Efficient Methods for Linear Least Squares: A Householder-Based QR Approach

Project NoML-22

Master Degree in Computer in Science (Artificial Intelligence Curriculum), University of Pisa
Computational Mathematics for Learning and Data Analysis course (646AA)
Professors: Antonio Frangioni, Federico Poloni
Academic Year: 2024-2025

Alberto Dicembre
a.dicembre@studenti.unipi.it
Roll number: 668377

Giulia Ghisolfi
g.ghisolfi@studenti.unipi.it
Roll number: 664222

1 Problem description

We are considering the linear least squares problem

$$\min_x \|\hat{A}x - \hat{b}\|,$$

where

$$\hat{A} = \begin{bmatrix} A^T \\ \lambda I \end{bmatrix}, \quad \hat{b} = \begin{bmatrix} b \\ 0 \end{bmatrix},$$

with $A \in \mathbb{R}^{m \times n}$ a tall-and-thin matrix, $\lambda > 0$ is a regularization parameter, and $b \in \mathbb{R}^m$ is a random vector. We assume $m \geq n$.

Firstly, the problem is approached through the utilization of a classical thin QR factorization with Householder reflectors. Subsequently, a variant is implemented that makes use of the structure of the matrix $\hat{A}$ and, in particular, the zeros in the scaled identity block, with a view to reducing the computational cost.

2 Problem Formulation

The linear least squares problem can be efficiently minimized by computing the QR factorization of $\hat{A} \in \mathbb{R}^{(m+n)\times m}$ as follows:

$$\|\hat{A}x - \hat{b}\| = \|QRx - \hat{b}\| = \|Q^T(QRx - \hat{b})\| = \|IRx - Q^T\hat{b}\| = \|Rx - Q^T\hat{b}\|,$$

where $Q \in \mathbb{R}^{(m+n)\times(m+n)}$ is an orthonormal matrix, $R \in \mathbb{R}^{(m+n)\times m}$ is an upper triangular matrix, and $I \in \mathbb{R}^{(m+n)\times(m+n)}$ is the identity matrix.

Given that Q is an orthonormal matrix, it satisfies the fundamental property $Q^{-1} = Q^T$. This allows us to simplify the least squares norm expression by premultiplying both sides by Q^T, effectively preserving the norm due to the orthogonality of Q.

Since R is upper triangular, we can rewrite it as

$$R = \begin{bmatrix} R_0 \\ 0 \end{bmatrix},$$

with $R_0 \in \mathbb{R}^{m \times m}$. We can also rewrite Q^T as

$$Q^T \hat{b} = \begin{bmatrix} Q_1^T \hat{b} \\ Q_2^T \hat{b} \end{bmatrix},$$

with $Q_1 \in \mathbb{R}^{(m+n) \times m}$ and $Q_2 \in \mathbb{R}^{(m+n) \times n}$.

Consequently, the problem can be reformulated as follows:

$$\|Rx - Q^T \hat{b}\| = \left\| \begin{bmatrix} R_0 \\ 0 \end{bmatrix} x - \begin{bmatrix} Q_1^T \hat{b} \\ Q_2^T \hat{b} \end{bmatrix} \right\|.$$

The lower block does not involve x, so we can ignore it. The upper block gives a square linear system:

$$R_0 x = Q_1^T \hat{b}.$$

The term $Q_1^T \hat{b}$ can be computed in $m(m+n)$ operations, and the subsequent computation of x can be performed via back-substitution in m^2 operations since R_0 is upper triangular. Therefore, the total cost is $\mathcal{O}(m^2)$, given that $m > n$.

The most computationally expensive step remains the factorization of $\hat{A}$, which must therefore be performed optimally to ensure efficiency. The objective of this study is to optimize this step in order to reduce the overall cost of solving the least squares problem. This aspect will be analyzed in the following section.

2.1 QR Factorization

To efficiently compute the reduced QR factorization, we apply the Householder transformation method, which constructs Q implicitly while reducing $\hat{A}$ to an upper triangular form.

The reduced QR factorization decomposes $\hat{A}$ as QR, where $\hat{A} \in \mathbb{R}^{(m+n) \times m}$ is the matrix described in Section 1, $Q \in \mathbb{R}^{(m+n) \times m}$ is an orthonormal matrix, and $R \in \mathbb{R}^{m \times m}$ is an upper triangular matrix.

The factorization is computed iteratively, starting with the initialization of the matrix R as $\hat{A}$. At each step, the bottom-right block of the matrix is extracted, and the Householder vector is computed to eliminate all entries below the diagonal in the current column. This vector is then used to construct the Householder matrix, which is applied to R, progressively reducing it to an upper triangular form. At the same time, the matrix Q is updated accordingly to ensure consistency with the transformations applied to R. By the end of the process, R_0 is in an upper triangular form, while Q_1 represents the implicit orthonormal transformations accumulated throughout the iterations.

This factorization differs from the full QR factorization because only the first m columns of Q are retained, reducing the storage requirement while maintaining numerical stability. The retained matrix components are only those necessary for solving the problem, as shown in Section 2.

Note that from this section onward, R refers to the upper triangular part of the matrix, which in Section 2 is denoted as R_0, while Q refers to Q_1 from the previous Section 2.

This approach guarantees numerical stability and involves a total computational cost of $2(m+n)m^2 - \frac{2}{3}m^3$, which simplifies to $\frac{4}{3}m^3 + 2m^2n$ operations, ultimately leading to a complexity of $O(m^3)$. Consequently, this step represents the computational bottleneck, as solving the linear system after the QR factorization requires only $O(m^2)$ operations.

3 Problem Properties

3.1 Existence and Uniqueness of the Solution

The first step to ensure the existence and uniqueness of the solution to the minimization problem is to verify that the QR factorization of the matrix $\hat{A}$ exists and is unique.

The existence of the QR factorization is always guaranteed, and, for any matrix that admits a QR factorization of $\hat{A}$, the existence of a reduced QR factorization is also ensured, as stated in Theorem 7.1 of Trefethen and Bau [1997]. Furthermore, if $\hat{A}$ is full rank, the factorization is also unique, as established in Theorem 7.2 of Trefethen and Bau [1997].

Accordingly, we need to verify that $\hat{A}$ is full rank, meaning that all the columns of $\hat{A}$ are linearly independent. Given that $\hat{A} \in \mathbb{R}^{(m+n)\times m}$, we want to ensure that $rank(\hat{A}) = m$. This holds if and only if the homogeneous system $\hat{A}x = 0$ admits only the trivial solution se $x = 0$. Given the structure of $\hat{A}$,

$$\hat{A} = \begin{bmatrix} A^T \\ \lambda I \end{bmatrix},$$

the equation $\lambda I x = 0$ directly implies that $x = 0$. Therefore, all the columns of $\hat{A}$ are linearly independent, regardless of the choice of A.

For the solution of the system to exist, we also need to ensure that R is invertible so that we can compute $x = R^{-1}Q^T b$.

We require $det(R) \neq 0$, R. Since R is an upper triangular matrix, its determinant is given by the product of its diagonal elements:

$$det(R) = \prod_{i=1}^{m} R_{ii}.$$

Consequently, we need $R_{ii} \neq 0 \ \forall i \in 1, \dots, m$. This condition requires that R has full column rank, which is guaranteed by the uniqueness of the QR factorization, as established above.

Therefore, we can conclude that the solution to our problem always exists and is unique.

3.2 Stability of the Method

The QR factorization method based on Householder transformations guarantees numerical stability, which is a desirable property since it ensures that the numerical error remains bounded at a level proportional to the machine precision.

This is formally stated in Theorem 16.1 of Trefethen and Bau [1997], which shows that the computed factors Q and R satisfy:

$$QR = \hat{A} + \delta\hat{A}, \quad \frac{\|\delta\hat{A}\|}{\|\hat{A}\|} = O(\epsilon_{\text{machine}})$$

for some perturbation matrix $\delta\hat{A}$. This implies that the computed decomposition is a slightly perturbed version of the exact QR factorization.

Furthermore, this stability property extends to the solution of the linear system $\hat{A}x = b$ via QR factorization. Specifically, Theorem 16.2 of Trefethen and Bau [1997] demonstrates that the algorithm for solving $\hat{A}x = b$ using QR factorization is backward stable, meaning that the computed solution $\tilde{x}$ satisfies:

$$(\hat{A} + \delta\hat{A})\tilde{x} = b, \quad \frac{\|\delta\hat{A}\|}{\|\hat{A}\|} = O(\epsilon_{\text{machine}}),$$

for some perturbation δA. This result confirms that the numerical solution corresponds to the exact solution of a slightly perturbed system.

3.3 Conditioning of the matrix $\hat{A}$

The condition number quantifies how much the output of a function can vary in response to small changes in its input. In other words, it measures the sensitivity of the problem to numerical errors.

This quantity provides an estimate of the numerical stability of the system. If the condition number $\kappa(\hat{A})$ is close to 1, the matrix is well-conditioned, meaning that small perturbations in the input data or numerical inaccuracies will not significantly affect the solution. Conversely, if $\kappa(\hat{A})$ is large, the problem is ill-conditioned, implying that even minor variations in the input may lead to substantial changes in the output, making the solution highly unstable.

To compute the condition number in our case, where $\hat{A} \in \mathbb{R}^{(m+n)\times m}$, we first calculate the Moore-Penrose pseudoinverse $\hat{A}^+$. The condition number is then defined as:

$$\kappa(\hat{A}) = \|\hat{A}\|\|\hat{A}^+\|,$$

where $\|\cdot\|$ denotes the 2-norm, which corresponds to the largest singular value of the matrix.

To compute the largest singular value of the matrix $\hat{A} \in \mathbb{R}^{(m+n)\times m}$ and of its Moore-Penrose pseudoinverse $\hat{A}^+$, we can use the Singular Value Decomposition (SVD). The SVD of $\hat{A}$ is given by:

$$\hat{A} = U\Sigma V^T,$$

where U is an orthogonal matrix of size $(m+n)\times(m+n)$, Σ is a diagonal matrix of size $(m+n)\times m$ containing the singular values of $\hat{A}$, V^T is an orthogonal matrix of size $m \times m$.

The decomposition of the Moore-Penrose pseudoinverse $\hat{A}^+$ is then computed as:

$$\hat{A}^+ = V\Sigma^+U^T,$$

where Σ^+ is the pseudoinverse of the diagonal matrix Σ. To obtain Σ^+, we invert the non-zero singular values in Σ and keep zeros in place of the zero singular values. For instance, if Σ is:

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_m \end{pmatrix},$$

then Σ^+ is:

$$\Sigma^+ = \begin{pmatrix} 1/\sigma_1 & 0 & \dots & 0 \\ 0 & 1/\sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/\sigma_m \end{pmatrix},$$

where $\sigma_1, \sigma_2, \dots, \sigma_m$ are the non-zero singular values of $\hat{A}$.

Once the pseudoinverse $\hat{A}^+$ is computed, the condition number of $\hat{A}$ can be determined as:

$$\kappa(\hat{A}) = \|\hat{A}\|\|\hat{A}^+\| = \|U\Sigma V^T\|\|V\Sigma^+U^T\| = \frac{\sigma_1}{\sigma_m},$$

where σ_1 is the largest singular value, and σ_m is the smallest non-zero singular value of $\hat{A}$.

