$, that means we have $C\boldsymbol{I}_D \succeq (\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top) L(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \mathbb{R}^D$, where $\boldsymbol{I}_D$ is a $D \times D$ identity matrix, *i.e.* $C\boldsymbol{I}_D - (\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top) L$ is a positive semi-definite matrix. Using Proposition **??**, we have

$$\begin{aligned} L(\boldsymbol{\theta}') &\leq L(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) + \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^\top C\boldsymbol{I}_D(\boldsymbol{\theta}' - \boldsymbol{\theta}) \\ &= L(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) + \frac{C}{2}\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|_2^2\,. \end{aligned} \tag{6.1.6}$$

Taking $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$ and $\boldsymbol{\theta}' = \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})$ by (6.1.3), in (6.1.6), we also have

$$\begin{aligned} L(\boldsymbol{\theta}^{(t+1)}) &\leq L(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top \cdot -\eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) + \frac{C}{2} \| -\eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \|_2^2 \\ &= L(\boldsymbol{\theta}^{(t)}) - \eta \left(\frac{2 - C\eta}{2} \right) \| \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \|_2^2 \,. \end{aligned} \tag{6.1.7}$$

Further assume that $0 < \eta \leq 1/C$, which is valid in general as the step size η should be small so as to avoid the overshooting problem. Then the (6.1.7) yields

$$L(\boldsymbol{\theta}^{(t+1)}) \leq L(\boldsymbol{\theta}^{(t)}) - \frac{\eta}{2} \| \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \|_2^2 \,. \tag{6.1.8}$$

Therefore, by noting that $\| \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \|_2^2 > 0$ when $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \neq \mathbf{0}$, the difference $L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^{(t)})$ is always negative such that the loss function L is strictly decreasing for every iteration steps until $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) = \mathbf{0}$.

Next, let $\boldsymbol{\theta}^*$ be the ultimate stationary point where $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^*) = \mathbf{0}$. The convexity of L guarantees that

$$L(\boldsymbol{\theta}^*) \geq L(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) \,. \tag{6.1.9}$$

Substituting (6.1.9) into (6.1.8) and further using $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) = (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)})/\eta$ by (6.1.3), we have

$$\begin{aligned} L(\boldsymbol{\theta}^{(t+1)}) &\leq L(\boldsymbol{\theta}^*) - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) - \frac{\eta}{2} \| \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) \|_2^2 \\ &= L(\boldsymbol{\theta}^*) - \frac{1}{\eta} (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) - \frac{1}{2\eta} \| \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)} \|_2^2 \\ &= L(\boldsymbol{\theta}^*) + \frac{1}{2\eta} (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)})^\top \Big(2(\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) - (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) \Big) \\ &= L(\boldsymbol{\theta}^*) + \frac{1}{2\eta} \Big((\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*) - (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*) \Big)^\top \Big(- (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*) - (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*) \Big) \\ &= L(\boldsymbol{\theta}^*) + \frac{1}{2\eta} \Big(\| \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \|_2^2 - \| \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^* \|_2^2 \Big) \,. \end{aligned} \tag{6.1.10}$$

Summing (6.1.10) over $i = 0, \cdots, t-1$ yields

$$\begin{aligned} \sum_{i=0}^{t-1} L(\boldsymbol{\theta}^{(i+1)}) &\leq \sum_{i=0}^{t-1} \left(L(\boldsymbol{\theta}^*) + \frac{1}{2\eta} \Big(\| \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^* \|_2^2 - \| \boldsymbol{\theta}^{(i+1)} - \boldsymbol{\theta}^* \|_2^2 \Big) \right) \\ \sum_{i=0}^{t-1} \Big(L(\boldsymbol{\theta}^{(i+1)}) - L(\boldsymbol{\theta}^*) \Big) &\leq \frac{1}{2\eta} \Big(\| \boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^* \|_2^2 - \| \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \|_2^2 \Big) \\ &\leq \frac{1}{2\eta} \| \boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^* \|_2^2 \,. \end{aligned} \tag{6.1.11}$$

Finally, with $L(\boldsymbol{\theta}^{(t)})$ is decreasing in t and so $L(\boldsymbol{\theta}^{(t)}) \leq 1/t \sum_{i=0}^{t-1} L(\boldsymbol{\theta}^{(i+1)})$,

$$L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*) \leq \frac{1}{t} \sum_{i=0}^{t-1} \Big(L(\boldsymbol{\theta}^{(i+1)}) - L(\boldsymbol{\theta}^*) \Big) \leq \frac{1}{2\eta t} \| \boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^* \|_2^2 \,, \tag{6.1.12}$$

implying that we can run $\| \boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^* \|_2^2 / (2\eta\delta)$ iterations for the gradient descent algorithm to reach the bound of $L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*) < \delta$.

Further assume that $L(\boldsymbol{\theta})$ is a μ-strongly convex function, *i.e.* for an $\mu > 0$,

$$L(\boldsymbol{\theta}') \geq L(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) + \frac{\mu}{2} \| \boldsymbol{\theta}' - \boldsymbol{\theta} \|_2^2 \,, \qquad \text{for any} \quad \boldsymbol{\theta}', \boldsymbol{\theta} \in \mathbb{R}^D \,. \tag{6.1.13}$$

Taking $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$ and $\boldsymbol{\theta}' = \boldsymbol{\theta}^*$ in (6.1.13), we have

$$L(\boldsymbol{\theta}^*) \geq L(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) + \frac{\mu}{2} \| \boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)} \|_2^2 \,, \tag{6.1.14}$$

or

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*) \geq L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*) + \frac{\mu}{2}\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 \,. \tag{6.1.15}$$

Also taking $\boldsymbol{\theta}' = \boldsymbol{\theta}^{(t)}$ and $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ in (6.1.6), we have:

$$L(\boldsymbol{\theta}^*) \leq L(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) + \frac{C}{2}\|\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}\|_2^2 \,,$$

which implies

$$\begin{aligned}
L(\boldsymbol{\theta}^*) - L(\boldsymbol{\theta}^{(t)}) &\leq \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) + \frac{C}{2}\|\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}\|_2^2 \\
&= \frac{1}{2}\left(\|\sqrt{C}(\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)})\|_2^2 + 2\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) + \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})/\sqrt{C}\|_2^2 - \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})/\sqrt{C}\|_2^2 \right) \\
&= \frac{1}{2}\left(\|\sqrt{C}(\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})/\sqrt{C}\|_2^2 - \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})/\sqrt{C}\|_2^2 \right) \\
&\leq -\frac{1}{2C}\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 \,,
\end{aligned}$$

and therefore,

$$2C(L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) \geq \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 \,. \tag{6.1.16}$$

Next, consider

$$\begin{aligned}
\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2 &= \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* - \eta\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 \\
&= \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - 2\eta\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*) + \eta^2\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 \\
&\leq \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - 2\eta\left((L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) + \frac{\mu}{2}\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 \right) + \eta^2\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 \\
&\leq (1 - \eta\mu)\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - 2\eta(L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) + 2\eta^2 C(L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) \\
&= (1 - \eta\mu)\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 - 2\eta(1 - \eta C)(L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) \,.
\end{aligned} \tag{6.1.17}$$

where the first inequality follows by using (6.1.15) and the last inequality is due to (6.1.16). With the assumption that $0 < \eta \leq 1/C$, we have $\eta C < 1$ such that $-2\eta(1 - \eta C) < 0$, together with the fact that $L(\boldsymbol{\theta}^{(t)}) \geq L(\boldsymbol{\theta}^*)$, (6.1.17) becomes

$$\begin{aligned}
\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2 &\leq (1 - \eta\mu)\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 \\
&\leq (1 - \eta\mu)^{t+1}\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2^2 \,,
\end{aligned}$$

which indicates a linear convergence with a rate of $O(r^t)$, where $r := (1 - \eta\mu)$ with $0 < r < 1$. In other words, we can run $(\ln(1/\delta) + 2\ln\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2)/(-\ln(1 - \eta\mu)) \sim O(1/\eta)$ iterations for the gradient descent algorithm to reach the bound of $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 < \delta$.

To conclude, with the assumption that $\nabla_{\boldsymbol{\theta}} L$ is C-Lipschitz continuity and the learning rate $\eta < 1/C$, we have the convergence rate of $O(1/\delta)$; while with an additional assumption of μ-strong convexity of $L(\boldsymbol{\theta})$, we have a faster convergence rate of $O(\ln(1/\delta))$, and this strong convexity assumption can be warranted by adding $\mathcal{L}^2$-regularization to L, the resulting penalized loss function is

$$L(\boldsymbol{\theta}) + \frac{\mu}{2}\|\boldsymbol{\theta}\|_2^2 \,,$$

for a hyperparameter $\mu > 0$.

6.2 Higher Order Gradient Descent Method

In this section, we shall discuss four gradient-based optimization methods, namely Newton-Raphson method, Gauss-Newton algorithm, BFGS, and L-BFGS, with the implementations in **R** and Python environments; see their respective subsections for more discussion.

6.2.1 Newton-Raphson Method

Newton-Raphson method, also known as the Newton's method developed by Isaac Newton and Joseph Raphson, is an iterative root finding algorithm for a $\mathbb{R}^D$-valued function, which is at least twice differentiable. In machine learning and deep learning, we are often required to minimize the chosen loss function $L : \mathbb{R}^D \to \mathbb{R}$, in which we can adopt an inexact (*i.e.* adaptive but uneven stepsize) line search in recursively finding a point with a smaller loss value. Mathematically,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)} =: \boldsymbol{\theta}^{(t)} + \boldsymbol{\delta}^{(t)} \qquad \text{such that} \quad L(\boldsymbol{\theta}^{(t+1)}) = L(\boldsymbol{\theta}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)}) < L(\boldsymbol{\theta}^{(t)})\,, \tag{6.2.1}$$

where $\alpha^{(t)} > 0$ is a constant, known as step length at the t^{th} iteration, and $\boldsymbol{p}^{(t)}$ is the step direction. In other words, at each iteration t, it approximates the roots $\boldsymbol{\theta}^{(t)}$ by adding it with a revision size $\boldsymbol{\delta}^{(t)} := \alpha^{(t)}\boldsymbol{p}^{(t)}$, and aims to produce an improved approximation. The minimization problem can be transformed into a root, if interior, finding problem, *i.e.*

$$\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \qquad \Rightarrow \qquad \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^*) := \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} = \mathbf{0}\,.$$

However, solving $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \mathbf{0}$ will only provide a stationary solution, which may be an inflection point, or even a local maximum. Moreover, Newton-Raphson method does not guarantee convergence if we start far away from the optimal point $\boldsymbol{\theta}^*$. We therefore need some assumptions for more precise results and they will be discussed in the next subsection.

To begin with, by the first-order Taylor expansion of $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ around $\boldsymbol{\theta}^{(t)}$,

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t+1)}) \approx \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) + \left(\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(t)})\right)(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) + \left(\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(t)})\right)\boldsymbol{\delta}^{(t)}\,. \tag{6.2.2}$$

By equating the rightmost term in (6.2.2) with 0, we have

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) + \left(\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(t)})\right)\boldsymbol{\delta}^{(t)} = 0 \qquad \Rightarrow \qquad \boldsymbol{\delta}^{(t)} = -\left(\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(t)})\right)^{-1}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\,, \tag{6.2.3}$$

which leads to the update

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left(\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(t)})\right)^{-1}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\,. \tag{6.2.4}$$

Based on the above settings, we implement the following Newton-Raphson algorithm in **R** environment as follows:

```
> source("Linear_Model.R")
>
> Newton <- function(X, y, epochs=1e3, tol=1e-8){
+     X <- cbind(1, X)                        # add a column of ones into X
+     theta_t <- rep(0, ncol(X))              # Initialize θ^(0) = 0
+     for (epoch in 1:epochs){
+         dg <- Gradient(X, y, theta_t)       # ∇_θ L(θ^(t))
```

```
+          dH <- Hessian(X, y, theta_t)       # ∇_θ ∇_θ^⊤ L(θ^(t))
+          # θ^(t+1) = θ^(t) − (∇_θ ∇_θ^⊤ L(θ^(t)))^{-1} ∇_θ L(θ^(t))
+          theta_new <- theta_t - solve(dH)%*%dg
+          delta <- theta_new - theta_t       # δ = θ^(t+1) − θ^(t)
+          theta_t <- theta_new
+          if (sum(delta^2) < tol^2){         # ||θ^(t+1) − θ^(t)||_2^2
+              return (list(Method="Newton-Raphson", nsteps=epoch,
+                           b=theta_t[1], omega=theta_t[-1]))
+          }
+      }
+      return (list(Method="Newton-Raphson", nsteps=epoch,
+                   b=theta_t[1], omega=theta_t[-1]))
+ }
```

Programme 6.2.1: Newton-Raphson algorithm in **R** saved as `Newton_Raphson.R`.

Here the `Gradient()` and `Hessian()` functions from `Linear_Model` returns respectively the gradient vector and the Hessian matrix with respect to the parameters $\boldsymbol{\theta}$ of a chosen loss function.

Similarly, in Python:

```
import numpy as np
from Linear_Model import Gradient, Hessian

def Newton(X, y, epochs=1e3, tol=1e-8):
    if len(X.shape) == 1:       # reshape X into a column vector if X is 1-d
        X = X.reshape(-1, 1)
    # in the following, add a column of ones into X
    intercept = np.ones(len(X)).reshape(-1,1)
    X, y = np.hstack((intercept, X)), np.array(y)
    # Initialize θ^(0) = 0
    theta_t = np.zeros(np.shape(X)[1])
    for epoch in range(int(epochs)):
        dg = Gradient(X, y, theta_t)                    # ∇_θ L(θ^(t))
        dH = Hessian(X, y, theta_t)                     # ∇_θ ∇_θ^⊤ L(θ^(t))
        # θ^(t+1) = θ^(t) − (∇_θ ∇_θ^⊤ L(θ^(t)))^{-1} ∇_θ L(θ^(t))
        theta_new = theta_t - np.linalg.inv(dH) @ dg
        delta = theta_new - theta_t                     # δ = θ^(t+1) − θ^(t)
        theta_t = theta_new
        if sum(delta**2) < tol**2:                      # ||θ^(t+1) − θ^(t)||_2^2
            return ({"Method": "Newton-Raphson", "nsteps": epoch+1,
                     "b": theta_t[0], "omega": theta_t[1:]})

    return ({"Method": "Newton-Raphson", "nsteps": epoch+1,
             "b": theta_t[0], "omega": theta_t[1:]})
```

Programme 6.2.2: Newton-Raphson algorithm in Python saved as `Newton_Raphson.py`.

6.2.1.1 Quadratic Convergence Rate of Newton-Raphson Method

We first call out the notion of quadratic convergence:

Definition 6.2.1. Suppose that the sequence $\{\boldsymbol{\theta}^{(t)}\}$ converges to $\boldsymbol{\theta}^*$, we say the convergence is *quadratic* if

$$\limsup_{t\to\infty} \frac{\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2}{\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2} < \infty\,.$$

Theorem 6.2.1. (see *e.g.* [2]) Assume that $L \in C^3$, and support that there exists a root $\boldsymbol{\theta}^* \in \mathbb{R}^D$ such that $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^*) = \mathbf{0}$ and $\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^*)$ is invertible. Then, there exists $\varepsilon > 0$ such that for any initial point $\boldsymbol{\theta}^{(0)}$ in the open ball $B(\boldsymbol{\theta}^*, \varepsilon)$ the root finding iteration in (6.2.4) converges to $\boldsymbol{\theta}^*$, *i.e.* $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 \to 0$, and the convergence rate is quadratic.

To prove this result, we start with the following lemma:

Lemma 6.2.1. Let $\boldsymbol{A} : \mathbb{R}^D \to \mathbb{R}^{D\times D}$ be a matrix-valued function that is continuous at $\boldsymbol{u} \in \mathbb{R}^D$. If $\boldsymbol{A}(\boldsymbol{u})^{-1}$ exists, then there exists a scalar $\varepsilon > 0$ such that $\boldsymbol{A}(\boldsymbol{v})^{-1}$ also exists for all $\boldsymbol{v} \in B(\boldsymbol{u}, \varepsilon)$.

This Lemma follows easily by first noting that the function $|\boldsymbol{A}(\boldsymbol{x})|$ is continuous, and so there should be a neighbourhood in $\boldsymbol{x}$ around $\boldsymbol{u}$ so that $|\boldsymbol{A}(\boldsymbol{x})| \neq 0$, the claim then follows by a simple application of Cramer's rule [2, 7, 14, 17].

Since $L \in \mathcal{C}^3$ (thrice continuously differentiable) and $\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^*)$ is invertible, by Lemma 6.2.1, there exists $\varepsilon > 0$, $c_1 > 0$, and $c_2 > 0$ such that if $\boldsymbol{\theta}^{(0)}, \boldsymbol{\theta} \in \{\tilde{\boldsymbol{\theta}} : \|\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}^*\|_2 \leq \varepsilon\}$, in light of Taylor expansion for $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ with a remainder up to the second order:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(0)}) + \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}) + \frac{1}{2}\left(\sum_{i=1}^D \sum_{j=1}^D (\theta_i - \theta_i^{(0)}) \cdot (\theta_j - \theta_j^{(0)}) \cdot \frac{\partial^3 L(\boldsymbol{\theta}^{(0)})}{\partial\theta_i \partial\theta_j \partial\theta_d}\right)_{d=1}^D,$$

and that

$$\left\| \left(\sum_{i=1}^D \sum_{j=1}^D (\theta_i - \theta_i^{(0)}) \cdot (\theta_j - \theta_j^{(0)}) \cdot \frac{\partial^3 L(\boldsymbol{\theta}^{(0)})}{\partial\theta_i \partial\theta_j \partial\theta_d}\right)_{d=1}^D \right\|_2 \leq 2c_1 \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}\|_2^2\,,$$

leading to

$$nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(0)}) - \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})\|_2 \leq c_1 \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}\|_2^2\,, \tag{6.2.5}$$

and the inverse of $\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta})$ is well-defined such that

$$\|(\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}))^{-1}\|_2 \leq c_2\,. \tag{6.2.6}$$

Substituting $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ and using the fact that $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^*) = \mathbf{0}$, (6.2.5) becomes

$$\|-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(0)}) + \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*)\|_2 \leq c_1 \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2^2\,. \tag{6.2.7}$$

For the first iteration step $\boldsymbol{\theta}^{(1)} = \boldsymbol{\theta}^{(0)} - \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(0)})$, substracting both sides with $\boldsymbol{\theta}^*$ and then

taking norm yields

$$\begin{aligned}\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2 &= \|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*-\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})^{-1}\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(0)})\|_2\\ &= \left\|\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})^{-1}\left(\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*)-\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(0)})\right)\right\|_2\\ &\le \|\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})^{-1}\|_2\cdot\|\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*)-\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(0)})\|_2\\ &\le c_1c_2\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2^2\,, \end{aligned} \tag{6.2.8}$$

where the first inequality is due to Lemma 3.3.1 and the second inequality is due to (6.2.6) and (6.2.7).

Suppose that $\boldsymbol{\theta}^{(0)}$ is chosen such that

$$\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2 \le \frac{\alpha}{c_1c_2}\,, \quad \text{for some} \quad \alpha\in(0,1)\,,$$

then (6.2.8) yields that

$$\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2 \le c_1c_2\left(\frac{\alpha}{c_1c_2}\right)^2 = \frac{\alpha^2}{c_1c_2} \le \frac{\alpha}{c_1c_2}\,, \tag{6.2.9}$$

hence, using the above argument, by replacing $\boldsymbol{\theta}^{(0)}$ by $\boldsymbol{\theta}^{(1)}$, we immediately conclude that $\|\boldsymbol{\theta}^{(2)}-\boldsymbol{\theta}^*\|_2 \le c_1c_2\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2^2$, and then $\|\boldsymbol{\theta}^{(2)}-\boldsymbol{\theta}^*\|_2 \le \alpha/(c_1c_2)$. We actually further have

$$\|\boldsymbol{\theta}^{(2)}-\boldsymbol{\theta}^*\|_2 \le c_1c_2\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2^2 \le c_1c_2(c_1c_2)^2\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2^4 \le (c_1c_2)^3\left(\frac{\alpha}{c_1c_2}\right)^4 = \frac{\alpha^{2^2}}{c_1c_2}\,.$$

Generally, at the t-iteration, the above argument can be used by replacing $\boldsymbol{\theta}^{(0)}$ by $\boldsymbol{\theta}^{(t-1)}$ indeed, if $\|\boldsymbol{\theta}^{(t-1)}-\boldsymbol{\theta}^*\|_2 \le \alpha/(c_1c_2)$, we can obtain that $\|\boldsymbol{\theta}^{(t+1)}-\boldsymbol{\theta}^*\|_2 \le c_1c_2\|\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\|_2^2$ from which, we have the quadratic rate of convergence that:

$$\limsup_{t\to\infty}\frac{\|\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\|_2}{\|\boldsymbol{\theta}^{(t-1)}-\boldsymbol{\theta}^*\|_2^2} \le c_1c_2 < \infty\,;$$

besides, we also have $\|\boldsymbol{\theta}^{(t+1)}-\boldsymbol{\theta}^*\|_2 \le \alpha/(c_1c_2)$. Furthermore, by successive substitutions and $\alpha\in(0,1)$,

$$\|\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\|_2 \le \frac{\alpha^{2^t}}{c_1c_2}\,, \tag{6.2.10}$$

therefore, the limit $\lim_{t\to\infty}\|\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\|_2 = 0$ and thus the sequence $\{\boldsymbol{\theta}^{(t)}\}$ converges to $\boldsymbol{\theta}^*$.

