Optimization Techniques for Multi-Dimensional Linear Least-Squares Regression

Guilherme Marcello
Graduate School of AI
Pohang University of Science and Technology
77 Cheongam-ro, Pohang-si
`marcello@postech.ac.kr`

Abstract

This report presents a comparative analysis of optimization techniques for multi-dimensional linear least-squares regression. We implement and evaluate four algorithms: Steepest Descent, Stochastic Gradient Descent, Newton's Method, and L-BFGS. Their performance is tested on two diverse datasets: the high-dimensional ICVL hand tracking dataset and the low-dimensional 'Student Habits vs Academic Performance' dataset. Our results illuminate the trade-offs between computational efficiency and convergence speed, providing practical guidance on algorithm selection for various regression challenges.

1 Introduction

Fitting models to data is a core challenge in machine learning, often solved through optimization. This report provides a comparative analysis of classical iterative optimization algorithms for multi-dimensional linear least-squares regression. While analytical solutions exist, iterative methods are crucial for large-scale problems and serve as a basis for more complex models.

Our analysis tests the algorithms on two diverse datasets to evaluate their versatility. The first is the high-dimensional ICVL hand tracking dataset, a computer vision task. The second is the lower-dimensional 'Student Habits vs Academic Performance' dataset [1], which assesses generalizability. This study aims to illuminate the practical trade-offs between convergence speed, computational cost, and complexity to guide the selection of appropriate optimization strategies.

2 Theoretical Foundation

This section lays out the mathematical framework for the multi-dimensional linear least-squares regression problem and the optimization algorithms used to solve it.

2.1 Problem Formulation

We are given a training dataset of N instances $\{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^{N}$, where each input feature vector $\boldsymbol{x}_j \in \mathbb{R}^n$ and its corresponding output target vector $\boldsymbol{y}_j \in \mathbb{R}^m$. Our goal is to find a linear regression function $f : \mathbb{R}^n \to \mathbb{R}^m$ of the form:

$$f(\boldsymbol{x}_j) = \boldsymbol{W}^T \boldsymbol{x}_j \tag{1}$$

[1]Available at `https://www.kaggle.com/datasets/jayaantanaath/student-habits-vs-academic-performance/data`

where $\boldsymbol{W} \in \mathbb{R}^{n\times m}$ is the matrix of parameters we want to determine. The objective is to minimize the sum of squared errors (or squared L_2 norm of the residuals):

$$\mathcal{E}'(\boldsymbol{W}) = \sum_{j=1}^{N} ||f(\boldsymbol{x}_j) - \boldsymbol{y}_j||_2^2 = \sum_{j=1}^{N} ||\boldsymbol{W}^T\boldsymbol{x}_j - \boldsymbol{y}_j||_2^2 \tag{2}$$

Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{n\times N}$ be the matrix of input features and $\boldsymbol{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N] \in \mathbb{R}^{m\times N}$ be the matrix of target outputs. The objective function, as given in Eq. (2), can be expressed in matrix form using the Frobenius norm $||\cdot||_F$:

$$\mathcal{E}'(\boldsymbol{W}) = ||\boldsymbol{W}^T\boldsymbol{X} - \boldsymbol{Y}||_F^2 \tag{3}$$

2.2 Gradient of the Objective Function

The gradient of Eq. (3) with respect to $\boldsymbol{W}$ is given by:

$$\nabla_{\boldsymbol{W}}\mathcal{E}'(\boldsymbol{W}) = 2\boldsymbol{X}(\boldsymbol{W}^T\boldsymbol{X} - \boldsymbol{Y})^T \tag{4}$$

This gradient has the same dimensions as $\boldsymbol{W}$, namely $\mathbb{R}^{n\times m}$.

2.3 Hessian Matrix (for Newton's Method)

For Newton's method, we require the Hessian matrix of the objective function $\mathcal{E}'(\boldsymbol{W})$. To express this as a standard matrix, we first vectorize the parameter matrix $\boldsymbol{W} \in \mathbb{R}^{n\times m}$ into a vector $\boldsymbol{w} = \text{vec}(\boldsymbol{W}) \in \mathbb{R}^{nm}$. The operator $\text{vec}(\cdot)$ stacks the columns of a matrix into a single column vector. The gradient with respect to this vectorized $\boldsymbol{w}$ is $\nabla_{\boldsymbol{w}}\mathcal{E}'(\boldsymbol{w}) = \text{vec}(\nabla_{\boldsymbol{W}}\mathcal{E}'(\boldsymbol{W}))$.