3.4 Conditioning of the Least Squares Problem

The condition number of a Least Squares problem is more intricate, involving various quantities, and offers valuable insight into how sensitive the solution is to perturbations in the input data, particularly the matrix $\hat{A}$ and the vector $\hat{b}$.

Computing these quantities is crucial for assessing the stability and accuracy of the solution, especially when dealing with ill-conditioned or nearly singular matrices.

To compute these quantities, we refer to the table structure outlined in Theorem 18.1 of Trefethen and Bau [1997], which is presented below for clarity.

Theorem 18.1. Let $\hat{b} \in \mathbb{C}^m$ and $\hat{A} \in \mathbb{C}^{m\times n}$ of full rank be fixed. The least squares problem has the following 2-norm relative condition numbers describing the sensitivities of $y = \hat{A}x$ and x to perturbations in $\hat{b}$ and $\hat{A}$, that will be reported below

	y	x
$\hat{b}$	$\frac{1}{\cos\theta}$	$\frac{\kappa(\hat{A})}{\eta\cos\theta}$
$\hat{A}$	$\frac{\kappa(\hat{A})}{\cos\theta}$	$\kappa(\hat{A}) + \frac{\kappa(\hat{A})^2\tan\theta}{\eta}$

Table 1: Formulas for relative condition number of least squares problems.

The results in the first row are exact, being attained for certain perturbations $\delta\hat{b}$, and the results in the second row are upper bounds.

The quantities θ and η are computed as follows:

$$\theta = \arcsin\left(\frac{\|\hat{b} - y\|}{\|\hat{b}\|}\right), \quad \eta = \frac{\|\hat{A}\| \cdot \|x\|}{\|y\|}.$$

4 Optimized QR Factorization

Given the block structure of $\hat{A}$, it can be utilized to reduce the computational cost of the QR decomposition, which is the bottleneck of the process.

In order to optimize the algorithm, we start from the version of the QR Householder factorization that does not explicitly form the matrix Q. This introduce unnecessary computation, since it it only use is to compute the product $Q^T y$ to solve the Least Squares problem. This is due to the fact that the Householder arithmetic allows us to calculate the product implicitly with the Householder vectors directly, and in a more efficient way.

As outlined in Algorithm 10.2 of Trefethen and Bau [1997], it is possible to write Q as a product of factors:

$$Q^T = Q_n \cdots Q_2 Q_1.$$

This allows for an implicit computation of the product $Q^T b$, which can be performed as in Algorithm 1.

Algorithm 1 Implicit Calculation of a Product $Q^T b$

Input: $V \in \mathbb{R}^{p \times q}$ (matrix containing Householder vectors), $b \in \mathbb{R}^p$
Output: $b \in \mathbb{R}^p$ (transformed vector $b \leftarrow Q^T b$)

```
1 Function Implicit_Product($V$, $b$)
2     for $i = 1$ to $n$ do
3         $b_{i:m} = b_{i:m} - 2v_i(v_i^T b_{i:m})$
4     return $b$
```

By storing only the Householder vectors instead to the full matrix, two advantages are gained. The explicit Q matrix is larger and more dense than the Householder vectors, thus ensuring superior memory efficiency. Furthermore, the complexity of calculating implicitly the product $Q^T y$ is reduced from $O(mn^2)$ to $O(mn)$, thereby enhancing computational efficiency.

In Algorithm 2 below, the complete QR algorithm using the implicit computation of Q is presented. From now on, for the sake of notational simplicity, we will denote the dimensions of

Algorithm 2 Reduced QR Factorization with implicit Q

```
Input: $\hat{A} \in \mathbb{R}^{p \times q}$
Output: $U \in \mathbb{R}^{p \times q}$ (Reduced Householder vectors), $R \in \mathbb{R}^{q \times q}$ (upper triangular matrix)
5  Function Implicit_QR_Factorization($\hat{A}$)
6      $(p, q) \leftarrow \text{size}(\hat{A})$
7      $R \leftarrow \hat{A}$
8      $U \leftarrow \text{zeros}(p, q)$
9      for $i = 1$ to $q$ do
10         $u \leftarrow \text{Householder\_Vector}(R[i:p, i])$
11         $U[i:p, i] \leftarrow u$
12         $R[i:p, i:q] \leftarrow R[i:p, i:q] - 2u(u^T R[i:p, i:q])$
13     return $U, R$
```

$\hat{A}$ as p and q, where $p = m + n$ and $q = m$. Meanwhile, m and n will continue to refer to the dimensions of the tall-and-thin matrix A. Thus, we have $\hat{A} \in \mathbb{R}^{p \times q}$.

At this point, the main computational steps in Algorithm 2 involve matrix operations and constitute the bulk of the computational workload. In the following, we aim to optimize these computations by exploiting the block structure of $\hat{A}$

The first operation we encounter is the computation of the Householder vector, which usually involves the entire column $R_{i:p,i}$, when working with a dense matrix. In our case, the relevant information is contained only in the first $n + 1$ elements. Therefore, instead of computing the Householder vector as $u = \text{Householder\_Vector}(R_{i:p,i})$, we can reduce the operation to

$$u_0 = \text{Householder\_Vector}(R_{i:n+1,i}),$$

avoiding unnecessary computations on zero elements.

Subsequently, instead of computing the full matrix-vector product $u^T R_{i:p,i:q}$, we can restrict the computation to

$$u_0^T R_{i:n+1,i:q},$$

considering only the first $n + 1$ elements of the Householder vector.

Since u_0 has only $n + 1$ elements instead of p, the computational complexity is reduced from $O(qp)$ to $O(q(n + 1))$, which simplifies to $O(qn)$ and, in terms of the original matrix A, results in $O(mn)$.

The result is then multiplied by u_0, leading to the computation:

$$u_0 \cdot u_0^T R_{i:n+1,i:q}.$$

This operation produces a matrix of size $(n + 1) \times q$, instead of $p \times q$, significantly reducing the computational cost. As a consequence, the complexity of this multiplication decreases from $O(mp)$ to $O(mn)$, making the overall computation more efficient.

This is followed by a scalar multiplication:

$$-2 \cdot u_0 \cdot u_0^T R_{i:n+1,i:q},$$

where the previously obtained $(n + 1) \times q$ matrix is scaled by -2. For a full $m \times p$ matrix, this operation would have a complexity of $O(mp)$. However, due to the sparsity of $\hat{A}$, the computation is restricted to only $n + 1$ rows, reducing the complexity to $O(mn)$.

Finally, we perform the subtraction:

$$R_{i:n+1,i:q} - 2 \cdot u_0 \cdot u_0^T R_{i:n+1,i:q}.$$

This operation involves two matrices of size $(n + 1) \times q$, resulting in a final computational complexity of $O(mn)$, consistent with the previous steps.

The final optimized algorithm is presented in Algorithm 3 below.

Algorithm 3 Optimized Reduced QR Factorization

Input: $\hat{A} \in \mathbb{R}^{p \times q}$ (a $p \times q$ matrix, structured as a transposed tall-and-thin matrix with a scaled identity block appended to its bottom), n (number of dense rows of the $\hat{A}$ matrix)
Output: $U_0 \in \mathbb{R}^{p \times q}$ (Reduced Householder vectors), $R \in \mathbb{R}^{q \times q}$ (upper triangular matrix)

```
14 Function Optimized_QR_Factorization($\hat{A}$)
15     $(p, q) \leftarrow \text{size}(\hat{A})$
16     $R \leftarrow \hat{A}$
17     $U_0 \leftarrow \text{zeros}(n + 1, q)$
18     for $i = 1$ to $q$ do
19         $n = n + 1$
20         $u_0 \leftarrow \text{Householder\_Vector}(R[i : n, i])$
21         $U_0[i : n, i] \leftarrow u_0$
22         $R[i : n, i : q] \leftarrow R[i : n, i : q] - 2u_0(u_0^T R[i : n, i : q])$
23     return $U_0, R$
```

Given that each step, which costs $O(mn)$, is performed $q = m$ times, the total computational cost is $O(mn) \cdot m = O(m^2 n)$. This represents a substantial enhancement in comparison with the non-optimised implementation.

5 Code Implementation

In this section, the code that has been discussed in the previous sections is presented.

The solution to the Least Squares problem, as described in Section 2, is provided in Algorithm 4. The reduced QR factorization, detailed in Section 2.1, is shown in Algorithm 5. The function to extract Householder vectors is presented in Algorithm 6. Together with the previously introduced optimized QR factorization algorithm in Algorithm 3 in Section 4, these constitute the implemented algorithms.

Algorithm 4 QR Algorithm for Least Squares

Input: $\hat{A} \in \mathbb{R}^{p\times q}$ (matrix with $p \geq q$ and full column rank), $\hat{b} \in \mathbb{R}^p$ (vector)
Output: $x \in \mathbb{R}^q$ (solution to $\min \|\hat{A}x - b\|_2$)

```
Function QR_Solve_LeastSquares($\hat{A}$, $\hat{b}$):
    // Compute the QR factorization of $A$ using Householder transformations
    $\hat{A} = QR$
    // Compute the modified right-hand side
    $c = Q^\top b$
    // Extrapolate first $n$ components of $c$ ($c_1 \in \mathbb{R}^q$)
    $c_1 = c[: q]$
    // Solve the triangular system using back substitution
    $R_0 x = c_1$
    return $x$
```

Algorithm 5 Reduced QR Factorization via Householder Reflections

Input: $\hat{A} \in \mathbb{R}^{(m+n)\times m}$ (matrix with $m \geq n$ and full column rank)
Output: $Q_1 \in \mathbb{R}^{(m+n)\times m}$ (implicit orthonormal matrix), $R_0 \in \mathbb{R}^{m\times m}$ (upper triangular matrix)

```
Function QR_Reduced($\hat{A}$)
    // Initialize matrices
    $R \leftarrow \hat{A}$
    $Q_1 \leftarrow I[:, : m]$ // Only first $m$ columns of $I$
    for $k = 1$ to $m$ do
        // Extract submatrix (bottom-right block of $R$ starting at $k$)
        $R_k \leftarrow R[k : m+n, k : m]$
        // Compute Householder vector for the first column of $R_k$
        $v_k \leftarrow \text{Householder\_Vector}(R_k[:, 1])$
        // Form Householder matrix
        $H_k \leftarrow I - 2\frac{v_k v_k^T}{v_k^T v_k}$
        // Apply $H_k$ to $R_k$ to update $R$
        $R[k : m+n, k : m] \leftarrow H_k R_k$
        // Apply $H_k$ to update only the first $m$ columns of $Q_1$
        $Q_1[k : m+n, k] \leftarrow H_k Q_1[k : m+n, k]$
    return $Q_1, R_0$
```

Algorithm 6 Householder Vector Extraction

```
Input: $A \in \mathbb{R}^{m \times n}$
Output: $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ (Householder vectors)
Function Householder_Vector($A$)
    for $j = 1$ to $\min(m, n)$ do
        $x = A[j : m, j]$
        $\mathbf{v}_j = x + \text{sign}(x_1)\|x\|_2\mathbf{e}_1$
        $\|\mathbf{v}_j\|_2 = 1$ // Normalize $\mathbf{v}_j$
        // Apply the Householder transformation
        $A[j : m, j : n] = A[j : m, j : n] - 2\mathbf{v}_j(\mathbf{v}_j^T A[j : m, j : n])$
    return $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$
```

The code was implemented in `Python`, and the `NumPy` library was used for array manipulation.

