UNIVERSITÀ DI PISA

Master Degree in Computer Science

Project 22 - QR factorization

Computational Mathematics for Learning and Data Analysis

Report

Francesco Botrugno 545713 — Master Degree in Computer Science
Francesco Caprari 580154 — Master Degree in Computer Science

2023/2024

Contents

1 Introduction

1.1 The Problem

In this report, we will analyze a linear least squares problem with a specific form:

$$\min_{w} ||\hat{X}w - \hat{y}|| \tag{P}$$

the norm is $\|\cdot\|_2$. The matrix $\hat{X}$ is defined as following:

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

Where X is a tall and thin matrix with dimensions $m \times n$, $\lambda > 0$, while $\hat{y}$ has the following form:

$$\hat{y} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

Where y is a random vector.

The resulting matrix $\hat{X}$ has dimensions $r \times m$ where $r = n + m$

$$\hat{X} = \begin{bmatrix} X^T \\ \lambda I \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \\ \lambda_{n+1,1} & 0 & \cdots & 0 \\ 0 & \lambda_{n+2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{r,m} \end{bmatrix}$$

The idea is to use the thin QR factorization with Householder matrices to solve the linear system and also develop an alternative algorithm that achieves the same results as the thin QR but leverages the structure of the matrix, particularly the scaled identity sub-matrix, with quadratic complexity in m. In the next section, we describe the algorithms we used.

1.2 Notation

We define a generic matrix as $X \in \mathbb{R}^{m \times n}$. We define a generic vector as $x \in \mathbb{R}^m$. When we use square brackets associated with a matrix $X[i]$, we refer to the i-th column of the matrix, while with this notation, $X[i : m \ ; \ i : n]$, we indicate the submatrix that takes the specified range of rows and columns within square brackets. When we use the symbol $^{(i)}$ as a superscript of a vector or matrix, we refer to that matrix or vector at the i-th iteration, e.g. $R^{(i)}, u^{(i)}$. When we use $\dot{R}$, we refer to the submatrix of an original matrix R on which we are going to operate

1.3 Thin QR factorization with Householder Reflectors

A matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$, can be factorized into $A = Q_1 R_1$ where $Q_1 \in \mathbb{R}^{m \times n}$ is orthogonal where, $Q = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} \in \mathbb{R}^{m \times m}$, $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \in \mathbb{R}^{m \times n}$ with $R_1 \in \mathbb{R}^{n \times n}$ upper triangular.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} = \underbrace{\begin{bmatrix} q_{11} & q_{12} & \dots & q_{1n} \\ q_{21} & q_{22} & \dots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \dots & q_{mn} \end{bmatrix}}_{Q} \times \underbrace{\begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ 0 & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & r_{nn} \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}}_{R}$$

By implementing the thin QR, we reduce the computational cost by looping only over the number of columns of the matrix. We have also applied the usual optimizations related to the calculation of the matrix R and addressed the issue of cancellation when creating the Householder matrix. In the thin QR that we implemented, we do not directly compute the matrix Q, instead, we store the vectors u from the Householder matrix, which we then use later to calculate the matrix Q. This approach reduces the computational cost of multiplying the matrices $Q^{(i)}$ and $Q^{(i+1)}$.

1.4 Variant of thin QR factorization with Householder Reflectors

In the variant of the algorithm we implemented, we took advantage of the scaled identity matrix λI, leveraging the sparsity of the lower part of the matrix. Indeed, we can reduce the size of the R matrix involved in the multiplication reducing the number of rows, and we can do the same for the calculation of Q. This is possible because, at each iteration, we always have the same number of non-zero elements to zero out.

