QR factorization and limited quasi Newton methods:

Linear least square problem

Giuseppe Gabriele Russo 583744
Marco Dell'Acqua 686081

June 10, 2025

Contents

1 Introduction **3**

2 Quasi-Newton Methods **4**
2.1 Wolfe and Strong Wolfe Conditions 5
2.2 The Limited-Memory BFGS Method (L-BFGS) 7
2.3 Applicability to the Problem at Hand 9

3 QR Factorization **11**
3.1 Classical QR via Modified Gram-Schmidt 11
3.2 Compact QR Factorization Using Householder Reflectors 11
3.3 Solving Least Squares Problems via Householder QR 12

4 L-BFGS experimental Results **13**
4.1 Effect of the Memory Parameter 13
4.2 Effect of the Regularization Parameter λ 14
4.3 Verification of the Bound K_0 14
4.3.1 Case 1: Moderate Regularization ($\lambda = 1$) 15
4.3.2 Case 2: Low Regularization ($\lambda = 10^{-2}$) 15
4.4 Verification of the Superlinear Convergence Bound 16
4.5 Relative Error Across Regularization Regimes 18

5 QR experimental Results **21**
5.1 Verification of QR Decomposition Stability 21
5.2 Numerical Accuracy of QR Solution via Augmented System Bound 22

6 Comparison: QR vs. L-BFGS **24**

Bibliography **26**

1 Introduction

This project focuses on the numerical solution of a structured least-squares problem, where the objective is to estimate a vector $w \in \mathbb{R}^m$ that minimizes the discrepancy between a data-dependent transformation and an augmented target vector. The problem is expressed as:

$$\min_w \left\| \begin{bmatrix} X^T \\ \lambda I_m \end{bmatrix} w - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|_2,$$

where $X \in \mathbb{R}^{m \times n}$ is a data matrix obtained from the ML-cup dataset, $y \in \mathbb{R}^m$ is a fixed response vector, I_m is the identity matrix of size m, and $\lambda > 0$ is a scalar hyperparameter.

Although the formulation above might not immediately resemble standard regularization, it is in fact mathematically equivalent to the minimization of the expression:

$$\|X^T w - y\|_2^2 + \lambda^2 \|w\|_2^2,$$

which is a well-known instance of Tikhonov regularization, commonly referred to as ridge regression [6, Chapter 3.4]. The regularization term $\lambda^2 \|w\|_2^2$ is added to penalize large coefficients in w, which improves the numerical stability of the solution and helps to prevent overfitting.

Two computational strategies are explored to solve this problem:

- A first-order method that uses limited-memory approximations to second-order curvature, making it suitable for large-scale optimization tasks where memory efficiency is critical.
- A numerically stable algorithm based on QR factorization using Householder reflectors, implemented in a compact form that avoids explicitly forming the orthogonal matrix. The reflectors are instead applied through a series of implicit matrix-vector operations.

The main objective of this work is to design both solvers from the ground up, without relying on pre-built numerical libraries. Special care is taken to study how the performance of the proposed algorithms depends not only on the size of the input data, but also on the regularization parameter λ and the structure of the target vector y. In particular, the quasi-Newton approach may exhibit varying convergence rates depending on the conditioning of the objective function, while the QR-based method maintains a predictable computational cost once the problem structure is fixed.

2 Quasi-Newton Methods

Quasi-Newton methods are iterative algorithms used to find the local minimum of a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. They belong to the class of second-order optimization techniques and aim to approximate the Newton-Raphson method while avoiding the explicit computation of the Hessian matrix $\nabla^2 f(w)$, which may be computationally expensive or infeasible to obtain in large-scale problems [11, Chapter 6, p. 135-136].

Motivation and General Idea

In the classical Newton method, the update rule for minimizing $f(w)$ is given by:

$$w_{k+1} = w_k - \left[\nabla^2 f(w_k)\right]^{-1} \nabla f(w_k)$$

This approach converges rapidly under suitable conditions (e.g., when f is strongly convex and twice continuously differentiable), but computing and inverting the Hessian can be prohibitively expensive, especially when the dimension n is large [11, Section 3.4].

Quasi-Newton methods overcome this limitation by iteratively constructing an approximation B_k to the Hessian (or its inverse H_k) using only gradient information. The core idea is to use *gradient differences* to capture curvature information, thus building up second-order structure incrementally [11, Theorem 3.6].

Update Strategy: Secant Condition

The key requirement for updating the Hessian approximation is that it satisfies the **secant condition**:

$$B_{k+1} s_k = y_k,$$

where:

$$s_k = w_{k+1} - w_k, \quad y_k = \nabla f(w_{k+1}) - \nabla f(w_k)$$

This condition ensures that the updated matrix approximates the curvature of f in the direction s_k. Since s_k and y_k are vectors in $\mathbb{R}^n$, this condition alone does not uniquely determine B_{k+1}, leaving room for multiple update strategies [11, Section 6.1, p. 137].

General Form of the Quasi-Newton Step

At each iteration, the search direction p_k is computed as:

$$p_k = -H_k \nabla f(w_k),$$

where $H_k \approx \left[\nabla^2 f(w_k)\right]^{-1}$ is the current approximation to the inverse Hessian. The step length α_k is typically chosen using a **line search** that satisfies Wolfe or strong Wolfe conditions to ensure sufficient descent and curvature [15] (In the next section we will discuss these conditions in detail).

Then the parameter vector is updated as:

$$w_{k+1} = w_k + \alpha_k p_k$$

2.1 Wolfe and Strong Wolfe Conditions

In quasi-Newton methods, the step length α_k is typically selected via a line search that ensures sufficient descent in the objective function while maintaining numerical stability. To formalize these requirements, the *Wolfe conditions* are commonly used [11, Section 3.1].

Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable, $w_k \in \mathbb{R}^n$ the current iterate, and $p_k \in \mathbb{R}^n$ a descent direction (i.e., $\nabla f(w_k)^T p_k < 0$). A step length $\alpha_k > 0$ is said to satisfy the **Wolfe conditions** if:

$$f(w_k + \alpha_k p_k) \leq f(w_k) + c_1 \alpha_k \nabla f(w_k)^T p_k \tag{W1}$$

$$\nabla f(w_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f(w_k)^T p_k \tag{W2}$$

with constants $0 < c_1 < c_2 < 1$, typically chosen as $c_1 = 10^{-4}$, $c_2 = 0.9$.

Interpretation.

- Condition (W1) ensures that the function value decreases sufficiently — this is known as the *Armijo condition*.
- Condition (W2) ensures that the slope of the function along p_k is not too small — it prevents the algorithm from selecting points too close to a minimizer prematurely.

Strong Wolfe Conditions. An alternative and stricter criterion replaces the curvature condition (W2) with an absolute value bound:

$$f(w_k + \alpha_k p_k) \leq f(w_k) + c_1 \alpha_k \nabla f(w_k)^T p_k \tag{SW1}$$

$$|\nabla f(w_k + \alpha_k p_k)^T p_k| \leq c_2 |\nabla f(w_k)^T p_k| \tag{SW2}$$

These are known as the **strong Wolfe conditions**, and they guarantee a more controlled and symmetric reduction in directional derivatives. They are particularly important in quasi-Newton methods to ensure the satisfaction of the *secant condition* and the positive definiteness of updates [11, Theorem 6.6].

Theoretical Convergence Guarantees

The convergence behavior of quasi-Newton methods has been extensively studied in the optimization literature. While they do not require the explicit computation of the Hessian, under appropriate conditions, they can achieve convergence rates that closely match those of Newton's method.