Using the gradient from Eq. (4): $\nabla_{\boldsymbol{W}}\mathcal{E}'(\boldsymbol{W}) = 2(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W} - \boldsymbol{X}\boldsymbol{Y}^T)$. Vectorizing this expression, we get: $\nabla_{\boldsymbol{w}}\mathcal{E}'(\boldsymbol{w}) = \text{vec}(2\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W} - 2\boldsymbol{X}\boldsymbol{Y}^T) = 2\text{vec}((\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{W}) - 2\text{vec}(\boldsymbol{X}\boldsymbol{Y}^T)$. Using the identity $\text{vec}(\boldsymbol{ABC}) = (\boldsymbol{C}^T \otimes \boldsymbol{A})\text{vec}(\boldsymbol{B})$, we set $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{X}^T$, $\boldsymbol{B} = \boldsymbol{W}$, and $\boldsymbol{C} = \boldsymbol{I}_m$ (the $m \times m$ identity matrix). This yields: $\text{vec}((\boldsymbol{X}\boldsymbol{X}^T)\boldsymbol{W}) = (\boldsymbol{I}_m^T \otimes (\boldsymbol{X}\boldsymbol{X}^T))\text{vec}(\boldsymbol{W}) = (\boldsymbol{I}_m \otimes (\boldsymbol{X}\boldsymbol{X}^T))\boldsymbol{w}$. So, the vectorized gradient is:

$$\nabla_{\boldsymbol{w}}\mathcal{E}'(\boldsymbol{w}) = 2(\boldsymbol{I}_m \otimes (\boldsymbol{X}\boldsymbol{X}^T))\boldsymbol{w} - 2\text{vec}(\boldsymbol{X}\boldsymbol{Y}^T) \tag{5}$$

The Hessian matrix $\boldsymbol{H}_{\boldsymbol{w}} \in \mathbb{R}^{nm\times nm}$ is the derivative of $\nabla_{\boldsymbol{w}}\mathcal{E}'(\boldsymbol{w})$ with respect to $\boldsymbol{w}^T$:

$$\boldsymbol{H}_{\boldsymbol{w}} = \frac{\partial(\nabla_{\boldsymbol{w}}\mathcal{E}'(\boldsymbol{w}))}{\partial \boldsymbol{w}^T} = 2(\boldsymbol{I}_m \otimes (\boldsymbol{X}\boldsymbol{X}^T)) \tag{6}$$

The matrix $\boldsymbol{X}\boldsymbol{X}^T = \sum_{j=1}^{N} \boldsymbol{x}_j\boldsymbol{x}_j^T$ is positive semi-definite, and $\boldsymbol{I}_m$ is positive definite. The Kronecker product $(\boldsymbol{I}_m \otimes (\boldsymbol{X}\boldsymbol{X}^T))$ is therefore positive semi-definite. Thus, $\boldsymbol{H}_{\boldsymbol{w}}$ is positive semi-definite, which is expected for a convex least squares problem.

2.4 Jacobian of Residuals (for Gauss-Newton Method)

The Gauss-Newton method is an iterative algorithm often used to solve non-linear least squares problems. It approximates the Hessian matrix using the Jacobian of the residual functions. In our linear regression case, the residuals are linear with respect to $\boldsymbol{W}$.

Let the residual for the j-th data instance be $\boldsymbol{r}_j(\boldsymbol{W}) \in \mathbb{R}^m$:

$$\boldsymbol{r}_j(\boldsymbol{W}) = \boldsymbol{W}^T\boldsymbol{x}_j - \boldsymbol{y}_j \tag{7}$$

As in the Hessian derivation, we vectorize the parameter matrix $\boldsymbol{W} \in \mathbb{R}^{n\times m}$ into $\boldsymbol{w} = \text{vec}(\boldsymbol{W}) \in \mathbb{R}^{nm}$. The k-th component of $\boldsymbol{r}_j(\boldsymbol{W})$ is $(\boldsymbol{r}_j(\boldsymbol{W}))_k = (\boldsymbol{w}^{(k)})^T\boldsymbol{x}_j - (\boldsymbol{y}_j)_k$, where $\boldsymbol{w}^{(k)}$ is the k-th column of $\boldsymbol{W}$ (i.e., $\boldsymbol{W} = [\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(m)}]$).