6.2.2 Modified Gauss-Newton Algorithm

Gauss-Newton algorithm is a modification of the classical Newton-Raphson's method which is used to numerically find the minimum point of a broad class of functions. Particularly, it is designed for solving for many practical least square problems subject to regularization; also see Section 5.2, and hence it is widely applied in non-parametric and semi-parametric regressions in machine learning and statistics. The algorithm essentially makes use of linear approximation for the following residual functions $r_n(\boldsymbol{\theta})$ so that the step of calculating the second or higher order derivatives is not necessary. This algorithm first appeared in the work of Gauss (1809), and we shall present a modified version below. Consider the parameters $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_D)^\top \in \mathbb{R}^D$ and the residual function

$$\boldsymbol{r}(\boldsymbol{\theta}) = (r_1(\boldsymbol{\theta}), \cdots, r_N(\boldsymbol{\theta}))^\top, \qquad \text{where} \quad r_n(\boldsymbol{\theta}) = y_n - f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\,,$$

with $D \leq N$, and the sum of squares

$$S(\boldsymbol{\theta}) := \boldsymbol{r}(\boldsymbol{\theta})^\top \boldsymbol{r}(\boldsymbol{\theta}) = \sum_{n=1}^{N} r_n^2(\boldsymbol{\theta})\,.$$

Denote the Jacobian (rectangular) matrix of derivatives of $\boldsymbol{r}(\boldsymbol{\theta})$ with respect to the coordinates of $\boldsymbol{\theta}$ by:

$$\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}) = \left(\frac{\partial r_n(\boldsymbol{\theta})}{\partial \theta_d}\right)_{n=1,d=1}^{N,D}, \tag{6.2.11}$$

The procedure of the algorithm is illustrated as below:

Algorithm 6.4 Modified Gauss-Newton Algorithm

1) Initialize $\boldsymbol{\theta}^{(0)}$ and compute $S(\boldsymbol{\theta}^{(0)})$. Label $t = 0$.
2) For $n = 1, \cdots, N$ and $d = 1, \cdots, D$, compute

$$\left(a_{nd}^{(t)}\right)_{n=1,d=1}^{N,D} \triangleq \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)}) = \left(\frac{\partial r_n(\boldsymbol{\theta}^{(t)})}{\partial \theta_d}\right)_{n=1,d=1}^{N,D} =: \boldsymbol{J}_{\boldsymbol{r}}^{(t)} \in \mathbb{R}^{N\times D}\,,$$

and

$$\boldsymbol{b}^{(t)} = \left(b_d^{(t)}\right)_{d=1}^{D} := (\boldsymbol{J}_{\boldsymbol{r}}^{(t)})^\top \boldsymbol{r}(\boldsymbol{\theta}^{(t)}) = \left(\sum_{n=1}^{N} r_n(\boldsymbol{\theta}^{(t)}) a_{nd}^{(t)}\right)_{d=1}^{D}.$$

3) Solve $\boldsymbol{p}^{(t)}$ the solution of the following linear system:

$$(\boldsymbol{J}_{\boldsymbol{r}}^{(t)})^\top \boldsymbol{J}_{\boldsymbol{r}}^{(t)} \boldsymbol{p}^{(t)} = -\boldsymbol{b}^{(t)}\,.$$

If $\operatorname{rank}\left((\boldsymbol{J}_{\boldsymbol{r}}^{(t)})^\top \boldsymbol{J}_{\boldsymbol{r}}^{(t)}\right) = D$, the system admits a unique solution $\boldsymbol{p}^{(t)} = -\left((\boldsymbol{J}_{\boldsymbol{r}}^{(t)})^\top \boldsymbol{J}_{\boldsymbol{r}}^{(t)}\right)^{-1} \boldsymbol{b}^{(t)}$; otherwise take $\boldsymbol{p}^{(t)} = -\boldsymbol{b}^{(t)}$.
4) Set $q(\lambda) := S\left(\boldsymbol{\theta}^{(t)} + \lambda \boldsymbol{p}^{(t)}\right)$. Find $\lambda^{(t)} \arg\min_\lambda q(\lambda)$, for instance, using the golden-section search method (see Subsubsection6.2.2.1). Also set $\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} + \lambda^{(t)} \boldsymbol{p}^{(t)}$ and then compute $S(\boldsymbol{\theta}^{(t+1)})$.
5) Repeat Steps (2) to (4) for $t \leftarrow t + 1$ until the usual termination condition is satisfied, *i.e.*

$$\left|S(\boldsymbol{\theta}^{(t+1)}) - S(\boldsymbol{\theta}^{(t)})\right| < \varepsilon \qquad \text{and} \qquad \left\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right\|_2 < \varepsilon\,,$$

for some small enough preassigned $\varepsilon > 0$.
6) Return $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t+1)}$ if the termination condition is reached.

6.2.2.1 Derivation of the Convergence of Modified Gauss-Newton Algorithm

Let us motivate the formulation of the numerical scheme. Given an initial point $\boldsymbol{\theta}^{(0)}$, which is supposedly close to the unknown minimum point

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} S(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} (y_n - f_{\boldsymbol{\theta}}(\boldsymbol{x}_n))^2 \, .$$

Approximating the function $\boldsymbol{r}$ up to the first order gives

$$\boldsymbol{r}(\boldsymbol{\theta}) \approx \boldsymbol{r}(\boldsymbol{\theta}^{(0)}) + \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}), \qquad \text{where} \quad \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{r}(\boldsymbol{\theta}) = \left(\frac{\partial r_n(\boldsymbol{\theta})}{\partial \theta_d} \right)_{n=1,d=1}^{N,D} ,$$

and the minimization of $S(\boldsymbol{\theta})$ is "nearly" equivalent to

$$\min_{\boldsymbol{\theta}} \left\| \boldsymbol{r}(\boldsymbol{\theta}^{(0)}) + \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}) \right\|_2^2 , \tag{6.2.12}$$

where $\|\cdot\|_2$ is the Euclidean norm. The latter minimization problem is no different from a linear least square problem in terms of $\Delta := \boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}$. By the multiple regression formula, the first order condition for (6.2.12) for solving for $\hat{\Delta} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^{(0)}$ should satisfy that

$$\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})^\top \left(\boldsymbol{r}(\boldsymbol{\theta}^{(0)}) + \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)}) \hat{\Delta} \right) = \boldsymbol{0} \, , \tag{6.2.13}$$

which gives

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(0)} - \left(\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})^\top \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)}) \right)^{-1} \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})^\top \boldsymbol{r}(\boldsymbol{\theta}^{(0)}) \, , \tag{6.2.14}$$

with which, generally, the iteration scheme reads as:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left(\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)})^\top \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)}) \right)^{-1} \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)})^\top \boldsymbol{r}(\boldsymbol{\theta}^{(t)}) \, , \tag{6.2.15}$$

which is mostly plaussible as $\text{rank}\{\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)})\} = D$ that can be pretty often realised in practice. Particularly, for a general non-linear regression which aims to find the regressor $y = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ for the dataset $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, the residual r_n is given by:

$$r_n(\boldsymbol{\theta}) = y_n - f_{\boldsymbol{\theta}}(\boldsymbol{x}_n), \qquad \text{and so} \quad \frac{\partial r_n(\boldsymbol{\theta})}{\partial \theta_d} = -\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)}{\partial \theta_d} \, , \qquad d = 1, \cdots, N \, ,$$

therefore the iteration scheme (6.2.15) becomes:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \left(\boldsymbol{J}_f(\boldsymbol{\theta}^{(t)})^\top \boldsymbol{J}_f(\boldsymbol{\theta}^{(t)}) \right)^{-1} \boldsymbol{J}_f(\boldsymbol{\theta}^{(t)})^\top \boldsymbol{r}(\boldsymbol{\theta}^{(t)}), \quad \text{where} \quad \boldsymbol{J}_f(\boldsymbol{\theta}) = \left(\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)}{\partial \theta_d} \right)_{n=1,d=1}^{N,D} . \tag{6.2.16}$$

For each iteration scheme, the system of equations (6.2.15) or particularly (6.2.16) in $\hat{\Delta}^{(t)} = \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}$ can be solved by applying Cholesky decomposition (see Section 3.3 or the general QR factorization, also see Atkinson (2008)) to the symmetric matrix $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)})^\top \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}^{(t)})$. Often, for a large system with a large value of N, iterative methods such as conjugate gradient method [16] can still be used.

Alternatively, this modified Gauss-Newton algorithm can also be derived from the celebrated Newton-Raphson's method, and hence the convergence follows by the discussion in Subsubsection 6.2.1.1; indeed, we aim to seek for the root(s) of $\nabla_{\boldsymbol{\theta}} S(\boldsymbol{\theta}) = \boldsymbol{0}$, by using Newton's method, we first see that

$$\nabla_{\boldsymbol{\theta}} S(\boldsymbol{\theta}^{(0)}) + \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top S(\boldsymbol{\theta}^{(0)})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)}) + O(\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}\|_2^2) = \boldsymbol{0} \, ,$$

based on which the recursive relation for the updates can be defined via

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left(\boldsymbol{H}(\boldsymbol{\theta}^{(t)}) \right)^{-1} \nabla_{\boldsymbol{\theta}} S(\boldsymbol{\theta}^{(t)}) \, , \tag{6.2.17}$$

where $\boldsymbol{H}(\boldsymbol{\theta})$ is the Hessian matrix of $S(\boldsymbol{\theta})$, which is

$$\left(\frac{\partial^2 S(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\right)_{i=1,j=1}^{D,D}.$$

Since $S(\boldsymbol{\theta}) = \sum_{n=1}^{N} r_n^2(\boldsymbol{\theta})$, we actually have:

$$\nabla_{\boldsymbol{\theta}} S(\boldsymbol{\theta}) = \left(2\sum_{n=1}^{N} r_n(\boldsymbol{\theta})\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_1}, \cdots, 2\sum_{n=1}^{N} r_n(\boldsymbol{\theta})\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_D}\right),$$

and we further differentiate this last gradient leads us to the Hessian $\boldsymbol{H} = (H_{ij})_{i=1,j=1}^{D,D}$, where

$$H_{ij}(\boldsymbol{\theta}) = 2\sum_{n=1}^{N}\left(\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_i}\cdot\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_j} + r_n(\boldsymbol{\theta})\frac{\partial^2 r_n(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\right), \qquad i,j = 1,\cdots,D\,.$$

If one ignores the second-order derivatives terms in the above expression, the Hessian is approximately equal to:

$$H_{ij}(\boldsymbol{\theta}) \approx 2\sum_{n=1}^{N}(\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}))_{ni}(\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}))_{nj}\,,$$

where $(\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}))_{n\ell}$ is the (n,ℓ) entry of the Jacobian matrix $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta})$. By substituting

$$\nabla_{\boldsymbol{\theta}} S(\boldsymbol{\theta}) = 2\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta})^{\top}\boldsymbol{r}(\boldsymbol{\theta}) \qquad \text{and} \qquad \boldsymbol{H}(\boldsymbol{\theta}) \approx 2\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta})^{\top}\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta})\,,$$

into (6.2.17), we again arrive at (6.2.15) as desired.

The effectiveness of the convergence of the modified Gauss-Newton algorithm can be much enhanced especially when the magnitude of the second-order cross term is relatively small, for instance, in the sense that

$$\left|r_n(\boldsymbol{\theta})\frac{\partial^2 r_n(\boldsymbol{\theta})}{\partial\theta_i\partial\theta_j}\right| \ll \left|\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_i}\cdot\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_j}\right|, \qquad \forall\, i,\, j,\, n\,,$$

which is plausible particularly when $r_n(\boldsymbol{\theta})$'s are small and stable, and when these $r_n(\boldsymbol{\theta})$'s are nearly flattened around the minimum point, making its order to zero being twice of that $\frac{\partial r_n(\boldsymbol{\theta})}{\partial\theta_d}$. In general, the convergence rate of the algorithm is linear in n; but also under some mild conditions, we can obtain a quadratic rate; indeed, the local convergence rate for the Gauss-Newton method can be much inferior to that of the Newton-Raphson's method if the residuals $\boldsymbol{r}(\boldsymbol{\theta})$ are large. To this end, we shall state two fundamental results from [3]:

Lemma 6.2.2. [3] Suppose that $\boldsymbol{r}(\boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}^N$ is continuously differentiable on $\mathbb{R}^D$. Then, for any $\boldsymbol{\theta}, \boldsymbol{\theta}+\boldsymbol{\Delta} \in \mathbb{R}^D$, where $\boldsymbol{\Delta} \in \mathbb{R}^D$ is a non-zero perturbation, we have

$$\boldsymbol{r}(\boldsymbol{\theta}+\boldsymbol{\Delta}) = \boldsymbol{r}(\boldsymbol{\theta}) + \int_0^1 \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}+t\boldsymbol{\Delta})\boldsymbol{\Delta}\,dt\,. \tag{6.2.18}$$

Proof. We first consider an individual $r_n(\boldsymbol{\theta}) \in \mathbb{R}$ and the simple application of the fundamental theorem of calculus gives

$$r_n(\boldsymbol{\theta}+1\cdot\boldsymbol{\Delta}) = r_n(\boldsymbol{\theta}+0\cdot\boldsymbol{\Delta}) + \int_0^1 \frac{\partial r_n(\boldsymbol{\theta}+t\cdot\boldsymbol{\Delta})}{\partial t}\,dt\,,$$

by noting that,

$$\frac{\partial r_n(\boldsymbol{\theta}+t\boldsymbol{\Delta})}{\partial t} = \sum_{d=1}^{D}\frac{\partial r_n(\boldsymbol{\theta}+t\boldsymbol{\Delta})}{\partial(\theta_d+t\Delta_d)}\cdot\frac{\partial(\theta_d+t\Delta_d)}{\partial t} = \nabla_{\boldsymbol{\theta}} r_n(\boldsymbol{\theta}+t\boldsymbol{\Delta})^{\top}\boldsymbol{\Delta}\,.$$

by recaling the definition of the Jacobian matrix $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}) := \nabla_{\boldsymbol{\theta}}\boldsymbol{r}(\boldsymbol{\theta})$, we complete the proof. □

Theorem 6.2.2. [3] Suppose that $S/N : \boldsymbol{\theta} \in \mathbb{R}^D \mapsto (\boldsymbol{r}(\boldsymbol{\theta})^\top/\sqrt{N})(\boldsymbol{r}(\boldsymbol{\theta})/\sqrt{N}) =: \tilde{\boldsymbol{r}}(\boldsymbol{\theta})^\top \tilde{\boldsymbol{r}}(\boldsymbol{\theta}) \in \mathcal{C}^2$ on $\mathbb{R}^D$, and there exists $\boldsymbol{\theta}^* \in \mathbb{R}^D$ so that $\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)^\top \tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*) = \nabla_{\boldsymbol{\theta}} S(\boldsymbol{\theta}^*)/(2N) = \mathbf{0}$. Assume that:

(**A1**) (*Lipschitz continuity*) The Jacobian matrix $\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}) := \nabla_{\boldsymbol{\theta}} \tilde{\boldsymbol{r}}(\boldsymbol{\theta})$ is Lipschitz continuous on $\mathbb{R}^D$ with a Lipschitz constant $C > 0$:

$$\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}) - \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}')\|_2 \le C\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2\,, \qquad \forall\, \boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^D\,; \tag{6.2.19}$$

(**A2**) $\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})\|_2 \le \alpha$ for all $\boldsymbol{\theta} \in \mathbb{R}^D$;

(**A3**) There exists an $\sigma \ge 0$ such that

$$\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})^\top \tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)\|_2 \le \sigma\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|_2\,, \qquad \forall\, \boldsymbol{\theta} \in \mathbb{R}^D\,; \tag{6.2.20}$$

(**A4**) $\sigma < \lambda_{\min}$, where $\lambda_{\min}$ is the smallest eigenvalue of the matrix $\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)^\top \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)$.

Then for any $\gamma \in (1, \lambda_{\min}/\sigma)$, there exists $\varepsilon > 0$ such that for any initial point $\boldsymbol{\theta}^{(0)}$ in the open ball $B(\boldsymbol{\theta}^*, \varepsilon)$, the iteration scheme in (6.2.15) converges to $\boldsymbol{\theta}^*$ and satisfies the following recursive bounds:

$$\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 \le \frac{\gamma\sigma}{\lambda_{\min}}\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 + \frac{\gamma C\alpha}{2\lambda_{\min}}\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2\,. \tag{6.2.21}$$

Remark 6.2.1. From Assumption (**A1**) of Theorem 6.2.2 and the definition of $\boldsymbol{\theta}^*$, we have

$$\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})^\top \tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)\|_2 = \left\| \left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}) - \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*) \right)^\top \tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*) \right\|_2 \le \|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}) - \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)\|_2 \cdot \|\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)\|_2 \le C\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|_2 \cdot \|\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)\|_2\,.$$

Also notice that by the (weak) Law of Large Number,

$$\|\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)\|_2^2 = \frac{1}{N}\sum_{n=1}^N r_n^2(\boldsymbol{\theta}^*) \xrightarrow{p} \mathbb{E}\Big((\mathrm{y} - f_{\boldsymbol{\theta}^*}(\mathbf{x}))^2 \Big) \le C_2\,,$$

Therefore, we arrive at the inequality in Assumption (**A3**) that $\sigma := C \cdot C_2$.