Global Convergence. Global convergence refers to the property that the sequence $\{w_k\}$ produced by the algorithm converges to a minimizer w^*, regardless of the starting point w_0. For quasi-Newton methods, global convergence is generally not guaranteed without additional mechanisms. However, it can be achieved if the method is combined with a line search that satisfies the *Wolfe* or *strong Wolfe conditions*(2.1), and if the function f satisfies:

- f is bounded below and continuously differentiable,
- the gradient $\nabla f(w)$ is Lipschitz continuous,
- and the search directions p_k are descent directions, i.e., $\nabla f(w_k)^T p_k < 0$.

Under these assumptions, quasi-Newton methods such as BFGS are *globally convergent*, i.e., $\liminf_{k\to\infty} \|\nabla f(w_k)\| = 0$.[11, Theorem 6.5]

Superlinear Convergence. Under stronger assumptions, quasi-Newton methods can achieve **superlinear convergence**, meaning:

$$\lim_{k\to\infty} \frac{\|w_{k+1} - w^*\|}{\|w_k - w^*\|} = 0$$

This rate is faster than linear but slower than quadratic (as in Newton's method). Superlinear convergence is typically guaranteed if:

- f is *twice continuously differentiable* near the solution w^*,
- the inverse Hessian approximations H_k converge to the true inverse Hessian $\nabla^2 f(w^*)^{-1}$,
- and the *line search* yields steps α_k that satisfy the *curvature condition* (i.e., strong Wolfe conditions).

In particular, the BFGS update satisfies these conditions and is known to converge superlinearly on strongly convex problems [11, Theorem 6.6]. DFP also converges superlinearly under similar assumptions, though BFGS is generally preferred in practice due to better numerical stability and robustness.

Comparison to Newton's Method. While Newton's method can achieve *quadratic convergence*, it requires access to the exact Hessian and its inverse. Quasi-Newton methods trade this precision for efficiency, constructing good approximations over time. In many real-world scenarios, especially with noisy or high-dimensional data, the superlinear convergence of BFGS often provides a better balance between speed and cost.

Popular Quasi-Newton Updates

Several schemes for updating B_k or H_k exist, among which the most well-known are:

- **DFP (Davidon-Fletcher-Powell)**: One of the earliest quasi-Newton methods, updating H_k directly [3].
- **BFGS (Broyden-Fletcher-Goldfarb-Shanno)**: Builds H_{k+1} using a rank-two update that maintains symmetry and positive definiteness [2].
- **SR1 (Symmetric Rank-One)**: Simpler update, often used when the secant condition does not enforce positive definiteness [11].

Among these, BFGS and its limited-memory variant L-BFGS are the most widely used in large-scale optimization and machine learning applications, due to their robustness and convergence properties [8].

2.2 The Limited-Memory BFGS Method (L-BFGS)

While the BFGS method is highly effective in medium-scale optimization problems, its applicability to large-scale problems is limited by the storage and computational costs associated with maintaining and updating a full $n \times n$ approximation of the inverse Hessian matrix H_k. The Limited-memory BFGS (L-BFGS) algorithm addresses this limitation by avoiding the explicit construction of H_k, and instead maintains a history of only the most recent updates.

Memory Efficiency. Rather than storing and updating a dense matrix, L-BFGS retains only the last *limit* pairs of vectors (s_k, y_k), where:

$$s_k = w_{k+1} - w_k, \quad y_k = \nabla f(w_{k+1}) - \nabla f(w_k)$$

These vector pairs are used to implicitly define a low-rank approximation to the inverse Hessian, which is applied directly to the gradient vector to compute the search direction. Typically, $limit \in [3, 20]$, making the method extremely memory-efficient and suitable for high-dimensional problems [8].

The full algorithm is accessible in the code shown 1

Algorithm 1 L-BFGS Optimization Algorithm

Require: Initial point w_0, memory size *limit*

Initialize: $k \leftarrow 0$, empty lists for $\{s_i\}$ and $\{y_i\}$
Compute gradient $g_0 = \nabla f(w_0)$
while $\|\nabla f(w_k)\|_2 > \epsilon$ **do**
    Compute search direction $p_k = \texttt{TwoLoopRecursion}(g_k, \{s_i, y_i\})$
    Perform line search to obtain step size α_k
    Update: $w_{k+1} = w_k + \alpha_k p_k$
    Compute gradient $g_{k+1} = \nabla f(w_{k+1})$
    $s_k = w_{k+1} - w_k$
    $y_k = \nabla f(w_{k+1}) - \nabla f(w_k)$
    Store s_k, y_k; discard oldest if $\#\{s_i\} > limit$
    $k \leftarrow k + 1$
end while

Two-Loop Recursion. The core computational component of L-BFGS is the *two-loop recursion*, an efficient algorithm for computing the product $p_k = -H_k \nabla f(w_k)$ without explicitly forming H_k. The algorithm proceeds as reported in 2.

Algorithm 2 TwoLoopRecursion$(g_k, \{s_i, y_i\})$

Require: Gradient $q = g_k$, memory vectors $\{s_i, y_i\}$
Ensure: Search direction $p_k = -H_k g_k$

for $i = k-1, k-2, \ldots, k-limit$ **do**
    $\rho_i \leftarrow \frac{1}{y_i^T s_i}$
    $\alpha_i \leftarrow \rho_i s_i^T q$
    $q \leftarrow q - \alpha_i y_i$
end for
$H_k^0 \leftarrow \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}} I$
$r \leftarrow H_k^0 q$
for $i = k-limit, \ldots, k-1$ **do**
    $\beta_i \leftarrow \rho_i y_i^T r$
    $r \leftarrow r + s_i(\alpha_i - \beta_i)$
end for
return $p_k = -r$

The cost of both algorithms is only $O(mn)$ time and memory, a significant improvement over the $O(n^2)$ cost of full BFGS. The choice of H_k^0 (the initial approximation) affects performance but does not alter convergence guarantees.

Convergence Properties. Despite its reduced memory usage, L-BFGS preserves many of the desirable convergence properties of BFGS. Under the same assumptions (e.g., Lipschitz continuity of the gradient, bounded level sets, and line

search satisfying Wolfe conditions), the method is globally convergent [11]. Furthermore, although it does not store a full Hessian approximation, L-BFGS has been shown to exhibit superlinear convergence in practice for well-conditioned problems.

Comparison with Stochastic Methods. Unlike stochastic optimization algorithms (e.g., SGD or Adam), L-BFGS is a deterministic method and assumes access to the full gradient at each iteration. It is therefore more suitable for problems with moderate data size or where batch gradients are tractable.However, recent variants such as *online L-BFGS* and *stochastic quasi-Newton methods* attempt to bridge the gap between curvature-aware optimization and the scalability of stochastic algorithms. These include approaches like stochastic BFGS [13], regularized updates [9], and memory-efficient stochastic L-BFGS with guaranteed linear convergence [10, 1].

2.3 Applicability to the Problem at Hand

To determine whether the theoretical results on quasi-Newton convergence apply in our case, we analyze the properties of the objective function:

$$f(w) = \left\| \begin{bmatrix} X^T \\ \lambda I_m \end{bmatrix} w - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|_2^2 = \|X^T w - y\|_2^2 + \lambda^2 \|w\|_2^2.$$

This is a strongly convex quadratic function, which satisfies all standard assumptions required for the convergence theory of quasi-Newton methods:

- **Twice differentiable:** f is a quadratic function with constant Hessian $H = \nabla^2 f(w) = 2XX^T + 2\lambda^2 I$, which is symmetric and positive definite when $\lambda > 0$.
- **Lipschitz gradient:** The gradient $\nabla f(w) = 2XX^T w - 2Xy + 2\lambda^2 w$ is Lipschitz continuous with constant $L = \|H\|_2$.
- **Strong convexity:** The function is μ-strongly convex with $\mu = \lambda_{\min}(H) > 0$, ensuring uniqueness and stability of the minimizer.
- **Line search:** Our implementation uses a backtracking line search satisfying the strong Wolfe conditions, as required for theoretical guarantees [11, Theorem 6.6].