The Jacobian of $\boldsymbol{r}_j(\boldsymbol{W})$ with respect to $\boldsymbol{w}^T$, denoted $\boldsymbol{J}_{\boldsymbol{r}_j}(\boldsymbol{w}) \in \mathbb{R}^{m\times nm}$, is given by the matrix calculus identity $\frac{\partial(\boldsymbol{A}^T\boldsymbol{x})}{\partial \text{vec}(\boldsymbol{A})^T} = \boldsymbol{I}_p \otimes \boldsymbol{x}^T$ (where $\boldsymbol{A}$ is $q \times p$). In our case, $\boldsymbol{A}$ corresponds to $\boldsymbol{W}$ ($n \times m$), $\boldsymbol{x}$ to $\boldsymbol{x}_j$ ($n \times 1$), and p to m. Thus:

$$\boldsymbol{J}_{\boldsymbol{r}_j}(\boldsymbol{w}) = \frac{\partial \boldsymbol{r}_j(\boldsymbol{W})}{\partial \boldsymbol{w}^T} = \boldsymbol{I}_m \otimes \boldsymbol{x}_j^T \tag{8}$$

Here, $\boldsymbol{I}_m$ is the $m \times m$ identity matrix, and $\boldsymbol{x}_j^T$ is the transpose of the j-th input feature vector (a $1 \times n$ row vector). The resulting Jacobian $\boldsymbol{J}_{\boldsymbol{r}_j}(\boldsymbol{w})$ has dimensions $m \times nm$.

For the Gauss-Newton method, we typically stack all N residual vectors into a single large residual vector $\mathcal{R}(\boldsymbol{W}) \in \mathbb{R}^{Nm}$:

$$\mathcal{R}(\boldsymbol{W}) = \begin{pmatrix} \boldsymbol{r}_1(\boldsymbol{W}) \\ \boldsymbol{r}_2(\boldsymbol{W}) \\ \vdots \\ \boldsymbol{r}_N(\boldsymbol{W}) \end{pmatrix} \tag{9}$$

The Jacobian of this stacked residual vector, $\boldsymbol{J}_{\mathcal{R}}(\boldsymbol{w}) \in \mathbb{R}^{Nm \times nm}$, is then formed by vertically stacking the individual Jacobians:

$$\boldsymbol{J}_{\mathcal{R}}(\boldsymbol{w}) = \frac{\partial \mathcal{R}(\boldsymbol{W})}{\partial \boldsymbol{w}^T} = \begin{pmatrix} \boldsymbol{J}_{\boldsymbol{r}_1}(\boldsymbol{w}) \\ \boldsymbol{J}_{\boldsymbol{r}_2}(\boldsymbol{w}) \\ \vdots \\ \boldsymbol{J}_{\boldsymbol{r}_N}(\boldsymbol{w}) \end{pmatrix} = \begin{pmatrix} \boldsymbol{I}_m \otimes \boldsymbol{x}_1^T \\ \boldsymbol{I}_m \otimes \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{I}_m \otimes \boldsymbol{x}_N^T \end{pmatrix} \tag{10}$$

This Jacobian $\boldsymbol{J}_{\mathcal{R}}(\boldsymbol{w})$ is used in the Gauss-Newton update rule, where the Hessian is approximated as $2\boldsymbol{J}_{\mathcal{R}}(\boldsymbol{w})^T \boldsymbol{J}_{\mathcal{R}}(\boldsymbol{w})$. For this linear least squares problem, this approximation $2(\sum_{j=1}^N (\boldsymbol{I}_m \otimes \boldsymbol{x}_j)(\boldsymbol{I}_m \otimes \boldsymbol{x}_j^T)) = 2\sum_{j=1}^N (\boldsymbol{I}_m \otimes (\boldsymbol{x}_j \boldsymbol{x}_j^T)) = 2(\boldsymbol{I}_m \otimes (\boldsymbol{X}\boldsymbol{X}^T))$ is identical to the true Hessian $\boldsymbol{H}_{\boldsymbol{w}}$ given in Eq. (6). This occurs because the second-order derivatives of the residuals (which are omitted in the Gauss-Newton approximation) are zero for linear residuals.

