Received January 5, 2025, February 30, 2025, date of publication May 1, 2025.

Digital Object Identifier 10.21608/ijaici.2025.340085.1004

Using the Quasi-Newton Method to Solve Nonlinear Least Squares Regression Problems

Esraa S. Atallah[1], Ahmed Hagag[1], Eman M. Ali[1], Tamer A. Abassy[1]

[1] Faculty of Computers and Artificial Intelligence, Benha University, Benha, Qalyubia Governorate, Egypt.

Corresponding author: Ahmed Hagag (e-mail: ahagag@fci.bu.edu.eg).

ABSTRACT In regression modeling, Gradient Descent (GD) is widely used to update parameters by iteratively minimizing a cost function. However, GD often converges slowly and may suffer from instability due to its reliance on first-order derivatives only. To improve convergence speed and stability, Newton's method utilizes second-order derivative information, aiming to find the point where the gradient vanishes. While Newton's method offers faster convergence, computing the exact Hessian matrix is often computationally expensive or infeasible.

Quasi-Newton methods overcome this limitation by approximating the Hessian matrix. These methods iteratively update an estimate of the Hessian, typically denoted as $H_k \approx \nabla^2 f(x_k)$, to guide the search direction. A notable quasi-Newton algorithm is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method.

In this paper, we apply BFGS and its variants—including Memoryless BFGS, Limited-memory BFGS (L-BFGS), and Scaled BFGS—to nonlinear least squares regression problems. Their performance is compared with traditional Gradient Descent and Newton's method, focusing on convergence behavior and optimization efficiency. The results demonstrate the potential advantages of quasi-Newton approaches in practical regression scenarios.

INDEX TERMS BFGS method, Gradient Descant, Newton method, Nonlinear least square, Quasi-Newton.

I. INTRODUCTION

Regression is a foundational technique in machine learning used to predict a target variable based on linear or nonlinear relationships between independent and dependent variables.

$$\hat{\boldsymbol{y}} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x_1} + \boldsymbol{\theta_2 x_2} + .. + \boldsymbol{\theta x_n} + \boldsymbol{\epsilon} \quad (1)$$

Where $\hat{\boldsymbol{y}}$ is the predicted value (dependent variable), $\boldsymbol{x_1, x_2, .., x_n}$ *are* independent variables, $\boldsymbol{\theta_0}$ is the intercept (constant value), $\boldsymbol{\theta_1, \theta_2, .., \theta_n}$ are coefficients for independent variables, and $\boldsymbol{\epsilon}$ is the error term.

The performance of a regression model is often evaluated using a cost function, with Mean Squared Error (MSE) being a common choice:

$$\text{MSE} = \frac{1}{2\text{m}} \sum_{1=0}^{\text{m}-1} \left(\hat{\text{y}}^{(\text{i})} - \text{y}^{(\text{i})}\right)^2 \quad (1)$$

To minimize this cost function, Gradient Descent (GD) is widely employed. It iteratively updates the model parameters by computing the gradient of the cost function with respect to each parameter:

$$\theta_n = \theta_n - \alpha \frac{\partial J}{\partial \theta_n} \quad (2)$$

Where $\boldsymbol{\alpha}$ is the learning rate (step size).

GD is popular due to its simple implementation and low memory requirements ($\boldsymbol{O(n)}$), as it only uses first-order derivatives. It performs effectively when the initial guess is far from the optimal solution $\boldsymbol{x}^*$, which is typically the case in early iterations. However, as it approaches the minimum, the updates tend to zigzag, which slows convergence and makes the method inefficient [3].

Gradient Descant algorithm

1. ***Initialization****: initialize* x_0. *Determine* $g_0 = \nabla F(x_0)$ *Set* $d_0 = -g_0$
2. *for* $k = 1, 2, \ldots$ *until convergence do*
3. *set stepsize* $\alpha_k > 0$ *satisfying the Wolfe line search conditions (3)(4).*
4. *Compute* x_{k+1} *by (2)*
5. $g_{K+1} = \nabla F(x_{K+1})$, $d_{K+1} = g_{K+1}$

TABLE. I. ALGORITHM 1

Although GD is straightforward and requires minimal memory, it often converges slowly and becomes

inefficient as it approaches the optimal solution due to zigzagging. To address these issues, optimization methods that incorporate second-order information have been developed. Newton's method uses the Hessian matrix to improve convergence. However, computing and inverting the Hessian matrix is computationally expensive, particularly for high-dimensional data.