Therefore, both global convergence and local superlinear convergence results for BFGS and L-BFGS apply rigorously to our setting.

Convergence Rate and Iteration Complexity Under the aforementioned assumptions—strong convexity, smoothness, constant Hessian, and line search satisfying the strong Wolfe conditions—the L-BFGS method exhibits superlinear convergence behavior.

A first global non-asymptotic analysis was recently proposed by Rodomanov and Nesterov [12, Theorem 4.2], who prove that, for all $k \geq 1$, the iterates of classical quasi-Newton methods satisfy:

$$\lambda_k \leq \left[e^{\frac{d}{k} \ln \frac{L}{\mu}} - 1 \right]^{k/2},$$

where λ_k denotes the local norm of the gradient, d is a problem-dependent constant (commonly proportional to the problem dimension n), and $\kappa = L/\mu$ is the condition number of the Hessian. This estimate holds globally and does not require proximity to the minimizer.

Rodomanov and Nesterov also define an estimate for the starting moment of superlinear convergence, indicating when the curvature approximation becomes sufficiently accurate. This is given by:

$$K_0^{\text{BFGS}} = 4n \ln \left(\frac{L}{\mu} \right).$$

An alternative local non-asymptotic analysis was proposed by Jin and Mokhtari [7, Theorem 1 and Corollary 3, Section 4.3], who prove that, under the additional assumption that the iterates are sufficiently close to the minimizer w^*, the iterates of quasi-Newton methods satisfy:

$$\frac{\|\nabla^2 f(w^*)^{1/2}(w_k - w^*)\|}{\|\nabla^2 f(w^*)^{1/2}(w_0 - w^*)\|} \leq \left(\frac{1}{k} \right)^{k/2}.$$

This expression uses the Hessian-induced norm:

$$\|v\|_H := \sqrt{v^T H v} = \|\nabla^2 f(w^*)^{1/2} v\|_2.$$

Since in our case the Hessian H is constant and positive definite, the Hessian norm and the Euclidean norm are equivalent up to a constant factor:

$$\mu \|v\|_2^2 \leq \|v\|_H^2 \leq L \|v\|_2^2,$$

thus the bound can be expressed as:

$$\|w_k - w^*\|_2 \leq \mathcal{O}\left(\left(\frac{1}{k} \right)^{k/2} \right).$$

To estimate the number of iterations required to achieve an accuracy ϵ, we solve:

$$\left(\frac{1}{k} \right)^{k/2} \leq \epsilon \quad \Longrightarrow \quad \frac{k}{2} \log k \geq \log \left(\frac{1}{\epsilon} \right),$$

leading to the asymptotic estimate:

$$k = \mathcal{O}\left(\frac{\log(1/\epsilon)}{\log \log(1/\epsilon)} \right).$$

This rate significantly improves over the classical linear convergence bound $\mathcal{O}(\log(1/\epsilon))$.

3 QR Factorization

QR factorization is a core algorithm in numerical linear algebra, widely used to solve least squares problems and to construct orthonormal bases [14, 5]. Given a matrix $A \in \mathbb{R}^{m \times n}$ with full column rank and $m \geq n$, the QR decomposition factorizes A as:

$$A = QR,$$

where $Q \in \mathbb{R}^{m \times n}$ has orthonormal columns ($Q^T Q = I_n$) and $R \in \mathbb{R}^{n \times n}$ is upper triangular. This decomposition enables efficient and numerically stable solutions to least squares problems.

3.1 Classical QR via Modified Gram-Schmidt

The Modified Gram-Schmidt (MGS) process improves the numerical stability of the classical Gram-Schmidt algorithm by orthogonalizing one vector at a time and immediately updating the remaining vectors.

Algorithm 3 Modified Gram-Schmidt QR Factorization

Require: Matrix $A \in \mathbb{R}^{m \times n}$
Ensure: Orthonormal matrix $Q \in \mathbb{R}^{m \times n}$, upper triangular matrix $R \in \mathbb{R}^{n \times n}$

for $j = 1$ to n **do**
  $v_j \leftarrow a_j$ ▷ Extract column j of A
  for $i = 1$ to $j - 1$ **do**
    $R_{i,j} \leftarrow q_i^T v_j$
    $v_j \leftarrow v_j - R_{i,j} q_i$
  end for
  $R_{j,j} \leftarrow \|v_j\|_2$
  $q_j \leftarrow v_j / R_{j,j}$
end for
Set $Q = [q_1, \ldots, q_n]$

The computational cost of the algorithm 3 is $O(mn^2)$, and although more stable than classical Gram-Schmidt, it remains sensitive to round-off errors when compared to Householder-based QR [14].

3.2 Compact QR Factorization Using Householder Reflectors

For large-scale problems, QR factorization is more efficiently computed via Householder transformations. Each transformation takes the form:

$$H_k = I - 2\frac{u_k u_k^T}{u_k^T u_k},$$

which zeroes all entries below the diagonal in column k. Rather than constructing the full orthogonal matrix Q, the algorithm 4 stores only the reflectors u_k, leading to a compact and numerically stable representation [5].

Algorithm 4 QR Factorization (Compact Householder Form)

Require: Matrix $A \in \mathbb{R}^{m \times n}$
Ensure: Upper triangular matrix $R \in \mathbb{R}^{m \times n}$, Householder vectors $\{u_k\}_{k=1}^{n}$
  $R \leftarrow A$
  for $k = 1$ to n **do**
    $x \leftarrow R_{k:m,k}$
    $u_k \leftarrow x + \text{sign}(x_1)\|x\|_2 e_1$
    $u_k \leftarrow u_k / \|u_k\|_2$
    $R_{k:m,k:n} \leftarrow R_{k:m,k:n} - 2u_k(u_k^T R_{k:m,k:n})$
  end for

The overall complexity remains $O(mn^2)$, but this version improves stability and reduces memory usage by not forming Q explicitly [4].

3.3 Solving Least Squares Problems via Householder QR

Given an overdetermined system $Ax \approx b$, with $A \in \mathbb{R}^{m \times n}$ and full rank, the goal is to compute:

$$\min_x \|Ax - b\|_2.$$

Using the compact QR factorization $A = QR$, the problem reduces to solving:

$$Rx = Q^T b.$$

This is done in three stages:

1. **QR factorization via Householder reflectors**: costs $O(mn^2)$, assuming $m \geq n$,
2. **Apply reflectors to compute** $y = Q^T b$: requires $O(mn)$,
3. **Solve triangular system** $Rx = y_{1:n}$: requires $O(n^2)$.

Hence, the total cost for solving the least squares problem using compact QR is:

$$\boxed{O(mn^2) + O(mn) + O(n^2)}.$$

In practice, the term $O(mn^2)$ dominates, especially when $m \gg n$. This method avoids forming the full orthogonal matrix Q, reduces numerical error, and is significantly more stable than solving the normal equations $A^T Ax = A^T b$ [5, 4].

4 L-BFGS experimental Results

All configurations were tested using 3 randomly generated response vectors y, and each experiment was repeated 10 times per configuration to account for variance due to randomness and numerical effects. The metrics reported here are the averages across these repetitions.