To measure performance in terms of memory usage and execution time, the `tracemalloc` and `time` libraries were used, respectively, as already described in Section 7.

For visualization of the results, the `matplotlib`, `seaborn`, and `pandas` libraries were employed.

A more detailed overview of the folder structure and the files containing the code is provided in Table 2, in Appendix.

The `README.md` file contains the instructions to run the experiments.

6 Data

To reproduce the experiment, the data used to construct the matrix A is taken from the ***ML-CUP*** dataset of the 2024-2025 *Machine Learning* course.

$A \in \mathbb{R}^{250 \times 15}$ is a rectangular matrix, with nonzero real numbers ranging in $[-9.749, 25.804]$, a mean of 1.077, and a standard deviation of 4.027.

To generate the vector $b \in \mathbb{R}^{250}$, we sample a random vector from a normal distribution $\mathcal{N}(1.077, 4.027)$.

With this data, we construct the final matrices used for the experiment, $\hat{A}$ and $\hat{b}$. Given that $m = 250$ and $n = 15$, the resulting dimensions are $\hat{A} \in \mathbb{R}^{265 \times 250}$, and $\hat{b} \in \mathbb{R}^{265}$.

6.1 Singular Values of A

Below are the singular values of A, sorted from largest to smallest:

$$\begin{bmatrix} 244.1377 & 63.1381 & 23.1181 & 21.3151 & 18.8556 & 8.8574 & 7.7268 & 6.4049 \\ 4.5256 & 2.7512 & 1.2218 & 8.2950 & 0.7436 & 0.0534 & 0.0036 & \end{bmatrix}$$

We can now compute the conditioning number for the matrix A:

$$\kappa(A) = \frac{\sigma_1}{\sigma_n} = \frac{244.137749}{0.003562} \approx 6.8529 \cdot 10^4.$$

Therefore, we have a conditioning number $\kappa(A) \approx 6.8529 \cdot 10^4$, which indicates that the matrix A is ill-conditioned.

6.2 Singular Values and Condition Number of $\hat{A}$

In Figure 1, we show how the minimum and maximum singular values of $\hat{A}$ vary as a function of the regularization parameter λ.

We observe that both the largest and the smallest singular values of $\hat{A}$ increase with increasing λ. However, the minimum singular value increases significantly faster than the maximum. As a consequence, the condition number of $\hat{A}$ decreases as λ increases.

Figure 2 illustrates this behavior: the condition number of $\hat{A}$ decreases rapidly for small values of λ, and then stabilizes, reaching a plateau for larger values.

We can conclude that the matrix $\hat{A}$ is best conditioned for values of $\lambda > 10^2$, where the condition number flattens and becomes relatively stable.

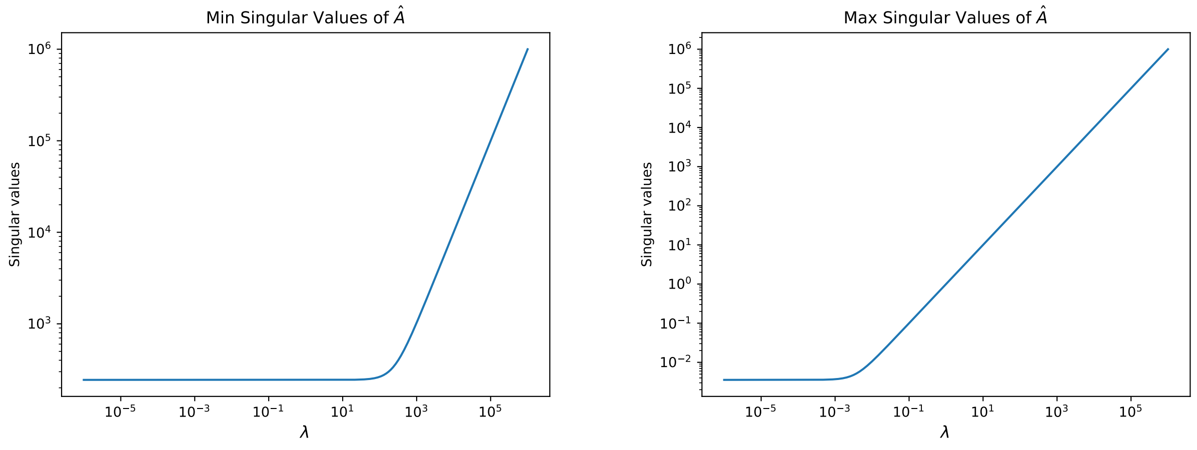

Figure 1: Minimum (left) and maximum (right) singular value of $\hat{A}$ for different values of λ ranging from 10^{-6} to 10^6. Both axis are in logarithmic scale.

6.3 Conditioning of the Least Squares Problem

The condition numbers were calculated as described in Section 3.4.

Figure 3 reports the plots of the variation of the condition numbers for the least squares problem, computed as the values of the regularization parameter λ, ranging from 10^{-6} to 10^6.

The four subplots of figure 3 show the behavior of the condition number of the output variables y and x with respect to perturbations in the matrix $\hat{A}$ and the right-hand side $\hat{b}$.

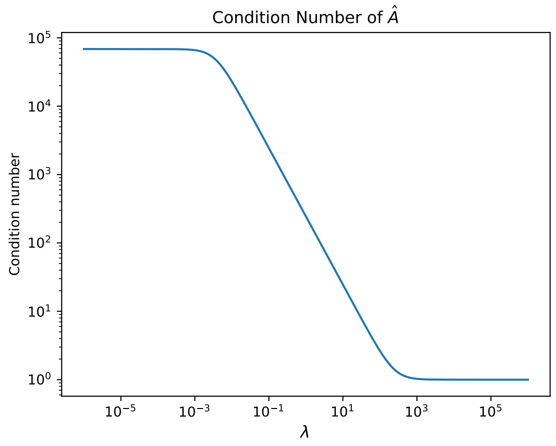

Figure 2: Condition number of $\hat{A}$ for different values of λ ranging from 10^{-6} to 10^6. Both axis are in logarithmic scale.

The red circles mark the values of λ corresponding to the smallest condition number in each case, emphasizing the most stable configurations.

For perturbations in $\hat{A}$, the condition numbers exhibit a non monotonic behavior, with a well-defined minimum at an intermediate value of λ. This indicates the existence of an optimal regularization level that balances the trade-off between sensitivity to noise and approximation accuracy. In particular, the condition number of x with respect to $\hat{A}$ reaches its minimum with λ beetween 10^2 and 10^3, showing a sharp decrease from the ill-conditioned regime at small λ.

On the other hand, the condition number with respect to $\hat{b}$ increases monotonically with λ for y, indicating that strong regularization amplifies the system's sensitivity to perturbations in the data vector. For x, the behavior is different: the condition number with respect to $\hat{b}$ displays a minimum around $\lambda \approx 10^2$.

Moreover, we observe that the condition numbers with respect to perturbations in $\hat{A}$ are consistently higher than those related to $\hat{b}$. This suggests that perturbations in the matrix data have a more significant impact on the solution than perturbations in the right-hand side vector. This behavior is likely due to the way in which $\hat{b}$ is constructed.

7 Performance Measurement

The performance was evaluated by considering five main metrics: execution time, memory usage, relative error, factorization error, and residual norm.

The **memory space** required to solve the problem was measured in kilobytes using the `tracemalloc` library, while the **execution time** was measured in seconds using the `time` library.

The **relative error** was computed using the following formula:

$$\text{Relative Error} = \frac{\|x_{\text{true}} - x_{\text{pred}}\|}{\|x_{\text{true}}\|}.$$

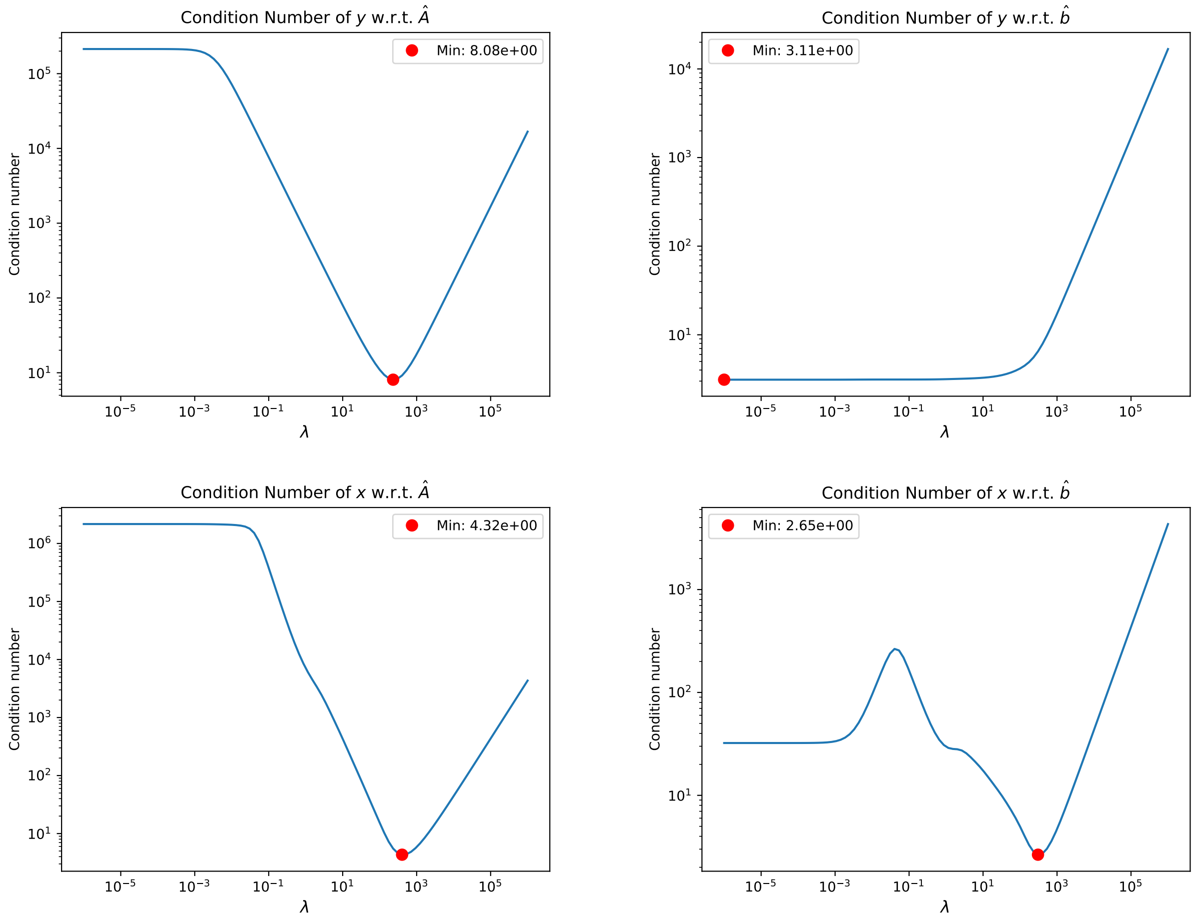

Figure 3: Relative condition numbers for the least squares problem computed for different values of the regularization parameter λ, ranging from 10^{-6} to 10^{6}. The red circles mark the values of λ corresponding to the smallest condition number in each case, emphasizing the most stable configurations. Both axis are in logarithmic scale.