Proof of Theorem 6.2.2. From Lemma 6.2.2 and Assumption (**A1**), we immediately have

$$\begin{aligned} \|\tilde{\boldsymbol{r}}(\boldsymbol{\theta} + \boldsymbol{\Delta}) - \tilde{\boldsymbol{r}}(\boldsymbol{\theta}) - \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})\boldsymbol{\Delta}\|_2 &\le \int_0^1 \|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta} + t\boldsymbol{\Delta}) - \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})\|_2 \cdot \|\boldsymbol{\Delta}\|_2\, dt \\ &\le \int_0^1 C\|t\boldsymbol{\Delta}\|_2 \cdot \|\boldsymbol{\Delta}\|_2\, dt \\ &= C\|\boldsymbol{\Delta}\|_2^2 \int_0^1 t\, dt = \frac{C}{2}\|\boldsymbol{\Delta}\|_2^2\,. \end{aligned} \tag{6.2.22}$$

Since $\lambda_{\max} \ge \lambda_{\min} > \sigma \ge 0$, by Property 22 in Section 3.3, we have $1/\lambda_{\max} > 0$ as the smallest eigenvalue of $(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)^\top \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*))^{-1}$, *i.e.* it is invertible. On the other hand, $1/\lambda_{\min}$ is the largest eigenvalue of $(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)^\top \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*))^{-1}$ such that

$$\left\| \left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*)^\top \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^*) \right)^{-1} \right\|_2 \le \frac{1}{\lambda_{\min}}\,.$$

Let γ be a constant that $1 < \gamma < \lambda_{\min}/\sigma$. By Lemma 6.2.1, and the continuity of the $\mathcal{L}^2$-norm of $(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})\boldsymbol{J}_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\theta}))^{-1}$, there exists a scalar $\varepsilon > 0$ such that the inverse of $\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})^\top \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})$ also exists for all $\boldsymbol{\theta} \in B(\boldsymbol{\theta}^*, \varepsilon)$, and the following inequality holds:

$$\left\| \left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta})^\top \boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}) \right)^{-1} \right\|_2 \le \frac{\gamma}{\lambda_{\min}}\,. \tag{6.2.23}$$

Then, we consider

$$\begin{aligned}\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2 &= \left\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*-\left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})\right)^{-1}\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})\right\|_2 \\ &= \left\|\left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})\right)^{-1}\left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*)-\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})\right)\right\|_2 \\ &\leq \left\|\left(\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})\right)^{-1}\right\|_2\cdot\left\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*)-\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})\right\|_2,\end{aligned}$$

where the inequality is due to the Lemma 3.3.1. Furthermore, by (6.2.23) and telescoping the second factor with the term $\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)$, we have

$$\begin{aligned}\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2 &\leq \frac{\gamma}{\lambda_{\min}}\cdot\left\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)-\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\left(\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)-\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})-\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^*-\boldsymbol{\theta}^{(0)})\right)\right\|_2 \\ &\leq \frac{\gamma}{\lambda_{\min}}\left(\left\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)\right\|_2+\left\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})^\top\left(\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)-\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})-\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^*-\boldsymbol{\theta}^{(0)})\right)\right\|_2\right).\end{aligned}$$

Finally, by Assumptions (**A1**), (**A2**), and (**A3**), we arrive

$$\begin{aligned}\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2 &\leq \frac{\gamma}{\lambda_{\min}}\left(\sigma\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2+\|\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})\|_2\cdot\|\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^*)-\tilde{\boldsymbol{r}}(\boldsymbol{\theta}^{(0)})-\boldsymbol{J}_{\tilde{\boldsymbol{r}}}(\boldsymbol{\theta}^{(0)})(\boldsymbol{\theta}^*-\boldsymbol{\theta}^{(0)})\|_2\right) \\ &\leq \frac{\gamma\sigma}{\lambda_{\min}}\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2+\frac{\gamma C\alpha}{2\lambda_{\min}}\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2^2,\end{aligned}$$

where the last inequality is due to (6.2.22) with $\boldsymbol{\theta}^*=\boldsymbol{\theta}+\boldsymbol{\Delta}$. Therefore, the recursive claim follows. Next, if we choose the radius of the open ball $B(\boldsymbol{\theta}^*,\tilde{\varepsilon})$ that

$$\tilde{\varepsilon}=\min\left\{\varepsilon,\frac{\lambda_{\min}-\gamma\sigma}{\gamma C\alpha}\right\},$$

we then have

$$\begin{aligned}\|\boldsymbol{\theta}^{(1)}-\boldsymbol{\theta}^*\|_2 &\leq \|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2\left(\frac{\gamma\sigma}{\lambda_{\min}}+\frac{\gamma C\alpha}{2\lambda_{\min}}\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2\right) \\ &\leq \|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2\left(\frac{\gamma\sigma}{\lambda_{\min}}+\frac{\gamma C\alpha}{2\lambda_{\min}}\cdot\frac{\lambda_{\min}-\gamma\sigma}{\gamma C\alpha}\right) \\ &= \frac{\gamma\sigma+\lambda_{\min}}{2\lambda_{\min}}\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2.\end{aligned}$$

Since $\gamma<\lambda_{\min}/\sigma$, we have

$$\rho:=\frac{\gamma\sigma+\lambda_{\min}}{2\lambda_{\min}}\in(0.5,1).$$

Similar argument applies to all other $\boldsymbol{\theta}^{(t)}$:

$$\|\boldsymbol{\theta}^{(t)}-\boldsymbol{\theta}^*\|_2\leq\rho\|\boldsymbol{\theta}^{(t-1)}-\boldsymbol{\theta}^*\|_2\leq\rho^2\|\boldsymbol{\theta}^{(t-2)}-\boldsymbol{\theta}^*\|_2\leq\cdots\leq\rho^t\|\boldsymbol{\theta}^{(0)}-\boldsymbol{\theta}^*\|_2.$$

Therefore, the sequence of updates $\{\boldsymbol{\theta}^{(t)}\}$ converges to $\boldsymbol{\theta}^*$. □

From the derivation above, we build the Gauss-Newton algorithm in **R**:

```
> source("Linear_Model.R")
>
> Gauss_Newton <- function(X, y, epochs=1e3, tol=1e-8){
+     X <- cbind(1, X)                          # add a column of ones into X
+     theta_t <- rep(0, ncol(X))                # Initialize θ^(0) = 0
+     for (epoch in 1:epochs){
+         r_t <- Error(X, y, theta_t)
+         J_r <- Error_Gradient(X, y, theta_t)
+         # θ^(t+1) = θ^(t) - (J_r(θ^(t))^T J_r(θ^(t)))^{-1} J_r(θ^(t))^T r(θ^(t))
+         theta_new <- theta_t - solve(t(J_r) %*% J_r) %*% t(J_r) %*% r_t
```

```
+          delta <- theta_new - theta_t                # δ = θ^(t+1) − θ^(t)
+          theta_t <- theta_new
+          if (sum(delta^2) < tol^2){                  # ||θ^(t+1) − θ^(t)||_2^2
+              return (list(Method="Gauss-Newton", nsteps=epoch,
+                           b=theta_t[1], omega=theta_t[-1]))
+          }
+      }
+      return (list(Method="Gauss-Newton", nsteps=epoch,
+                   b=theta_t[1], omega=theta_t[-1]))
+ }
```

Programme 6.2.3: Gauss-Newton algorithm in **R** saved as `Gauss_Newton.R`.

Here the `Error()` and `Error_Gradient()` functions from `Linear_Model` return $\boldsymbol{r}(\boldsymbol{\theta})$ and $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta})$ respectively.

Similarly, in Python:

```
import numpy as np
from Linear_Model import Error, Error_Gradient

def Gauss_Newton(X, y, epochs=1e3, tol=1e-8):
    if len(X.shape) == 1:       # reshape X into a column vector if X is 1-d
        X = X.reshape(-1, 1)
    # in the following, add a column of ones into X
    intercept = np.ones(len(X)).reshape(-1,1)
    X, y = np.hstack((intercept, X)), np.array(y)
    # Initialize θ^(0) = 0
    theta_t = np.zeros(np.shape(X)[1])
    for epoch in range(int(epochs)):
        r_t = Error(X, y, theta_t)
        J_r = Error_Gradient(X, y, theta_t)
        # θ^(t+1) = θ^(t) − (J_r(θ^(t))^T J_r(θ^(t)))^(−1) J_r(θ^(t))^T r(θ^(t))
        theta_new = theta_t - np.linalg.inv(J_r.T @ J_r) @ J_r.T @ r_t
        delta = theta_new - theta_t                          # δ = θ^(t+1) − θ^(t)
        theta_t = theta_new
        if sum(delta**2) < tol**2:                           # ||θ^(t+1) − θ^(t)||_2^2
            return ({"Method": "Gauss-Newton", "nsteps": epoch+1,
                     "b": theta_t[0], "omega": theta_t[1:]})

    return ({"Method": "Gauss-Newton", "nsteps": epoch+1,
             "b": theta_t[0], "omega": theta_t[1:]})
```

Programme 6.2.4: Gauss-Newton algorithm in Python saved as `Gauss_Newton.py`

```
    r_i = H_0_inv @ q_i                                  # r^(t-M) = (H_0^(t))^{-1} q^(t-M)
    for i in range(M):
        # β^(i) = (g^(i))^T r^(i) / ((g^(i))^T δ^(i))
        beta[i] = g_list[i].T @ r_i / (g_list[i].T @ delta_list[i])
        # r^(i+1) = r^(i) + δ^(i)(α^(i) - β^(i))
        r_i = r_i + delta_list[i] * (alpha[i] - beta[i])
    return r_i                                           # (H^(t))^{-1} ∇_θ L(θ^(t)) = r^(t)

def L_BFGS(X, y, eta=1, epochs=1e3, tol=1e-8, M=10):
    if len(X.shape) == 1:       # reshape X into a column vector if X is 1-d
        X = X.reshape(-1, 1)
    # in the following, add a column of ones into X
    intercept = np.ones(len(X)).reshape(-1,1)
    X, y = np.hstack((intercept, X)), np.array(y)
    # Initialize θ^(0) = 0 and (H^(0))^{-1} = I_D
    theta_t = np.zeros(np.shape(X)[1])
    g_list, delta_list = deque(maxlen=M), deque(maxlen=M)
    inv_H = np.identity(np.shape(X)[1])
    for epoch in range(int(epochs)):
        dg = Gradient(X, y, theta_t)
        p_t = -L_BFGS_two_loop(dg, delta_list, g_list, inv_H, min(epoch, M))
        theta_new = theta_t + eta * np.squeeze(p_t)          # θ^(t+1) = θ^(t) + η p^(t)
        dg_new = Gradient(X, y, theta_new)

        g_list.append((dg_new - dg).reshape(-1, 1))
        delta_list.append((theta_new - theta_t).reshape(-1, 1))
        delta = theta_new - theta_t                          # δ = θ^(t+1) - θ^(t)
        theta_t = theta_new

        if sum(delta**2) < tol**2:                           # ||θ^(t+1) - θ^(t)||_2^2
            return ({"Method": "L-BFGS", "nsteps": epoch+1,
                     "b": theta_t[0], "omega": theta_t[1:]})

    return ({"Method": "L-BFGS", "nsteps": epoch+1,
             "b": theta_t[0], "omega": theta_t[1:]})
```

Programme 6.2.8: L-BFGS algorithm in Python saved as `L_BFGS.py`.

Here the `deque()` function in the `collections` library with `maxlen=M` argument generates a fixed length of M list-like container. It starts as an empty container and once the container is full, i.e. with length M or has M items, we append this container at the either end and removes one item at the opposite end such that the length remains at M.

6.2.5 Example with Advertising Dataset

Finally, we implement all Algorithms in **R** and Python using the "`Advertising.csv`" first introduced in Subsection 6.1.2, involving information from 200 markets about their budgets on *Television* (TV) advertisement $x_n^{(1)}$, budgets on *Radio* advertisement $x_n^{(2)}$, budgets on *Newspaper* advertisement $x_n^{(3)}$, and their annual

Sales y_n of their product, for $n = 1, \cdots, 200$.

As an illustration, consider a simple linear regression model, and the dataset $\mathcal{S} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ and the predicting model is given by:

$$f_{\boldsymbol{\omega},b}(\boldsymbol{x}) = \omega_1 x^{(1)} + \omega_2 x^{(2)} + \omega_3 x^{(3)} + b := \boldsymbol{\omega}^\top \boldsymbol{x} + b\,, \tag{6.2.46}$$

where $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^\top \in \mathbb{R}^3$ and $\boldsymbol{x} = (x^{(1)}, x^{(2)}, x^{(3)})$. For simplicity, we define $x^{(0)} = 1$ being the intercept term such that

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \omega_1 x^{(1)} + \omega_2 x^{(2)} + \omega_3 x^{(3)} + b x^{(0)} = \boldsymbol{\theta}^\top \boldsymbol{x}\,, \tag{6.2.47}$$

where $\boldsymbol{\theta} = (b, \omega_1, \omega_2, \omega_3)^\top$ and $\boldsymbol{x} = (1, x^{(1)}, x^{(2)}, x^{(3)})$. Adopting MSE $\mathcal{E}(\boldsymbol{\theta})$ as the loss function,

$$\mathcal{E}(\boldsymbol{\theta}) := \frac{1}{N}\sum_{n=1}^N \Big(y_n - f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)\Big)^2 = \frac{1}{N}\sum_{n=1}^N (y_n - \boldsymbol{\theta}^\top \boldsymbol{x}_n)^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})\,, \tag{6.2.48}$$

where $\boldsymbol{y} = (y_1, \cdots, y_N)^\top \in \mathbb{R}^N$ and $\boldsymbol{X} = (\boldsymbol{1}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \in \mathbb{R}^{N\times(1+D)}$. The gradient of (6.2.48) with respect to $\boldsymbol{\theta}$ is

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \frac{1}{N}\nabla_{\boldsymbol{\theta}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = \frac{1}{N}(-\boldsymbol{X}^\top \boldsymbol{y} - \boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta}) = -\frac{2}{N}\boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})\,. \tag{6.2.49}$$

And the Hessian matrix of (6.2.48) with respect to $\boldsymbol{\theta}$ is

$$\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta}) = \frac{2}{N}\boldsymbol{X}^\top \boldsymbol{X}\,. \tag{6.2.50}$$

Meanwhile, for the Gauss-Newton algorithm, the error and the gradient of the error are respectively:

$$\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta} \qquad \text{and} \qquad \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{\theta}) = -\boldsymbol{X}\,.$$

We therefore have the following functions in `Linear_Model.R` written in **R**:

```
> Gradient <- function(X, y, theta){
+     -2 * t(X)%*%(y - X%*%theta) / length(y)    # ∇_θ L(θ) = −2X^⊤(y − Xθ)/N
+ }
>
> Hessian <- function(X, y, theta){
+     2 * t(X)%*%X / length(y)                   # H = 2X^⊤X/N
+ }
>
> Error <- function(X, y, theta){
+     y - X%*%theta                              # r = y − Xθ
+ }
>
> Error_Gradient <- function(X, y, theta){
+     -X                                         # J_r(θ) = −X
+ }
```

Programme 6.2.9: Functions for gradients, hessians, error, and gradient of error in **R**, saved as `Linear_Model.R`.

Similarly, in Python:

```
def Gradient(X, y, theta):
    return -2 * X.T @ (y - X @ theta) / len(y)  # ∇_θ L(θ) = −2X^⊤(y − Xθ)/N
```

```
3
4  def Hessian(X, y, theta):
5      return 2 * X.T @ X / len(y)                    # H = 2X^T X/N
6
7  def Error(X, y, theta):
8      return y - X @ theta                           # r = y - Xθ
9
10 def Error_Gradient(X, y, theta):
11     return -X                                      # J_r(θ) = -X
```

Programme 6.2.10: Functions for gradients, hessians, error, and gradient of error in Python, saved as `Linear_Model.py`.

Next, in **R**, we import Programme 6.2.1 for Newton-Raphson algorithm, Programme 6.2.3 for Gauss-Newton algorithm, Programme 6.2.5 for BFGS algorithm, and Programme 6.2.7 for limited-memory BFGS algorithm. For all four algorithms, `Newton()`, `Gauss_Newton()`, `BFGS()`, and `L_BFGS()` functions, also record the number of steps via the counter `steps` for the respective algorithm to converge, *i.e.* $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\|_2$ is smaller than the tolerance level, for instance, `tol=1e-8`.

```
> source("Newton_Raphson.R"); source("Gauss_Newton.R")
> source("BFGS.R"); source("L_BFGS.R")
>
> dataset <- read.csv("Advertising.csv", row.names=1)
> X <- subset(dataset, select=-c(sales)); y <- dataset["sales"]
> X <- as.matrix(X); y <- as.vector(t(y))
>
> Newton(X, y)
$Method
[1] "Newton-Raphson"

$nsteps
[1] 2

$b
[1] 2.938889

$omega
[1]  0.045764645  0.188530017 -0.001037493

> Gauss_Newton(X, y)
$Method
[1] "Gauss-Newton"

$nsteps
[1] 2

```

```
$b
[1] 2.938889

$omega
[1]  0.045764645  0.188530017 -0.001037493

> BFGS(X, y)
$Method
[1] "BFGS"

$nsteps
[1] 12

$b
[1] 2.938889

$omega
[1]  0.045764645  0.188530017 -0.001037493

> L_BFGS(X, y)
$Method
[1] "L-BFGS"

$nsteps
[1] 12

$b
[1] 2.938889

$omega
[1]  0.045764645  0.188530017 -0.001037493

>
> model <- lm(y~X)
> list(b=coef(model)[1], omega=coef(model)[-1])
$b
(Intercept)
   2.938889

$omega
        XTV        Xradio    Xnewspaper
0.045764645  0.188530017 -0.001037493
```

Programme 6.2.11: Different optimization algorithms with "`Advertising.csv`" dataset in **R**.

From the output of Programme 6.2.11 above, all four methods produce an estimate of the parameters $\boldsymbol{\theta}$ extremely close to the solutions computed by the least square method, *i.e.* computed by the `lm()` function. Moreover, the respective numbers of iterations required for Newton-Raphson and Gauss-Newton to converge is just 2 steps, while those for BFGS and limited-memory BFGS need 12 steps to converge. One major reason is that Newton-Raphson method builds upon the exact second order of $L(\boldsymbol{\theta})$, but BGFS and L-BFGS builds upon the approximation of the second order of $L(\boldsymbol{\theta})$. Finally, BFGS and L-BFGS have an implicit step for the approximation of the Hessian matrix for each iteration, the actual number of iteration should be doubled.

Similarly, in Python, we import Programme 6.2.2 for Newton-Raphson algorithm, Programme 6.2.4 for Gauss-Newton algorithm, Programme 6.2.6 for BFGS algorithm, and Programme 6.2.8 for limited-memory BFGS algorithm.

```
import pandas as pd
from Newton_Raphson import Newton
from Gauss_Newton import Gauss_Newton
from BFGS import BFGS
from L_BFGS import L_BFGS

dataset = pd.read_csv("Advertising.csv", index_col=0)
X, y = dataset.drop(columns="sales").values, dataset["sales"].values

print(Newton(X, y))
print(Gauss_Newton(X, y))
print(BFGS(X, y))
print(L_BFGS(X, y))

from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X, y)
print({"b": model.intercept_, "omega": model.coef_})
```

```
{'Method': 'Newton-Raphson', 'nsteps': 2, 'b': 2.9388893694594147,
 'omega': array([ 0.04576465,  0.18853002, -0.00103749])}
{'Method': 'Gauss-Newton', 'nsteps': 2, 'b': 2.9388893694594147,
 'omega': array([ 0.04576465,  0.18853002, -0.00103749])}
{'Method': 'BFGS', 'nsteps': 12, 'b': 2.9388893694595626,
 'omega': array([ 0.04576465,  0.18853002, -0.00103749])}
{'Method': 'L-BFGS', 'nsteps': 12, 'b': 2.938889369459563,
 'omega': array([ 0.04576465,  0.18853002, -0.00103749])}
{'b': 2.9388893694594085,
 'omega': array([ 0.04576465,  0.18853002, -0.00103749])}
```

Programme 6.2.12: Different optimization algorithms with "`Advertising.csv`" dataset in Python.

6.3 Enhancing Gradient Descent Method

In practice, there are often two major matters with (higher order) gradient descent method:

1. it is too sensitive to the choice of the learning rate hyperparameter η;
2. the Hessian matrix is unavailable but just the feature $\boldsymbol{x}_n$ and response y_n; and
3. it is computationally inefficient with large datasets. Especially in the era of big data, it is highly ineffective to update the parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ by taking care of the whole datasets $\mathcal{S}$ at each epoch.

6.3.1 Enhanced Gradient Descent Method

One possible approach on resolving the Problem 1 in the last paragraph is to vary the learning rate η at different steps of training. Suppose that $L(\boldsymbol{\omega}^{(t)})$ is the loss function in the parameter $\boldsymbol{\omega}^{(t)} = (\omega_1^{(t)}, \cdots, \omega_D^{(t)})^\top$ at the t-th iterations, $\eta^{(0)}$ is the initial learning rate, and the gradients at the current point:
$g_d^{(t)} := \dfrac{\partial L}{\partial \omega_d}(\boldsymbol{\omega}^{(t)}), \qquad \text{for} \quad d = 1, \cdots, D \qquad \text{or equivalently set} \qquad \boldsymbol{g}^{(t)} := \nabla_{\boldsymbol{\omega}^{(t)}} L(\boldsymbol{\omega}^{(t)}) = (g_1^{(t)}, \cdots, g_D^{(t)})^\top .$
In the following, we shall describe four different common ways of boosting up the performance of the Gradient Decent learning.

1. **Adaptive Gradient (Adagrad) Algorithm**: This method was first proposed by Duchi et al. (2011), the purpose is to rescale the learning rates for different parameters according to the average magnitudes. In principle, it is desirable to have a learning rate such that

   (i) it is small for very large gradients;

   (ii) it is large for very small gradients; or

   (iii) while more steps are taken, the estimated parameters are supposed to get closer to the ultimate optimal solution, and we shall be more conservative on the estimation progress by requiring a smaller learning rate as more steps have been taken, as as to avoid overshooting at the last few steps.

   Adagrad proposes to essentially divide the learning rate η by the square root of the sum of squares of all historical gradients for updating a parameter estimate:
   $$\omega_d^{(t+1)} = \omega_d^{(t)} - \frac{\eta}{\sqrt{\sum_{s=1}^t \left(g_d^{(s)}\right)^2 + \varepsilon}} g_d^{(t)}, \qquad \text{for} \quad d = 1, \cdots, D\,.$$
   Or equivalently in matrix form:
   $$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} - \eta \left((\boldsymbol{G}^{(t)})^{1/2} + \varepsilon \boldsymbol{I}_D \right)^{-1} \boldsymbol{g}^{(t)}, \qquad \text{where} \quad \boldsymbol{G}^{(t)} = \operatorname{diag}\left(\sum_{s=1}^t \left(g_1^{(s)}\right)^2, \cdots, \sum_{s=1}^t \left(g_D^{(s)}\right)^2 \right),$$
   where ε is another hyperparameter being a small number so as to avoid division by zero. One major disadvantage of Adagrad is as more gradient terms accumulate, the adaptive learning rate diminishes making the parameters cease further update.

2. **RMSprop** (Root Mean Square Propagation): to avoid the possibility of vanishing learning rate, Tieleman and Hinton (2012) suggested replacing the sum of square of past gradients by a weighted

average:

$$\omega_d^{(t+1)} = \omega_d^{(t)} - \frac{\eta}{\sqrt{\nu_d^{(t)} + \varepsilon}} g_d^{(t)}, \qquad \text{for} \quad d = 1, \cdots, D,$$

where

$$\nu_d^{(t)} = \gamma \nu_d^{(t-1)} + (1-\gamma)\left(g_d^{(t)}\right)^2, \qquad \text{where} \quad \gamma \in [0,1] \quad \text{and} \quad \nu_d^{(0)} = 0.$$

Or in matrix form:

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} - \eta\left((V^{(t)})^{1/2} + \varepsilon \boldsymbol{I}_D\right)^{-1} \boldsymbol{g}^{(t)}, \qquad \text{where} \quad \boldsymbol{V}^{(t)} = \gamma \boldsymbol{V}^{(t-1)} + (1-\gamma)\text{diag}(\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}).$$

RMSprop is an adaptive learning rate optimization algorithm proposed by Geoffrey Hinton in his online Coursera Lecture 6e "*Divide the gradient by a running average of its recent magnitude*"[2]. Hinton suggests that $\gamma = 0.9$ and $\eta = 0.001$ would be good values for the algorithm.