2.5 Hessian Matrix (for Newton's Method)

Newton's method requires the Hessian matrix. Since $\boldsymbol{W}$ is a matrix, the Hessian is technically a 4th-order tensor. However, the problem can be vectorized. Let $\boldsymbol{w}_{vec} = \text{vec}(\boldsymbol{W}) \in \mathbb{R}^{nm \times 1}$, where $\text{vec}(\cdot)$ stacks the columns of a matrix into a single vector. The objective function $\mathcal{E}'(\boldsymbol{W})$ from Eq. (2) can be seen as m independent least-squares problems, one for each column of $\boldsymbol{W}$ (i.e., for each output dimension). Let $\boldsymbol{w}^{(s)} = \boldsymbol{W}_{:,s}$ be the s-th column of $\boldsymbol{W}$, and let $\boldsymbol{y}_j^{(s)}$ be the s-th component of $\boldsymbol{y}_j$. Then the objective for the s-th output dimension is $\mathcal{E}'_s(\boldsymbol{w}^{(s)}) = \sum_{j=1}^N ((\boldsymbol{w}^{(s)})^T \boldsymbol{x}_j - \boldsymbol{y}_j^{(s)})^2 = ||\boldsymbol{X}^T \boldsymbol{w}^{(s)} - \boldsymbol{Y}_{s,:}^T||_2^2$, where $\boldsymbol{Y}_{s,:}$ is the s-th row of $\boldsymbol{Y}$. The gradient for this single-output problem is $\nabla_{\boldsymbol{w}^{(s)}} \mathcal{E}'_s(\boldsymbol{w}^{(s)}) = 2\boldsymbol{X}(\boldsymbol{X}^T \boldsymbol{w}^{(s)} - \boldsymbol{Y}_{s,:}^T)$. The Hessian for $\mathcal{E}'_s(\boldsymbol{w}^{(s)})$ with respect to $\boldsymbol{w}^{(s)} \in \mathbb{R}^n$ is $\boldsymbol{H}_s = 2\boldsymbol{X}\boldsymbol{X}^T \in \mathbb{R}^{n \times n}$. Since the total objective $\mathcal{E}'(\boldsymbol{W}) = \sum_{s=1}^m \mathcal{E}'_s(\boldsymbol{W}_{:,s})$, the Hessian of $\mathcal{E}'(\boldsymbol{W})$ with respect to $\boldsymbol{w}_{vec}$ is a block-diagonal matrix:

$$\boldsymbol{H}_{w_{vec}} = \text{diag}(\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_m) = \boldsymbol{I}_m \otimes (2\boldsymbol{X}\boldsymbol{X}^T) \tag{11}$$

where $\otimes$ denotes the Kronecker product, and $\boldsymbol{I}_m$ is the $m \times m$ identity matrix. The size of $\boldsymbol{H}_{w_{vec}}$ is $(nm \times nm)$.

2.6 Jacobian of Residuals (for Gauss-Newton Method)

The Gauss-Newton method is applied to non-linear least-squares problems of the form $\min \frac{1}{2} \sum_k r_k(\boldsymbol{\theta})^2$. Our objective, as defined in Eq. (2), is $\mathcal{E}'(\boldsymbol{W}) = \sum_{j=1}^N ||\boldsymbol{r}_j(\boldsymbol{W})||_2^2$, where the residual vectors are $\boldsymbol{r}_j(\boldsymbol{W}) = \boldsymbol{W}^T \boldsymbol{x}_j - \boldsymbol{y}_j \in \mathbb{R}^m$. Let the overall matrix of residuals be $\boldsymbol{R}(\boldsymbol{W}) = \boldsymbol{W}^T \boldsymbol{X} - \boldsymbol{Y} \in \mathbb{R}^{m \times N}$. Let $\boldsymbol{r}_{vec} = \text{vec}(\boldsymbol{R}(\boldsymbol{W})) \in \mathbb{R}^{mN \times 1}$ be the vectorized residuals, stacking columns of $\boldsymbol{R}$. The parameters are $\boldsymbol{w}_{vec} = \text{vec}(\boldsymbol{W}) \in \mathbb{R}^{nm \times 1}$. The Jacobian of $\boldsymbol{r}_{vec}$ with respect to $\boldsymbol{w}_{vec}^T$ is an $mN \times nm$ matrix, denoted $\boldsymbol{J}_{\boldsymbol{r}_{vec}}$. Using standard results for derivatives of vectorized matrix products:

$$\boldsymbol{J}_{\boldsymbol{r}_{vec}} = \frac{\partial \text{vec}(\boldsymbol{W}^T \boldsymbol{X} - \boldsymbol{Y})}{\partial (\text{vec}(\boldsymbol{W}))^T} = \boldsymbol{X}^T \otimes \boldsymbol{I}_m \tag{12}$$

This matrix has dimensions $(mN \times nm)$.