The ground truth solution x_{true} was obtained using the `linalg.lstsq` function, from the `Numpy` library, which computes the least-squares solution to a linear matrix equation. This solution was used as a reference to assess the accuracy of the predictions produced by the implemented algorithms.

The **factorization error** between the QR factorization and the original matrix $\hat{A}$ was computed as:

$$\text{Factorization Error} = \|\hat{A}_{\text{reconstructed}} - \hat{A}\|.$$

where $\hat{A}_{\text{reconstructed}}$ denotes the reconstruction of $\hat{A}$ using the matrices Q and R, in order to evaluate the accuracy of the QR factorization.

Finally the **residual norm** assess how well the computed solution satisfies the original system.

$$\text{Residual Norm} = \|\hat{A}x - \hat{b}\|.$$

8 Experimental Setup

The implemented functions are those described in Sections 4 and 5.

In order to assess the performance of the implemented algorithms, three sets of experiments were performed.

The first set varied the regularization parameter λ. The results and discussion are reported in Section 9. The second set involved adding noise to assess the robustness of the methods, and results are presented in Section 10. Both of these first two sets of tests used the dataset described in Section 6.

The third set, reported in Section 11, used randomly generated matrices A of increasing size to evaluate the scalability of the execution time and memory usage of the methods with respect to the matrix dimensions.

The experiments were conducted using both versions of the QR factorization to solve the minimization problem introduced in Section 1 and solved through Algorithm 4, as discussed in Section 5.

The version of the QR factorization based on the reduced QR factorization via Householder reflections, described in Algorithm 5, will be referred to as *reduced* in the following.

Conversely, the version that uses the optimized QR factorization, described in Algorithm 3, will be referred to as *optimized.*

All metrics were collected in the same environment to ensure consistency across runs. All computations were performed in double precision to minimize rounding errors.

A fixed random seed was set to ensure reproducibility of the experiment.

9 Experiments

The aim of the following experiments is to compare the two versions of the algorithm described above, in terms of computational performance.

The experiment was performed using different values of λ, ranging from 10^{-8} to 10^{4}, sampled on a logarithmic scale.

Tables 3 and 4, in Appendix, report the results of the experiments using the reduced and optimized versions of the algorithm, respectively.

Figures 4 and 5 show how the execution time and relative error vary with respect to λ for both algorithm versions.

A plot for the memory usage is not included, as the values remain nearly constant across different λ, although the optimized version consistently requires significantly less memory than the reduced one.

From the experimental results, we observe that the optimized version requires significantly less memory (less than 188 kB compared to 2812 kB for the reduced version) and achieves faster execution times in solving the problem.

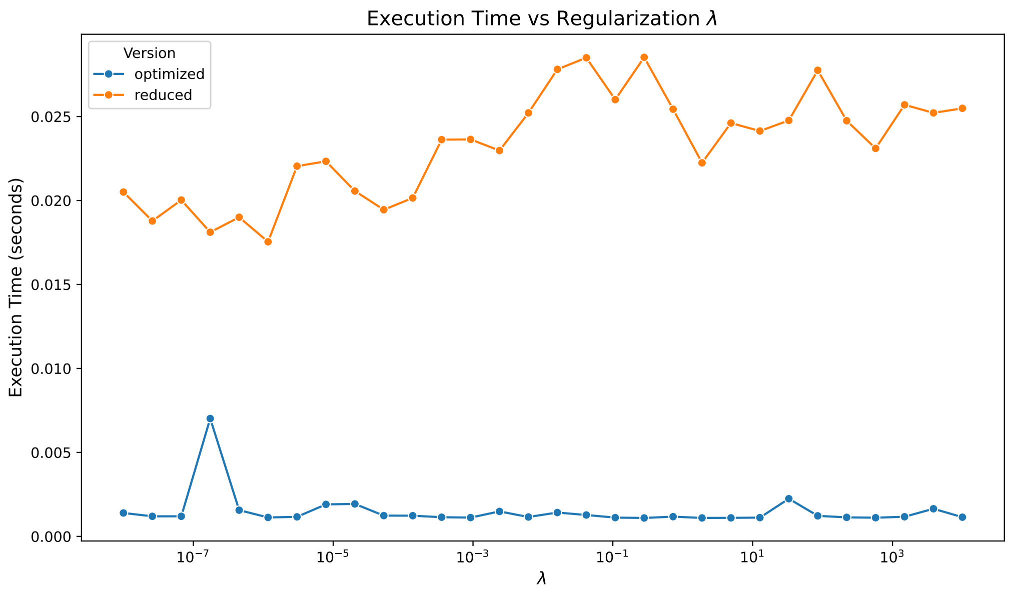

Figure 4: Execution time (in seconds) as a function of λ for the reduced and optimized versions of the algorithm. The x-axis is in logarithmic scale.

We also note that the relative error, in both the optimized and reduced versions, exhibits fluctuations as λ varies. In particular, it reaches a peak around λ equal to 10^{-2}, then decreases, achieving the lowest error values between 1 and 10^3, and finally increases rapidly for $\lambda > 10^3$.

Figures 6 and 7 show the behavior of the factorization error and the residual norm across different values of λ, for both algorithm versions.

From the results, we can conclude that both factorizations allow us to reconstruct the original matrix $\hat{A}$ with comparable accuracy. In both cases, the reconstruction error varies with the value of λ, increasing as λ increases, but always remaining below the order of 10^{-11}.

Note that in the *reduced* version, $\hat{A}_{\text{reconstructed}}$ is simply computed as the product QR. In the *optimized* version, instead, since Q is represented implicitly via Householder vectors, $\hat{A}_{\text{reconstructed}}$ is obtained by applying the sequence of Householder transformations to the matrix R, exploiting the implicit product calculation described in Algorithm 1.

Figure 7 reports the value of the residual norm $\|\hat{A}x - \hat{b}\|$ achieved by each method, showing that the minimization problem solved using both the reduced and optimized QR factorizations leads to the same solution.

Results and Discussion

From the execution time metric, shown in Figure 4 and reported in Tables 4 and 3, the optimized version consistently achieves runtimes approximately one order of magnitude smaller than the reduced version. This demonstrates that the *optimized* implementation is superior in terms of computational efficiency.

With respect to memory usage, theory indicates that the *reduced* QR version stores the entire matrix Q explicitly, whereas the *optimized* version only stores the Householder vectors.

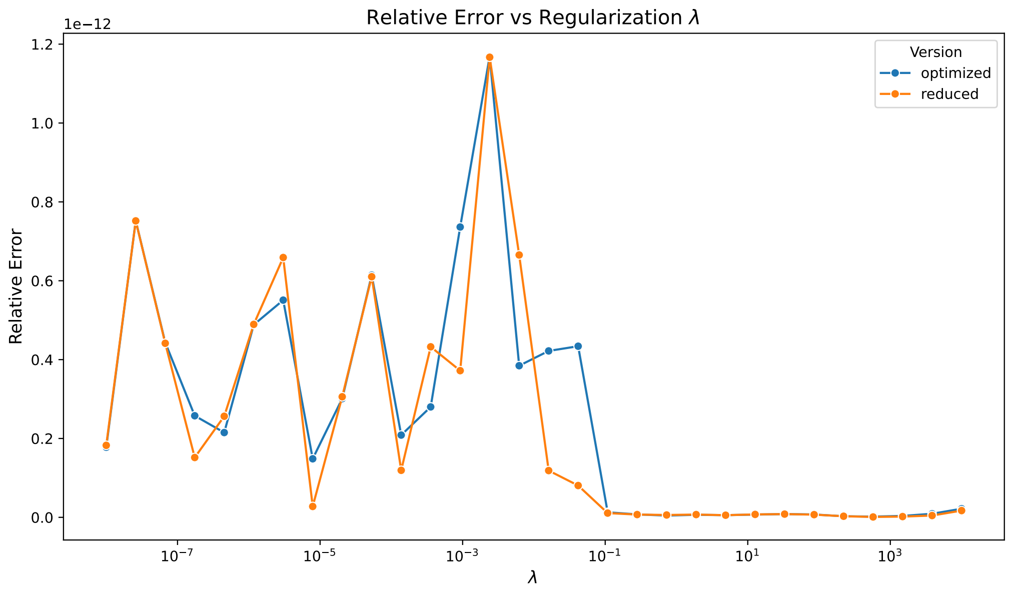

Figure 5: Relative error as a function of λ for both algorithm versions. The x-axis is in logarithmic scale.

Therefore, we expect the *optimized* version to require significantly less memory, which is indeed confirmed by the results reported in Tables 4 and 3 in the Appendix.

The *optimized* version uses less than 188 kB compared to 2812 kB for the *reduced* version. In all the experiments described in Section 9, the matrix dimensions remain constant, therefore, the memory usage for each method is also constant.

The relative error plot in Figure 5 shows that both versions produce errors on the order of 10^{-12} for all tested values of λ. According to the stability results of the QR factorization, already discussed in Section 3.2, the factorization error should be proportional to machine precision, $O(\epsilon_{\text{machine}}) \approx 10^{-16}$ in double precision, as used here. Our results are close to this bound, considering that we are measuring a norm over the entire matrix and that numerical operations accumulate rounding errors.

Moreover, we can observe that, for both versions, the relative error exhibits significant oscillations for λ values less than 10^{-1}, fluctuating within the range 10^{-12} to 10^{-14}. For λ values greater than 10^{-1}, the relative error stabilizes and remains nearly constant around 10^{-15}.

This behavior is consistent with the conditioning values of $\hat{A}$ reported in Section 6.2, which show that the condition number decreases as λ increases, leading to improved numerical stability and reduced error oscillations.

The factorization error plot, Figure 6, confirms that $\|\hat{A}_{\text{reconstructed}} - \hat{A}\|$ stays below 10^{-11} for both methods, which indicates that the reconstructed matrices closely approximate the original matrix with a very small numerical error.