3. **Momentum**: accelerates Gradient Descent method by orienting the gradient descent also along the prevalent direction, and therefore, reducing unnecessary oscillations to possibly speed up convergence.

$$\omega_d^{(t+1)} = \omega_d^{(t)} + \eta m_d^{(t)}, \qquad \text{for} \quad d = 1, \cdots, D,$$

where

$$m_d^{(t)} = \gamma m_d^{(t-1)} - (1-\gamma) g_d^{(t)}, \qquad \text{where} \quad \gamma \in [0,1] \quad \text{and} \quad m_d^{(0)} = 0.$$

Or in matrix form:

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} + \eta \boldsymbol{m}^{(t)}, \qquad \text{where} \quad \boldsymbol{m}^{(t)} = \gamma \boldsymbol{m}^{(t-1)} - (1-\gamma)\boldsymbol{g}^{(t)}.$$

In practice, $\gamma = 0.9$ is often adopted. Moreover, momentum can help to escape from a local minimum as the prevalent direction is often drifting towards the global minimum.

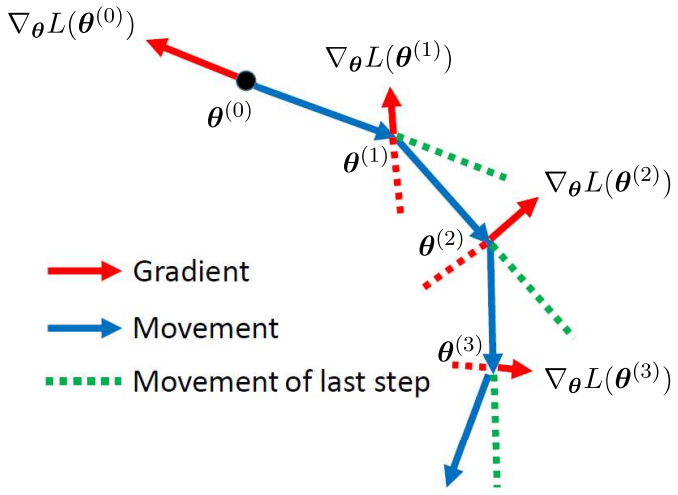


Figure 6.3.1: Momentum: movement of last step minus gradient at present

4. **Adam** (Adaptive Moment Estimation): incorporates both the ideas from RMSprop and Momentum methods with a bias correction:

$$\omega_d^{(t+1)} = \omega_d^{(t)} + \frac{\eta}{\sqrt{\hat{\nu}_d^{(t)} + \varepsilon}} \hat{m}_d^{(t)}, \qquad \text{for} \quad d = 1, \cdots, D,$$

[2]The lecture notes can be found in `http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf`

where

$$\begin{aligned} m_d^{(t)} &= \beta_1 m_d^{(t-1)} - (1-\beta_1) g_d^{(t)} \,, \\ \nu_d^{(t)} &= \beta_2 \nu_d^{(t-1)} + (1-\beta_2)\left(g_d^{(t)}\right)^2 \,, \\ \hat{m}_d^{(t)} &= \frac{m_d^{(t)}}{1-\beta_1^t} \,, \\ \hat{\nu}_d^{(t)} &= \frac{\nu_d^{(t)}}{1-\beta_2^t} \,. \end{aligned}$$

Adam uses the estimates of the first and the second "moments"of the negative gradients $-g_d^{(t)}$, which correspond to $m_d^{(t)}$ and $v_d^{(t)}$ respectively, so as to "adapt" the learning rate η; in other words, for $d = 1, \cdots, D$, we want to have

$$\widehat{\mathbb{E}}(m_d^{(t)}) = -\widehat{\mathbb{E}}(g_d^{(t)}) \qquad \text{and} \qquad \widehat{\mathbb{E}}(v_d^{(t)}) = \widehat{\mathbb{E}}\left[(g_d^{(t)})^2\right] \,,$$

with the naive initial condition that $m_d^{(0)} = 0$ and $v_d^{(0)} = 0$, which actually causes bias, that means, the estimators, $m_d^{(t)}$ and $v_d^{(t)}$, under this initial condition are both biased towards zero. To see such a bias, consider the expectation of $m_d^{(t)}$ and using the fact that $g_d^{(t)}$ are i.i.d., for $t = 1, 2, \cdots$, given the sample is,

$$\begin{aligned} \widehat{\mathbb{E}}(m_d^{(t)}) &= \widehat{\mathbb{E}}\left(\beta_1 m_d^{(t-1)} - (1-\beta_1) g_d^{(t)}\right) = \widehat{\mathbb{E}}\left((1-\beta_1)\sum_{s=1}^{t} \beta_1^{t-s}\left(-g_d^{(s)}\right)\right) \\ &= \left(-\widehat{\mathbb{E}}(g_d^{(1)})\right)(1-\beta_1)\sum_{s=1}^{t}\beta_1^{t-s} = -\widehat{\mathbb{E}}(g_d^{(1)})(1-\beta_1^t) \,, \end{aligned}$$

causing a bias with a factor of β_1^t. Similar argument also leads to see the biasedness of $v_d^{(t)}$. Therefore, we propose their respective unbiased estimators:

$$\hat{m}_d^{(t)} = \frac{m_d^{(t)}}{1-\beta_1^t} \qquad \text{and} \qquad \hat{v}_d^{(t)} = \frac{v_d^{(t)}}{1-\beta_2^t} \,.$$

Or in matrix form:

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} + \eta\left((\hat{\boldsymbol{V}}^{(t)})^{1/2} + \varepsilon \boldsymbol{I}_D\right)^{-1} \hat{\boldsymbol{m}}^{(t)} \,,$$

where

$$\begin{aligned} \boldsymbol{m}^{(t)} &= \beta_1 \boldsymbol{m}^{(t-1)} - (1-\beta_1)\boldsymbol{g}^{(t)} \,, \\ \boldsymbol{\nu}^{(t)} &= \beta_2 \boldsymbol{\nu}^{(t-1)} + (1-\beta_2)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)} \,, \\ \hat{\boldsymbol{m}}^{(t)} &= \frac{\boldsymbol{m}^{(t)}}{1-\beta_1^t} \,, \\ \hat{\boldsymbol{V}}^{(t)} &= \operatorname{diag}\left(\frac{\boldsymbol{\nu}^{(t)}}{1-\beta_2^t}\right) \,. \end{aligned}$$

We shall illustrate the application of these enhanced gradient descent methods in Python as follows:

```
import numpy as np

class SGD():
    def __init__(self, eta=1e-3):
        self.eta = eta

    def Delta(self, dg):
```

```
        return -self.eta * dg            # -ηg^(t-1)

class Adagrad():
    def __init__(self, eta=1e-3, epsilon=1e-8):
        self.eta = eta
        self.epsilon = epsilon
        self.past_g = 0

    def Delta(self, dg):
        self.past_g += dg**2
        # ((G^(t-1))^(1/2) + εI_D)^(-1)
        inv_G = np.diag((self.past_g**(1/2) + self.epsilon)**(-1))
        return -self.eta * inv_G @ dg

class RMSprop():
    def __init__(self, eta=1e-3, gamma=0.9, epsilon=1e-8):
        self.eta = eta
        self.gamma = gamma
        self.epsilon = epsilon
        self.v = 0

    def Delta(self, dg):
        self.v = self.gamma * self.v + (1 - self.gamma) * dg * dg
        # ((V^(t-1))^(1/2) + εI_D)^(-1)
        inv_V = np.diag((self.v**(1/2) + self.epsilon)**(-1))
        return -self.eta * inv_V @ dg

class Momentum():
    def __init__(self, eta=1e-3, gamma=0.9):
        self.eta = eta
        self.gamma = gamma
        self.m = 0

    def Delta(self, dg):
        # m^(t-1) - (1-γ)g^(t)
        self.m = self.gamma * self.m - (1 - self.gamma) * dg
        return self.eta * self.m

class Adam():
    def __init__(self, eta=1e-3, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.m, self.v = 0, 0
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
```

```
        self.eta = eta
        self.t = 0

    def Delta(self, dg):
        self.t += 1
        # m^(t) = β1 m^(t-1) − (1 − β1) g^(t)
        self.m = self.beta1 * self.m - (1 - self.beta1) * dg
        # ν^(t) = β2 ν^(t-1) + (1 − β2) g^(t) ⊙ g^(t)
        self.v = self.beta2 * self.v + (1 - self.beta2) * dg**2
        # m̂^(t) = m^(t) / 1 − β1^t
        hat_m = self.m/(1 - self.beta1**self.t)
        # v̂^(t) = v^(t) / 1 − β2^t
        hat_v = self.v/(1 - self.beta2**self.t)
        # ((V̂^(t-1))^(1/2) + εI_D)^(-1)
        inv_hat_V = np.diag((hat_v**(1/2) + self.epsilon)**(-1))
        return self.eta * inv_hat_V @ hat_m
```

Programme 6.3.1: Enhanced gradient descent in Python, saved as `Enhanced_Gradient_Descent.py`.

Next, the "`Advertising.csv`" dataset is used in a comparative study for these enhanced gradient methods against the batch gradient descent. We first define the `Gradient()` function, which returns the average gradient computed in a minibatch of size M, *i.e.* $\left(1/M \sum_{m=1}^{M} \boldsymbol{g}_m^{(t)}\right)_{d=0}^{D}$ where $d = 0$ denotes the intercept term.

```
def Gradient(X, y, theta):
    return -X.T @ (y - X @ theta) / len(y)

def Gradient_Descent(X, y, optimizer, epochs=1e5):
    intercept = np.ones(len(X)).reshape(-1,1)
    if len(X.shape) == 1:
        X = X.reshape(-1, 1)
    X, y = np.hstack((intercept, X)), np.array(y)
    theta_t = np.zeros(np.shape(X)[1])
    for epoch in range(int(epochs)):
        # calculate gradient for omega_t and b_t
        dg = Gradient(X, y, theta_t)
        theta_t += optimizer.Delta(dg)

    return({"Optimizer": optimizer.__class__.__name__,
            "b": theta_t[0], "omega": np.squeeze(theta_t[1:])})
```

Programme 6.3.2: `Gradient()` function and `Gradient_Descent()` function in Python.

Here the `Enhanced_Gradient_Descent()` function performs one of the chosen gradient descent methods in Programme 6.3.1 specified by the `optimizer` argument. Finally, it returns the name of the chosen optimizer with `optimizer.__class__.__name__`, the intercept b and the weight ω of the simple linear regression model in (6.1.5).

```
import pandas as pd
from Enhanced_Gradient_Descent import SGD, Adagrad, RMSprop, Momentum, Adam

dataset = pd.read_csv("Advertising.csv", index_col=0)
```

```
                        X, y = dataset["TV"].values, dataset["sales"].values

print(Gradient_Descent(X, y, SGD(eta=1e-5), epochs=1e6))
print(Gradient_Descent(X, y, Adagrad(eta=1), epochs=1e4))
print(Gradient_Descent(X, y, RMSprop(eta=1e-4), epochs=1e5))
print(Gradient_Descent(X, y, Momentum(eta=1e-4), epochs=2e5))
print(Gradient_Descent(X, y, Adam(eta=1), epochs=1e3))

from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X.reshape(-1, 1), y)
print({"b": model.intercept_, "omega": model.coef_[0]})
```

```
{'Optimizer': 'SGD', 'b': 6.4739923017155645, 'omega': array(0.05037336)}
{'Optimizer': 'Adagrad', 'b': 7.032593464887523, 'omega': array(0.04753664)}
{'Optimizer': 'RMSprop', 'b': 7.032543549195243, 'omega': array(0.04748664)}
{'Optimizer': 'Momentum', 'b': 6.988264681976378, 'omega': array(0.04776175)}
{'Optimizer': 'Adam', 'b': 7.0325935491277045, 'omega': array(0.04753664)}
{'b': 7.032593549127693, 'omega': 0.047536640433019764}
```

Programme 6.3.3: Enhanced gradient descent from Programme 6.3.1 with the "`Advertising.csv`" dataset in Python.

Here the `Enhanced_Gradient_Descent` comes from Programme 6.3.1. As mentioned in Subsection 6.1.2, the batch gradient descent method with a learning rate of `1e-3`, or even `1e-4`, overshoots the parameter updates. SGD also suffers from this situation that a learning rate of `1e-5` is used. However, as illustrated in the Programme 6.3.3, even with `1e6` numbers of epoch, SGD fails to reach close enough to the ultimate optimal point and this indicates that more epochs are still required; meanwhile, for Adagrad and Adam, a much larger learning rate can be allowed so as to enjoy a faster convergence to the optimal in the total number of steps. For RMSprop and Momentum, a learning rate of `1e-4` is adopted, a larger learning rate, let say `1e-3`, will encounter the same overshooting issue.

Next, we shall illustrate the effectiveness of various enhancing gradient descent methods via an example involving the Beale's function:

$$f(\boldsymbol{x}) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_2x_2^2)^2 + (2.625 - x_1x_2^3)^2\,. \tag{6.3.1}$$

The partial derivatives of f with respect to x_1 and x_2 are respectively:

$$\begin{aligned}\frac{\partial f}{\partial x_1} &= -12.75 + 3x_2 + 4.5x_2^2 + 5.25x_2^3 + 2x_1 \cdot (3 - 2x_2 - x_2^2 - 2x_2^3 + x_2^4 + x_2^6)\,,\\ \frac{\partial f}{\partial x_2} &= 6x_1 \cdot \left(0.5 + 1.5x_2 + 2.625x_2^2 + x_1 \cdot \left(-\frac{1}{3} - \frac{1}{3}\cdot x_2 - x_2^2 + \frac{2}{3}\cdot x_2^3 + x_2^5\right)\right)\,.\end{aligned} \tag{6.3.2}$$

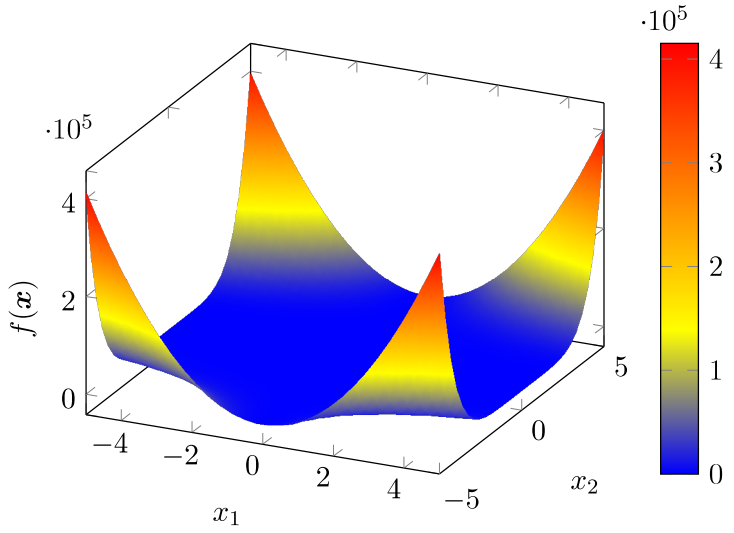


Figure 6.3.2: Beale's function in (6.3.1).

Therefore, with (6.3.1) and (6.3.2), we can build the Programme 6.3.4 in Python using various Enhancing gradient descent methods developed in Programme 6.3.1:

```
import numpy as np

Beale = lambda x, y: (1.5-x+x*y)**2 + (2.25-x+x*y**2)**2 + (2.625-x+x*y**3)**2

def Beale_Gradient(theta):
    x, y = theta
    dx = -12.75+3*y+4.5*y**2+5.25*y**3+2*x*(3-2*y-y**2-2*y**3+y**4+y**6)
    dy = 6*x*(0.5+1.5*y+2.625*y**2+x*(-1/3-1/3*y-y**2+2/3*y**3+y**5))
    return np.array([dx, dy])

def Beale_Gradient_Descent(theta_t, optimizer, max_epochs=1e5, tol=1e-5):
    paths = [theta_t]
    for epoch in range(int(max_epochs)):
        dg = Beale_Gradient(theta_t)
        theta_new = theta_t + optimizer.Delta(dg)
        paths.append(theta_new)
        if sum((theta_new - theta_t)**2) < tol**2:
            return (np.squeeze(paths))
        theta_t = theta_new

    return(np.squeeze(paths))

start = np.array([1, 1])
start_ = start.reshape(-1, 1)

SGD_path = Beale_Gradient_Descent(start, SGD(eta=1e-2))
Adagrad_path = Beale_Gradient_Descent(start, Adagrad(eta=1))
RMSprop_path = Beale_Gradient_Descent(start, RMSprop(eta=1e-3))
Momentum_path = Beale_Gradient_Descent(start, Momentum(eta=1e-1))
Adam_path = Beale_Gradient_Descent(start, Adam(eta=1))

print({"SGD": len(SGD_path)})
print({"Adagrad": len(Adagrad_path)})
print({"RMSprop": len(RMSprop_path)})
print({"Momentum": len(Momentum_path)})
```

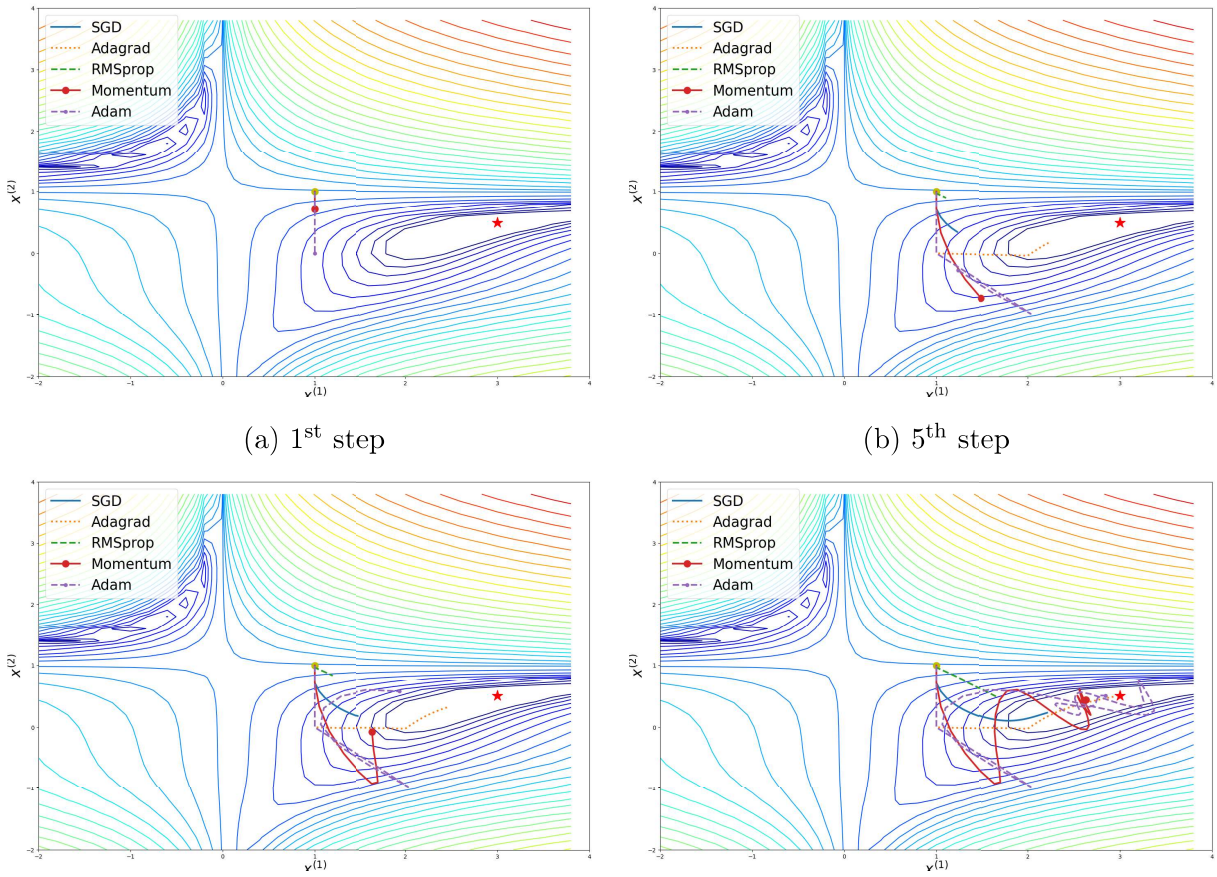
```
print({"Adam": len(Adam_path)})

print({"nstep": len(RMSprop_path), "eta": 1e-2, "RMSprop": RMSprop_path[-1]})

RMSprop_path = Beale_Gradient_Descent(start, RMSprop(eta=1e-1))
print({"nstep": len(RMSprop_path), "eta": 1e-1, "RMSprop": RMSprop_path[-1]})

RMSprop_path = Beale_Gradient_Descent(start, RMSprop(eta=1))
print({"nstep": len(RMSprop_path), "eta": 1, "RMSprop": RMSprop_path[-1]})
```

```
{'SGD': 1562}
{'Adagrad': 198}
{'RMSprop': 100001}
{'Momentum': 196}
{'Adam': 210}
{'nstep': 100001, 'eta': 0.01, 'RMSprop': array([2.6239 , 0.38584])}
{'nstep': 100001, 'eta': 0.1, 'RMSprop': array([2.77376, 0.49764])}
{'nstep': 100001, 'eta': 1, 'RMSprop': array([2.6239 , 0.38584])}
```

Programme 6.3.4: Enhancing gradient descent in Beale's function in Python.