Quasi-Newton methods provide a practical alternative. Instead of computing the Hessian directly, they approximate it using information from successive gradient evaluations. These methods, particularly the BFGS family, offer a balance between convergence speed and computational efficiency. The current study investigates the effectiveness of BFGS and its variants in optimizing nonlinear least squares regression models. To improve the stability and adaptiveness of the step size $\boldsymbol{\alpha}$, techniques such as the Wolfe line search are used [1, 2]:

$$F\ (x_k + \alpha_k d_k) < F(x_k) + \rho\ \alpha_k g_k^T d_k, \quad (3)$$

$$\nabla F(x_k + \alpha_k d_k)^T d_k > \nabla F(x_k)^T d_k \quad (4)$$

where $\mathbf{0} < \boldsymbol{\rho} < \boldsymbol{\sigma} < \mathbf{1}$ are constants, $\boldsymbol{x_k} = \boldsymbol{\theta_0}, \boldsymbol{\theta_1}, \dots, \boldsymbol{\theta_n}$.

In iterative optimization methods, we begin with an initial point $\boldsymbol{\theta_0}$ and, at each iteration, compute a search direction $\boldsymbol{d_k}$ and a step size $\boldsymbol{\alpha_k}$, updating the parameters as:

$$x_{k+1} = x_k + \alpha_k\ d_k \quad (5)$$

necessary that $\boldsymbol{F\ (x_{k+1}) < F\ (x_k)}$.

the objective function $\boldsymbol{F\ (x_{k+1})}$can be approximated by either a linear or a quadratic model,

$$\boldsymbol{F\ (x_{k+1}) \approx F(x_k) + g(x_k)^T\ d_k + \frac{1}{2} d_k{}^T H(x_k) d_k} \quad (6)$$

where $\boldsymbol{g(x_k) = \nabla F(x_k)}$ is the gradient and $\boldsymbol{H(x_k) = \nabla^2 F(x_k)}$ is the Hessian matrix of $\boldsymbol{F(x_k)}$

A. NEWTON METHOD

The condition $\nabla F(x^*) = 0$, where x* is the minimum point of the objective function $F(x)$, forms the basis of Newton's method, also known as the **Newton-Raphson method**—named after Isaac Newton and Joseph Raphson [3]. This method relies on a second-order Taylor series approximation (see Theorem 1) to find a local minimum by iteratively updating the input vector x. The update rule is defined as:

$$x_{k+1} = x_k - \alpha_k\ (\nabla^2 F(x_k))^{-1}\ \nabla f\ (x_k) \quad (7)$$

Where:

- $\nabla f\ (x_k)$ is the gradient at iteration k,
- $\nabla^2 F(x_k)$ is the Hessian matrix (second derivative),
- And α_k, is the step size.

Advantages of Newton's Method:

1. If $F(x)$ is a quadratic function, Newton's method converges in a single step, regardless of the initial starting point.
2. When the initial point x_0 is sufficiently close to the minimum, the method exhibits quadratic convergence—a highly desirable property for solving least squares problems.

In many optimization frameworks, Newton's method is considered a core component for achieving high convergence rates [4].

Despite its advantages, Newton's method also suffers from several notable drawbacks:

1. Lack of global convergence: In non-linear least squares problems, Newton's method may fail to converge. One key issue is that the computed search direction$d_k = (\nabla^2 F(x_k))^{-1}\ \nabla f\ (x_k)$ may not always be a descent direction, which is essential for ensuring the cost function decreases.
2. High computational cost: Each iteration requires evaluating the Hessian matrix and computing its inverse, both of which are computationally expensive, especially for high-dimensional data.
3. Singular Hessian issue: If the Hessian is singular or singular at any point, especially in the preliminary stages—Newton's method fails because it cannot solve the resulting system of equations.

Due to these limitations, Newton's method is often combined with more stable or approximate techniques, such as quasi-Newton methods, which aim to preserve its fast convergence without incurring the full computational cost.