Figure 7 shows that the residual norm $\|\hat{A}x - \hat{b}\|$ is identical for the two methods across all tested λ, confirming that both factorizations lead to the same minimizer. This matches the theoretical result from Section 3.1, where the uniqueness and existence of the solution were

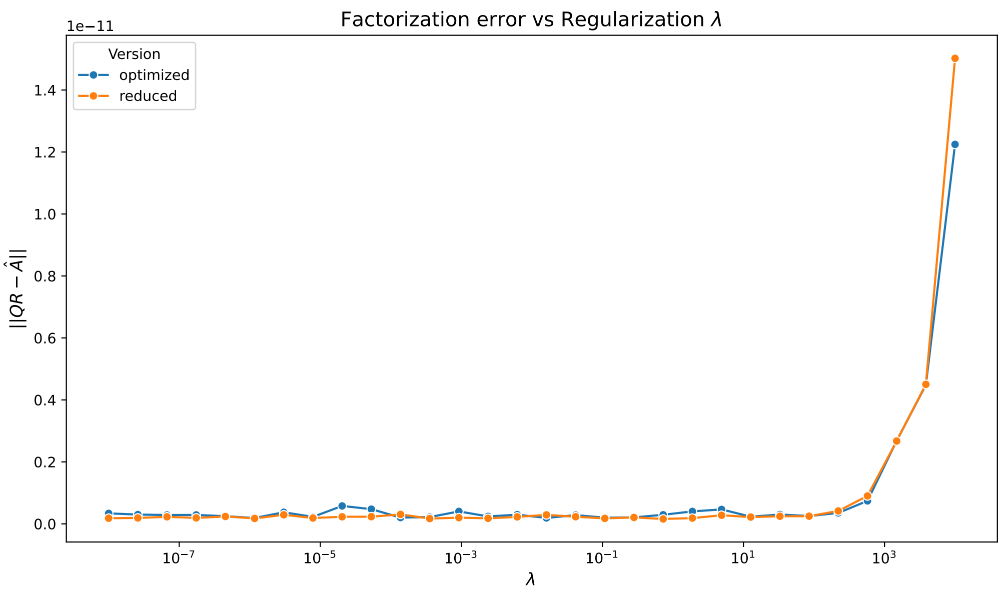

Figure 6: Factorization error as a function of λ for both algorithm versions. The x-axis is in logarithmic scale.

proven, and is further explained by the fact that both algorithms are algebraically equivalent up to floating-point round-off errors.

10 Noise Sensitivity Analysis

To assess the impact of noise on the stability and performance of the problem, we conducted a series of experiments by adding Gaussian noise with zero mean and varying standard deviations to both the matrix $\hat{A}$ and the vector $\hat{b}$.

Specifically, noise was sampled from $\mathcal{N}(0, \sigma_A)$ for $\hat{A}$ and from $\mathcal{N}(0, \sigma_b)$ for $\hat{b}$, with σ_A and σ_b taking different values to evaluate the robustness of the proposed methods under various noise levels.

The noise levels σ_A and σ_b were varied in the set $\{0, 10^{-8}, 10^{-6}, 10^{-4}\}$.

All experiments were carried out under the same conditions as described in the previous Section 9, using the dataset described in Section 6. The noise for each value of σ_A and σ_b was sampled using a fixed random seed, ensuring that the input provided to the two versions of the algorithm was exactly the same.

The regularization parameter λ was varied in the range $[10^{-8}, 10^4]$.

The noise on $\hat{A}$ and $\hat{b}$ was added independently, in order to assess whether perturbing each component separately would have a different impact on the results. The experiments covered all possible combinations of noise levels in the predefined set for both σ_A and σ_b.

As evaluation metrics, we used the relative error, the factorization error, and the residual norm, as defined in Section 7.

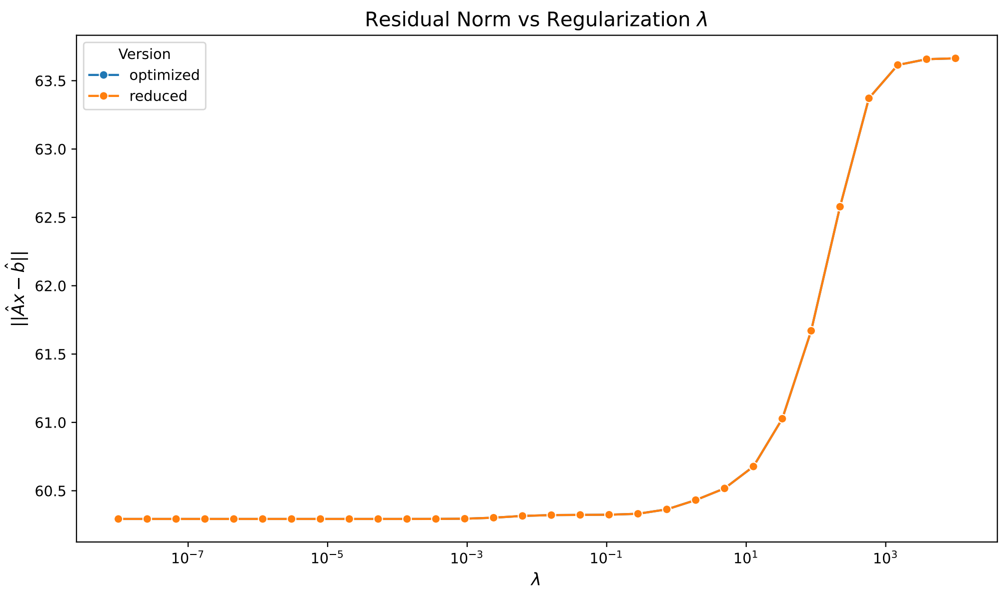

Figure 7: Residual norm as a function of λ for both algorithm versions. The x-axis is in logarithmic scale.

Figures 8, 9, and 10 report the heatmaps illustrating the behaviour of these metrics across the tested noise levels and λ values.

The metric values are displayed on a logarithmic scale, with darker colors representing lower values and lighter colors representing higher values.

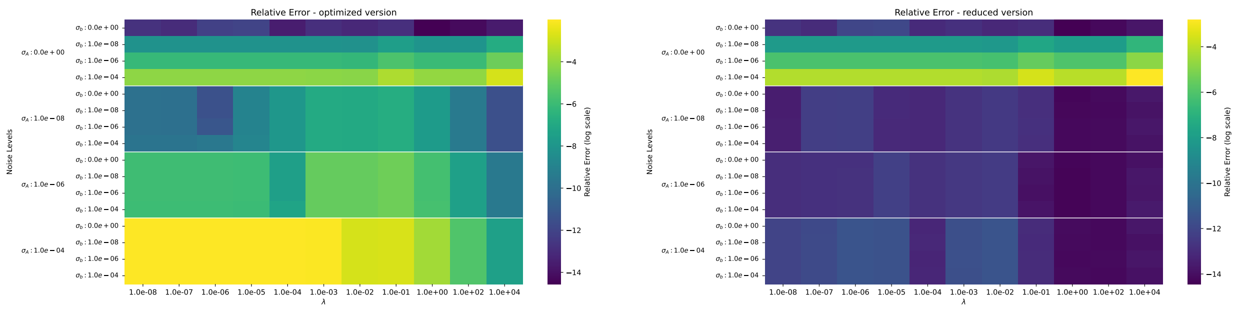

Figure 8: Heatmaps showing the behaviour of the relative error across different noise levels (σ_A, σ_b) and λ values. Results are reported for both versions, *reduced* (left) and *optimized* (right). Values are reported on a logarithmic scale, with darker colors indicating lower values and lighter colors indicating higher values. Lower values correspond to better performance.

The corresponding numerical values for factorization error, relative error, and residual norm are reported in Tables 6, 5, and 7 in the Appendix, respectively.

Results and Discussion

From the results reported in the heatmaps in Figures 8, 9, and 10, it can be observed that adding noise to $\hat{A}$ and $\hat{b}$ affects the considered metrics in different ways, as expected from the analysis reported in Section 6.3.

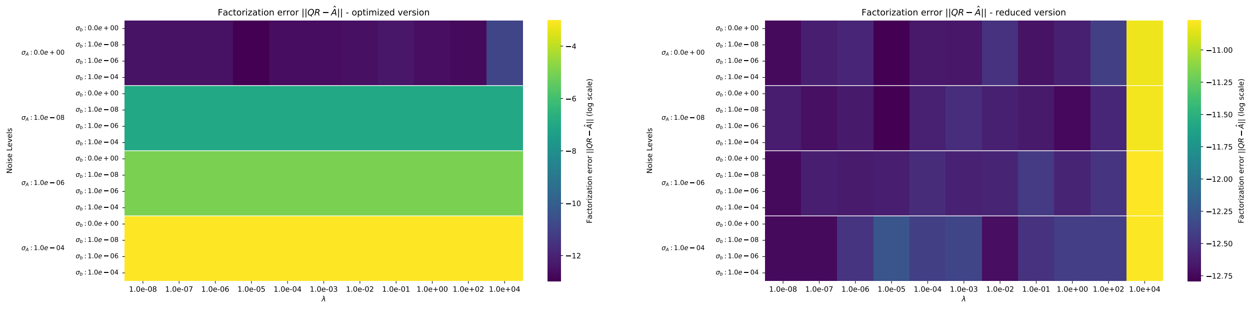

Figure 9: Heatmaps showing the behaviour of the factorization error across different noise levels (σ_A, σ_b) and λ values. Results are reported for both versions, *reduced* (left) and *optimized* (right). Values are reported on a logarithmic scale, with darker colors indicating lower values and lighter colors indicating higher values. Lower values correspond to better performance.

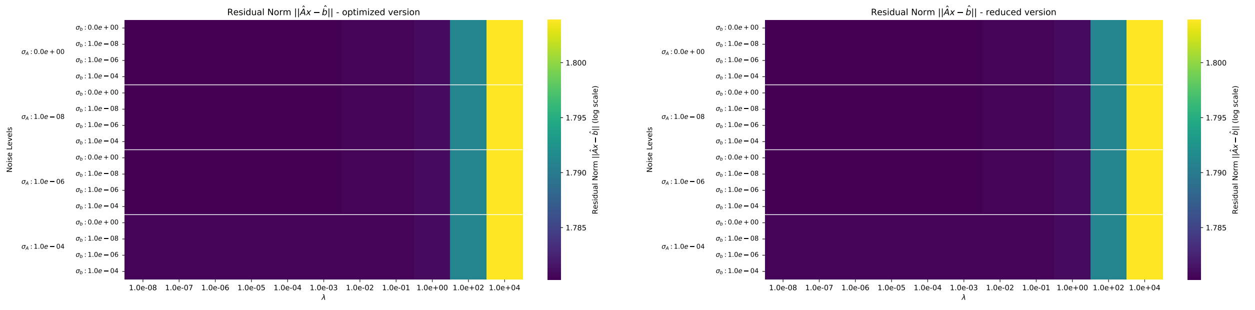

Figure 10: Heatmaps showing the behaviour of the residual norm across different noise levels (σ_A, σ_b) and λ values. Results are reported for both versions, *reduced* (left) and *optimized* (right). Values are shown on a logarithmic scale, with darker colors indicating lower values and lighter colors indicating higher values.

Regarding the *relative error*, for both versions the minimum values are obtained in the absence of noise. However, the error growth trends differ between the two implementations.

In the *optimized* version, increasing σ_A leads to a much more pronounced error growth compared to σ_b, indicating that noise in the matrix has a greater impact on the stability of the solution. Conversely, in the *reduced* version, the most evident increase in error occurs when noise is added to $\hat{b}$, with the lowest values reached when $\sigma_A = 0$, although an increase in error can still be observed when σ_A grows.

These observations, especially those regarding the *optimized* version, are in agreement with the theoretical analysis of the condition numbers reported in Section 6.3. In fact, the condition numbers with respect to perturbations in $\hat{A}$ are consistently higher than those related to $\hat{b}$, indicating that noise in the matrix has a stronger influence on the stability of the computed solution. This explains why, in the *optimized* version, the relative error is much more sensitive to σ_A than to σ_b.