Here the `lambda` is to define a function with inputs `x` and `y`, which is equivalent in effect to the `def` in Python. The algorithm stops when the condition that $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\|_2^2 < \delta^2$, where δ is the `tol`, is met, and we print the number of iterations for each gradient descent method required for convergence. From the output, we observe that RMSprop requires the largest number of iterations in order to converge, which even exceeds our maximum number of iterations of 100,000 as specified in the argument `max_epochs=1e5`; this is mainly due to the smaller learning rate `eta=1e-3` being adopted. However, when we set the learning rate by 10 folds larger to `eta=1e-2`, the required convergence still exceeds the iteration threshold, and the same situation is found when we further enlarge the learning rate to `eta=1e-1` and even `eta=1`. Among other available enhanced gradient methods, the required number of iterations for SGD is the second largest, which is roughly 8 times larger than that of the others; while those of the other methods are similar. In particular, different learning rates η are adopted in different methods, *e.g.* SGD with $\eta = 0.01$; Adagrad and Adam with $\eta = 1$; and Momentum with $\eta = 0.1$. They are chosen based on the *grid search* method, where we choose the largest η from a sequence of 10^n, where $n \in \mathbb{Z}$, so that the given descent method can acquire a higher learning rate η, so that a faster convergence can be guaranteed, yet without encountering the overshooting problem.

We now examine the paths of these five gradient descent methods, specifically the 1^{st}, 5^{th}, 10^{th}, and 50^{th} steps as depicted in Figure 6.3.3, while Figure 6.3.4 is a GIF (online version of this book) illustrating the entire paths up to 200 iterations for each method.



(a) 1^{st} step (b) 5^{th} step

(c) 10^{th} step (d) 50^{th} step

Figure 6.3.3: The 1^{st}, 5^{th}, 10^{th}, and 50^{th} steps of SGD, Adagrad, RMSprop, Momentum, and Adam.

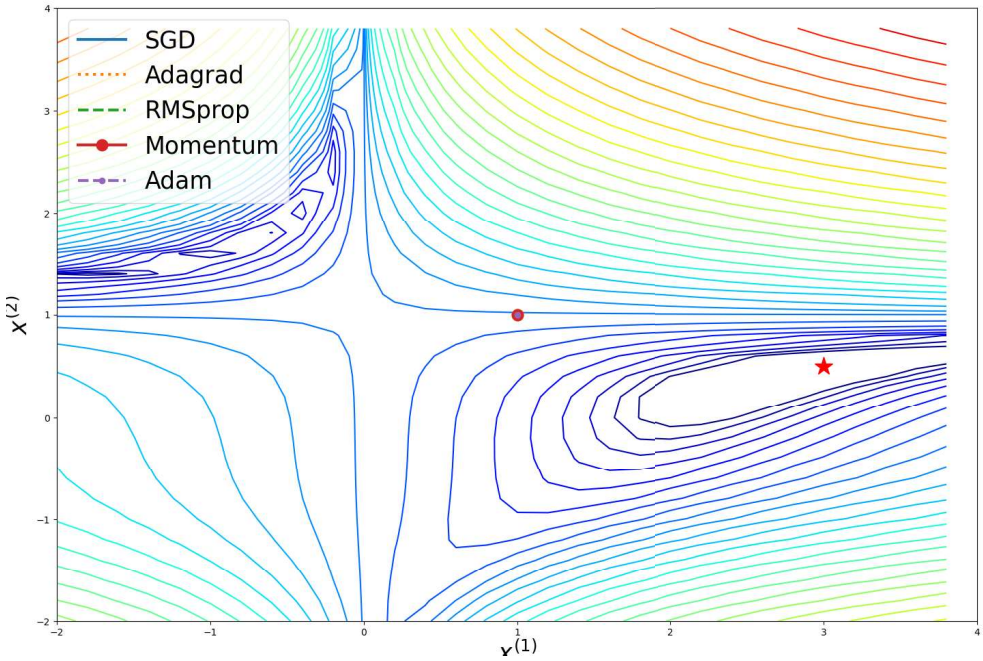


Figure 6.3.4: A GIF of the five stochastic gradient descent methods up to the first 200 steps.

6.3.2 Stochastic Gradient Descent and its Mini-batch Counterpart

For the problem 2 especially for big data, it is reasonable to update the estimation of parameters $\boldsymbol{\theta}$ by using only a portion of the dataset $\mathcal{S}$, and the corresponding method is often referred as **Mini-batch stochastic gradient descent** (Mini-batch SGD for short). Picking a loss $\ell : \mathbb{R}^D \times \mathbb{R} \times \mathbb{R}^D \to \mathbb{R}$, for simplicity, we assume the true label $y \in \mathbb{R}$, and defining the n-th loss residual $r_n^2 := \ell(\boldsymbol{\theta}; y_n, \boldsymbol{x}_n)$, where y_n is the true label of $\boldsymbol{x}_n$. With the Law of Large Number, the average loss function $L(\boldsymbol{\theta})$ can be approximated by:

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \ell(\boldsymbol{\theta}; y_n, \boldsymbol{x}_n) \approx \mathbb{E}(\ell(\boldsymbol{\theta}; \mathrm{y}_n, \mathbf{x}_n)|\boldsymbol{\theta}), \quad \text{with a very high probability with large } N. \tag{6.3.3}$$

Again, by the Law of Large Number, with a sufficiently large $M \ll N$, the $\mathbb{E}(\ell(\boldsymbol{\theta}; \mathrm{y}, \mathbf{x})|\boldsymbol{\theta})$ can be approximated instead by a small portion of the dataset $\mathcal{S}$, *i.e.*

$$L(\boldsymbol{\theta}) \approx \mathbb{E}(\ell(\boldsymbol{\theta}; \mathrm{y}_n, \mathbf{x}_n)|\boldsymbol{\theta}) \approx \frac{1}{M}\sum_{m=1}^{M} \ell(\boldsymbol{\theta}; y_m, \boldsymbol{x}_m) \quad \text{with a very high probability with large } N. \tag{6.3.4}$$

Therefore, with a very high probability with both large M and N,

$$\frac{1}{N}\sum_{n=1}^{N} \ell(\boldsymbol{\theta}; y_n, \boldsymbol{x}_n) \approx \frac{1}{M}\sum_{m=1}^{M} \ell(\boldsymbol{\theta}; y_m, \boldsymbol{x}_m), \tag{6.3.5}$$

and this is the main philosophy of Mini-batch SGD: replacing the overall loss, which involves much more computation, by a portion of losses. And this is viable especially when L is separable as commonly found in statistics and machine learning. For example, in (5.1.1), MSE is used as the average loss function such that the loss is now just the squared residuals, *i.e.* $r_n^2 := \ell(\boldsymbol{\theta}; y_n, \boldsymbol{x}_n) = (y_n - f_{\boldsymbol{\theta}}(\boldsymbol{x}_n))^2$, where $f : \mathbb{R}^D \to \mathbb{R}$. The parameter update for Mini-batch SGD is:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \cdot \frac{1}{M}\sum_{m=1}^{M} \ell(\boldsymbol{\theta}^{(t)}; y_m, \boldsymbol{x}_m). \tag{6.3.6}$$

In general, we shall even first shuffle the training set and then split it into B number of batches, each of equal size of M, *i.e.* $B = \lceil N/M \rceil$: Denote the shuffled dataset, so that $(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n) := (\tilde{\boldsymbol{x}}_{\pi(n)}, \tilde{\boldsymbol{y}}_{\pi(n)})$ for a permutation π on $\{1, 2, \cdots, N\}$ to itself, be $\mathcal{S}' := \{(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n)\}_{n=1}^N$, then

$$\mathcal{S}' = \{(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n)\}_{n=1}^N = (\mathcal{S}'_1, \mathcal{S}'_2, \cdots, \mathcal{S}'_B) = (\{(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n)\}_{n=1}^M, \{(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n)\}_{n=M+1}^{2M}, \cdots, \{(\tilde{\boldsymbol{x}}_n, \tilde{\boldsymbol{y}}_n)\}_{n=(B-1)\times M+1}^{N}).$$

Then we have the following algorithm:

Algorithm 6.8 Mini-batch Stochastic Gradient Descent

```
Initialize θ^(0).
for each epoch t = 1, ··· , T do
    for b = 1, ··· , B do
        S'_b is selected to train the model to obtain the estimate θ^(B(t−1)+b).
    end for
end for
```

In practice, the commonly used batch size is $M = 32 = 2^5 > 30$, the golden number for a random sample mean to a t-distribution and or a normal distribution. For two particular cases:

1. the common stochastic gradient descent (SGD) is a special case of $M = 1$, and now a randomly picked sample point is used for an update of the parameter estimate;
2. the (whole) batch gradient descent is also a special case in Mini-batch SGD with batch size $M = N$,

all training set is used at a time.

The general effect of using SGD, Mini-batch SGD, and batch SGD methods for the parameter updates is illustrated in Figure (6.3.5).

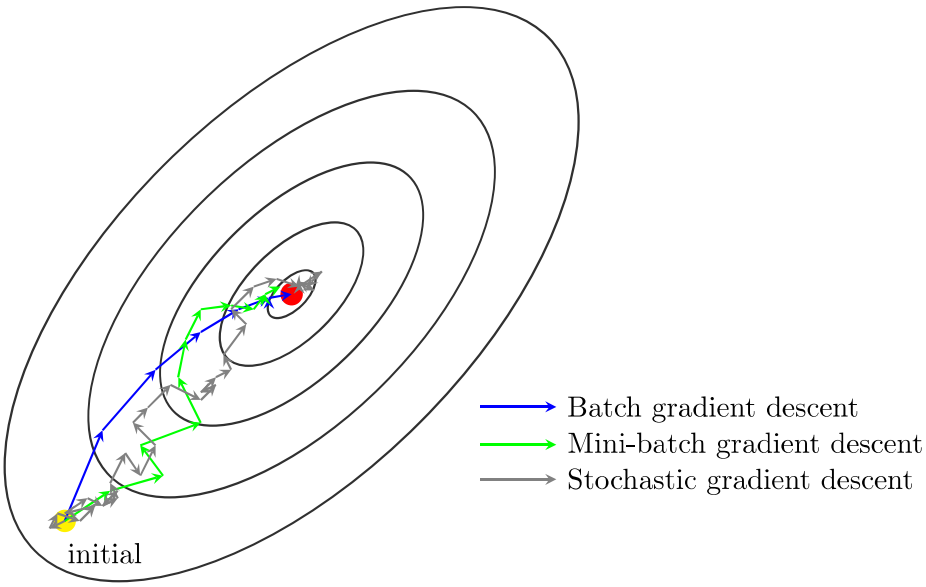


Figure 6.3.5: Gradient Descent for SGD, Mini-batch SGD, and batch SGD

The red spot represents the ultimate local minimum point we are looking for. The blue trace represents the sequence of updates after the batch gradient descent. It is the fastest possible geographically path straightly directing to the minimum point, and it involves the least number of steps, however the computational cost is much higher in each step than the other two methods. Even worst, in the computation of the overall average loss $L(\boldsymbol{\theta})$, we might encounter numeric overflow or underflow even with a decent computing machine due to a huge number of terms being dealt at one time.

For the stochastic gradient descent, *i.e.* $M = 1$, the gray trace, the path is so close to a typical sample path of a diffusion process (see Section 6.4), namely a solution of the *Stochastic Gradient Langevin Equation* (SGLE). The path is of zigzag shape where it apparently requires many more iterations, in comparison to batch gradient descent, to be sufficiently close to the ultimate minimum point. In contrast, in each iteration of the batch gradient descent, all the gradients of the loss $\nabla_{\boldsymbol{\theta}}\ell$ are needed to be calculated and then summed up in order to obtain the overall average loss $L(\boldsymbol{\theta})$, and this easily leads to the exploding gradient problem due to numerical overflow or underflow. On the other hand, SGD only requires to compute a single gradient of the loss $\nabla_{\boldsymbol{\theta}}\ell$ at each iteration step, the gradient computed at each iteration is small so that the exploding gradient problem can be avoided, while SGD has a high probability that it requires only a while to be sufficiently close to the ultimate minimum point, altogether, SGD yet requires much less computational steps (= number of zigzag steps $\sim O(1/\eta)$) to be sufficiently close to the ultimate minimum point than batch gradient descent (= number of steps $\times O(N) \sim O(1/\eta \cdot N)$); to be discussed in Subsection 6.3.4 and Section 6.4. Finally, although the path is of zigzag shape, the mean direction of the path is still pointing towards the ultimate minimum point.

For the Mini-batch gradient descent, the green path, it takes far less iterations than the stochastic gradient descent to get close to the minimum point, computational time $O(1/\eta \cdot M)$, and the path still follows the *Langevin dynamics* (See (6.4.6)) due to the use of the Law of Large Number and the Central Limit Theorem; indeed the Mini-batch SGD can be regarded as a discretization of a continuous-time diffusion process.

Next, we shall illustrate the convergence of SGD and the Mini-batch SGD in Python using a simple linear model:

$$y_n = \omega x_n + b + 0.1\varepsilon_n, \text{where} \quad \varepsilon_n \sim \mathcal{N}(0,1)\,.$$

We first generate the data with the true parameters being $\omega = 2$ and $b = 5$ in the domain $x_n \in [-1, 1]$. Then we shuffle the data with the seed argument `random_state=4012`:

```
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.utils import shuffle

np.random.seed(4012)
N = 100
x = np.linspace(-1, 1, N)
y = 2*x + 5 + np.random.randn(N)/100
x, y = shuffle(x, y, random_state=4012)

def gradient(x, y, w, b):
    return -x*(y-(w*x+b)), -(y-(w*x+b))
```

Programme 6.3.5: Generating dataset in Python.

Here the MSE L and then divided by 2 is used as the chosen loss function such that the gradients with respect to a and b are respectively:

$$\frac{\partial L}{\partial \omega} = \frac{1}{N}\sum_{n=1}^{N} -x_n \cdot (y_n - \omega x - b) \qquad \text{and} \qquad \frac{\partial L}{\partial b} = \frac{1}{N}\sum_{n=1}^{N} -(y_n - \omega x - b)\,. \tag{6.3.7}$$

Since we are performing SGD and Mini-batch SGD, we only use a simplified version of (6.3.7). Next, the SGD is as follow:

```
def SGD(inits, X, y, lr=1e-3, n_iter=10000, eta=1e-12):
    w, b = inits
    w_list, b_list = [w], [b]
    for i in range(n_iter):
        for j in range(len(y)):
            grad_w, grad_b = gradient(X[j], y[j], w, b)
            w -= lr*grad_w
            b -= lr*grad_b
            w_list.append(w)
            b_list.append(b)
            if (w - w_list[-2])**2 + (b - b_list[-2])**2 < eta:
                return w_list, b_list
    return w_list, b_list

start_time = time.time()
SGD_w, SGD_b = SGD([-4,0], x, y, lr=0.02, eta=1e-8)
SGD_time = time.time() - start_time
print({"Method":"SGD with eta=1e-8", "niter": len(SGD_w), "time": SGD_time,
       "w": SGD_w[-1], "b": SGD_b[-1]})

```

```
start_time = time.time()
SGD_w, SGD_b = SGD([-4,0], x, y, lr=0.02, eta=1e-12)
SGD_time = time.time() - start_time
print({"Method":"SGD with eta=1e-12", "niter": len(SGD_w), "time": SGD_time,
       "w": SGD_w[-1], "b": SGD_b[-1]})
```

```
{'Method': 'SGD with eta=1e-8', 'niter': 143, 'time': 0.0,
 'w': -0.10244944370777345, 'b': 4.976970857873605}
{'Method': 'SGD with eta=1e-12', 'niter': 991, 'time': 0.001994609832763672,
 'w': 1.9955364985358999, 'b': 5.000478973363515}
```

Programme 6.3.6: SGD in Python.

Here, the SGD in Programme 6.3.6 starts at the point $(x_0, y_0) = (-4, 0)$ and ends when $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\|_2^2 < \delta$ `eta`. However, from the output of Programme 6.3.6, we observe that δ taking value `eta=1e-8` is not enough for the estimator of ω to reach near its true value $\omega = 2$. Hence, a much smaller value of δ, let say `eta=1e-12`, is used.

Next, we have the Mini-batch SGD:

```
def MbSGD(inits, X, y, lr=1e-3, n_iter=10000, bs=32, eta=1e-8):
    ind = np.arange(len(y))
    w, b = inits
    w_list, b_list = [w], [b]
    batch_indices = [ind[i:(i+bs)] for i in range(0, len(y), bs)]
    for i in range(n_iter):
        for indices in batch_indices:
            grad_sum_w, grad_sum_b = 0, 0
                for j in indices:
                    grad_w, grad_b = gradient(X[j], y[j], w, b)
                    grad_sum_w += grad_w
                    grad_sum_b += grad_b

                w -= lr*grad_sum_w/len(indices)
                b -= lr*grad_sum_b/len(indices)
                w_list.append(w)
                b_list.append(b)

                if (w - w_list[-2])**2 + (b - b_list[-2])**2 < eta:
                    return w_list, b_list
    return w_list, b_list

start_time = time.time()
MbSGD_w, MbSGD_b = MbSGD([-4,0], x, y, lr=0.02, eta=1e-8)
MbSGD_time = time.time() - start_time
print({"Method":"Minibatch", "niter": len(MbSGD_w),
       "time": MbSGD_time, "w": MbSGD_w[-1], "b": MbSGD_b[-1]})
```

```
{'Method': 'Minibatch', 'niter': 857, 'time': 0.03291177749633789,
 'w': 1.9886699053277366, 'b': 5.001721600322803}
```

Programme 6.3.7: Mini-batch SGD in Python.

From the output, the delta having value `eta=1e-8` is enough for the Mini-batch SGD. However, the time required for Mini-batch SGD to reach near the ultimate optimal point is significantly larger than that of SGD, 0.0329118 seconds for Mini-batch SGD when compared with 0.0019946 seconds for SGD with a even smaller delta `eta=1e-12`. The path for SGD and Mini-batch SGD are shown in Figure 6.3.6:

```
w = np.linspace(-6.0, 10.0, N)
b = np.linspace(-2.0, 10.0, N)
Wgrid, Bgrid = np.meshgrid(w, b)
MSE = 0
for xs, ys in zip(x, y):
    MSE += (ys - (Wgrid*xs + Bgrid))**2 / N / 2

plt.figure(figsize=(8, 6))
cp = plt.contour(Wgrid, Bgrid, MSE, levels=25, linestyles="dashed")
plt.xlabel('w')
plt.ylabel('b')
plt.title('MSE')

plt.plot(SGD_w, SGD_b, color='red', label="SGD")
plt.plot(MbSGD_w, MbSGD_b, color='black', label="Mini-batch")
plt.legend(framealpha=1, frameon=True)
plt.savefig("SGD vs Minibatch.png")
plt.show()
```

Programme 6.3.8: Plotting the contour and path for SGD and Mini-batch SGD in Python.

The corresponding plot is

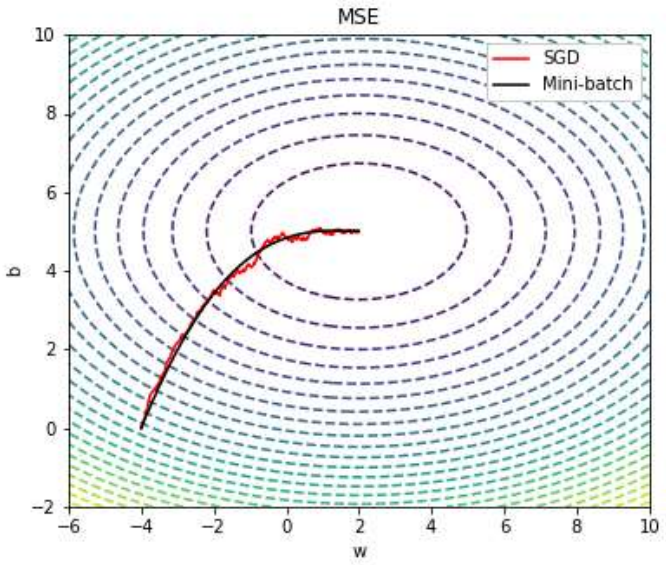


Figure 6.3.6: Path for SGD (red) and Mini-batch SGD (black) in the contour with MSE as the loss.