4.1 Effect of the Memory Parameter

To evaluate the influence of the memory parameter *limit* on the convergence of L-BFGS, we ran experiments across values $limit \in \{10, 20, 40\}$, fixing the tolerance to $\epsilon = 10^{-5}$ and testing a broad range of regularization parameters $\lambda \in \{10^{-4}, 10^{-2}, 1, 10^{2}, 10^{4}\}$.

Each data point in the plot represents the average number of iterations required to meet the stopping criterion, computed across 3 independent draws of y, with 10 runs for each. The results are summarized in Figure 1.

The trends highlight several key observations:

- For small values of λ, corresponding to ill-conditioned problems, increasing the memory size yields a substantial reduction in the number of iterations.
- As λ increases and the condition number improves, the effect of the memory parameter diminishes. The number of iterations stabilizes at a very low level regardless of *limit*.

These results empirically validate the theoretical intuition that the memory parameter enhances convergence speed by better approximating curvature information, especially when the problem is poorly conditioned.

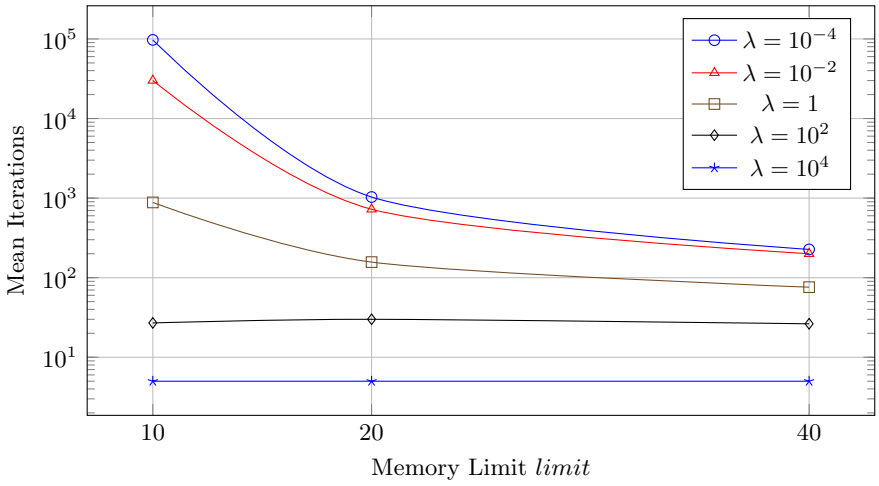

Figure 1: Average number of L-BFGS iterations as a function of the memory parameter *limit*, for different values of λ and fixed tolerance $\epsilon = 10^{-5}$.

4.2 Effect of the Regularization Parameter λ

To further understand how regularization affects convergence, we analyze the number of iterations required to meet the stopping criterion for different values of λ, comparing across memory sizes $limit = 10, 20, 40$. The tolerance is fixed at $\epsilon = 10^{-5}$. Each data point corresponds to the average of 10 runs over 3 different targets y.

As shown in Figure 2, increasing λ improves conditioning and significantly accelerates convergence. Moreover, larger memory budgets consistently reduce the number of iterations, especially in ill-conditioned regimes (low λ).

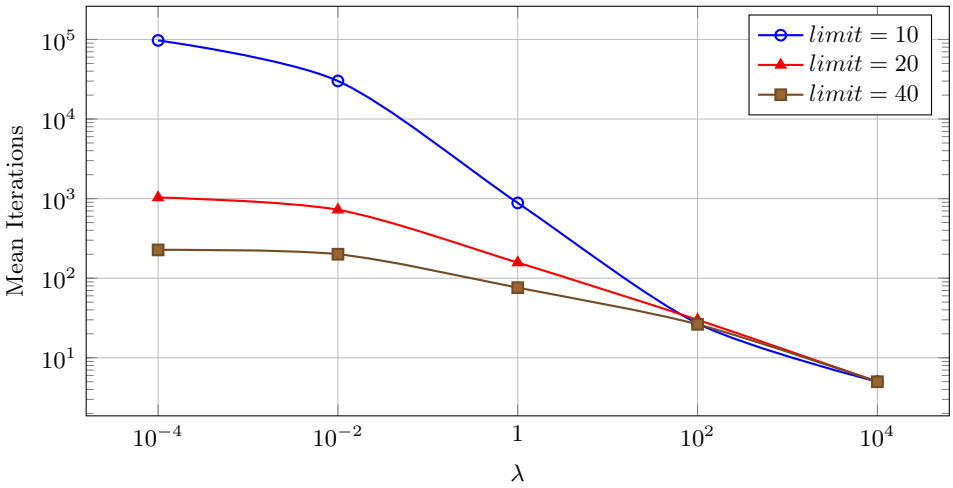

Figure 2: Effect of the regularization parameter λ on the number of L-BFGS iterations, for three different memory sizes and fixed tolerance $\epsilon = 10^{-5}$.

4.3 Verification of the Bound K_0

We empirically examine the theoretical threshold $K_0^{\text{BFGS}} = 4n \ln \kappa$, introduced by Rodomanov and Nesterov [12], which defines the iteration number after which superlinear convergence behavior is theoretically expected to occur in quasi-Newton methods.

This bound is derived in the context of global non-asymptotic convergence and reflects the number of steps required to accumulate sufficient curvature information through secant updates. Its magnitude depends logarithmically on the condition number κ of the Hessian H, which in our case takes the form:

$$H = 2XX^\top + 2\lambda^2 I.$$

We test this theoretical prediction using two experimental configurations: one with moderate regularization ($\lambda = 1$), and one with weak regularization ($\lambda = 10^{-2}$). In both cases, we track the evolution of the gradient norm $\|\nabla f(w_k)\|$

across iterations, and compare the empirical convergence behavior with the value of K_0^{BFGS}.

4.3.1 Case 1: Moderate Regularization ($\lambda = 1$)

For $\lambda = 1$, we consider L-BFGS with memory $limit = 20$ and tolerance $\epsilon = 10^{-5}$. The Hessian is reasonably well-conditioned:

$$\kappa \approx 8.8 \times 10^4 \quad \Rightarrow \quad K_0^{\text{BFGS}} \approx 22770.$$

In Figure 3, we plot the evolution of $\|\nabla f(w_k)\|$ for this configuration. Although the predicted K_0 lies far beyond the practical runtime of the algorithm (the run terminates well before iteration 150), the method already reaches machine-precision-level gradient norms after just a few dozen iterations. This stark contrast demonstrates that while the bound is theoretically sound, it is highly conservative in practice. This behavior confirms the observation made in [7] that these bounds may vastly overestimate the true onset of superlinear behavior.

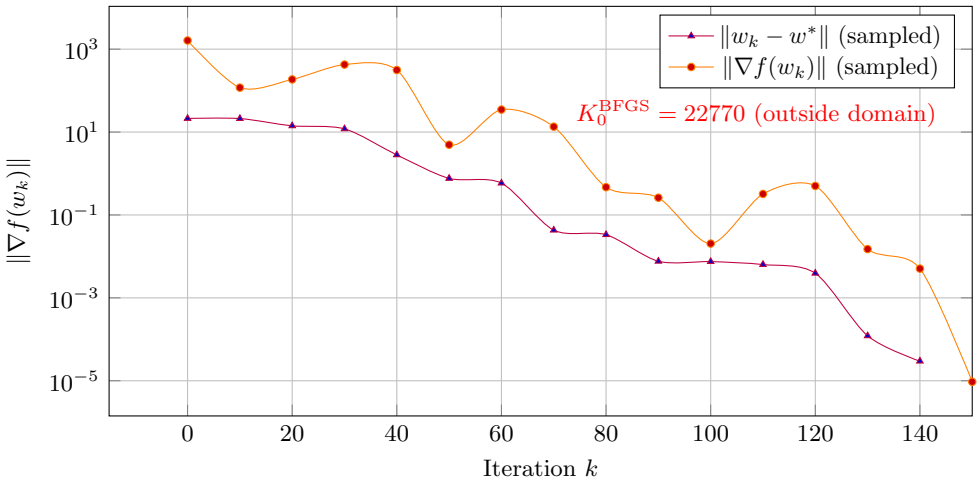

Figure 3: Gradient norm and distance to optimum across iterations for L-BFGS with $\lambda = 1$, $limit = 20$, and $\epsilon = 10^{-5}$.