Conversely, the *reduced* version exhibits a different trend, possibly due to its numerical structure, which may make it less sensitive to perturbations in $\hat{A}$ but slightly more to perturbations in $\hat{b}$.

Furthermore, the theoretical curves in Figure 3, in Section 6.3, indicate that the sensitivity to noise is λ-dependent, with minima in the condition numbers occurring around $\lambda \approx 10^2$. This is consistent with the experimental patterns observed in the heatmaps.

Figures 9 show the effect of noise on performance when measured through the *factorization error.* Also in this case, both versions achieve their minimum values in the absence of noise.

For the *optimized* version, the error increases with higher σ_A, while it remains essentially unchanged when noise is added to $\hat{b}$ with σ_A fixed. The effect of noise on $\hat{b}$ is marginal, as expected, since it does not directly alter the factorization of A.

The *reduced* version, on the other hand, does not show significant variations with the addition of noise, indicating that this implementation is more robust to perturbations.

Finally, the residual norm of the problem, $\|\hat{A}x - \hat{b}\|$, shown in Figures 10, remains approximately constant for a fixed value of λ, regardless of the amount of noise added. Moreover, both versions of the algorithm achieve essentially the same residual values, indicating that, despite the presence of noise, they converge to the same solution.

Overall, the results highlight that the robustness of the *optimized* method is mainly limited by perturbations in $\hat{A}$, while the effect of noise on $\hat{b}$ is comparatively smaller. In contrast, the *reduced* QR factorization shows greater stability to noise than the *optimized* version. Consequently, the *reduced* version may yield better relative and factorization error values for some noise configurations.

11 Scalability Analysis with Increasing Matrix Size

In this set of experiments, the scalability of the two QR-based solvers, *optimized* and *reduced*, was evaluated in terms of execution time and memory usage as the size of matrix A increased.

The problem formulation follows the definition in Section 2, with the augmented matrix $\hat{A} \in \mathbb{R}^{(m+n)\times n}$ constructed using a fixed regularization parameter $\lambda = 10^{-2}$.

The experiments were conducted with matrices $A \in \mathbb{R}^{m\times n}$ having a fixed number of columns $n = 15$ and a number of rows m ranging over the set:

$$m \in \{15, 50, 100, 150, 200, 250, 400, 800, 1000, 1500, 2000, 5000, 10000\}.$$

For each size m, A was generated with entries drawn from a standard normal distribution.

The right-hand side vector b was constructed following the same random generation process described in Section 6, with nonzero entries sampled from a standard normal distribution $\mathcal{N}(0, 1)$.

A random seed was set in order to perform, for each size m, the experiments for the two versions under the same conditions, providing the same $\hat{A}$ and $\hat{b}$ as input.

Execution time and memory usage were measured for each solver and compared against an ideal linear growth reference line to assess scalability. Figures 11 and 12 report the results for execution time and memory usage, respectively.

Numerical values for all the metrics introduced in Section 7 for the experiments described in this section are reported in Tables 8 and 9 in the Appendix.

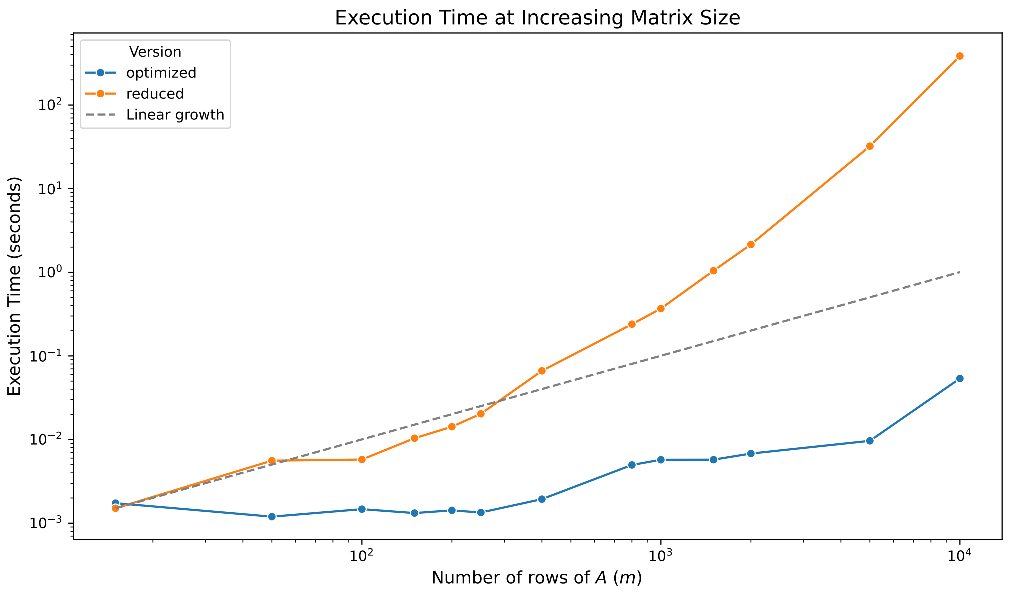

Figure 11: Execution time of the *optimized* and *reduced* QR-based solvers as a function of the number of rows m of matrix A. The experiments were performed with matrices A of fixed number of columns $n = 15$ and increasing number of rows, generated with random entries. Both axes are in logarithmic scale. The dashed gray line represents ideal linear growth, included as a reference for scalability assessment.

Results and Discussion

From the execution time plots in Figure 11, it is clear that the *optimized* version is consistently faster than the *reduced* one for all tested values of m. The *optimized* implementation shows a sub-linear growth compared to the reference linear growth line, whereas the *reduced* version exhibits linear or slightly sub-linear growth only for m values up to the order of 10^2. The performance gap between the two methods increases as the matrix size grows, highlighting the better scalability of the *optimized* implementation.

Regarding memory usage, the *reduced* version consistently requires less memory than the *optimized* one, in line with its structure, which avoids explicitly storing the entire Q matrix. Also in this test, the difference becomes more pronounced as the matrix size increases.

In conclusion, the scalability tests confirm the expected outcome: the *optimized* version achieves better performance in terms of both execution time and memory usage, scaling more efficiently with the size of A.

12 Conclusions

In this report, we analyzed two different QR-based solvers for the minimization of linear least squares problems, each relying on a different implementation of the QR factorization.

The first solver makes use of a classical thin QR factorization with Householder reflectors, *reduced* version. Subsequently, an optimized variant was implemented, exploiting the specific structure of the matrix $\hat{A}$, called *optimized*.

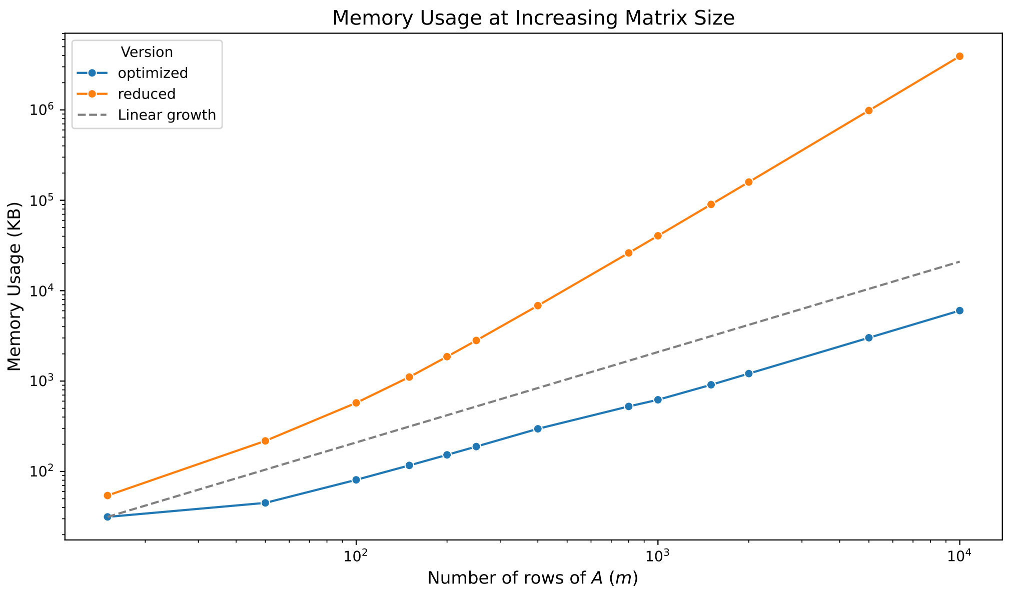

Figure 12: Memory usage of the *optimized* and *reduced* QR-based solvers as a function of the number of rows m of matrix A. The experiments were performed with matrices A of fixed number of columns $n = 15$ and increasing number of rows, generated with random entries. Both axes are in logarithmic scale. The dashed gray line represents ideal linear growth, included as a reference for scalability assessment.

Several experiments were conducted to analyze and compare the performance of the two implementations. In the first set of experiments, we compared the performance of the two versions using different values of λ. Then, a noise sensitivity analysis was performed to evaluate how both solvers respond to noise. Finally, a scalability analysis was carried out to study how the computational cost of the two algorithms scales with the problem size.

Experiments comparing the two version with varying λ demostrate that the *optimized* solver consistently outperforms the *reduced* one in terms of efficiency, achieving runtimes about one order of magnitude smaller. Both methods, however, achieve very small errors: relative errors remain on the order of between 10^{-12} and 10^{-15}, with oscillations for $\lambda > 10^{-1}$ that stabilize as conditioning improves for larger λ. Factorization errors are always below 10^{-11}, confirming numerical accuracy, and the residual norm is identical for both solvers, showing that they converge to the same minimizer despite differences in computational performance.

The noise sensitivity analysis shows that both solvers achieve their lowest errors in the absence of noise, but respond differently to perturbations: the *optimized* version is much more sensitive to noise in $\hat{A}$, while the *reduced* version is more affected by noise in $\hat{b}$ but shows overall greater robustness to matrix perturbations. Factorization errors confirm this trend, with the *reduced* solver being less influenced by noise, while the residual norm remains stable across all cases, showing that both solvers converge to solutions of comparable quality.

The scalability analysis reveals that the *optimized* version is consistently faster than the *reduced* one, independently of the problem size. Moreover, the execution time of the *optimized* solver grows sub-linearly with respect to the problem size, whereas the *reduced* solver scales linearly up to medium-sized problems and then exhibits super-linear growth for very large dimensions.

Regarding memory usage, the *optimized* version requires less memory overall and scales more efficiently with the size of A.

In conclusion, the results consistently show that the *optimized* solver is the most efficient in terms of runtime and scalability, achieving faster execution and lower memory costs across problem sizes, though at the expense of greater sensitivity to perturbations. Conversely, the *reduced* solver, despite being less efficient computationally, offers stronger robustness to noise and stable accuracy, making it a more reliable choice in settings where data are affected by perturbations.

References

Lloyd N. Trefethen and David Bau, III. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997. doi: 10.1137/1.9780898719574. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898719574`.