Given the initial matrix $\hat{X}$:

$$\hat{X} = \begin{bmatrix} X^T \\ \lambda I \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \\ \lambda_{n+1,1} & 0 & \cdots & 0 \\ 0 & \lambda_{n+2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{r,m} \end{bmatrix} = R^{(0)}$$

To obtain the final R upper triangular matrix, we start from the first column $\hat{X}[1]$ where we need to transform into zeros all the values, except the first one. Since our

column contains already an high number of zeros we can avoid to operate on the whole column: all the values below λ are already equal to zero, so we take into consideration only the first $n+1$ elements. At each step k we need to obtain the householder vector $u^{(k)}$ starting from the matrix column vector $R^{(k-1)}[k]$, and then find the new $R^{(k)}$ matrix. The fundamental step we can exploit is the multiplication between the $u^{(k)}$ and the matrix of the previous iteration $R^{(k-1)}$, where we can reduce the complexity by limiting our calculation on a subsection of the matrix R and a subsection of the householder vector u.

Example

- **Iteration 1** $k = 1$

  We define the section of $R^{(0)}$ as:

  $$\dot{R}^{(0)} = R^{(0)}[1 : (n+1); 1 : m]$$

  We take the first column vector:

  $$\dot{X}[1] = \dot{R}^{(0)}[1] = \begin{bmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \\ \lambda_{n+1,1} \end{bmatrix}$$

  From which we can obtain the householder vector $u^{(1)}$ with the same size $n+1$:

  $$u^{(1)} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \\ u_{n+1} \end{bmatrix}$$

  To obtain the new matrix $R^{(1)}$, to zero out the element in the first column, we don't need to change the whole matrix, we can restrict only to the non zero elements of the upper part, hence until row $n+1$ which contains λ.

  $$\dot{R}^{(1)} = R^{(1)}[1 : (n+1); 1 : m]$$

  We update the section of $R^{(1)}$ for the next iteration as:

  $$\dot{R}^{(1)} = \dot{R}^{(0)} - 2u^{(1)} \cdot \left(u^{(1)}\right)^T \cdot \dot{R}^{(0)}$$

  After performing the above operations, we obtain the new matrix $R^{(1)}$ as follows:

$$R^{(1)} = \begin{bmatrix} x'_{1,1} & x'_{1,2} & x'_{1,3} & \cdots & x'_{1,c} \\ 0 & x'_{2,2} & x'_{2,3} & \cdots & x'_{2,c} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & x'_{n,2} & x'_{n,3} & \cdots & x'_{n,c} \\ 0 & x'_{n+1,2} & x'_{n+1,3} & \cdots & x'_{n+1,c} \\ 0 & \lambda_{n+2,2} & 0 & \cdots & 0 \\ 0 & 0 & \lambda_{n+3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_{r,c} \end{bmatrix}$$

- **Iteration 2** $k = 2$

  We define the section of $R^{(1)}$ for this iteration as:

  $$\dot{R}^{(1)} = R^{(1)}[2 : (n+2); 2 : m]$$

  We obtain the new householder vector:

  $$\dot{R}^{(1)}[2] = \begin{bmatrix} x'_{2,2} \\ \vdots \\ x'_{n+1,2} \\ \lambda_{n+2,2} \end{bmatrix} \quad u^{(2)} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \\ u_{n+1} \end{bmatrix}$$

  In the second iteration the first row (and also the first column) of $R^{(1)}$ shall remain unchanged, consequently the element $x'_{1,2}$ is not involved in the process, but now we have an additional element $x'_{n+1,2}$ that in the previous iteration was a zero, so the number of elements of the householder vector is $(n+1) - 1 + 1$.

  We redefine the range of $R^{(2)}$ to update:

  $$\dot{R}^{(2)} = R^{(2)}[2 : (n+2); 2 : m]$$

  We update the section of $R^{(2)}$ for the next iteration as:

  $$\dot{R}^{(2)} = \dot{R}^{(1)} - 2u^{(2)} \cdot (u^{(2)})^T \cdot R^{(1)}$$

More in general:

- **Iteration** k **with** $1 \leq k \leq m$

  $$\dot{R}^{(k-1)} = R^{(k-1)}[k : n+k; k : m]$$

  $$\dot{R}^{(k-1)}[k] = \begin{bmatrix} x'_{k,k} \\ \vdots \\ x'_{n+(k-1),k} \\ \lambda_{n+k,k} \end{bmatrix} \quad u^{(k)} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \\ u_{n+1} \end{bmatrix}$$

$$\dot{R}^{(k)} = R^{(k)}[k : (n+k); k : m]$$

And calculate the matrix for the next iteration as:

$$\dot{R}^{(k)} = \dot{R}^{(k-1)} - 2u^{(k)} \cdot \left(u^{(k)}\right)^T \cdot \dot{R}^{(k-1)}$$

All the submatrices $\dot{R}$ on which we work at each iteration will have $n+1$ rows, while the number of columns decreases. Therefore, we achieve an improvement over the classical algorithm.

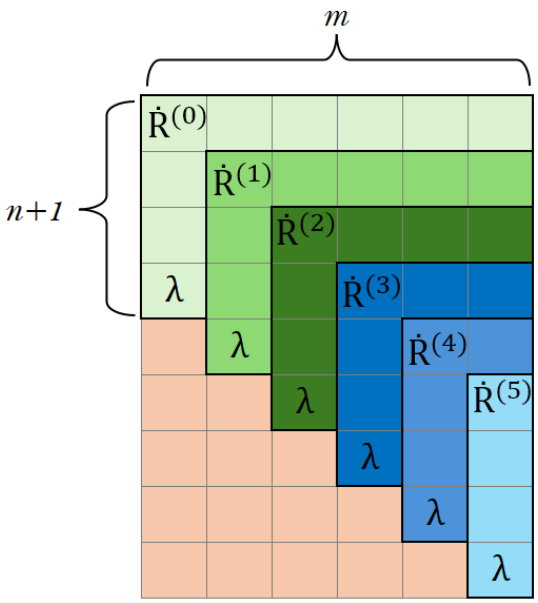

Figure 1: Example of $\dot{R}$ shape at each iteration

Similarly to how the computational cost is reduced when calculating the triangular matrix R, we can also reduce the computational cost when calculating the matrix Q. The matrix Q is initialized as the identity matrix, in each iteration, we update a smaller submatrix respect to the original QR. The size of the submatrix we update in each iteration is the same as for R, with $n+1$ rows, while the number of columns varies depending on the iteration.

1.5 Solve the system

After applying the factorization then we solve the new system:

$$||\hat{X}w - \hat{y}|| = ||QRw - \hat{y}|| = ||Q^T(QRw - \hat{y})|| = ||Rw - Q^T\hat{y}||$$

$$= \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \mathbf{w} - \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \hat{\mathbf{y}} \right\| = \left\| \begin{bmatrix} R_1 w - Q_1^T \hat{\mathbf{y}} \\ Q_2^T \hat{\mathbf{y}} \end{bmatrix} \right\|$$

$$R_1 w - Q_1^T \hat{y} = 0$$
$$c = Q_1^T \hat{y}$$
$$R_1 \mathbf{w} = c$$

Therefore, **w** will be the solution to the original problem. To solve the previous system, it is necessary that the matrix R_1 is invertible. R_1 is invertible if A has full column rank.

Proof. Regardless of whether we use the thin QR or one of its variants, we always obtain the factorization of $A = QR$.

$A^T A = (QR)^T (QR) = R^T Q^T Q R = R_1^T R_1$

If A is full column rank $\rightarrow$ $A^T A$ is positive definite $\rightarrow$ $R_1^T R_1$ is invertible $\rightarrow$ R_1 is invertible □

Once the solution is found, looking at the residual, we cannot say with certainty how close it is to x^* the solution of the problem.

2 What to expect from the algorithm(s)

The following algorithm 1 was used to compute the vector u for constructing the Householder matrix. To avoid numerical instability caused by subtracting the first element from the vector's norm, we change the sign of the latter if $x[1] \geq 0$.