4.3.2 Case 2: Low Regularization ($\lambda = 10^{-2}$)

We repeat the same analysis for $\lambda = 10^{-2}$, using $limit = 10$. The condition number is now extremely high:

$$\kappa \approx 8.8 \times 10^8 \quad \Rightarrow \quad K_0^{\text{BFGS}} \approx 41191.$$

Although this configuration is more ill-conditioned, it turns out to be more informative. As shown in Figure 4, the gradient norm decays smoothly over

thousands of iterations, and the shape of the curve resembles the predicted superlinear behavior. Although the iteration count remains well below the theoretical K_0, this behavior provides partial validation of the bound's predictive structure (if not its scale), especially in regimes with weak regularization.

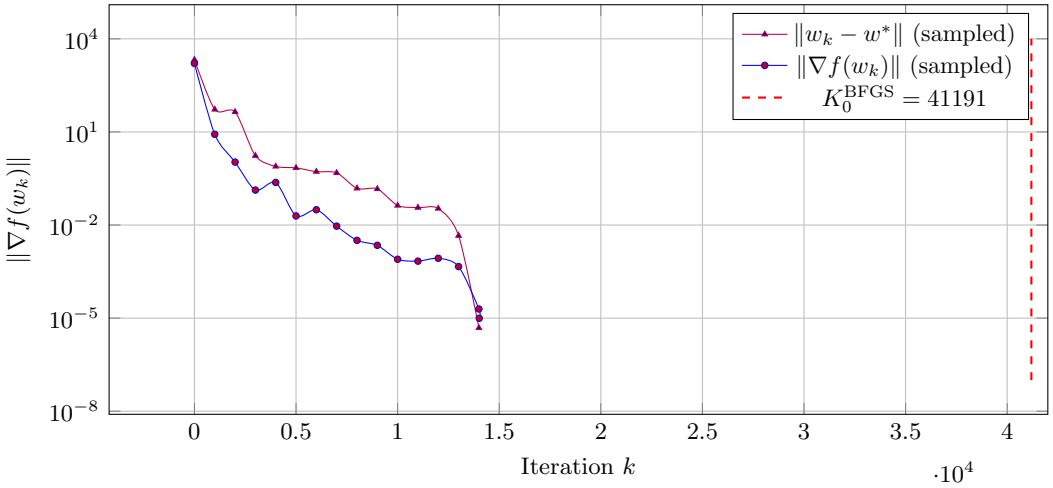

Figure 4: Gradient norm and distance to optimum for L-BFGS with $\lambda = 10^{-2}$, $limit = 10$, and $\epsilon = 10^{-5}$.

4.4 Verification of the Superlinear Convergence Bound

We now turn to the local analysis of superlinear convergence for quasi-Newton methods, focusing on the theoretical result by Jin and Mokhtari [7]. Their analysis shows that, under suitable assumptions, the distance to the minimizer should decay superlinearly in a very specific form.

Theoretical Background. Jin and Mokhtari prove that if f is twice continuously differentiable, strongly convex, and has a Lipschitz continuous gradient near the minimizer w^*, then BFGS satisfies:

$$\|w_k - w^*\|_H \leq \left(\frac{1}{k}\right)^{k/2} \cdot \|w_0 - w^*\|_H,$$

where $\|\cdot\|_H$ is the norm induced by the Hessian H. In our case, since H is constant and symmetric positive definite, the Euclidean and Hessian norms are equivalent. Thus, the bound can be safely reinterpreted as:

$$\|w_k - w^*\| \leq \|w_0 - w^*\| \cdot \left(\frac{1}{k}\right)^{k/2}.$$

This bound is *local*: it only holds when w_k lies sufficiently close to w^*, a region in which the secant approximation is accurate and the curvature is well-behaved.

Empirical Verification (Moderate Regularization, $\lambda = 1$). We first examine the standard configuration $\lambda = 1$, with *limit* $= 20$ and $\epsilon = 10^{-5}$. In this case, the method begins far from the minimizer, and the assumptions of the bound are not immediately satisfied.

In Figure 5, we observe that the empirical sequence $\|w_k - w^*\|$ initially lies above the theoretical curve. This is expected: the early iterations occur in the *transient regime*, before superlinear behavior emerges. However, after a number of steps, the empirical curve sharply decreases and begins to follow the predicted trend. This validates the asymptotic nature of the bound.

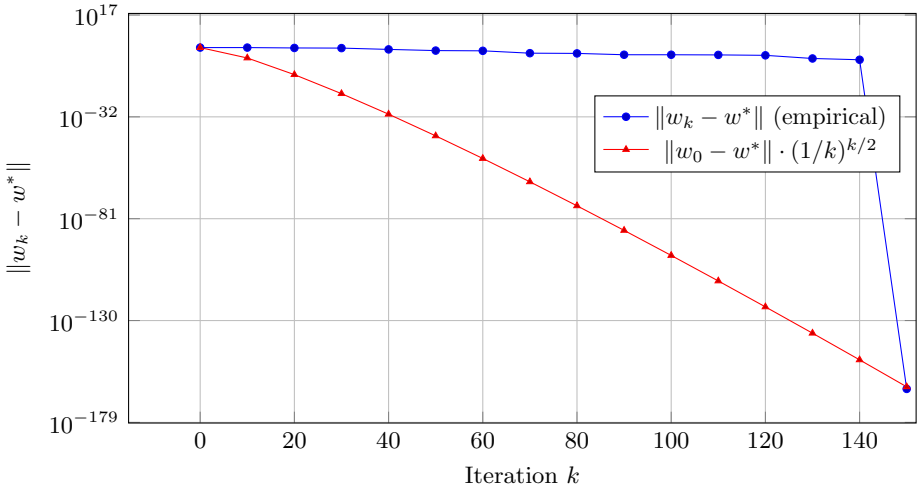

Figure 5: Empirical and theoretical comparison of $\|w_k - w^*\|$ in a moderately regularized setting ($\lambda = 1$), including $k = 0$.

Empirical Verification (Favorable Regime, $\lambda = 10^4$). To further validate the theory, we consider an extreme configuration with $\lambda = 10^4$. While this regularization is not practically useful (as it suppresses the data fidelity term), it yields an almost purely diagonal Hessian and places the starting point very close to the minimizer. As a result, the assumptions of the bound are satisfied *from the beginning*.

Figure 6 confirms this intuition. The empirical curve $\|w_k - w^*\|$ falls rapidly and *respects the theoretical bound from the first iteration*. This experiment serves to confirm that the superlinear bound is not only theoretically valid but also *empirically tight* under favorable conditions.