TABLE. II. ALGORITHM 2

Newton algorithm
1. ***Initialization***. *initialize* x_0. *Determine* $g_0 = \nabla F(x_0), H_0 = \nabla^2 f(x_0)$. *set* $d_0 = -{H_0}^{-1} g_0$.
2. *for* $k = 1, 2, \dots$ *until convergence do*
3. *set stepsize* $\alpha_k > 0$ *satisfying the Wolfe line search conditions (3)(4).*
4. $x_{k+1} = x_k + \alpha_k d_k$
5. $g_{k+1} = \nabla f(x_k), H_{k+1} = \nabla^2 f(x_k), d_{k+1} = -{H_{k+1}}^{-1} g_{k+1}$

B. QUASI-NEWTON METHOD

Newton's method is among the most fundamental methods for dealing with unconstrained optimization problems. A fundamental result of mathematics is that $\nabla F(x^*) = 0$ is a necessary condition for optimality. Finding the gradient function's zero is the goal of Newton's method. The method is iterative, with each iteration estimating the function

gradient using a linear approximation around the k_{th} iteration.

$$\nabla F(x_k + p_k) \approx \nabla F(x_k) + \nabla^2 F(x_k) p_k \quad (8)$$

For Newton's method to work effectively, the Hessian matrix. $\nabla^2 F(x_k)$ must be available. However, computing the Hessian is often computationally expensive or infeasible for large-scale problems. This challenge leads to the development of Quasi-Newton methods, which approximate the Hessian rather than compute it directly.

The central idea is to estimate the Hessian $H_k \approx \nabla^2 F(x_k)$, and then define the search direction using this approximation. The main problem becomes how to update the matrix H_k to H_{k+1} in a way that captures curvature information from the function as we move from x_k to x_{k+1} [5, 6]. These methods blend the structure of Newton's method with the computational efficiency of Gauss-Newton approximations.

In general, Quasi-Newton methods update the solution using (Equi 7) Where: d_k is the solution to $H_k d_k = -g_k$, α_k satisfies Wolfe line search conditions H_k approximates the true Hessian.

The matrix is updated by: $H_{k+1} = H_k + U_k$

To maintain symmetry and positive definiteness—key properties of the true Hessian—Quasi-Newton methods enforce the secant condition (also called the Quasi-Newton condition):

$\nabla F(x_k + \alpha_k d_k) \approx \nabla F(x_k) + \nabla^2 F(x_k)(x_{k+1} - x_k)$ (9)

Define:

$s_k = x_{k+1} - x_k = \alpha_k d_k, y_k = \nabla F(x_{k+1}) - \nabla F(x_k),$

Then the secant condition becomes: $y_k = H_k s_k$

Additionally, we ensure the updated Hessian approximation satisfies: $H_{k+1} s_j = H_k s_j \; ; \; j = k - 1, \ldots, k - n + 1.$

The d_{k+1} is given by: $H_{k+1} d_{k+1} = -g_{k+1},$ (10)

or directly using the inverse Hessian approximation

$$d_{k+1} = -B_{k+1} g_{k+1}, \quad (11)$$

where B_{k+1} approximates the inverse Hessian, $B_{k+1} = H_{k+1}^{-1}$

Starting from an initial point x_0, these methods iteratively produce x_{k+1} until convergence to the optimal solution x^*. Quasi-Newton methods achieve super-linear convergence near the optimum and are considered robust and efficient for a wide range of differentiable functions. Several variations exist for updating the inverse Hessian, including:

- Symmetric Rank-One (SR1) update [7, 8]
- Rank-Two updates such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [7-10]

These methods have become some of the most popular and widely used techniques in numerical optimization due to their balance of speed and accuracy.

II. PROBLEM DEFINITION

Regression models are a type of least squares problem, where the goal is to minimize the cost function. This minimization yields the optimal weights (parameters $\boldsymbol{\theta}^*$) as in (2). As previously mentioned, the cost function is typically optimized using the Gradient Descent method. However, this method suffers from slow convergence and instability. Therefore, it is proposed to employ second-order methods such as Quasi-Newton techniques.