Appendix

File/Folder	**Description**
`main.py`	Entry point for running the experiments.
`src/solver.py`	Implementation of the QR-based solvers.
`src/qr_factorization.py`	QR factorization implementations.
`src/dataloader.py`	Functions for loading and preparing the input data.
`src/utils/metrics.py`	Functions for computing performance metrics, as described in Section 7.
`src/utils/matrix_reconstruction.py`	Functions for reconstructing matrices from QR factorization, used to assess the quality of the factorization.
`src/utils/results_visualization.py`	Utilities for plotting and visualizing results.
`src/experiments/`	Scripts to run experiments and specific tests.
`CUP/`	Folder containing the **ML-CUP** dataset.
`figures/`	Folder containing plots and figures used in the report.
`notebooks/`	Jupyter notebooks for data analysis and visualization.
`results/`	CSV files with experiment outputs.
`README.md`	Project description and instructions to run experiments.

Table 2: Overview of the main source code files and folders.

λ	Time (s)	Memory (kB)	Relative Error	$\|\|QR - \hat{A}\|\|$	$\|\hat{A}x - \hat{b}\|$
1.000e-08	0.020503	2813.375000	1.826e-13	1.820e-13	60.293601
2.593e-08	0.018780	2812.812500	7.518e-13	1.905e-13	60.293601
6.723e-08	0.020028	2812.812500	4.414e-13	2.268e-13	60.293601
1.743e-07	0.018113	2812.812500	1.517e-13	1.918e-13	60.293601
4.520e-07	0.019003	2812.812500	2.564e-13	2.379e-13	60.293601
1.172e-06	0.017542	2812.812500	4.893e-13	1.749e-13	60.293601
3.039e-06	0.022047	2812.812500	6.589e-13	2.915e-13	60.293601
7.880e-06	0.022333	2812.812500	2.762e-14	1.878e-13	60.293601
2.043e-05	0.020564	2812.812500	3.058e-13	2.278e-13	60.293602
5.298e-05	0.019450	2812.812500	6.102e-13	2.311e-13	60.293608
1.374e-04	0.020151	2812.812500	1.192e-13	3.041e-13	60.293645
3.562e-04	0.023624	2812.812500	4.325e-13	1.707e-13	60.293893
9.237e-04	0.023636	2812.812500	3.720e-13	1.996e-13	60.295462
2.395e-03	0.022975	2812.812500	1.167e-12	1.784e-13	60.302797
6.210e-03	0.025216	2812.812500	6.654e-13	2.250e-13	60.315848
1.610e-02	0.027810	2812.812500	1.186e-13	2.891e-13	60.321888
4.175e-02	0.028500	2812.812500	8.065e-14	2.298e-13	60.323298
1.083e-01	0.026007	2812.812500	1.078e-14	1.767e-13	60.324614
2.807e-01	0.028526	2812.812500	7.070e-15	2.068e-13	60.331603
7.279e-01	0.025435	2812.998047	5.970e-15	1.578e-13	60.363792
1.887e+00	0.022243	2812.812500	7.048e-15	1.851e-13	60.431718
4.894e+00	0.024613	2812.812500	5.423e-15	2.773e-13	60.517183
1.269e+01	0.024138	2812.812500	7.442e-15	2.190e-13	60.677451
3.290e+01	0.024767	2812.812500	8.237e-15	2.445e-13	61.027126
8.532e+01	0.027754	2812.812500	6.878e-15	2.473e-13	61.670439
2.212e+02	0.024749	2812.812500	2.830e-15	4.163e-13	62.578008
5.736e+02	0.023102	2812.812500	9.726e-16	9.005e-13	63.371388
1.487e+03	0.025698	2812.812500	1.939e-15	2.673e-12	63.614915
3.857e+03	0.025216	2812.812500	4.592e-15	4.501e-12	63.657206
1.000e+04	0.025489	2812.812500	1.713e-14	1.502e-11	63.663654

Table 3: Experimental results using the *reduced* version of the QR factorization. For each value of λ, execution time, memory usage, relative error, factorization error, and residual norm are reported.

λ	Time (s)	Memory (kB)	Relative Error	$\|\|QR - \hat{A}\|\|$	$\|\hat{A}x - \hat{b}\|$
1.000e-08	0.001397	199.007812	1.778e-13	3.379e-13	60.293601
2.593e-08	0.001197	187.796875	7.522e-13	2.995e-13	60.293601
6.723e-08	0.001198	187.750000	4.434e-13	2.851e-13	60.293601
1.743e-07	0.007019	187.750000	2.576e-13	2.879e-13	60.293601
4.520e-07	0.001563	187.750000	2.150e-13	2.467e-13	60.293601
1.172e-06	0.001130	187.750000	4.884e-13	1.883e-13	60.293601
3.039e-06	0.001165	187.750000	5.509e-13	3.735e-13	60.293601
7.880e-06	0.001908	187.750000	1.485e-13	2.256e-13	60.293601
2.043e-05	0.001935	187.750000	3.011e-13	5.779e-13	60.293602
5.298e-05	0.001240	187.750000	6.142e-13	4.743e-13	60.293608
1.374e-04	0.001234	187.750000	2.087e-13	2.030e-13	60.293645
3.562e-04	0.001143	187.750000	2.801e-13	2.190e-13	60.293893
9.237e-04	0.001122	187.750000	7.363e-13	4.013e-13	60.295462
2.395e-03	0.001488	187.750000	1.169e-12	2.395e-13	60.302797
6.210e-03	0.001157	187.750000	3.846e-13	2.959e-13	60.315848
1.610e-02	0.001422	187.750000	4.220e-13	1.916e-13	60.321888
4.175e-02	0.001275	187.750000	4.340e-13	2.818e-13	60.323298
1.083e-01	0.001120	187.750000	1.275e-14	1.984e-13	60.324614
2.807e-01	0.001099	187.750000	7.458e-15	2.084e-13	60.331603
7.279e-01	0.001177	187.750000	4.549e-15	2.927e-13	60.363792
1.887e+00	0.001100	187.750000	6.487e-15	4.017e-13	60.431718
4.894e+00	0.001105	187.750000	5.540e-15	4.657e-13	60.517183
1.269e+01	0.001120	187.750000	6.974e-15	2.313e-13	60.677451
3.290e+01	0.002245	187.750000	8.349e-15	3.003e-13	61.027126
8.532e+01	0.001225	187.750000	7.274e-15	2.527e-13	61.670439
2.212e+02	0.001135	187.750000	2.720e-15	3.521e-13	62.578008
5.736e+02	0.001115	187.750000	1.810e-15	7.411e-13	63.371388
1.487e+03	0.001171	187.750000	3.607e-15	2.682e-12	63.614915
3.857e+03	0.001648	187.750000	9.054e-15	4.480e-12	63.657206
1.000e+04	0.001145	187.750000	2.183e-14	1.225e-11	63.663654

Table 4: Experimental results using the *optimized* version of the QR factorization. For each value of λ, execution time, memory usage, relative error, factorization error, and residual norm are reported.

		λ	1.0e-08	1.0e-07	1.0e-06	1.0e-05	1.0e-04	1.0e-03	1.0e-02	1.0e-01	1.0e+00	1.0e+02	1.0e+04
version	σ_A	σ_b											
optimized	0.0e+00	0.0e+00	1.778e-13	1.017e-13	7.744e-13	9.835e-13	2.720e-14	1.407e-13	7.070e-14	1.443e-13	2.786e-15	6.139e-15	2.183e-14
		1.0e-08	6.935e-09	6.935e-09	6.934e-09	6.934e-09	6.916e-09	6.748e-09	5.656e-09	2.769e-08	8.633e-09	8.434e-09	1.527e-07
		1.0e-06	6.935e-07	6.935e-07	6.935e-07	6.933e-07	6.916e-07	6.748e-07	5.656e-07	2.769e-06	8.633e-07	8.434e-07	1.527e-05
		1.0e-04	6.935e-05	6.935e-05	6.935e-05	6.933e-05	6.916e-05	6.749e-05	5.655e-05	2.769e-04	8.634e-05	8.434e-05	1.528e-03
	1.0e-08	0.0e+00	1.191e-10	1.077e-10	3.534e-12	1.113e-09	1.220e-08	1.139e-07	1.093e-07	1.649e-07	1.846e-08	3.512e-10	3.319e-12
		1.0e-08	1.191e-10	1.076e-10	3.564e-12	1.113e-09	1.220e-08	1.139e-07	1.093e-07	1.649e-07	1.846e-08	3.512e-10	3.318e-12
		1.0e-06	1.161e-10	1.047e-10	6.559e-12	1.116e-09	1.220e-08	1.139e-07	1.093e-07	1.649e-07	1.846e-08	3.512e-10	3.317e-12
		1.0e-04	1.855e-10	1.967e-10	3.067e-10	1.414e-09	1.250e-08	1.142e-07	1.096e-07	1.649e-07	1.846e-08	3.513e-10	3.317e-12
	1.0e-06	0.0e+00	1.202e-06	1.201e-06	1.190e-06	1.079e-06	3.076e-08	1.028e-05	1.077e-05	1.649e-05	1.846e-06	3.512e-08	3.322e-10
		1.0e-08	1.202e-06	1.201e-06	1.190e-06	1.079e-06	3.076e-08	1.028e-05	1.077e-05	1.649e-05	1.846e-06	3.512e-08	3.322e-10
		1.0e-06	1.201e-06	1.200e-06	1.189e-06	1.078e-06	3.106e-08	1.028e-05	1.077e-05	1.649e-05	1.846e-06	3.512e-08	3.322e-10
		1.0e-04	1.171e-06	1.170e-06	1.159e-06	1.048e-06	6.126e-08	1.031e-05	1.081e-05	1.650e-05	1.846e-06	3.513e-08	3.323e-10
	1.0e-04	0.0e+00	9.791e-03	9.791e-03	9.790e-03	9.783e-03	9.708e-03	8.454e-03	1.507e-03	1.681e-03	1.847e-04	3.512e-06	3.322e-08
		1.0e-08	9.791e-03	9.791e-03	9.790e-03	9.783e-03	9.708e-03	8.454e-03	1.507e-03	1.681e-03	1.847e-04	3.512e-06	3.322e-08
		1.0e-06	9.791e-03	9.790e-03	9.790e-03	9.783e-03	9.708e-03	8.454e-03	1.507e-03	1.681e-03	1.847e-04	3.512e-06	3.322e-08
		1.0e-04	9.776e-03	9.775e-03	9.775e-03	9.768e-03	9.693e-03	8.439e-03	1.503e-03	1.682e-03	1.847e-04	3.512e-06	3.323e-08
reduced	0.0e+00	0.0e+00	1.826e-13	1.005e-13	9.243e-13	1.143e-12	8.705e-14	1.410e-13	6.963e-14	1.069e-13	3.618e-15	6.231e-15	1.713e-14
		1.0e-08	6.935e-09	6.935e-09	6.934e-09	6.934e-09	6.916e-09	6.748e-09	5.656e-09	2.769e-08	8.633e-09	8.434e-09	1.527e-07
		1.0e-06	6.935e-07	6.935e-07	6.935e-07	6.933e-07	6.916e-07	6.748e-07	5.656e-07	2.769e-06	8.633e-07	8.434e-07	1.527e-05
		1.0e-04	6.935e-05	6.935e-05	6.935e-05	6.933e-05	6.916e-05	6.749e-05	5.655e-05	2.769e-04	8.634e-05	8.434e-05	1.528e-03
	1.0e-08	0.0e+00	3.120e-14	4.757e-13	5.197e-13	8.393e-14	6.987e-14	1.787e-13	2.946e-13	1.284e-13	5.234e-15	7.029e-15	1.979e-14
		1.0e-08	3.234e-14	4.730e-13	5.211e-13	8.357e-14	6.988e-14	1.758e-13	2.940e-13	1.338e-13	5.622e-15	6.687e-15	1.345e-14
		1.0e-06	3.262e-14	4.747e-13	5.197e-13	8.318e-14	6.960e-14	1.765e-13	2.941e-13	1.409e-13	6.229e-15	6.918e-15	1.791e-14
		1.0e-04	3.252e-14	4.744e-13	5.193e-13	8.146e-14	6.948e-14	1.776e-13	2.945e-13	1.380e-13	6.472e-15	6.897e-15	1.363e-14
	1.0e-06	0.0e+00	1.220e-13	1.496e-13	1.552e-13	5.868e-13	1.683e-13	2.833e-13	3.808e-13	1.618e-14	5.334e-15	6.678e-15	1.187e-14
		1.0e-08	1.202e-13	1.522e-13	1.576e-13	5.881e-13	1.647e-13	2.839e-13	3.803e-13	1.349e-14	4.715e-15	6.591e-15	1.810e-14
		1.0e-06	1.232e-13	1.504e-13	1.562e-13	5.880e-13	1.672e-13	2.818e-13	3.919e-13	1.038e-14	4.593e-15	6.555e-15	1.645e-14
		1.0e-04	1.225e-13	1.521e-13	1.579e-13	5.867e-13	1.698e-13	2.837e-13	3.901e-13	1.418e-14	4.809e-15	6.464e-15	2.239e-14
	1.0e-04	0.0e+00	7.999e-13	1.402e-12	3.604e-12	3.120e-12	6.139e-14	2.357e-12	3.818e-12	9.644e-14	7.885e-15	6.252e-15	1.815e-14
		1.0e-08	8.009e-13	1.410e-12	3.639e-12	3.126e-12	7.423e-14	2.361e-12	3.805e-12	9.082e-14	7.004e-15	6.392e-15	1.163e-14
		1.0e-06	7.948e-13	1.407e-12	3.628e-12	3.118e-12	6.058e-14	2.367e-12	3.817e-12	9.873e-14	6.768e-15	6.910e-15	1.618e-14
		1.0e-04	7.993e-13	1.406e-12	3.608e-12	3.128e-12	6.487e-14	2.350e-12	3.812e-12	9.753e-14	6.966e-15	6.621e-15	1.224e-14