From Figure 6.3.6, SGD, the red line, oscillates a lot, but SGD is still able to reach a point near the ultimate optimal point; meanwhile for Mini-batch SGD, the black line, it is much smoother than that of SGD and Mini-batch SGD gradually drops towards the ultimate optimal point. Finally, with the aid of array in Python, we can improve the efficiency of Mini-batch SGD:

```
def MbSGD_Mat(inits, X, y, lr=1e-3, n_iter=10000, bs=32, eta=1e-8):
```

```
    ind = np.arange(len(y))
    w, b = inits
    w_list, b_list = [w], [b]
    batch_indices = [ind[i:(i+bs)] for i in range(0, len(y), bs)]
    for i in range(n_iter):
        for indices in batch_indices:
            grad_w, grad_b = gradient(X[indices], y[indices], w, b)
            w -= lr*np.mean(grad_w)
            b -= lr*np.mean(grad_b)
            w_list.append(w)
            b_list.append(b)

            if (w - w_list[-2])**2 + (b - b_list[-2])**2 < eta:
                return w_list, b_list
    return w_list, b_list

start_time = time.time()
MbSGD_Mat_w, MbSGD_Mat_b = MbSGD_Mat([-4,0], x, y, lr=0.02, eta=1e-8)
MbSGD_Mat_time = time.time() - start_time
print({"Method":"Minibatch Matrix", "niter": len(MbSGD_Mat_w),
       "time": MbSGD_Mat_time, "w": MbSGD_Mat_w[-1], "b": MbSGD_Mat_b[-1]})
```

```
{'Method': 'Minibatch Matrix', 'niter': 857, 'time': 0.01795220375061035,
 'w': 1.9886699053277366, 'b': 5.001721600322803}
```

Programme 6.3.9: Mini-batch SGD using array in Python.

The output shows that it is at least two times faster when array is used.

6.3.3 Convergence of SGD

The convergence result of SGD for a sequence of learning rate $\eta_t \in \mathbb{R}^+$ such that

$$\sum_{t=1}^{\infty} \eta_t = \infty\,, \qquad \text{and} \qquad \sum_{t=1}^{\infty} \eta_t^2 < \infty\,,$$

can be found in, *e.g.* Bottou et al. (2018), and we shall provide a description as follows; indeed the learning rate $\eta_t \to 0$ as $\sum_{t=1}^{\infty} \eta_t^2$ is convergent. To begin with, we need the following assumptions:

(A1) (*Lipschitz continuity*) The loss function $L : \mathbb{R}^D \to \mathbb{R}$ is continuously differentiable and its gradient function $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ is Lipschitz continuous with a postive Lipschitz constant C, *i.e.*

$$\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}')\|_2 \le C\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2 \qquad \forall\, \boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^D\,. \tag{6.3.8}$$

(A2) (*Strong convexity*) $L(\boldsymbol{\theta})$ is a μ-strong convex function (also recall (6.1.13))

$$L(\boldsymbol{\theta}') \ge L(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top (\boldsymbol{\theta}' - \boldsymbol{\theta}) + \frac{\mu}{2}\|\boldsymbol{\theta}' - \boldsymbol{\theta}\|_2^2\,, \qquad \text{for any} \quad \boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^D\,. \tag{6.3.9}$$

(A3) For the sequence $\{\boldsymbol{\theta}^{(t)}\}$ contained in an open set over which L is bounded below, there exists scalars $\gamma_G \ge \gamma > 0$, $M_\sigma \ge 0$, and $M_V \ge 0$ such that, for all $t = 0, 1, \cdots$,

(i) $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})^\top \mathbb{E}[\nabla_{\boldsymbol{\theta}} \ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) | \boldsymbol{\theta}^{(t)}] \ge \gamma \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2$;

(ii) $\|\mathbb{E}[\nabla_{\boldsymbol{\theta}} \ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) | \boldsymbol{\theta}^{(t)}]\|_2 \le \gamma_G \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2$;

(iii) $\operatorname{tr}\left(\operatorname{Var}[\nabla_{\boldsymbol{\theta}} \ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) | \boldsymbol{\theta}^{(t)}]\right) \le M_\sigma + M_V \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2$.

where $\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) := \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}; y_{n_{t+1}}, \boldsymbol{x}_{n_{t+1}})$ is the gradient function computed at the single datum $(\boldsymbol{x}_{n_{t+1}}, y_{n_{t+1}})$ with $\boldsymbol{\theta}^{(t)}$ at the t^{th} step, so $\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)}) = \frac{1}{N}\sum_{n=1}^{N}\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})$.

Remark 6.3.1. In light of the Law of Large Number, we have, as $N \to \infty$,

$$\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)}) = \frac{1}{N}\sum_{n=1}^{N}\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) \to \mathbb{E}[\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})|\boldsymbol{\theta}^{(t)}]\,,$$

such that the condtions (i), (ii), (iii) in Assumption (**A3**) still be warranted.

Assumption (**A1**) limits the sensitivity change of the gradient $\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, as commonly used for various the basic assumption of gradient-based methods in Subsection 6.1.3. Furthermore, plugging $\boldsymbol{\theta}' = \boldsymbol{\theta}^{(t+1)}$ and $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$ in (6.1.6) yields

$$L(\boldsymbol{\theta}^{(t+1)}) \le L(\boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})^{\top}(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) + \frac{1}{2}C\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)})\|_2^2\,. \tag{6.3.10}$$

Comparing (6.3.10) with (6.3.9) under $\boldsymbol{\theta}' := \boldsymbol{\theta}^{(t+1)}$ and $\boldsymbol{\theta} := \boldsymbol{\theta}^{(t)}$, we actually implicitly assume $C \ge \mu$. Using Assumption (**A2**), substituting by $\boldsymbol{\theta}' := \boldsymbol{\theta}^* = \boldsymbol{\theta} - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})/\mu$ into (6.3.9) yields,

$$\begin{aligned}
L(\boldsymbol{\theta}^*) &\ge L(\boldsymbol{\theta}) + (\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}))^{\top}\left(\boldsymbol{\theta} - \frac{1}{\mu}\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) - \boldsymbol{\theta}\right) + \frac{\mu}{2}\left\|\boldsymbol{\theta} - \frac{1}{\mu}\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) - \boldsymbol{\theta}\right\|_2^2 \\
&\ge L(\boldsymbol{\theta}) - \frac{1}{\mu}(\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}))^{\top}\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) + \frac{1}{2\mu}\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})\|_2^2 \\
&= L(\boldsymbol{\theta}) - \frac{1}{2\mu}\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})\|_2^2\,,
\end{aligned}$$

or equivalently,

$$\frac{1}{2\mu}\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})\|_2^2 \ge L(\boldsymbol{\theta}) - L(\boldsymbol{\theta}^*)\,. \tag{6.3.11}$$

Recall the following identity, and then by Assumption (**A3**), we have

$$\begin{aligned}
\mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})\|_2^2\middle|\boldsymbol{\theta}^{(t)}\right] &= \operatorname{tr}\left(\operatorname{Var}[\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})|\boldsymbol{\theta}^{(t)}]\right) + \left\|\mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})\middle|\boldsymbol{\theta}^{(t)}\right]\right\|_2^2 \\
&\le M_\sigma + M_V\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 + \gamma_G^2\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 \\
&=: M_\sigma + M_G\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2\,,
\end{aligned} \tag{6.3.12}$$

where $M_G := M_V + \gamma_G^2 \ge \gamma^2 > 0$. Then for large enough t, we must have the learning rate $\eta_t \in (0, \gamma/(CM_G))$, and the SGD update in (6.3.6) with $M = 1$, *i.e.* $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})$, which is in turn plugged into (6.3.10) that:

$$L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^{(t)}) \le -\eta_t\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})^{\top}\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) + \frac{\eta_t^2}{2}C\|\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})\|_2^2\,. \tag{6.3.13}$$

Taking conditional expectation, given $\boldsymbol{\theta}$, on both sides of (6.3.13) yields

$$\begin{aligned}
\mathbb{E}\left[L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^{(t)})\middle|\boldsymbol{\theta}^{(t)}\right] &\le -\eta_t\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})^{\top}\mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})\middle|\boldsymbol{\theta}^{(t)}\right] + \frac{\eta_t^2}{2}C\mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})\|_2^2\middle|\boldsymbol{\theta}^{(t)}\right] \\
&\le -\eta_t\gamma\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{\eta_t^2}{2}C\left(M_\sigma + M_G\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2\right) \\
&= -\left(\gamma - \frac{\eta_t CM_G}{2}\right)\eta_t\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{\eta_t^2 CM_\sigma}{2}\,,
\end{aligned} \tag{6.3.14}$$

where the second inequality follows so that the first term is due to Assumption (**A3**) (i) and the second term

is due to (6.3.12). Since $\eta_t \in (0, \gamma/(CM_G))$ for large enough t, we have $\eta_t C M_G \leq \gamma$, and (6.3.14) becomes

$$\begin{aligned}\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^{(t)})|\boldsymbol{\theta}^{(t)}] &\leq -\frac{\eta_t C M_G}{2} \cdot \eta_t \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{\eta_t^2 C M_\sigma}{2} \\ &\leq -\frac{\eta_t C M_G}{2} \cdot \frac{\gamma}{C M_G} \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{\eta_t^2 C M_\sigma}{2} \\ &\leq -\eta_t \gamma \mu (L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) + \frac{\eta_t^2 C M_\sigma}{2}\,,\end{aligned}$$

where the last inequality is by (6.3.11) with $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$. By telescoping, we finally have

$$\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^*)|\boldsymbol{\theta}^{(t)}] \leq (1 - \eta_t \gamma \mu)(L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)) + \frac{\eta_t^2 C M_\sigma}{2}\,. \tag{6.3.15}$$

Taking expectation on both sides of (6.3.15)

$$\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^*)] \leq (1 - \eta_t \gamma \mu)\mathbb{E}[L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)] + \frac{\eta_t^2 C M_\sigma}{2}\,. \tag{6.3.16}$$

Finally, subtracting $\eta_t C M_\sigma/(2\mu\gamma)$ from both sides of (6.3.16) gives:

$$\begin{aligned}\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^*)] - \frac{\eta_t C M_\sigma}{2\mu\gamma} &\leq (1 - \eta_t \gamma \mu)\mathbb{E}[L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)] + \frac{\eta_t^2 C M_\sigma}{2} - \frac{\eta_t C M_\sigma}{2\mu\gamma} \\ &\leq (1 - \eta_t \gamma \mu)\left(\mathbb{E}[L(\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^*)] - \frac{\eta_t C M_\sigma}{2\mu\gamma}\right)\,.\end{aligned}$$

Hence, as $\eta_t \in (0, \gamma/(CM_G))$ for large enough t, $M_G \geq \gamma^2 > 0$, and $C \geq \mu$ as mentioned before,

$$0 < \eta_t \gamma \mu < \frac{\mu \gamma^2}{C M_G} \leq \frac{\mu \gamma^2}{C \gamma^2} = \frac{\mu}{C} \leq 1\,,$$

therefore

$$\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^*)] \to \frac{\eta_t C M_\sigma}{2\mu\gamma} \to 0\,, \qquad \text{as} \quad t \to \infty\,.$$

Remark 6.3.2. For if η_t is just a constant η, we still have the last convergence result but spelled as:

$$\lim_{t\to\infty} L(\boldsymbol{\theta}^{(t)}) = L(\boldsymbol{\theta}^*) + \frac{\eta C M_\sigma}{2\mu\gamma}\,.$$

With a sufficiently small η, the second term of error on the right-hand side will be negligible, and so in practice, with a small constant learning rate, we can still achieve the minimum of L effectively.

6.3.4 Convergence Rate of SGD

We now try to investigate how many steps on average we needed to be close enough to the optimal point $\boldsymbol{\theta}^*$. Firstly, we define a filtration $\{\mathcal{F}^t\}_{t=1}^{\infty}$ such that

$$\mathcal{F}^t := \sigma\left(\boldsymbol{\theta}^{(k)}, \mathbf{x}_k : k = 0, 1, 2, \cdots, t\right)\,, \qquad \text{for} \quad t = 1, 2, \cdots,$$

for our original, but unknown, probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Next, we only take on the Assumptions **(A1)** **(A3)**(iii).

In the literature, the analysis is mainly under the condition of the boundedness in norm of the gradient term $\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|_2$, and the requirement that $\mathrm{tr}(\mathrm{Var}[\nabla_{\boldsymbol{\theta}} \ell_{n_{t+1}}(\boldsymbol{\theta})|\boldsymbol{\theta}]) \leq M_\sigma < \infty$ in [12, 20, 25]. However, this boundedness assumption does not hold even for the commonly adopted square-loss functions, unless the parameter space is compact. Our present study herein does not require a bounded gradient; indeed, Assumption **(A3)**(iii) is a much relaxed condition, which says that the variance of the gradient noise can be decomposed into a constant additive part, together with a term proportional to the square-norm of the overall gradient noise. This weaker growth condition has been considered, for instance, [19, 21]; adopting

this, we can still able to show that $\mathbb{P}(\tau_\varepsilon < \infty) = 1$, and moreover $\mathbb{E}[\tau_\varepsilon] \le O(1/(\eta\varepsilon^2))$. To this end, we shall deduce an upper bound of the expected number of iterations of SGD:

Proposition 6.3.1. [9] Under Assumptions **(A1)**, **(A3)**, and $\eta \in \left(0, \frac{2\varepsilon^2}{C(\varepsilon^2(2+M_G)+M_\sigma)} \wedge \frac{2}{C(2+M_G)}\right)$, τ_ε is integrable with

$$\mathbb{E}[\tau_\varepsilon] \le \frac{L(\boldsymbol{\theta}^{(0)}) - L(\boldsymbol{\theta}^*)}{\eta\left(\varepsilon^2 - \frac{C\eta[\varepsilon^2(2+M_G)+M_\sigma]}{2}\right)}. \tag{6.3.17}$$

Proof. Let

$$\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)} = -\eta\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) = -\eta\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)}) - \eta\boldsymbol{\delta}^{(t+1)},$$

where $\boldsymbol{\delta}^{(t+1)} := \nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)}) - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})$. By Assumption **(A1)** and then recall (6.3.10), we have

$$\begin{aligned}
L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^{(t)}) &\le \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) + \frac{1}{2}C\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\|_2^2 \\
&= -\eta\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 - \eta\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})^\top\boldsymbol{\delta}^{(t+1)} + \frac{C\eta^2}{2}\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)}) + \boldsymbol{\delta}^{(t+1)}\|_2^2 \\
&\le -\eta\left(1 - \frac{C\eta}{2}\right)\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 - \eta(1 - C\eta)\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})^\top\boldsymbol{\delta}^{(t+1)} + \frac{C\eta^2}{2}\|\boldsymbol{\delta}^{(t+1)}\|_2^2. \quad (6.3.18)
\end{aligned}$$

We first note that $\mathbb{E}[\boldsymbol{\delta}^{(t+1)}|\mathcal{F}^t] = \mathbb{E}[\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})|\mathcal{F}^t] - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)}) = \mathbf{0}$. On the other hand, by Assumption **(A3)**(iii) leading to (6.3.12), we have

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{\delta}^{(t+1)}\|_2^2|\mathcal{F}^t] &= \mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell_{n_{t+1}}(\boldsymbol{\theta}^{(t)})\|_2^2|\mathcal{F}^t] + \mathbb{E}[\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2|\mathcal{F}^t] \\
&\le (M_\sigma + M_G\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2) + \|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2.
\end{aligned}$$

Therefore, taking conditional expectation given $\mathcal{F}^t$ on both sides of (6.3.18) yields that

$$\begin{aligned}
\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)})|\mathcal{F}^t] - L(\boldsymbol{\theta}^{(t)}) &\le -\eta\left(1 - \frac{C\eta}{2}\right)\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{C\eta^2}{2}\left(M_\sigma + (M_G+1)\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2\right) \\
&\le -\eta\left(1 - \frac{C\eta(2+M_G)}{2}\right)\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{C\eta^2M_\sigma}{2}. \quad (6.3.19)
\end{aligned}$$

Using (6.3.19), along with a standard optional stopping argument, we have for any $T > 0$,

$$\begin{aligned}
\mathbb{E}\left[L(\boldsymbol{\theta}^{(\tau_\varepsilon\wedge T)})\right] - L(\boldsymbol{\theta}^{(0)}) &= \mathbb{E}\left[\sum_{t=0}^{\tau_\varepsilon\wedge T-1}\left(L(\boldsymbol{\theta}^{(t+1)}) - L(\boldsymbol{\theta}^{(t)})\right)\right] \\
&= \sum_{t=0}^{T-1}\mathbb{E}\left[\left(\mathbb{E}[L(\boldsymbol{\theta}^{(t+1)})|\mathcal{F}^t] - L(\boldsymbol{\theta}^{(t)})\right)\mathbb{1}_{\{\tau_\varepsilon > t\}}\right] \\
&\le \sum_{t=0}^{T-1}\mathbb{E}\left[\left(-\eta\left(1 - \frac{C\eta(2+M_G)}{2}\right)\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2^2 + \frac{C\eta^2M_\sigma}{2}\right)\mathbb{1}_{\{\tau_\varepsilon > t\}}\right] \\
&\le -\eta\left(\varepsilon^2 - \frac{C\eta[\varepsilon^2(2+M_G)+M_\sigma]}{2}\right)\mathbb{E}[\tau_\varepsilon \wedge T],
\end{aligned} \tag{6.3.20}$$

where the last inequality follows from the fact that $\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)})\|_2 \ge \varepsilon$ on the event $\{\tau_\varepsilon > t\}$. By the choice of η, rearranging (6.3.20) yields

$$\mathbb{E}[\tau_\varepsilon \wedge T] \le \frac{L(\boldsymbol{\theta}^{(0)}) - \mathbb{E}[L(\boldsymbol{\theta}^{(\tau_\varepsilon\wedge T)})]}{\eta\left(\varepsilon^2 - \frac{C\eta[\varepsilon^2(2+M_G)+M_\sigma]}{2}\right)} \le \frac{L(\boldsymbol{\theta}^{(0)}) - L(\boldsymbol{\theta}^*)}{\eta\left(\varepsilon^2 - \frac{C\eta[\varepsilon^2(2+M_G)+M_\sigma]}{2}\right)}. \tag{6.3.21}$$

Since the right-hand side of (6.3.21) is independent of T, by using monotone convergence theorem $\mathbb{E}[\tau_\varepsilon] = \lim_{t\to\infty}\mathbb{E}[\tau_\varepsilon \wedge T]$, which is also bounded by the same number, thus $\mathbb{P}(\tau_\varepsilon < \infty) = 1$. □

6.4 Stochastic Gradient Langevin Equation (SGLE)

6.4.1 Itô's Diffusion Processes and Stochastic Differential Equations

Definition 6.4.1. Let $\{\mathcal{F}^t\}$ be an increasing family of σ-algebras, indexed by time t, of subsets of the sample space Ω. A process $\mathbf{x}_t(\boldsymbol{\omega}) : [0,\infty) \times \Omega \mapsto \mathbb{R}^D$ is called $\mathcal{F}^t$-adapted if for each $t \geq 0$, the function $\boldsymbol{\omega} \mapsto \mathbf{x}_t(\boldsymbol{\omega})$ is $\mathcal{F}^t$-measurable.

In layman term, $\mathcal{F}^t$ can be taken as the collection of all possible information by the time t. The increasing nature $\mathcal{F}^t$ means that more and more information can be acquired by the evolution of time. In other words, all the behaviour of $\mathbf{x}_s$ for $s \in [0,t]$ is known by the time t.

Definition 6.4.2. Brownian Motion: A stochastic process $\{\mathbf{w}_t\}$ in time is a $\mathbb{R}^D$ standard Brownian motion if it satisfies the following properties:

1. (**Starts at 0**) $\mathbf{w}_0 = \mathbf{0}$;
2. (**Stationary Increment**) for $s \geq 0$ and $t > 0$, $\mathbf{w}_{t+s} - \mathbf{w}_t \overset{d}{=} \mathbf{w}_s - \mathbf{w}_0 \sim \mathcal{N}(0, s\boldsymbol{I}_D)$;
3. (**Independent Increment**) for $0 = t_0 \leq t_1 \leq \cdots \leq t_N$, the collection of random vectors $\{\mathbf{w}_{t_n} - \mathbf{w}_{t_{n-1}}\}_{n=1,\cdots,N}$ are independent.

Definition 6.4.3. Itô's Diffusion Process: A stochastic process $\{\mathbf{x}_t\} \in \mathbb{R}^M$ in time is called Itô's Diffusion Process if there exist two adapted processes $\{\boldsymbol{a}_t\} \in \mathbb{R}^M$ and $\{\boldsymbol{\Sigma}_t\} \in \mathbb{R}^{D\times M}$ such that

$$d\mathbf{x}_t = \boldsymbol{a}_t\, dt + \boldsymbol{\Sigma}_t\, d\mathbf{w}_t\,, \qquad \text{for} \quad 0 \leq t \leq T\,,$$

so that both $\mathbb{E}(\|\boldsymbol{a}_t\|_2^2) < \infty$ and $\mathbb{E}(\|\boldsymbol{\Sigma}_t\|_2^2) < \infty$.