This paper aims to apply the BFGS method and its variants—including the Memoryless BFGS method [10, 11],the Limited-Memory BFGS method [11-14], the Scaled BFGS method [9], and the Double Scaled BFGS method [9]—to the least squares problem exemplified by the regression model, and to identify the most effective approach.

The most effective Quasi-Newton update for approximating the Hessian is the BFGS formula:

$$\boldsymbol{B_{k+1}} = \boldsymbol{B_k} + \frac{\boldsymbol{B_k s_k s_k^T B_k}}{\boldsymbol{s_k^T B_k s_k}} + \left(\frac{\boldsymbol{y_k y_k^T}}{\boldsymbol{y_k^T s_k}}\right) \quad (12)$$

This satisfies the secant equation (11) and represents a rank-two update. Applying the Sherman-Morrison-Woodbury formula [15] twice yields the update for the inverse Hessian in the BFGS method. Assuming $\boldsymbol{H_k}$ is the inverse approximation of the Hessian at iteration $\boldsymbol{k}$:

$$H_{k+1} = H_k - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k}\left(1 \frac{y_k^T H_k y_k}{y_k^T s_k}\right)\left(\frac{s_k s_k^T}{y_k^T s_k}\right) \quad (13)$$

A key property of BFGS is that $\boldsymbol{H_{k+1}}$ remains positive definite for any $\boldsymbol{k}$, provided that $\boldsymbol{H_k}$ is positive definite. If the inverse Hessian approximation $\boldsymbol{H_k}$ estimates the curvature of the objective function incorrectly—thus slowing down the iteration—it tends to correct itself in subsequent steps. This is one of the self-correcting features of the BFGS algorithm. The quality of the Wolfe line search implementation plays a significant role in preserving this self-correcting behavior. Using the initial step length $\boldsymbol{\alpha = 1}$ in the Wolfe line search leads to super-linear convergence. Due to its numerous advantages, the BFGS update is widely considered one of the most robust and effective Quasi-Newton methods [1, 2].

TABLE. III. ALGORITHM 3

symmetric rank two (BFGS)
1. *Initialization:*
initialize x_0.
Determine $g_0 = \nabla F(x_0)$
$H_0 = \nabla^2 F(x_0)$.
Set $d_0 = H_0^{-1} g_0$
1. *for* $k = 1, 2, \ldots$ *until convergence do*
2. *set stepsize* $\alpha_k > 0$ *conditions (3)(4).*
3. *set* $x_{k+1} = x_k + \alpha_k d_k$
4. *set* $s_k = x_{k+1} - x_k$ *and* $y_k = g_{k+1} - g_k$,
5. *use (13) to update the inverse Hessian.*
6. *Compute search direction* $d_{k+1} = -H_{k+1} g_{k+1}$.

III. METHODOLOGY

The BFGS method has notable characteristics, as previously mentioned, such as its self-correcting nature. If the current

Hessian approximation $\boldsymbol{H_k}$ is inaccurate, the BFGS method tends to correct it within a few iterations. It is also one of the most efficient and accurate methods for solving minimization problems. However, it requires a large amount of memory per iteration, making it computationally expensive with complexity $(\boldsymbol{O(g(n)) = n^2})$. As a result, it is primarily suitable for small- to medium-scale problems. To address high-dimensional problems, modified methods that reduce memory usage and computational complexity to $\boldsymbol{O(g(n)) = mn\ where\ m > n}$)have been proposed. These are known as the Memoryless BFGS and Limited-Memory BFGS (L-BFGS) methods. Additionally, to enhance performance, a modified version called the Self-Scaling BFGS has been introduced.

A. MEMORY-LESS BFGS METHOD

Memoryless BFGS eliminates the need to store and update the approximate Hessian matrix explicitly. Instead, it computes the search direction directly using past gradients and step sizes, significantly reducing memory usage and computational cost.

Assuming $\boldsymbol{H_k = I}$, the Memoryless BFGS method is defined by the update:

$$H_{k+1} = I - \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + \left(1 + \frac{y_k^T y_k}{y_k^T s_k}\right)\left(\frac{s_k s_k^T}{y_k^T s_k}\right) \quad (14)$$

The memoryless BFGS and memoryless SR1 methods differ significantly. The memoryless BFGS method is well-defined when the step size is chosen using Wolfe line search conditions, which guarantee $\boldsymbol{(s_k - y_k)^T y_k \neq 0}$ at each iteration. On the other hand, in the case of the SR1 method, Wolfe line search does not ensure that $\boldsymbol{(s_k - y_k)^T y_k \neq 0}$ [3, 7].