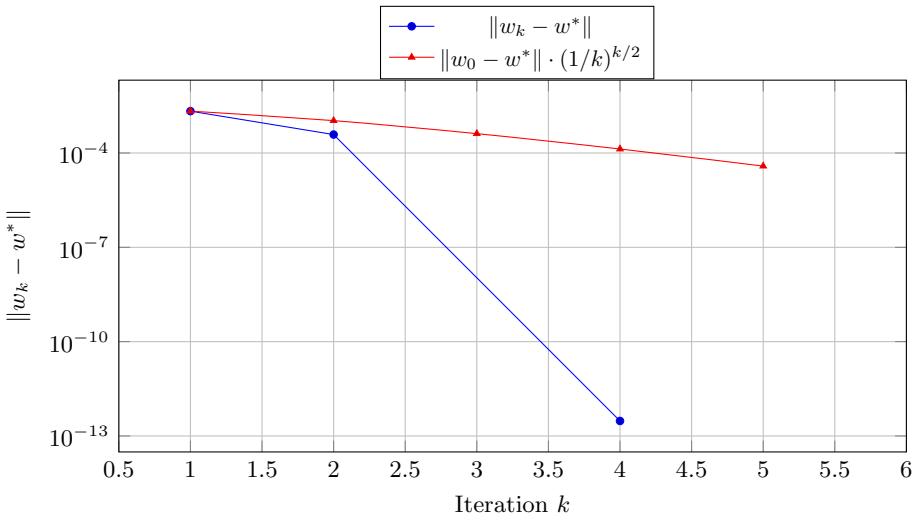

Figure 6: Empirical and theoretical comparison of $\|w_k - w^*\|$ in a highly regularized setting ($\lambda = 10^4$).

4.5 Relative Error Across Regularization Regimes

To complement our analysis of absolute convergence, we now examine the relative error

$$\frac{\|w_k - w^*\|}{\|w^*\|},$$

which expresses the approximation quality of the current iterate in proportion to the size of the true solution. This metric is especially informative in cases where the scale of the solution varies significantly with regularization, and helps to detect numerical instabilities or poorly scaled problems.

We consider three different values of λ representing weak, moderate, and strong regularization. In each case, we plot the relative error trajectory of L-BFGS over the course of optimization, starting from the same initial point $w_0 = 0$.

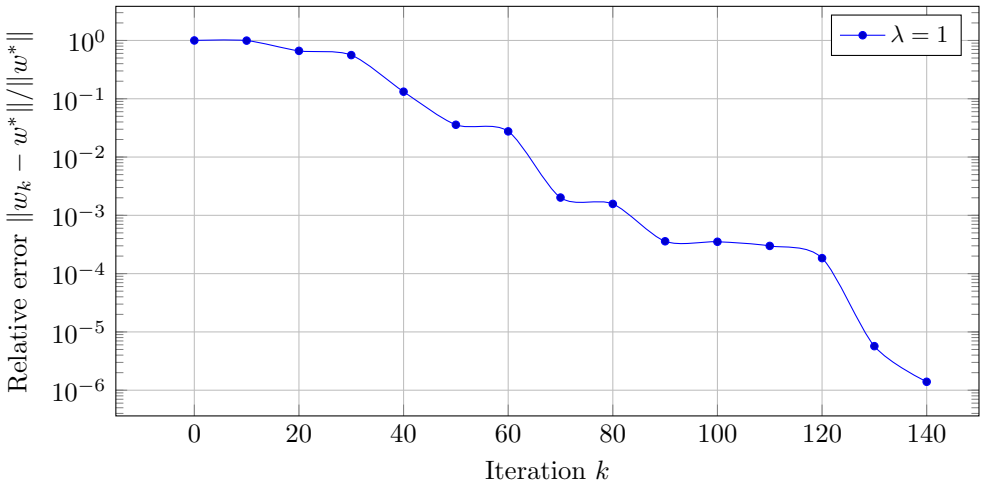

Figure 7: Relative error for L-BFGS with $\lambda = 1$, $limit = 20$, and $\epsilon = 10^{-5}$.

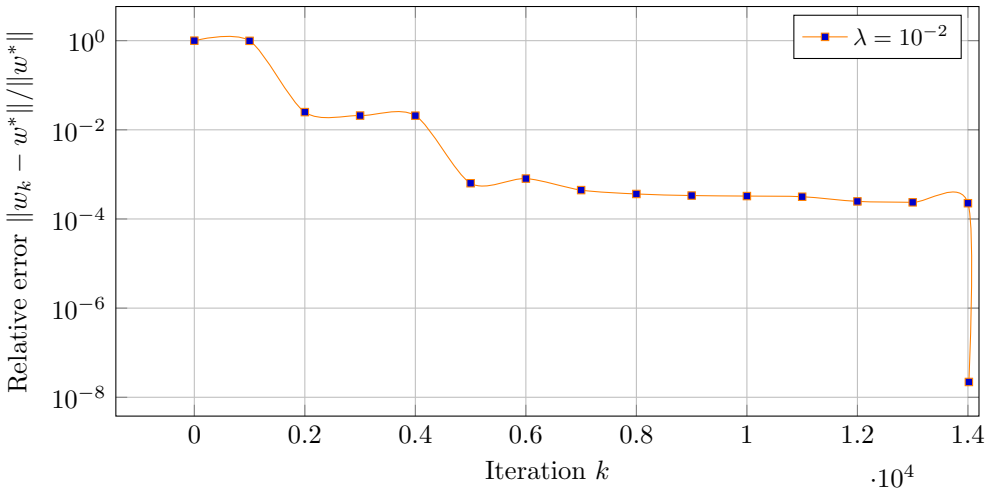

Figure 8: Relative error for L-BFGS with $\lambda = 10^{-2}$, $limit = 10$, and $\epsilon = 10^{-5}$.

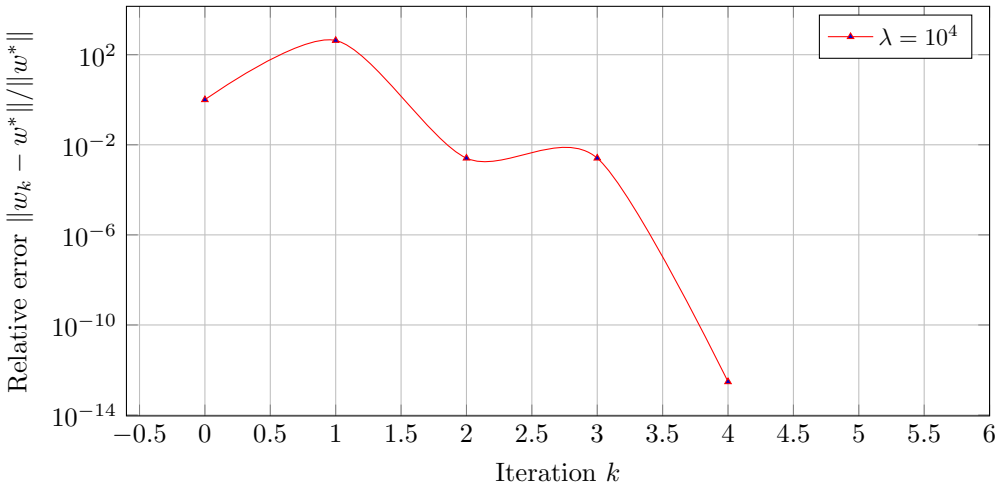

Figure 9: Relative error for L-BFGS with $\lambda = 10^4$, $limit = 10$, and $\epsilon = 10^{-5}$.

From the plots above, we observe that:

- For $\lambda = 1$, the relative error decays smoothly and quickly to machine precision within 150 iterations, confirming a well-conditioned optimization problem. The trajectory is stable and monotonically decreasing, matching theoretical expectations.