If $\{\boldsymbol{x}_t\}$ is an Itô's Diffusion Process and $f(t,\boldsymbol{x}) : \mathbb{R}^+ \times \mathbb{R}^M \to \mathbb{R}$ is $C^{1,2}$ differentiable function, *i.e.* continuously differentiable in time and twice continuously differentiable in spatial variables, then $\{f(t,\boldsymbol{x}_t)\}$ is also an Itô's Diffusion Process, so that its dynamics is:

$$df = \left(\frac{\partial f}{\partial t} + (\nabla_{\boldsymbol{x}} f)^\top \boldsymbol{a}_t + \frac{1}{2}\mathrm{tr}(\boldsymbol{\Sigma}_t^\top (\nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{x}}^\top f)\boldsymbol{\Sigma}_t)\right) dt + (\nabla_{\boldsymbol{x}} f)^\top \boldsymbol{\Sigma}_t\, d\mathbf{w}_t \ .$$

6.4.2 Approximation for SGD scheme by SGLE

Stochastic Gradient Langevin Equation (SGLE) is defined as follows:

$$d\boldsymbol{\theta}_t = -\boldsymbol{l}(\boldsymbol{\theta}_t)\, dt + \sigma \boldsymbol{I}_D\, d\mathbf{w}_t\,, \tag{6.4.1}$$

where $\boldsymbol{l} = \nabla_{\boldsymbol{\theta}}\mathbb{E}(F)$, and F is an individual loss function. For simplicity, we write $\boldsymbol{\theta}^{(k)} := \boldsymbol{\theta}_k$, then

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \underbrace{\nabla_{\boldsymbol{\theta}} F}_{\boldsymbol{l}=\mathbb{E} \text{ for this}} (\boldsymbol{\theta}_k) \tag{6.4.2}$$

For SGD, we pick up a sample data point $(\boldsymbol{x}_{n_k}, y_{n_k})$ at random. So $\eta\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \eta l_{n_k}(\boldsymbol{\theta}; \boldsymbol{x}_{n_k}, y_{n_k})$ has a mean of $\mathbb{E}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}))$ and a variance $\eta^2 \mathrm{Var}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}))$.

To motivate how we approximate a SGD scheme by Stochastic Gradient Langevin Equation (SGLE), define $\boldsymbol{\xi}_k = \frac{\eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k) - \text{ its mean}}{\text{its s.d.}}$, where its mean and its s.d. are approximately $O(\eta)$, so that $\boldsymbol{\xi}_k$ depends on $\boldsymbol{\theta}_k$ for all k. By the non-homogeneous version of CLT (for CLT martingale differences),

$$\sum_{i=1}^{k} \frac{\boldsymbol{\xi}_i}{\sqrt{k}} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_D)\,, \tag{6.4.3}$$

and further taking $\eta = (1/\Delta t)^{-1} = \Delta t$, (6.4.2) can be represented by:

$$\Delta \boldsymbol{\theta}_k = -\eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k) = \mathbb{E}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k))\Delta t + \sqrt{\Delta t}\sqrt{\Delta t}\,\boldsymbol{\xi}_k\,, \tag{6.4.4}$$

for $\boldsymbol{\xi}_k \sim \mathcal{N}(0, \boldsymbol{I}_D)$ such that $\sqrt{\Delta t}\,\boldsymbol{\xi}_k \sim \mathcal{N}(0, \Delta t \boldsymbol{I}_D)$. Then, (6.4.4) is approximately converging to

$$d\boldsymbol{\theta}_t = \mathbb{E}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}))\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} dt + \sqrt{\Delta t}\boldsymbol{I}_D\, d\mathbf{w}_t := \nabla_{\boldsymbol{\theta}} \mathbb{E}(L(\boldsymbol{\theta}))\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} dt + \sigma \boldsymbol{I}_D\, d\mathbf{w}_t\,, \tag{6.4.5}$$

where $\sigma := \sqrt{\Delta t}$.

Taking $\Phi(\boldsymbol{\theta}) = \mathbb{E}(L(\boldsymbol{\theta}; \boldsymbol{x}_1, y_1))$ such that $\boldsymbol{l}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta})$. Here we only consider radially symmetric F arround the minimizer $\boldsymbol{\theta}_*$. Consider now

$$d(\boldsymbol{\theta}_t - \boldsymbol{\theta}_*) = -\boldsymbol{l}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_* + \boldsymbol{\theta}_*)\, dt + \sigma \boldsymbol{I}_D\, d\mathbf{w}_t\,. \tag{6.4.6}$$

So without loss of generality, take $\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_*$, and take the minimizer as 0. Consider Φ and $\boldsymbol{l}$ are radially symmetric around 0. We are now interested in looking for the expected time for the Langevin dynamics needed to approach an ε-neighborhood of the origin; also see Figure 6.4.1.

After considering the original discrete SGD can well be approximated by a continuous SGLE, we here only focus on the continuous setting. If we assume that $\sigma = 0$,

1-dimensional case: When $D = 1$, we have

$$d\theta_t = -L(\theta_t)\, dt \qquad \Rightarrow \qquad \frac{d\theta_t}{\theta_t} = -\frac{L(\theta_t)}{\theta_t}\, dt\,.$$

Integrating both sides from 0 to t, we have

$$[\ln \theta_\tau]_0^t = -\int_0^t \frac{L(\theta_\tau)}{\theta_\tau}\, d\tau \qquad \Rightarrow \qquad \theta_t = \theta_0 \exp\left(-\int_0^t \frac{L(\theta_\tau)}{\theta_\tau}\, d\tau\right)\,.$$

Multi-dimensional case

We further assume that the number of feature variables D is also large, *i.e.* $D \gg 2$. Denote $R^2 := \sum_{i=1}^{D} \left(\theta_t^i\right)^2$, then consider its first and its second partial derivatives with respect to θ_t^i:

$$2R \cdot \frac{\partial R}{\partial \theta_t^i} = 2\theta_t^i \qquad \Rightarrow \qquad \frac{\partial R}{\partial \theta_t^i} = \frac{\theta_t^i}{R}\,, \tag{6.4.7}$$

and

$$\frac{\partial^2 R}{\partial \left(\theta_t^i\right)^2} = \frac{1}{R} - \frac{\theta_t^i}{R^2} \cdot \frac{\theta_t^i}{R} = \frac{1}{R} - \frac{\left(\theta_t^i\right)^2}{R^3}\,. \tag{6.4.8}$$

Consider $\boldsymbol{l} = \nabla_{\boldsymbol{\theta}} \Phi$, we assume Φ is now radially symmetric around the origin.

$$\boldsymbol{l} = \nabla_{\boldsymbol{\theta}} \Phi = \left(\frac{\partial \Phi}{\partial \theta_t^i}\right)_{i=1}^{D} = \left(\Phi'(R) \cdot \frac{\partial R}{\partial \theta_t^i}\right)_{i=1}^{D} = \left(\Phi'(R) \cdot \frac{\theta_t^i}{R}\right)_{i=1}^{D} =: (L^i)_{i=1}^{D}\,.$$

Then by the Itô's Lemma and the law of total derivatives,

$$\begin{aligned} dR_t &= \sum_{i=1}^{D} \frac{\partial R}{\partial \theta_t^i}\, d\theta_t^i + \frac{1}{2}\sum_{i=1}^{D}\sum_{j=1}^{D} \frac{\partial^2 R}{\partial \theta_t^i \partial \theta_t^j} \cdot (d\theta_t^i)(d\theta_t^j) \\ &= \sum_{i=1}^{D} \frac{\partial R}{\partial \theta_t^i}\Big(- L^i\, dt + \sigma\, dW_t^i \Big) + \frac{1}{2}\sum_{i=1}^{D}\sum_{j=1}^{D} \frac{\partial^2 R}{\partial \theta_t^i \partial \theta_t^j} \cdot \Big(- L^i\, dt + \sigma\, dW_t^i \Big)\Big(- L^j\, dt + \sigma\, dW_t^j \Big), \end{aligned}$$

where the last equality is the direct substitution of (6.4.6). Then, using the martingale property, (6.4.7), and (6.4.8),

$$\begin{aligned} dR_t &= \sum_{i=1}^{D} \frac{\theta_t^i}{R}(-L^i)\, dt + \sigma \sum_{i=1}^{D} \frac{\partial R}{\partial \theta_t^i}\, dW_t^i + \frac{1}{2}\sum_{i=1}^{D} \frac{\partial^2 R}{\partial \left(\theta_t^i\right)^2} \cdot (\sigma^2\, dt) \\ &= \sum_{i=1}^{D} \frac{\theta_t^i}{R}\left(-\Phi'(R)\frac{\theta_t^i}{R}\right) dt + \sigma \sum_{i=1}^{D} \frac{\theta_t^i}{R} dW_t^i + \frac{1}{2}\sum_{i=1}^{D}\left(\frac{1}{R} - \frac{\left(\theta_t^i\right)^2}{R^3}\right) \cdot (\sigma^2\, dt) \\ &= \left(-\Phi'(R) + \frac{1}{2}\left(\frac{D}{R} - \frac{1}{R}\right)\sigma^2\right) dt + \sigma \sum_{i=1}^{D} \frac{\theta_t^i}{R}\, dW_t^i =: \left(-\Phi' + \frac{(D-1)\sigma^2}{2R}\right) dt + \sigma \sum_{i=1}^{D} \frac{\theta_t^i}{R}\, dW_t^i, \end{aligned}$$

where the second last equality is by the definition of $R^2 := \sum_{i=1}^{D} \left(\theta_t^i\right)^2$. Next, consider

$$\begin{aligned} d(\ln R_t) &= \frac{1}{R_t}\, dR_t - \frac{1}{2}\frac{1}{R_t^2}\, (dR_t)^2 = \left(-\frac{\Phi'}{R} \cdot \frac{(D-1)\sigma^2}{2R^2}\right) dt - \frac{1}{2}\frac{1}{R_t^2}\left(\sigma^2 \sum_{i=1}^{D} \frac{\left(\theta_t^i\right)^2}{R^2}\, dt\right) + \sigma \sum_{i=1}^{D} \frac{\theta_t^i}{R}\, dW_t^i \\ &= \left(-\frac{\Phi'}{R} + \frac{(D-2)\sigma^2}{2R^2}\right) dt + \sigma \sum_{i=1}^{D} \frac{\theta_t^i}{R}\, dW_t^i . \end{aligned}$$

We further assume that $\Phi'(R)/R$ is always positive such that

$$0 < C_L \le \inf \frac{\Phi'(R)}{R} \le \sup \frac{\Phi'(R)}{R} \le C_u < \infty . \tag{6.4.9}$$

Consider the example $\Phi'(R) = R$, which corresponds to $R^2/2$ square-loss function for an individual deviation.

6.4.3 Expected Hitting time for SGLE on ε-neighborhood of Origin

We now study the hitting time for ε-neighborhood of origin. Define the hitting time $\tau_\delta := \inf\{t : R_t = \delta\}$. Also take $\tau = \min(\tau_\varepsilon, \tau_B)$, where $\varepsilon \ll B$. By simple first step analysis, $\tau < \infty$.

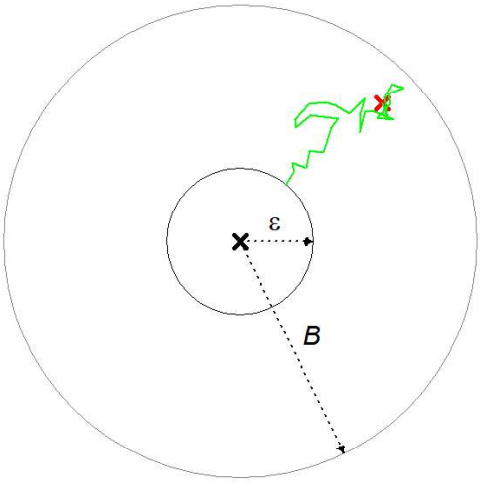


Figure 6.4.1: A path hitting the ε-neighborhood of origin with time $\tau = \min(\tau_\varepsilon, \tau_B) = \tau_\varepsilon$.

Consider any $f \in C^{1,2}(t, \mathbb{R}^D)$, or just the independent case, where $f \in C^2(\mathbb{R}^D)$.

$$\begin{aligned} df(R_t) &= f'(R_t)\, dR_t + \frac{1}{2} f''(R_t)(dR_t)^2 \\ &= \left(-\Phi' + \frac{(D-1)\sigma^2}{2R}\right) f'\, dt + f' \cdot \sigma \sum_{i=1}^{D} \frac{\theta_t^i}{R}\, dW_t^i + \frac{1}{2} f''(R_t) \left(\sigma^2 \sum_{i=1}^{D} \frac{\left(\theta_t^i\right)^2}{R^2}\, dt\right) \\ &= \left[\left(-\Phi' + \frac{(D-1)\sigma^2}{2R}\right) f' + \frac{\sigma^2}{2} f''\right] dt + \sigma f' \sum_{i=1}^{D} \frac{\theta_t^i}{R}\, dW_t^i \,. \end{aligned}$$

We want to set the dt component 0, so

$$\left(-\Phi' + \frac{(D-1)\sigma^2}{2R}\right) f' + \frac{\sigma^2}{2} f'' = 0 \,.$$

Also assume that f is radially symmetric

$$f'' - \left(\frac{2}{\sigma^2}\Phi' - \frac{D-1}{R}\right) f' = 0 \,. \tag{6.4.10}$$

In solving (6.4.10), we first notice that

$$\frac{\partial}{\partial R}\left(\frac{2}{\sigma^2}\Phi - (D-1)\ln R\right) = \frac{2}{\sigma^2}\Phi' - \frac{D-1}{R} \,,$$

we have the integrating factor

$$\exp\left[-\left(\frac{2}{\sigma^2}\Phi - (D-1)\ln R\right)\right] = R^{D-1} \exp\left(-\frac{2}{\sigma^2}\Phi\right) \,, \tag{6.4.11}$$

where Φ, in some sense, is of quadratic growth by our assumption. Therefore, using this integrating factor in (6.4.11), the solution of the first order differential equation on f' reads

$$R^{D-1} \exp\left(-\frac{2}{\sigma^2}\Phi\right) f' = C \,. \tag{6.4.12}$$

Finally, integrating f' from 1 to R gives

$$f(R) = C \int_1^R u^{-(D-1)} \exp\left(\frac{2}{\sigma^2}\Phi(u)\right) du \,. \tag{6.4.13}$$

In simplicity, we take $\varepsilon \ll 1$, and $f(R)$ explodes to ∞ with R, the rate is very fast indeed, almost of order $\exp\left(\frac{1}{\sigma^2}\Phi(R)\right)$.

Theorem 6.4.1. Optional Stopping Theorem: Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a filtered probability space with $\{\mathcal{F}^t\}_{t\in[0,\infty)}$ as a filtration, let $\{M_t\}$ be an $\mathcal{F}^t$-martingale. Given any finite $\mathcal{F}^t$-stopping time $\tau \geq 0$ such that M_τ is integrable and

$$\lim_{n\to\infty} \mathbb{E}(|M_n| : \tau > n) = 0 \,,$$

we then have

$$\mathbb{E}(M_\tau) = M_0 \,.$$

Since R is uniformly bounded before hitting ε or B, by the Optional Stopping Theorem,

$$\mathbb{E}(f(R_\tau)) = f(R_0) \,,$$

or

$$\mathbb{P}(\text{hitting } B) \cdot f(B) + \mathbb{P}(\text{hitting } \varepsilon) f(\varepsilon) = f(R_0) \,. \tag{6.4.14}$$

Note that as $\tau < \infty$ a.s., $\mathbb{P}(\text{hitting } B) + \mathbb{P}(\text{hitting } \varepsilon) = 1$. Therefore, (6.4.14) can be rewritten as

$$\mathbb{P}(\text{hitting } \varepsilon) = \frac{f(B) - f(R_0)}{f(B) - f(\varepsilon)} \,.$$

Consider when $B \to \infty$, $f(B)$ increases to ∞ at a much faster rate than B such that

$$\mathbb{P}(\text{hitting } \varepsilon \text{ first before going to } \infty) = 1\,.$$

Moreover,

$$\mathbb{P}(\text{hitting } B \text{ before } \varepsilon) = 1 - \left(1 - O\left(\frac{1}{f(B)}\right)\right) = O\left(\frac{1}{f(B)}\right)\,.$$

Now, we know that R_t must hit inside an ε-neighborhood of the origin,

$$\mathbb{E}(\ln R_\tau - \ln R_0) = \mathbb{E}\left(\int_0^\tau -\frac{\Phi'(R_t)}{R_t} + \frac{(D-2)\sigma^2}{2R_t^2}\,dt\right) + 0\,,$$

where the zero term is due to the martingale. But

$$\mathbb{E}(\ln R_\tau) = \ln R_{\tau_\varepsilon} \cdot \mathbb{P}(\text{hitting } \varepsilon) + \ln R_{\tau_B} \cdot \mathbb{P}(\text{hitting } B)\,.$$

Further taking the limit as $B \to \infty$, we have

$$\lim_{B\to\infty} \mathbb{E}(\ln R_\tau) = \ln \varepsilon \cdot 1 + \lim_{B\to\infty} \ln B \cdot O\left(\frac{1}{f(B)}\right) = \ln \varepsilon\,,$$

as $f(B)$ goes to ∞ much faster than $\ln B$. Hence,

$$\mathbb{E}\left(\int_0^{\tau_\varepsilon} \frac{\Phi'(R_t)}{R_t} + \left(-\frac{(D-2)\sigma^2}{2R_t^2}\right)\,dt\right) = -\ln\varepsilon + \ln R_0\,.$$

Consider the lower bound and the upper bound:

$$\mathbb{E}\left(\int_0^{\tau_\varepsilon} \frac{\Phi'(R_t)}{R_t} - \frac{(D-2)\sigma^2}{2R_t^2}\,dt\right) \geq \mathbb{E}\left(\int_0^{\tau_\varepsilon} C_L - \frac{(D-2)\sigma^2}{2\varepsilon^2}\,dt\right) = \left(C_L - \frac{(D-2)\sigma^2}{2\varepsilon^2}\right)\mathbb{E}\left(\tau_\varepsilon\right)$$

and noting that the last term is negative,

$$\mathbb{E}\left(\int_0^{\tau_\varepsilon} \frac{\Phi'(R_t)}{R_t} - \frac{(D-2)\sigma^2}{2R_t^2}\,dt\right) \leq \mathbb{E}\left(\int_0^{\tau_\varepsilon} C_u\,dt\right) = C_u\mathbb{E}(\tau_\varepsilon)\,.$$

Therefore

$$C_L - \frac{D-2}{2\varepsilon^2}\sigma^2 \leq \frac{-\ln\varepsilon + \ln R_0}{\mathbb{E}(\tau_\varepsilon)} \leq C_u\,.$$

As we know $\sigma \approx \sqrt{\Delta t} = \sqrt{\eta}$, so $\frac{D-2}{2}\left(\frac{\sigma}{\varepsilon}\right)^2$ would not be too large and it is much smaller than C_L, *i.e.* $\ll C_L$.

$$\mathbb{E}(\tau_\varepsilon) \sim O(-\ln\varepsilon + \ln R_0)\,.$$

Finally, the expected number of iterations for SGD can be understood as:

$$\frac{\mathbb{E}(\tau_\varepsilon)}{\Delta t} = O\left(-\frac{\ln\varepsilon}{\eta}\right)\,.$$

6.A Appendices

6.A.1 More Convergence Results for SGD

We first state the lemmas and theorems used:

Lemma 6.A.1. Let $\{z_t\}$ be a sequence of positive numbers such that

$$z_{t+1} \leq z_t(1-a_t) + c_t\,, \tag{6.A.1}$$

with

$$0 < a_t < 1\,, \qquad \sum_t a_t = +\infty\,, \qquad c_t > 0\,, \qquad \sum_t c_t < +\infty\,.$$

Then $z_t \to 0$, as $t \to +\infty$.