Table 5: Numerical values of the relative error for all tested noise levels (σ_A, σ_b) and λ values.

		λ	1.0e-08	1.0e-07	1.0e-06	1.0e-05	1.0e-04	1.0e-03	1.0e-02	1.0e-01	1.0e+00	1.0e+02	1.0e+04
optimized	0.0e+00	0.0e+00	3.379e-13	3.014e-13	3.011e-13	1.068e-13	2.316e-13	2.369e-13	2.719e-13	4.232e-13	2.495e-13	1.996e-13	1.225e-11
		1.0e-08	3.379e-13	3.014e-13	3.011e-13	1.068e-13	2.316e-13	2.369e-13	2.719e-13	4.232e-13	2.495e-13	1.996e-13	1.225e-11
		1.0e-06	3.379e-13	3.014e-13	3.011e-13	1.068e-13	2.316e-13	2.369e-13	2.719e-13	4.232e-13	2.495e-13	1.996e-13	1.225e-11
		1.0e-04	3.379e-13	3.014e-13	3.011e-13	1.068e-13	2.316e-13	2.369e-13	2.719e-13	4.232e-13	2.495e-13	1.996e-13	1.225e-11
	1.0e-08	0.0e+00	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08
		1.0e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08
		1.0e-06	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08
		1.0e-04	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08	9.577e-08
	1.0e-06	0.0e+00	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06
		1.0e-08	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06
		1.0e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06
		1.0e-04	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06	9.577e-06
	1.0e-04	0.0e+00	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04
		1.0e-08	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04
		1.0e-06	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04
		1.0e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04	9.577e-04
reduced	0.0e+00	0.0e+00	1.820e-13	2.383e-13	2.632e-13	1.619e-13	2.194e-13	2.141e-13	3.154e-13	2.049e-13	2.446e-13	3.830e-13	1.502e-11
		1.0e-08	1.820e-13	2.383e-13	2.632e-13	1.619e-13	2.194e-13	2.141e-13	3.154e-13	2.049e-13	2.446e-13	3.830e-13	1.502e-11
		1.0e-06	1.820e-13	2.383e-13	2.632e-13	1.619e-13	2.194e-13	2.141e-13	3.154e-13	2.049e-13	2.446e-13	3.830e-13	1.502e-11
		1.0e-04	1.820e-13	2.383e-13	2.632e-13	1.619e-13	2.194e-13	2.141e-13	3.154e-13	2.049e-13	2.446e-13	3.830e-13	1.502e-11
	1.0e-08	0.0e+00	2.299e-13	1.948e-13	2.221e-13	1.619e-13	2.482e-13	2.921e-13	2.435e-13	2.206e-13	1.774e-13	2.658e-13	1.561e-11
		1.0e-08	2.299e-13	1.948e-13	2.221e-13	1.619e-13	2.482e-13	2.921e-13	2.435e-13	2.206e-13	1.774e-13	2.658e-13	1.561e-11
		1.0e-06	2.299e-13	1.948e-13	2.221e-13	1.619e-13	2.482e-13	2.921e-13	2.435e-13	2.206e-13	1.774e-13	2.658e-13	1.561e-11
		1.0e-04	2.299e-13	1.948e-13	2.221e-13	1.619e-13	2.482e-13	2.921e-13	2.435e-13	2.206e-13	1.774e-13	2.658e-13	1.561e-11
	1.0e-06	0.0e+00	1.814e-13	2.401e-13	2.262e-13	2.409e-13	2.869e-13	2.473e-13	2.588e-13	3.643e-13	2.588e-13	3.263e-13	1.692e-11
		1.0e-08	1.814e-13	2.401e-13	2.262e-13	2.409e-13	2.869e-13	2.473e-13	2.588e-13	3.643e-13	2.588e-13	3.263e-13	1.692e-11
		1.0e-06	1.814e-13	2.401e-13	2.262e-13	2.409e-13	2.869e-13	2.473e-13	2.588e-13	3.643e-13	2.588e-13	3.263e-13	1.692e-11
		1.0e-04	1.814e-13	2.401e-13	2.262e-13	2.409e-13	2.869e-13	2.473e-13	2.588e-13	3.643e-13	2.588e-13	3.263e-13	1.692e-11
	1.0e-04	0.0e+00	1.812e-13	1.838e-13	3.280e-13	5.505e-13	3.867e-13	4.201e-13	1.937e-13	3.134e-13	3.777e-13	3.794e-13	1.658e-11
		1.0e-08	1.812e-13	1.838e-13	3.280e-13	5.505e-13	3.867e-13	4.201e-13	1.937e-13	3.134e-13	3.777e-13	3.794e-13	1.658e-11
		1.0e-06	1.812e-13	1.838e-13	3.280e-13	5.505e-13	3.867e-13	4.201e-13	1.937e-13	3.134e-13	3.777e-13	3.794e-13	1.658e-11
		1.0e-04	1.812e-13	1.838e-13	3.280e-13	5.505e-13	3.867e-13	4.201e-13	1.937e-13	3.134e-13	3.777e-13	3.794e-13	1.658e-11

Table 6: Numerical values of the factorization error for all tested noise levels (σ_A, σ_b) and λ values.

		λ	1.0e-08	1.0e-07	1.0e-06	1.0e-05	1.0e-04	1.0e-03	1.0e-02	1.0e-01	1.0e+00	1.0e+02	1.0e+04
optimized	0.0e+00	0.0e+00	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
	1.0e-08	0.0e+00	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
	1.0e-06	0.0e+00	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
	1.0e-04	0.0e+00	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
reduced	0.0e+00	0.0e+00	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
	1.0e-08	0.0e+00	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
	1.0e-06	0.0e+00	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.029e+01	6.030e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
	1.0e-04	0.0e+00	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-08	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-06	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01
		1.0e-04	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.032e+01	6.038e+01	6.180e+01	6.366e+01

Table 7: Numerical values of the residual norm for all tested noise levels (σ_A, σ_b) and λ values.

m	Time (s)	Memory (kB)	Relative Error	$\|\|QR - \hat{A}\|\|$	$\|\|\hat{A}x - \hat{b}\|\|$
15	0.001507	54.197266	4.967e-15	7.878e-15	0.154926
50	0.005593	217.929688	1.864e-15	1.767e-14	6.053295
100	0.005766	574.054688	1.820e-15	3.509e-14	8.667522
150	0.010378	1108.031250	1.425e-15	3.858e-14	10.951310
200	0.014238	1863.109375	1.039e-15	4.369e-14	12.554129
250	0.020336	2813.562500	1.668e-15	4.789e-14	14.757725
400	0.066336	6836.804688	1.471e-15	6.549e-14	18.695068
800	0.239584	26159.312500	2.244e-15	1.018e-13	27.654088
1000	0.368349	40508.562500	1.894e-15	1.381e-13	30.767335
1500	1.045602	90051.781250	1.804e-15	1.511e-13	38.230543
2000	2.150609	159125.039062	1.846e-15	2.291e-13	43.957457
5000	32.253042	983727.281250	1.014e-15	4.385e-13	70.288600
10000	385.773162	3920563.242188	2.159e-15	4.323e-13	100.305820

Table 8: Experimental results for the *reduced* QR factorization with increasing problem size. Matrices A have a fixed number of columns $n = 15$ and an increasing number of rows m, with entries generated from a standard normal distribution. The regularization parameter λ was fixed at 10^{-2}.

m	Time (s)	Memory (kB)	Relative Error	$\|\|QR - \hat{A}\|\|$	$\|\|\hat{A}x - \hat{b}\|\|$
15	0.001735	31.441406	6.275e-15	7.571e-15	0.154926
50	0.001197	44.929688	1.460e-15	1.227e-14	6.053295
100	0.001471	80.804688	1.701e-15	2.323e-14	8.667522
150	0.001321	116.742188	1.197e-15	2.057e-14	10.951310
200	0.001424	152.679688	8.986e-16	3.932e-14	12.554129
250	0.001342	188.679688	1.655e-15	3.725e-14	14.757725
400	0.001938	296.554688	1.341e-15	4.164e-14	18.695068
800	0.004969	524.320312	2.571e-15	7.057e-14	27.654088
1000	0.005735	621.195312	2.076e-15	1.076e-13	30.767335
1500	0.005748	909.960938	1.642e-15	1.310e-13	38.230543
2000	0.006803	1210.742188	1.941e-15	1.874e-13	43.957457
5000	0.009656	3015.429688	1.869e-15	3.527e-13	70.288600
10000	0.053716	6023.242188	1.471e-15	2.591e-13	100.305820

Table 9: Experimental results for the *optimized* QR factorization with increasing problem size. Matrices A have a fixed number of columns $n = 15$ and an increasing number of rows m, with entries generated from a standard normal distribution. The regularization parameter λ was fixed at 10^{-2}.