B. LIMITED MEMORY BFGS

Limited-memory BFGS (L-BFGS) stores only a limited number of previous updates to the gradient and position vectors (typically the most recent m updates), allowing it to approximate the inverse Hessian efficiently without the need to handle large matrices. This makes L-BFGS particularly suitable for large-scale optimization problems. L-BFGS applies the BFGS update using information from only the most recent m iterations to update the base matrix H_0 multiple times, forming H_{k+1}. Its implementation is identical to that of the standard BFGS method, except that the inverse Hessian approximation is not formed explicitly. Instead, it is represented using a limited number of BFGS updates. This method often yields a fast linear convergence rate and requires only vector operations, significantly reducing memory demands.

Equation (13) is derived from:

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \left(\frac{s_k s_k^T}{y_k^T s_k}\right) \quad (15)$$

Let: $V_k = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right)$, $p_k = \frac{1}{y_k^T s_k}$

Then: $H_{k+1} = [V_k^T \dots V_{k-m+1}^T] H_{k-m+1} [V_k \dots V_{k-m+1}] + p_{k-m+1} [V_k^T \dots V_{k-m+1}^T] s_{k-m+1} s_{k-m+1}^T [V_k \dots V_{k-m+1}] + \dots + s_k s_k^T$ (16)

This final formula avoids matrix storage and uses only vector operations, making it suitable for large-scale optimization problems.

C. SCALED BFGS METHODS

The standard BFGS method may perform poorly on non-convex minimization problems when using an exact line search. To enhance its performance, self-scaling BFGS methods have been introduced. These methods are based on the concept of scaling the eigenvalue structure of the BFGS approximation to better match the true Hessian matrix.

The scaled BFGS update formula is:

$$H_{k+1} = H_k - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k} + \left(\frac{1}{\gamma_k} + \frac{y_k^T H_k y_k}{y_k^T s_k}\right)\left(\frac{s_k s_k^T}{y_k^T s_k}\right) \quad (17)$$

Here, γ_k is a positive parameter that must be determined during the optimization process. This algorithm is simple to implement, but it is applicable only to small- and medium-scale unconstrained problems.

IV. RESULT

In this section, we investigate the efficiency of the BFGS method and its modified variants—Memoryless BFGS (ML-BFGS) and Limited-memory BFGS (L-BFGS). These methods are compared to the traditional Gradient Descent (GD) method and the Newton method. The performance metric used for comparison is the Mean Squared Error (MSE).

A. EXPERIMENT 1:

For this experiment, we used the dataset from the Elo Merchant Category Recommendation competition [16, 17]. Elo is one of the largest payment companies in Brazil. In this competition, Elo collaborated with various merchants to offer exclusive deals or discounts to their cardholders.

The key research questions in this context include:

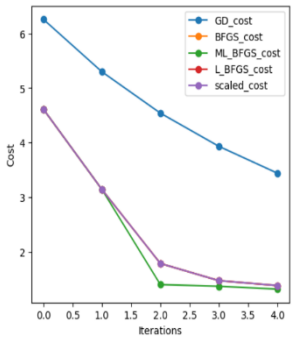
- Do these promotions benefit the merchant or the customer?
- Do customers find their experiences enjoyable?
- Do merchants observe an increase in repeat customers?

Given the importance of personalization, the objective of the dataset is to analyze customer behavior and assess whether such promotions influence loyalty or recurring purchases [16].

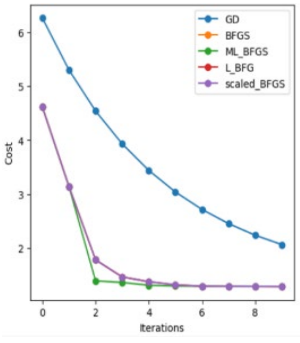
When the number of iterations is set to 5, it is observed that Gradient Descent (GD) fails to converge, whereas BFGS and its modified variants demonstrate greater stability and faster convergence. As illustrated in Figure 1(a), the cost function computed by GD remains high, while BFGS and its variants maintain lower and more stable cost values.