2.7 Iterative Optimization Algorithms

The general update rule for the parameter matrix $\boldsymbol{W}$ at iteration k is:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + \alpha_k \boldsymbol{A}_k \tag{13}$$

where α_k is the step size (determined by backtracking line search) and $\boldsymbol{A}_k \in \mathbb{R}^{n \times m}$ is the update direction matrix.

2.7.1 Steepest Descent

The update direction is the negative gradient (from Eq. (4)):

$$\boldsymbol{A}_k^{SD} = -\nabla_{\boldsymbol{W}} \mathcal{E}'(\boldsymbol{W}_k) = -2(\boldsymbol{X}\boldsymbol{X}^T \boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T) \tag{14}$$

So the update rule is:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - 2\alpha_k(\boldsymbol{X}\boldsymbol{X}^T \boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T) \tag{15}$$

2.7.2 Newton's Method

The update for the vectorized parameters $\boldsymbol{w}_{vec}$ is $\text{vec}(\Delta \boldsymbol{W}_k) = -\boldsymbol{H}_{w_{vec},k}^{-1} \text{vec}(\nabla_{\boldsymbol{W}} \mathcal{E}'(\boldsymbol{W}_k))$. Using $\boldsymbol{H}_{w_{vec},k} = \boldsymbol{I}_m \otimes (2\boldsymbol{X}\boldsymbol{X}^T)$ from Eq. (11), its inverse is:

$$\boldsymbol{H}_{w_{vec},k}^{-1} = (\boldsymbol{I}_m \otimes (2\boldsymbol{X}\boldsymbol{X}^T))^{-1} = \boldsymbol{I}_m \otimes (\frac{1}{2}(\boldsymbol{X}\boldsymbol{X}^T)^{-1}) \tag{16}$$

So, $\text{vec}(\Delta \boldsymbol{W}_k) = -(\boldsymbol{I}_m \otimes (\frac{1}{2}(\boldsymbol{X}\boldsymbol{X}^T)^{-1}))\text{vec}(2(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T))$. This simplifies to $\text{vec}(\Delta \boldsymbol{W}_k) = -(\boldsymbol{I}_m \otimes ((\boldsymbol{X}\boldsymbol{X}^T)^{-1}))\text{vec}((\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T))$. Using the property $\text{vec}(\boldsymbol{P}\boldsymbol{Q}\boldsymbol{R}) = (\boldsymbol{R}^T \otimes \boldsymbol{P})\text{vec}(\boldsymbol{Q})$, if we let $\boldsymbol{P} = (\boldsymbol{X}\boldsymbol{X}^T)^{-1}$, $\boldsymbol{Q} = (\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T)$, and $\boldsymbol{R} = \boldsymbol{I}_m$, then: $\text{vec}((\boldsymbol{X}\boldsymbol{X}^T)^{-1}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T)\boldsymbol{I}_m) = (\boldsymbol{I}_m \otimes ((\boldsymbol{X}\boldsymbol{X}^T)^{-1}))\text{vec}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T)$. Thus, the update direction in matrix form is $\Delta \boldsymbol{W}_k$:

$$\boldsymbol{A}_k^{Newton} = \Delta \boldsymbol{W}_k = -(\boldsymbol{X}\boldsymbol{X}^T)^{-1}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T) \tag{17}$$

The update rule (Eq. (13)) becomes:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \alpha_k(\boldsymbol{X}\boldsymbol{X}^T)^{-1}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T) \tag{18}$$

Note that for this method, $(\boldsymbol{X}\boldsymbol{X}^T)$ must be invertible. For its implementation, a regularizer should be added.