Algorithm 1 Compute Householder vector

Input: x
$s \leftarrow ||x||$
if $x[1] \geq 0$ **then**
  $s \leftarrow -s$
end if
$v \leftarrow x$
$v[1] \leftarrow v[1] - s$
$u \leftarrow \frac{v}{||v||}$
return $[u, s]$

2.1 Thin QR factorization

Algorithm 2 Compute thinQR factorization

Input: A
$[r, m] \leftarrow size(A)$
$R \leftarrow A$
$Q = I_{m \times n}$
for $k \leftarrow 1$ to $min(r-1, m)$ **do**
  $[u, s] \leftarrow householder(R[k : end, k])$
  $R[k, k] \leftarrow s$
  $R[k+1 : end, k] \leftarrow 0$
  $R[k : end, k+1 : end] \leftarrow R[k : end, k+1 : end) - 2 \cdot u \cdot (u' * R[k : end, k+1 : end])$
  $ulist[k] \leftarrow u$
end for
for $k \leftarrow min(r-1, m)$ to 1 **do**
  $Q[k : end, k : end] = Q[k : end, k : end] - 2 \cdot ulist[k] \cdot (ulist[k]' \cdot Q[k : end, k : end])$
end for
return $[Q, R(1 : n, 1 : n)]$

In the QR factorization process, when computing the orthogonal matrix Q and the triangular matrix R, we do not directly construct the Householder matrices $\mathbf{H}_k$. Instead, we implicitly perform the product using the vector $\mathbf{u}$ associated with each Householder transformation this reduces the computational cost from cubic to quadratic.

The reason we start the construction of Q from the transformations $\mathbf{Q}_n \mathbf{Q}_{n-1} \dots \mathbf{Q}_1$ is that this approach allows us to involve only a small part of the vector at each step, if this sparsity property is exploited, a speed-up is achieved.[1]

Additionally, we limit ourselves to calculating only the first n reduced vectors, as these are sufficient to represent the matrix Q in its reduced form, further reducing computational costs without compromising the Error of the solution.

2.1.1 Complexity

Given a matrix A of dimension $r \times m$. The cost of computing the Householder vector is determined by the norm calculation, so for each iteration, we will have a cost of $O(r-k+1)$. Then we have the cost of multiplying R by the Householder vector, which will have a cost of $O((r-k+1) \times (m-k))$. $\sum_{k=1}^{min(r,m)} (r-k+1)(m-k) = O(rm^2)$. For Q, the cost is very similar to that of R, since the dimensions of the matrices involved are the same as those of R. Since the cost is dominated by multiplication to calculate R and Q, the total cost of the algorithm is therefore $O(rm^2)$, which is cubic in the number of columns, because $r \approx m$, $r = m + n$ and so the complexity is $O(m^3)$.

2.1.2 Stability

The *thinQR* is stable, in fact we can prove that $\tilde{Q}\tilde{R} = qr(A + \Delta_A)$, so $\tilde{Q}\tilde{R} = \hat{A} = A + \Delta$, $||\Delta|| = O(u)||A||$.

Proof. At each step of the algorithm, we view each calculation as a perturbation of A: $\tilde{R_0} = A$

$\tilde{R_1} = \tilde{Q_1} \cdot \tilde{R_0} = \tilde{Q_1}(A + \Delta_1)$

$\tilde{R_2} = \tilde{Q_2} \cdot \tilde{R_1} = \tilde{Q_2}(\tilde{R_1} + \Delta_2) = \tilde{Q_2}(\tilde{Q_1}(A + \Delta_1) + \Delta_2)$

This is true because at each step when we compute R, we have new perturbation with norm $||\Delta_k|| = O(u)||\tilde{R_{k-1}} = O(u)(||A|| + O(u))$. □