TABLE. IV. EXPERIMENT 1: Elo Merchant Category

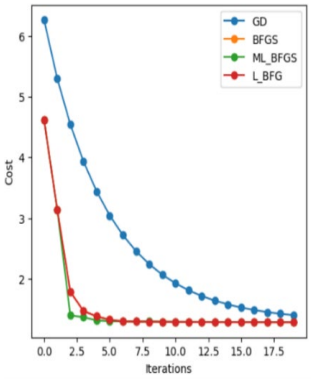
Model	score	MSR	Time
Five iterations			
GD	0.533	6.63423	00.189930
BFGS	0.808	2.72375	00.881986
ML_BFGS	0.817	2.60933	01.053210
L_BFGS	0.808	2.72375	01.025684
S_BFGS	0.808	2.72375	01.113555
Ten iterations			
GD	0.71	4.03207	00.752
BFGS	0.82	2.72375	01.167
ML_BFGS	0.82	2.5655	02.510
L_BFGS	0.81	2.5614	02.33
S_BFGS	0.81	2.5614	02.54
Twenty iterations			
GD	0.805	2.77	01.46
BFGS	0.82	2.54	03.748
ML_BFGS	0.82	2.55	05.16
L_BFGS	0.82	2.54	04.53
S_BFGS	0.82	2.54	04.31
Thirty iterations			
GD	0.82	2.60	03.49
BFGS	0.83	2.54	08.00



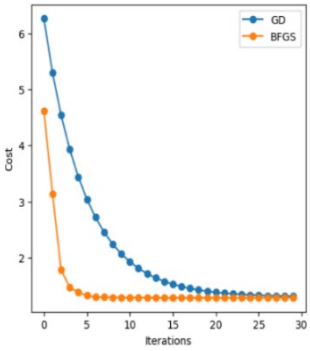
a) 5 iter



a) 5 iter



a) 5 iter



a) 5 iter

Fig 1 EXPERIMENT 1: Elo Merchant Category Recommendation

Increasing the number of iterations to 10 still does not yield acceptable results for GD. As shown in Figure 1(b), the cost function from GD remains high and unstable, while BFGS and its modified methods continue to exhibit robustness and stability, achieving satisfactory cost values. At 20 iterations, Figure 1(c) shows that GD begins to converge, yet the results are still not stable. In contrast, BFGS-based methods remain consistently effective.

To further explore the iteration threshold at which GD becomes comparable in stability to BFGS, the number of iterations increased to 30, as depicted in Figure 1(d). While all BFGS methods—standard BFGS, Memoryless BFGS (ML-BFGS), Limited-memory BFGS (L-BFGS), and Scaled BFGS—produce satisfactory and stable results, ML-BFGS achieves the lowest Mean Squared Error (MSE) and demonstrates superior performance.

TABLE. V. Different tolerance

Model	score	MSR	Time
tolerance= 0.001			
GD	2.577	36	05.45
BFGS	2.556	11	02.78
ML_BFGS	2.563	11	02.87
L_BFGS	2.556	11	02.48
S_BFGS	2.556	11	02.52
tolerance= $10 * e^{-5}$			
GD	2.566	49	9.79
BFGS	2.54	35	05.35
ML_BFGS	2.563	32	07.74
L_BFGS	2.54	35	07.44
S_BFGS	2.54	36	06.40

The compared models were updated to terminate early when the cost function falls below a predefined tolerance threshold (tolerance = 0.001). This modification highlights the faster convergence of BFGS and its modified methods in comparison to Gradient Descent (GD). As demonstrated in Table V and Figure 2, the BFGS-based methods reach the stopping criterion more quickly, thereby reducing computational time and demonstrating greater efficiency than GD.