- For $\lambda = 10^{-2}$, the decay is much slower and non-monotonic. The algorithm requires over 14,000 iterations to reach convergence. This reflects a poorly conditioned problem, where curvature information is harder to accumulate and the landscape is more sensitive to noise.

- For $\lambda = 10^4$, the behavior is extreme. The true minimizer w^* is very close to zero, so the relative error explodes after the first step (since the denominator is small) and then collapses rapidly. Although the relative error is numerically unstable in early steps, the optimizer still converges to machine precision within five iterations.

Overall, the relative error offers a useful scale-invariant metric for comparing convergence across different regularization regimes. However, as seen in the strong regularization case, care must be taken when interpreting the relative error when the optimal solution is near zero, as small numerator perturbations can lead to large (and potentially misleading) ratios.

5 QR experimental Results

5.1 Verification of QR Decomposition Stability

To validate the correctness and numerical robustness of our custom QR factorization algorithm based on Householder reflections, we conduct a backward stability analysis. Specifically, for each regularization parameter λ, we compute:

$$\|\Delta\| = \|\hat{X} - \hat{X}_{QR}\|,$$

where $\hat{X}$ is the regularized design matrix and $\hat{X}_{QR} = QR$ is its reconstruction from the computed QR factorization. The result is then compared to the expected backward error threshold:

$$\varepsilon \cdot \|\hat{X}\|,$$

where $\varepsilon \approx 2.22 \times 10^{-16}$ is the machine epsilon in double precision. This benchmark represents the maximum discrepancy allowed by a backward stable algorithm.

The results in Figure 10 show that for all values of λ, the computed backward error remains very close to the predicted bound. This confirms that the implementation respects backward stability guarantees and introduces no systematic error.

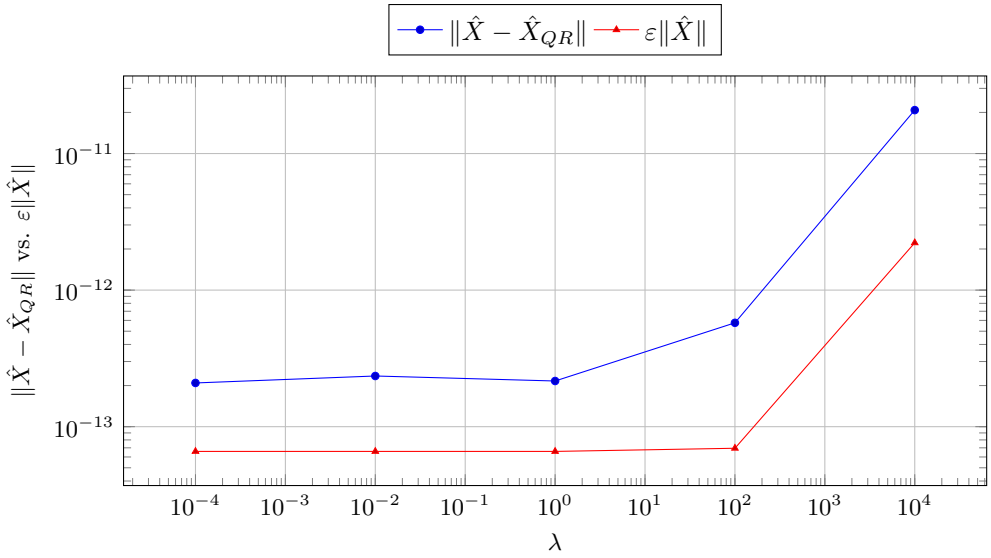

Figure 10: Backward error $\|\hat{X} - \hat{X}_{QR}\|$ and predicted bound $\varepsilon\|\hat{X}\|$ across different values of λ.

As λ increases, the matrix $\hat{X} = \begin{bmatrix} X \\ \lambda I \end{bmatrix}$ grows in norm, since the regularization term becomes dominant. Consequently, both the backward error and the ex-

pected bound increase proportionally. This is fully consistent with the theory of backward stable algorithms and confirms that our QR implementation remains accurate even in highly regularized settings.

5.2 Numerical Accuracy of QR Solution via Augmented System Bound

To validate the numerical accuracy of our QR-based solver, we compute both sides of the augmented system bound for the regularized least squares problem:

$$\frac{\|\tilde{w} - w\|}{\left\| \begin{bmatrix} -\frac{1}{\alpha}\tilde{r} \\ w \end{bmatrix} \right\|} \leq \kappa \left(\begin{bmatrix} \alpha I & \hat{X} \\ \hat{X}^T & 0 \end{bmatrix} \right) \cdot \frac{\left\| \frac{1}{\alpha} \hat{X}^T \tilde{r} \right\|}{\|y\|},$$

where $\tilde{r} := \hat{X}\tilde{w} - \hat{y}$ is the residual of the augmented system.

We compute this inequality for several values of the regularization parameter λ, across three different realizations of the target vector y. For each configuration, we average the results across three values of the perturbation parameter α, starting from $\|\hat{X}^T\|_2 \cdot \varepsilon_{\text{mach}}$ and increasing by powers of 10.

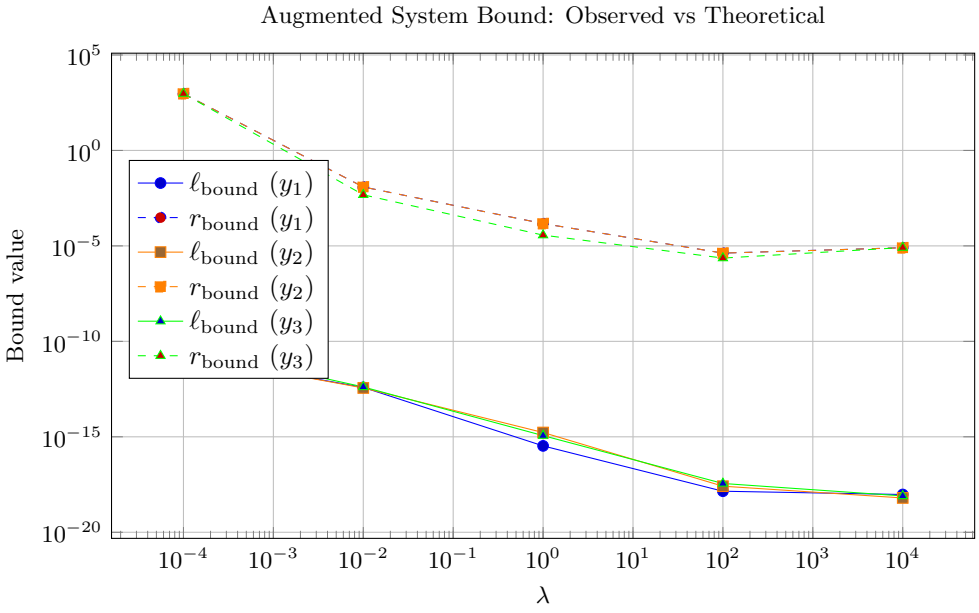

Figure 11: Comparison between observed relative error (ℓ_{bound}) and theoretical upper bound (r_{bound}) for different values of λ and multiple realizations of y.

The plot in Figure 11 shows that the empirical relative error ℓ_{bound} remains consistently below the theoretical upper bound r_{bound}, often by several orders of

magnitude. This confirms the stability and correctness of our QR-based solver.

As expected, increasing λ leads to a better-conditioned augmented system, resulting in smaller values for both ℓ_{bound} and r_{bound}. In all tested configurations, the empirical error is close to machine precision, suggesting the solution is numerically accurate.

We emphasize that α acts as a perturbation scale for the residual, and should be chosen to reflect realistic levels of floating-point roundoff.