2.7.3 Gauss-Newton Method

For a linear least-squares problem, the objective function $\mathcal{E}'(\boldsymbol{W})$ is quadratic. The Gauss-Newton method approximates the Hessian of a general sum-of-squares objective $\frac{1}{2}\sum r_i^2$ by $\boldsymbol{J}^T\boldsymbol{J}$. For our objective $\mathcal{E}'(\boldsymbol{W}) = \sum ||\boldsymbol{r}_j(\boldsymbol{W})||_2^2$, the exact Hessian is $2\sum_j(\boldsymbol{J}_j^T\boldsymbol{J}_j + \sum_i r_{ji}\nabla^2 r_{ji})$. Since the residuals $\boldsymbol{r}_j(\boldsymbol{W})$ (defined in 2.6) are linear in $\boldsymbol{W}$, their second derivatives $\nabla^2 r_{ji}$ are zero. Thus, the Gauss-Newton approximation of the Hessian, $2\boldsymbol{J}_{\text{eff}}^T\boldsymbol{J}_{\text{eff}}$, becomes identical to the true Hessian of $\mathcal{E}'(\boldsymbol{W})$. Consequently, for linear least-squares problems, the Gauss-Newton method converges to Newton's method. The update direction $\boldsymbol{A}_k^{GN}$ is therefore the same as $\boldsymbol{A}_k^{Newton}$:

$$\boldsymbol{A}_k^{GN} = -(\boldsymbol{X}\boldsymbol{X}^T)^{-1}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T) \tag{19}$$

And the update rule is:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \alpha_k(\boldsymbol{X}\boldsymbol{X}^T)^{-1}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}_k - \boldsymbol{X}\boldsymbol{Y}^T) \tag{20}$$

The practical implementation will be identical to Newton's method.

2.8 Backtracking Line Search

To determine the step size α_k in Eq. (13) at each iteration k, backtracking line search is employed. It starts with an initial estimate for α (e.g., $\alpha_{init} = 1$) and iteratively reduces it by a factor $\tau \in (0, 1)$ (e.g., $\tau = 0.5$) until the Armijo-Goldstein condition is satisfied:

$$\mathcal{E}'(\boldsymbol{W}_k + \alpha\boldsymbol{A}_k) \leq \mathcal{E}'(\boldsymbol{W}_k) + c_1\alpha\text{Tr}((\nabla_{\boldsymbol{W}}\mathcal{E}'(\boldsymbol{W}_k))^T\boldsymbol{A}_k) \tag{21}$$

where $c_1 \in (0, 1)$ is a constant (e.g., $c_1 = 10^{-4}$) and $\boldsymbol{A}_k$ is the current search direction. The term $\text{Tr}((\nabla_{\boldsymbol{W}}\mathcal{E}'(\boldsymbol{W}_k))^T\boldsymbol{A}_k)$ represents the directional derivative $g_k^T p_k$. For descent directions, this term is negative.

3 Experiments and Methodology

3.1 Datasets

Two distinct datasets were used for training and evaluation:

1. **ICVL Hand Tracking Dataset:** A high-dimensional dataset where the task is to predict hand keypoint locations from depth images. It consists of $n = 2048$ features and $m = 63$ targets. The dataset was partitioned into 16,008 training samples and 1,596 testing samples.
2. **Student Habits and Performance:** A low-dimensional dataset exploring the relationship between students' habits and their academic performance. It contains $n = 7$ features and $m = 2$ targets, with 800 samples for training and 200 for testing.

3.2 Optimization Algorithms

We implemented and compared four iterative optimization algorithms against the direct analytical solution:

- **Steepest Descent (SD)**.
- **Stochastic Gradient Descent (SGD):** A variant of gradient descent that computes the gradient on a small, random subset (mini-batch) of the training data (batch size of 128).
- **Newton's Method (NM):** The system was solved efficiently via a pre-computed LU decomposition of XX^T. A small regularization term ($\lambda = 10^{-5}$) was used. It is worth noting that for the linear least-squares problem, the Gauss-Newton method is mathematically equivalent to Newton's method. Thus, a separate implementation was not required.
- **Limited-memory BFGS (L-BFGS):** A quasi-Newton method that approximates the inverse Hessian using a history of the last 10 updates.