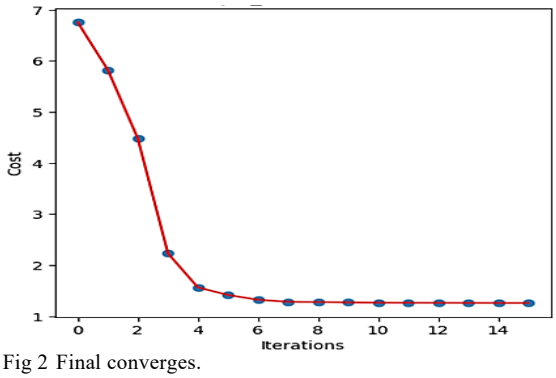


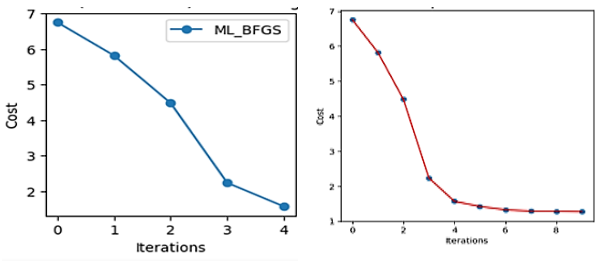
Fig 2 Final converges.

B. EXPERIMENT 2:

The 'GreyLivingstone' notebook was utilized for feature extraction, and its output was employed to evaluate the proposed models[18] . The resulting dataset consists of 770 features and 201,917 records. This experiment represents the second phase of our evaluation and was conducted using Google Colab with a T4 GPU environment to ensure efficient processing and execution.

TABLE. VI. EXPERIMENT 2: Th 'GreyLivingstone'

Model	score	MSR	Time
five iterations			
BFGS	0.78	3.09	17.50
ML_BFGS	0.78	3.09	16.48
L_BFGS	0.78	3.09	17.16
S_BFGS	0.78	3.09	19.09
Ten iterations			
BFGS	0.82	2.52	33.67
ML_BFGS	0.82	2.52	34.5
L_BFGS	0.82	2.52	36.08
S_BFGS	0.82	2.52	36.95
Twenty iterations			
BFGS	0.82	2.50	01:06.60
ML_BFGS	0.82	2.50	01:05.87
L_BFGS	0.82	2.50	01:12.43
S_BFGS	0.82	2.50	01:12.39



a) Five iter b) Ten iter

Fig 3 EXPERIMENT 2: The 'GreyLivingstone'

We updated the compared models to include an early stopping criterion, where training halts if the cost function drops below a specified tolerance value (tolerance = 0.001). This enhancement demonstrates that the BFGS method and its modified variants achieve convergence faster than other regression approaches, with all BFGS-based models successfully converging within sixteen iterations. This significantly reduces computational time, as illustrated in Table VII and Figure 4.

TABLE. VII. Different regressors

Model	Train accuracy	Test accuracy	MSR	Time
BFGS	0.83	0.82	2.50	30.08
ML_BFGS	0.83	0.82	2.50	29.51
Linear Regression	0.83	0.82	2.50	11.35
Ridge Regression	0.83	0.82	2.50	01.93
Lasso Regression	0.74	0.73	3.83	01.33
Elastic Net	0.65	0.64	4.98	01.39
Decision Tree	1.00	0.64	5.08	02:58.97
K-Nearest	0.61	0.37	8.89	00.75
Neural Network (MLP)	0.84	0.82	2.50	01:48.72

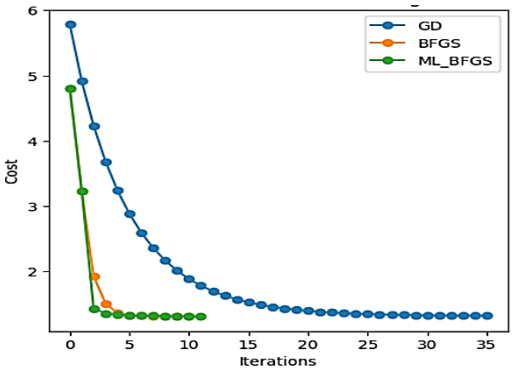


Fig 4 final converge (ML_BFGS)

V. CONCLUSION

In regression models, Gradient Descent (GD) is a traditional optimization technique; however, it often suffers from slow convergence and instability. To address these issues, second-order methods such as Quasi-Newton algorithms are utilized. Among them, the BFGS method is known for its fast convergence on classical problems. Additionally, several modified versions of BFGS have been developed to improve performance, particularly in large-scale settings. These include the Memoryless BFGS (ML-BFGS), Limited-Memory BFGS (L-BFGS), and Scaled BFGS. Among these, ML-BFGS demonstrated the best overall performance, offering a balance of accuracy and efficiency, and proved to be the fastest in convergence.