Proof. From (6.A.1), we have $z_{t+1} - z_t \leq c_t - z_t a_t$, then by the telescoping sum:

$$-z_0 \leq z_t - z_0 = \sum_{k=0}^{t-1}(z_{k+1} - z_k) \leq \sum_{k=0}^{t-1} c_k - \sum_{k=0}^{t-1} z_k a_k \leq \sum_{k=0}^{t-1} c_k < +\infty\,.$$

From the assumption, we get that z_t is bounded,

$$\sum_t z_t a_t < +\infty \qquad \text{and} \qquad \sum_{k=0}^{t-1} z_k a_k \leq z_0 + \sum_{k=0}^{t-1} c_k\,. \tag{6.A.2}$$

Since z_t is bounded, we can consider the limit points of this sequence. Suppose z^* is a strictly positive limit point. If z_{t_h} is a subsequence converging to z^*, we have

$$\sum_h z_{t_h} a_{t_h} < \sum_t z_t a_t < +\infty\,.$$

We get a contradiction with the assumption on the sequence a_t. Therefore $z^* = 0$ is the only possible limit point. This implies that $z_t \to 0$, which completes the proof. □

We shall use in the sequel a very important convergence theorem for supermartingales due to Doob:

Theorem 6.A.1 (DOOB's MARTINGALE CONVERGENCE THEOREM)**.** Consider an $\mathcal{F}^n$ supermartingale x_n, hence $\mathcal{F}^n$ is a filtration and

$$\mathbb{E}[\mathrm{x}_{n+1}|\mathcal{F}^n] \leq \mathrm{x}_n.$$

Assume also that $\mathrm{x}_n \geq \mathrm{y}$ with $\mathbb{E}(\mathrm{y}^-) < +\infty$. Then x_n converges a.s..

This result is the stochastic analogue of the deterministic result that a monotone (decreasing) sequence, which is bounded below, converges. Moreover, this results extends to the case when the supermartingale x_n satisfies $\mathrm{x}_n \geq -\mathrm{y}_n$, y_n is positive, adapted to the filtration $\mathcal{F}^n$, $\mathrm{y}_n \uparrow \mathrm{y}$, with $\mathrm{y} < +\infty$, a.s.. Indeed, for any M, let $\tau_M = \inf\{n : \mathrm{y}_n > M\}$. The process $\mathrm{x}_{n\wedge\tau_M}$, where $n \wedge \tau_M := \min(n, \tau_M)$, satisfies $\mathrm{x}_{n\wedge\tau_M} \geq -M$ and is an $\mathcal{F}^{n\wedge\tau_M}$ supermartingale based on optimal stopping theorem. Therefore, $\mathrm{x}_{n\wedge\tau_M}$ converges a.s. for any M. Since, for $M > \mathrm{y}$, $\mathrm{x}_{n\wedge\tau_M} = \mathrm{x}_n$, x_n converges a.s..

Lemma 6.A.2 (Stochastic Analogue of Lemma 6.A.1)**.** Let z_t be a positive process adapted to a filtration $\mathcal{F}^t$, with

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] \leq \mathrm{z}_t(1 - a_t + b_t) + c_t\,, \tag{6.A.3}$$

where $\{a_t\}, \{b_t\}, \{c_t\}$ are adapted positive sequences, $a_t < 1 + b_t$, and

$$\sum_t a_t = +\infty\,, \qquad \sum_t b_t < +\infty\,, \qquad \sum_t c_t < +\infty\,, \qquad \text{a.s.}\,.$$

Then $\mathrm{z}_t \to 0$, a.s..

Proof. We first check that we can take $b_t = 0$; indeed, from (6.A.3),

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] \le (1+b_t)\mathrm{z}_t\left(1 - \frac{a_t}{1+b_t}\right) + c_t$$

$$\mathbb{E}\left[\frac{\mathrm{z}_{t+1}}{\prod_{k=0}^{t-1}(1+b_k)}\right] \le (1+b_t)\frac{\mathrm{z}_t}{\prod_{k=0}^{t-1}(1+b_k)}\left(1 - \frac{a_t}{1+b_t}\right) + \frac{c_t}{\prod_{k=0}^{t-1}(1+b_k)}$$

$$\mathbb{E}\left[\frac{\mathrm{z}_{t+1}}{\prod_{k=0}^{t}(1+b_k)}\right] \le \frac{\mathrm{z}_t}{\prod_{k=0}^{t-1}(1+b_k)}\left(1 - \frac{a_t}{1+b_t}\right) + \frac{c_t}{\prod_{k=0}^{t-1}(1+b_k)}$$

$$\mathbb{E}[\mathrm{z}'_{t+1}|\mathcal{F}^t] \le \mathrm{z}'_t(1-a'_t) + c'_t\,,$$

where

$$\mathrm{z}'_t = \frac{\mathrm{z}_t}{\prod_{k=0}^{t-1}(1+b_k)}\,, \qquad c'_t = \frac{c_t}{\prod_{k=0}^{t-1}(1+b_k)}\,, \qquad a'_t = \frac{a_t}{1+b_t}\,,$$

with

$$\mathrm{z}'_0 = \mathrm{z}_0\,, \quad c'_0 = c_0\,, \quad a'_0 = \frac{a_0}{1+b_0}\,, \qquad \text{and} \qquad 0 < a'_t < 1\,, \quad \sum_t c'_t < +\infty\,, \quad \sum_t a'_t = +\infty\,.$$

We next consider if $\sum_t a'_t < +\infty$, then from

$$\sum_t a_t = \sum_t a'_t + \sum_t a'_t b_t \qquad \text{and} \qquad \sum_t a'_t b_t < +\infty\,,$$

as $\sum_t b_t < +\infty$, which contradicts $\sum_t a_t = +\infty$. Hence, we must have $\sum_t a'_t = +\infty$. We therefore can simplify the process in (6.A.3) to

$$\mathbb{E}[\mathrm{z}'_{t+1}|\mathcal{F}^t] \le \mathrm{z}'_t(1-a'_t) + c'_t\,, \tag{6.A.4}$$

where $0 < a'_t < 1$, $c'_t > 0$ are adapted to a filtration $\mathcal{F}^t$ and satisfy $\sum_t c'_t < +\infty$ and $\sum_t a'_t = +\infty$. Define $\mathrm{x}_0 = \mathrm{z}'_0$ and

$$\mathrm{x}_n = \mathrm{z}'_n + \sum_{t=0}^{n-1} a'_t \mathrm{z}'_t - \sum_{t=0}^{n-1} c'_t\,, \qquad n = 1, 2, 3, \cdots\,. \tag{6.A.5}$$

Then

$$\mathrm{x}_{n+1} - \mathrm{x}_n = \mathrm{z}'_{n+1} - \mathrm{z}'_n + a'_n \mathrm{z}'_n - c'_n = \mathrm{z}'_{n+1} - [\mathrm{z}'_n(1-a'_t) + c'_t]\,,$$

which implies $\mathbb{E}[\mathrm{x}_{n+1}|\mathcal{F}^n] \le \mathrm{x}_n$ a.s.. So x_n is an $\mathcal{F}^n$ supermartingale. Moreover, with $\mathrm{z}'_t, a'_t > 0$,

$$\mathrm{x}_n \ge -\sum_{t=0}^{n-1} c'_t \qquad \text{and} \qquad \sum_{t=0}^{n-1} c'_t \uparrow \sum_t c'_t < +\infty\,.$$

Therefore x_n converges a.s.. Next, from (6.A.5) and $\mathrm{z}'_{n+1} > 0$,

$$\sum_{t=0}^{n} a'_t \mathrm{z}'_t \le \sum_{t=0}^{n} c'_t + \mathrm{x}_{n+1}\,,$$

hence $\sum_t a'_t \mathrm{z}'_t < +\infty$, a.s.. From the assumption on the sequence a'_t, we obtain $\mathrm{z}'_t \to 0$, a.s.. Finally, since $\prod_{k=0}^{t-1}(1+b_k)$ converges as $t \to +\infty$, it follows that $\mathrm{z}_t \to 0$ a.s.. □

6.A.1.1 ROBBINS-MONRO THEOREM

We consider $\ell(\boldsymbol{\theta}; \mathbf{x})$, where $\mathbf{x}$ is a random variable, $\boldsymbol{\theta} \in \mathbb{R}^D, \mathbf{x} \in \mathbb{R}^Q$. We assume that

(A1) Uniform Lipshitz property:

$$\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta};\mathbf{x}) - \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}',\mathbf{x})\|_2 \leq C\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|_2\,, \tag{6.A.6}$$

(A2) $\mathcal{L}^2$-boundedness:

$$\mathbb{E}\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta};\mathbf{x})\|_2^2 < +\infty\,, \qquad \forall\, \boldsymbol{x}\,. \tag{6.A.7}$$

We define $L(\boldsymbol{\theta}) = \mathbb{E}[\ell(\boldsymbol{\theta};\mathbf{x})]$. From (6.A.6), it follows that

$$\|\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}')\|_2 \leq C\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|_2 \quad \Rightarrow \quad \|\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}^\top L(\boldsymbol{\theta})\|_2 \leq C\,, \quad \text{a.e..} \tag{6.A.8}$$

with respect to Lebesgue measure, implying by the Labesgue Differentiation theorem.

We assume also that

(A3) Uniform Convexity:

$$(\boldsymbol{\theta}-\boldsymbol{\theta}')^\top(\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}')) \geq \mu\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|_2^2\,, \qquad \mu > 0\,. \tag{6.A.9}$$

Hence the function $L(\boldsymbol{\theta})$ has a unique minimum $\hat{\boldsymbol{\theta}}$, which satisfies $\nabla_{\boldsymbol{\theta}}L(\hat{\boldsymbol{\theta}}) = \mathbf{0}$. Note that $\mu < C$.

Theorem 6.A.2 (ROBBINS-MONRO THEOREM)**.** We assume (6.A.6), (6.A.9) and $\sum_t \eta_t = +\infty$, $\sum_t \eta_t^2 < +\infty$, then

$$\boldsymbol{\theta}^{(t)} \to \hat{\boldsymbol{\theta}}\,, \qquad \text{a.s..} \tag{6.A.10}$$

Proof. We define $\mathcal{F}^t := \sigma(\mathbf{x}_1, \cdots, \mathbf{x}_t)$. We shall consider the sequence $\mathrm{z}_t := \|\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}}\|_2^2$. By the definition of the stochastic gradient descent algorithm, we have

$$\boldsymbol{\theta}^{(t+1)} - \hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}} - \eta_t\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})\,.$$

Hence, after taking square of $\mathcal{L}^2$-norm on both sides,

$$\|\boldsymbol{\theta}^{(t+1)} - \hat{\boldsymbol{\theta}}\|_2^2 := \mathrm{z}_{t+1} = \mathrm{z}_t - 2\eta_t(\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}})^\top\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) + \eta_t^2\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})\|_2^2\,.$$

Therefore, further taking conditional expectation on the filtration $\mathcal{F}^t$ yields:

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] = \mathrm{z}_t - 2\eta_t(\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}})^\top\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}^{(t)}) + \eta_t^2\mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})\|_2^2|\mathcal{F}^t]\,.$$

Hence, with $\nabla_{\boldsymbol{\theta}}L(\hat{\boldsymbol{\theta}}) = \mathbf{0}$ and the assumption in (6.A.9),

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] \leq \mathrm{z}_t - 2\eta_t\mu\mathrm{z}_t + \eta_t^2\mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})\|_2^2|\mathcal{F}^t]\,. \tag{6.A.11}$$

In particular, we use a simple telescoping argument and note that for any $a, b \in \mathbb{R}$, $(a-b)^2 \leq 2a^2 + 2b^2$,

$$\mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})\|_2^2|\mathcal{F}^t] \leq 2\mathbb{E}[(\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) - \nabla_{\boldsymbol{\theta}}\ell(\hat{\boldsymbol{\theta}}, \mathbf{x}_{t+1})\|_2^2)|\mathcal{F}^t] + 2\mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell(\hat{\boldsymbol{\theta}}, \mathbf{x})\|_2^2]\,,$$

and from the assumption (6.A.6), it follows that

$$\mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})\|_2^2|\mathcal{F}^t] \leq 2C^2\mathrm{z}_t + 2\sigma^2\,,$$

where

$$\sigma^2 := \mathbb{E}[\|\nabla_{\boldsymbol{\theta}}\ell(\hat{\boldsymbol{\theta}}, \mathbf{x})\|_2^2]\,. \tag{6.A.12}$$

Hence, from (6.A.11), we can then write

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] \leq \mathrm{z}_t(1 - a_t + b_t) + c_t\,, \tag{6.A.13}$$

with

$$a_t = 2\eta_t\mu\,, \qquad b_t = 2\eta_t^2C^2\,, \qquad c_t = 2\sigma^2\eta_t^2\,.$$

From the assumptions on the sequence η_t, we see that the assumptions of Lemma 6.A.2 are satisfied, so

$$\sum_t a_t \to \infty\,, \qquad \sum_t b_t < \infty\,, \qquad \sum_t c_t < \infty\,.$$

Hence $\mathrm{z}_t \to 0$, a.s., which concludes the proof. □

6.A.1.2 KIEFER-WOLFOWITZ THEOREM

It is a variant of Robbins-Monro, in which $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1})$ is approximated by a finite difference $\boldsymbol{y}_t \in \mathbb{R}^D$ with

$$y_t^{(d)} := \frac{\ell(\boldsymbol{\theta}^{(t)} + \alpha_t \boldsymbol{e}_d, \mathbf{x}_{t+1}) - \ell(\boldsymbol{\theta}^{(t)} - \alpha_t \boldsymbol{e}_d, \mathbf{x}_{t+1})}{2\alpha_t}\,, \quad \alpha_t > 0\,, \tag{6.A.14}$$

where $d = 1, \cdots, D$ denotes the d^{th} component and $\boldsymbol{e}_d$ is the d^{th} unit vector. Now the algorithm is modified to:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \boldsymbol{y}_t\,. \tag{6.A.15}$$

We can write

$$\boldsymbol{y}_t = \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) + \frac{\alpha_t}{2}\boldsymbol{\Theta}^{(t)}\,, \tag{6.A.16}$$

where the error term $\boldsymbol{\Theta}^{(t)} = (\Theta_1^{(t)}, \cdots, \Theta_D^{(t)})^\top \in \mathbb{R}^D$ is defined as:

$$\Theta_d^{(t)} = \boldsymbol{e}_i^\top \left(\int_0^1 \int_0^1 \lambda \left(\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top \ell(\boldsymbol{\theta}^{(t)} + \lambda\mu\alpha_t \boldsymbol{e}_d, \mathbf{x}_{t+1}) - \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^\top \ell(\boldsymbol{\theta}^{(t)} - \lambda\mu\alpha_t \boldsymbol{e}_d, \mathbf{x}_{t+1}) \right) d\lambda d\mu \right) \boldsymbol{e}_d\,,$$

and

$$\|\boldsymbol{\Theta}^{(t)}\|_2 \le C\sqrt{D}\,. \tag{6.A.17}$$

Theorem 6.A.3 (KIEFER-WOLFOWITZ THEOREM)**.** We assume (6.A.8), (6.A.9) and $\sum_t \eta_t = +\infty$, $\sum_t \eta_t^2 < +\infty$, $\sum_t \eta_t \alpha_t < +\infty$. Then

$$\boldsymbol{\theta}^{(t)} \to \hat{\boldsymbol{\theta}}\,, \qquad \text{a.s..} \tag{6.A.18}$$

Proof. We write

$$\boldsymbol{\theta}^{(t+1)} - \hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}} - \eta_t \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) - \frac{\eta_t \alpha_t}{2}\boldsymbol{\Theta}^{(t)}\,,$$

and define $\mathrm{z}_t := \|\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}}\|_2^2$. After taking square of $\mathcal{L}^2$-norm on both sides, we obtain that

$$\mathrm{z}_{t+1} = \mathrm{z}_t - 2\eta_t(\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}})^\top \left(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}) + \frac{\alpha_t}{2}\boldsymbol{\Theta}^{(t)} \right) + \eta_t^2 \left\| \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) + \frac{\alpha_t}{2}\boldsymbol{\Theta}^{(t)} \right\|_2^2\,. \tag{6.A.19}$$

Hence, further taking conditional expectation on the filtration $\mathcal{F}^t$ yields:

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] \le \mathrm{z}_t - 2\eta_t\mu\mathrm{z}_t + \eta_t^2 \mathbb{E}\left[\left\| \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) + \frac{\alpha_t}{2}\boldsymbol{\Theta}^{(t)} \right\|_2^2 \middle| \mathcal{F}^t \right] - \eta_t\alpha_t(\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}})^\top \mathbb{E}(\boldsymbol{\Theta}^{(t)}|\mathcal{F}^t)\,.$$

We use

$$\mathbb{E}\left[\left\| \nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)}, \mathbf{x}_{t+1}) + \frac{\alpha_t}{2}\boldsymbol{\Theta}^{(t)} \right\|_2^2 \middle| \mathcal{F}^t \right] \le 2C^2\mathrm{z}_t + 2\mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\ell(\hat{\boldsymbol{\theta}}, \mathbf{x})\|_2 + \frac{\alpha_t}{2}C\sqrt{D} \right]^2\,,$$

and the simple inequality that for any $a, b \in \mathbb{R}$, $(a+b)^2 = a^2 + b^2 + 2ab \ge 0$ such that $-ab \le (a^2+b^2)/2$:

$$-(\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}})^\top \mathbb{E}(\boldsymbol{\Theta}^{(t)}|\mathcal{F}^t) \le \frac{1}{2}\left(\|\boldsymbol{\theta}^{(t)} - \hat{\boldsymbol{\theta}}\|_2^2 + \mathbb{E}(\|\boldsymbol{\Theta}^{(t)}\|_2^2|\mathcal{F}^t) \right) \le \frac{1}{2}(\mathrm{z}_t + C^2 D)\,.$$

We obtain finally

$$\mathbb{E}[\mathrm{z}_{t+1}|\mathcal{F}^t] \le \mathrm{z}_t \left(1 - 2\eta_t\mu + 2\eta_t^2 C^2 + \frac{\eta_t\alpha_t}{2} \right) + 2\eta_t^2 \mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\ell(\hat{\boldsymbol{\theta}}, \mathbf{x})\|_2 + \frac{\alpha_t}{2}C\sqrt{D} \right]^2 + \eta_t\alpha_t\frac{C^2 D}{2}\,. \tag{6.A.20}$$

From the assumptions on the sequences η_t, α_t, using again Lemma 6.A.2, we obtain $\mathrm{z}_t \to 0$, a.s. and conclude the proof of the theorem. □

BIBLIOGRAPHY

[1] Atkinson, K. E. (2008). *An introduction to numerical analysis*. John Wiley & Sons.

[2] Atkinson, K., and Han, W. (2005). Theoretical numerical analysis (Vol. 39, pp. xviii+-576). Berlin: Springer.

[3] Björck, Å. (1996). Numerical methods for least squares problems. Society for Industrial and Applied Mathematics, 341-342.

[4] Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. Siam Review, 60(2), 223-311.

[5] Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations. IMA Journal of Applied Mathematics, 6(1), 76-90.

[6] Broyden, C. G., Dennis Jr, J. E., and Moré, J. J. (1973). On the local and superlinear convergence of quasi-Newton methods. IMA Journal of Applied Mathematics, 12(3), 223-245.

[7] Cramer, G. (1750). Introduction à l'analyse des lignes courbes algébriques. chez les frères Cramer et C. Philibert.

[8] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7).

[9] Fan, N.K., Ng, T. H., and Yam, S. C. P. (2022). When to Stop a SGD: A Stopping Time Approach. Proceeding.

[10] Fletcher, R. (1970). A new approach to variable metric algorithms. The computer journal, 13(3), 317-322.

[11] Gauss, C. F. (1809). *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss. sumtibus Frid.* Perthes et IH Besser.

[12] Ghadimi, S., and Lan, G. (2013). Stochastic first-and zeroth-order methods for nonconvex stochastic programming. SIAM Journal on Optimization, 23(4), 2341-2368.

[13] Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. Mathematics of computation, 24(109), 23-26.

[14] Golub, G. H., and Van Loan, C. F. (2013). Matrix computations. JHU press.

[15] Gratton, S., Lawless, A. S., and Nichols, N. K. (2007). Approximate Gauss–Newton methods for nonlinear least squares problems. SIAM Journal on Optimization, 18(1), 106-132.

[16] Hestenes, M. R., and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems (Vol. 49, No. 1). Washington, DC: NBS.

[17] Horn, R. A., and Johnson, C. R. (2012). Matrix analysis. Cambridge university press.

[18] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.

[19] Jin, R., and He, X. (2020). Convergence of Momentum-Based Stochastic Gradient Descent. In 2020 IEEE 16th International Conference on Control & Automation (ICCA) (pp. 779-784). IEEE.

[20] Karimi, H., Nutini, J., and Schmidt, M. (2016). Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 795-811). Springer, Cham.

[21] Lei, Y., Hu, T., Li, G., and Tang, K. (2019). Stochastic gradient descent for nonconvex learning without bounded gradient assumptions. IEEE transactions on neural networks and learning systems, 31(10), 4394-4400.

[22] Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage. Mathematics of computation, 35(151), 773-782.

[23] Nocedal, J., and Wright, S. (2006). Numerical optimization. Springer Science & Business Media.

[24] Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. Mathematics of computation, 24(111), 647-656.

[25] Shamir, O., and Zhang, T. (2013). Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In International conference on machine learning (pp. 71-79). PMLR.

[26] Tieleman, T., and Hinton, G. (2012). Divide the gradient by a running average of its recent magnitude. COURSERA Neural Netw. Mach. Learn, 6, 26-31. `http://www.cs.toronto.edu/$\sim$tijmen/csc321/slides/lecture_slides_lec6.pdf`