3.3 Experimental Setup

All experiments were conducted on a machine with an Apple M1 processor and 16GB of memory. For iterative methods, we ran two variants: one with weights initialized to zeros, and another with random initialization. Each run was subject to a 60-second time limit, a 500-iteration maximum, or a gradient norm tolerance of 10^{-8}. Performance was evaluated by the Mean Squared Error (MSE) on the test set.

4 Results and Discussion

The performance of the optimization algorithms was analyzed based on their convergence speed and the final test set MSE, using the analytical solution as a baseline. The results are available at `https://github.com/guilherme-marcello/linear-regression-optimizer-benchmark`.

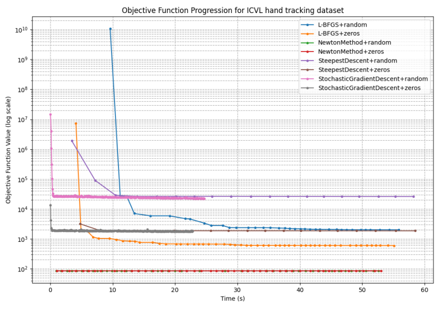
4.1 ICVL Hand Tracking Dataset

On this high-dimensional dataset, the analytical solution provided a strong baseline with an MSE of 0.00088, computed in approximately 1.39 seconds.

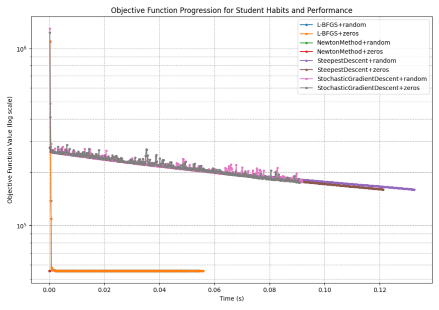
As shown in Table 1 and Figure 1a, Newton's Method came closest to matching the analytical solution's accuracy, albeit at a much higher computational cost. Interestingly, **L-BFGS with zero-initialization achieved an even lower MSE (0.00061)** than the regularized analytical solution, suggesting it found a better minimum. However, its sensitivity to initialization highlights a potential lack of robustness. The first-order methods (SD and SGD) failed to converge to a competitive solution within the time limit.

4.2 Student Habits and Performance Dataset

The performance differences were more dramatic on the low-dimensional dataset. The analytical solution was computed almost instantaneously (less than 0.1ms) and achieved an MSE of 35.077.



(a) ICVL Hand Tracking Dataset



(b) Student Habits and Performance Dataset

Figure 1: Convergence plots for the optimizers on both datasets. The y-axis represents the objective function value on a logarithmic scale, and the x-axis represents the computation time in seconds.

Table 1: Performance on the ICVL Hand Tracking Dataset.

Optimizer	**Initialization**	**Test MSE**	**Time (s)**
Analytical	-	0.00088	**1.39**
Newton's Method	random	0.00096	52.6
	zeros	0.00096	53.0
L-BFGS	zeros	**0.00061**	55.1
	random	0.00262	55.9
Steepest Descent	zeros	0.00191	58.4
	random	0.01837	58.2
SGD	zeros	0.00175	22.8
	random	0.01690	24.6

Table 2: Performance on the Student Habits and Performance Dataset.

Optimizer	**Initialization**	**Test MSE**	**Time (s)**
Analytical	-	**35.077**	**<0.001**
Newton's Method	random	**35.077**	**<0.001**
	zeros	**35.077**	**<0.001**
L-BFGS	random	**35.077**	0.054
	zeros	**35.077**	0.056
Steepest Descent	zeros	95.547	0.121
	random	95.456	0.132
SGD	zeros	104.746	0.091
	random	105.884	0.092

Here, both **Newton's Method and the analytical solution were unequivocally superior**, finding the optimal solution in negligible time (Table 2). L-BFGS also achieved the same MSE but took longer. This demonstrates that for low-dimensional, well-conditioned problems, direct or second-order methods are vastly more efficient. As seen in Figure 1b, first-order methods were again unable to find the optimal solution.

5 Conclusion

The analytical solution serves as a critical performance benchmark. For low-dimensional problems, it is exceptionally fast and accurate. For high-dimensional problems, its computational cost increases, but it remains a strong baseline. Iterative methods become necessary when the direct solution is too costly. Among them, second-order (Newton's) and quasi-Newton (L-BFGS) methods consistently outperform first-order techniques, converging to solutions of similar or even better quality than the regularized analytical solution.