Refferences

[1] P. WOLFE, "[CONVERGENCE CONDITIONS FOR ASCENT METHODS]," in SIAM REVIEW vol. 11, ed, 1969, pp. 226-&.

[2] P. WOLFE, "[CONVERGENCE CONDITIONS FOR ASCENT METHODS .2. SOME CORRECTIONS]," in SIAM REVIEW vol. 13, ed, 1971, pp. 185-&.

[3] C. N. L. Eu, "Numerical Analysis in Nonlinear Least Squares Methods and Applications," 2017.

[4] B. S. Goh, "Greatest descent algorithms in unconstrained optimization," Journal of optimization theory and applications, vol. 142, pp. 275-289, 2009.

[5] M. Al-Baali, E. Spedicato, and F. Maggioni, "[Broyden's quasi-Newton methods for a nonlinear system of equations and unconstrained optimization: a review and open problems]," in OPTIMIZATION METHODS & SOFTWARE vol. 29, ed, 2014, pp. 937-954.

[6] J. H. Runnoe, "Quasi-Newton methods for unconstrained optimization," 2020.

[7] N. Andrei, "[A note on memory-less SR1 and memory-less BFGS methods for large-scale unconstrained optimization]," in NUMERICAL ALGORITHMS vol. 90, ed, 2022, pp. 223-240.

[8] S. Nakayama, Y. Narushima, and H. Yabe, "A memoryless symmetric rank-one method with sufficient descent property for unconstrained optimization," Journal of the Operations Research Society of Japan, vol. 61, no. 1, pp. 53-70, 2018.

[9] N. Andrei, "[An adaptive scaled BFGS method for unconstrained optimization]," in NUMERICAL ALGORITHMS vol. 77, ed, 2018, pp. 413-432.

[10] N. Andrei, "[A double parameter self-scaling memoryless BFGS method for unconstrained optimization]," in COMPUTATIONAL & APPLIED MATHEMATICS vol. 39, ed, 2020.

[11] D. Saputro and P. Widyaningsih, "[Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) Method for The Parameter Estimation on Geographically Weighted Ordinal Logistic Regression Model (GWOLR)]," in 4TH INTERNATIONAL CONFERENCE ON RESEARCH, IMPLEMENTATION, AND EDUCATION OF MATHEMATICS AND SCIENCES (ICRIEMS): RESEARCH AND EDUCATION FOR DEVELOPING SCIENTIFIC ATTITUDE IN SCIENCES AND MATHEMATICS vol. 1868, ed, 2017.

[12] H. Liu, Y. Li, and M. Zhang, "[An Active Set Limited Memory BFGS Algorithm for Machine Learning]," in SYMMETRY-BASEL vol. 14, ed, 2022.

[13] H. Tankaria, S. Sugimoto, and N. Yamashita, "A regularized limited memory BFGS method for large-scale unconstrained optimization and its efficient implementations," Computational Optimization and Applications, vol. 82, no. 1, pp. 61-88, 2022.

[14] G. Yuan, Z. Wei, and S. Lu, "[Limited memory BFGS method with backtracking for symmetric nonlinear equations]," in MATHEMATICAL AND COMPUTER MODELLING vol. 54, ed, 2011, pp. 367-377.

[15] Y. Hao and V. Simoncini, "[The Sherman-Morrison-Woodbury formula for generalized linear matrix equations and applications]," in NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS vol. 28, ed, 2021.

[16] B. E. Addison Howard, Phil Culliton. Elo Merchant Category Recommendation [Online] Available: https://kaggle.com/competitions/elo-merchant-category-recommendation

[17] GreyLivingstone, "Elo_Merchant_Category_Recommendation," ed, 2023.

[18] kaggle kernels output greylivingstone/elo-merchant-category-recommendation -p /path/to/dest