6 Comparison: QR vs. L-BFGS

To better visualize the performance impact of the memory parameter *limit*, we split the comparison into two separate plots. Each figure shows average execution time across 10 runs, on a fixed realization of the response vector y, as a function of the regularization parameter λ.

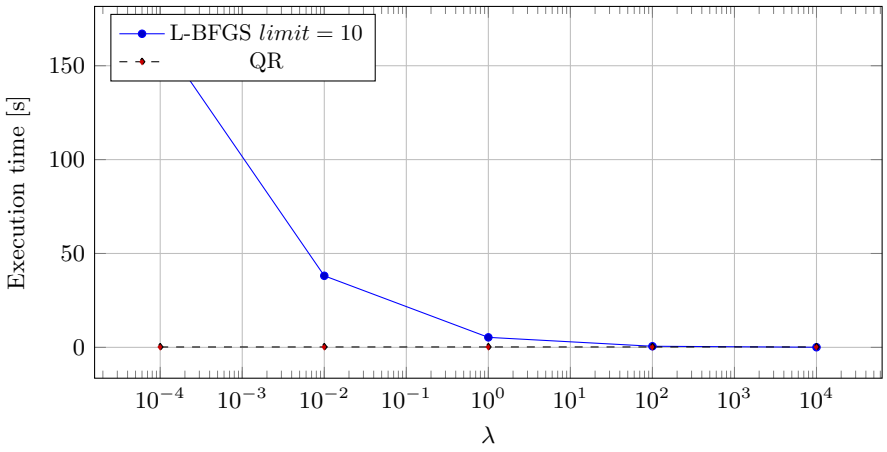

Figure 12: Execution time for L-BFGS with $limit = 10$ vs. QR, across different values of λ.

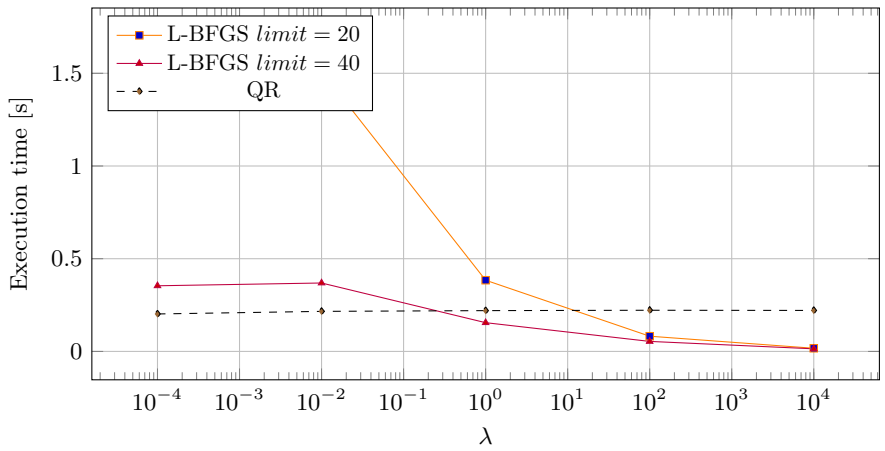

Figure 13: Execution time for L-BFGS with $limit = 20$ and $limit = 40$ vs. QR, across different values of λ.

These plots clearly illustrate the effect of the memory parameter on the efficiency of L-BFGS. When $limit = 10$, the method is extremely inefficient

under weak regularization ($\lambda \leq 10^{-2}$), requiring hundreds of iterations and leading to prohibitively long execution times. In contrast, with $limit = 20$ or $limit = 40$, the method becomes much more robust and competitive—even outperforming QR for moderately or strongly regularized problems ($\lambda \geq 1$).

The QR baseline, as expected from a direct method, displays stable execution time across all λ. However, it is unable to exploit problem structure, leading to higher costs for strongly regularized cases where L-BFGS can converge in a handful of iterations.

Direct comparison with error bars. We now compare directly the average execution time (with variance) between QR and the best L-BFGS configuration ($limit = 40$) on the same response vector y, for each regularization level λ.

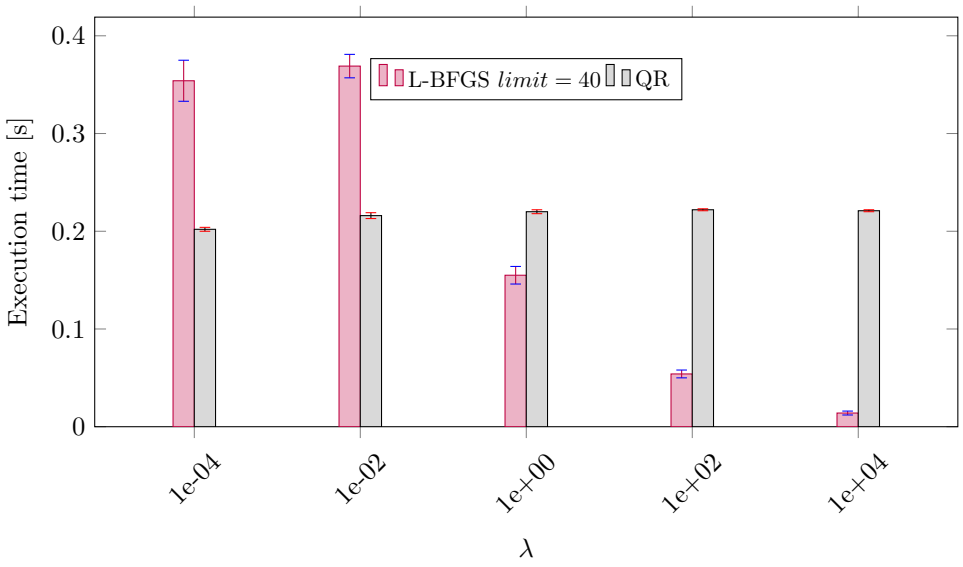

Figure 14: Direct comparison of execution time with standard deviation for QR and L-BFGS ($limit = 40$) on the same y vector.

This bar chart reinforces the observations made previously. For weak regularization, QR outperforms L-BFGS in both speed and stability. However, as λ increases, L-BFGS with $limit = 40$ becomes more efficient than QR, both in mean time and variance. The low variance values for both methods confirm the consistency of the implementations across repeated runs.

References

[1] Albert S Berahas, Raghu Bollapragada, and Jorge Nocedal. Global convergence of a stochastic quasi-newton method for nonconvex optimization. *SIAM Journal on Optimization*, 29(4):2607–2633, 2019.

[2] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[3] William C Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[4] James W Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[5] Gene H Golub and Charles F Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4 edition, 2013.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009.

[7] Qiang Jin and Aryan Mokhtari. Non-asymptotic superlinear convergence of standard quasi-newton methods. *Mathematical Programming*, 194(1-2):435–468, 2022.

[8] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[9] Aryan Mokhtari and Alejandro Ribeiro. Res: Regularized stochastic bfgs algorithm. *IEEE Transactions on Signal Processing*, 62(23):6089–6104, 2014.

[10] Philipp Moritz, Robert Nishihara, and Michael I Jordan. Linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258. PMLR, 2016.

[11] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[12] Artem Rodomanov and Yurii Nesterov. New results on superlinear convergence of classical quasi-newton methods. *Journal of Optimization Theory and Applications*, 188(3):744–769, 2021.

[13] Nicol N Schraudolph, Jin Yu, and Simon Günter. Stochastic quasi-newton methods for online learning. In *Advances in neural information processing systems*, pages 793–800, 2007.

[14] Lloyd N Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, 1997.

[15] Philip Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.