2.2 Variant QR factorization

Algorithm 3 Compute Variant thin QR factorization

Input: A
$[r, m] \leftarrow size(A)$
$R \leftarrow A$
$Q = I_{r \times m}$
$n \leftarrow (r - m)$
for $k \leftarrow 1$ to $min(r - 1, m)$ **do**
  $[u, s] \leftarrow householder(R[k : n + k, k])$
  $R[k, k] \leftarrow s$
  $R[k + 1 : n + k, k] \leftarrow 0$
  $R[k : n+k, k+1 : end] \leftarrow R[k : n+k, k+1 : end] - 2 \cdot u \cdot (u' \cdot R[k : n+k, k+1 : end])$
  $ulist[k] \leftarrow u$
end for
$i \leftarrow 0$
for $k \leftarrow min(r - 1, m)$ to 1 **do**
  $Temp \leftarrow ulist(k)' \cdot Q[k : end - i, k : end]$
  $Q[k : end - i, k : end] \leftarrow Q[k : end - i, k : end] - 2 \cdot ulist(k) \cdot Temp$
end for
return $[Q, R(1 : n, 1 : n)]$

In this variant of the thin QR, we apply the same improvements that we have already considered in the classical algorithm. The main difference lies in the fact that now the matrices R and Q that we update during the iterative process have a smaller size, as shown in section 1.4, which allows us to achieve computational improvements.

2.2.1 Complexity

Given a matrix A of dimension $r \times m$. We start from $k = 1$ to m, so we repeat the entire calculation m times. At each step we compute the norm of the householder vector of size $n + 1$ which costs $O(n + 1)$. In addition, we have the product between the householder vector and the matrix $\dot{R}$ of size $(n + 1) \times (m - k)$ (see Figure 1). The computational cost of this multiplication is given by $\sum_{k=1}^{m}(n + 1)(m - k)$ which corresponds to $O(m((n + 1)(m - k)) \leq O(nm^2)$. Given $n \ll m$ we can approximate $O(nm^2) \simeq O(m^2)$. Also for the calculation of matrix Q, iterating from $k = \ m$ *to* 1,

considering that the matrices involved have the same size, we obtain the same cost $O(nm^2)$. Therefore, the total computational cost of this variant is $2m^2{=}O(m^2)$.

2.2.2 Stability

The variant of the factorization algorithm is still stable because the algorithm is essentially identical to the classical one, except that the matrices R and Q to be multiplied are of smaller dimensions. Even though we are updating only a part of the matrix at each iteration, the orthogonal properties of the matrices Q_k and the triangular form of the matrices R_k ensure that the algorithm remains numerically stable. The perturbations are limited and the growth of rounding errors is controlled, ensuring the stability of the entire iterative process.

2.3 Solving the system

After calculating the factorization, it is sufficient to perform a multiplication and solve a triangular system to obtain the solution to the original problem 1.1.

Algorithm 4 Compute the Result of linear system system

Input: X, y
$[Q, R] \leftarrow thinQR(X)$
$c \leftarrow Q' \cdot y$
$w \leftarrow R \backslash c$
return w

This procedure is backward stable because, as we have previously demonstrated, both the thin QR factorization and its variant that exploits the structure of the matrix are backward stable and the subsequent two operations are also backward stable, this is because the product of an orthogonal matrix Q and a vector y is stable and the same applies to solving a triangular system. As for the cost, $c = Q' \cdot y$ has a cost equal to $O(rm)$, while the solution of the system $Rw = y$ it has a cost equal to $O(m^2)$. Thus, in the end, the cost is dominated by the QR factorization.
One way to increase the speed of solving the system is to iteratively compute the product Uy, leveraging the Householder vectors within the QR factorization process of matrix A. This way, we won't return the vector Q but will directly obtain the vector $c = Qy$ [2]. Taking into consideration the Variant QR show previously, this is the modified algorithm 5.

Algorithm 5 Compute Qy and R

Input: A, y
$[r, m] \leftarrow size(A)$
$R \leftarrow A$
$Qy = y$
$n \leftarrow r - m$
for $k \leftarrow 1$ to $min(r-1, m)$ **do**
  $[u, s] \leftarrow householder(R[k : n + k, k])$
  $R[k, k] \leftarrow s$
  $R[k + 1 : n + k, k] \leftarrow 0$
  $Temp \leftarrow u' \cdot R[k : n + k, k + 1 : end]$
  $R[k : n + k, k + 1 : end] \leftarrow R[k : n + k, k + 1 : end] - 2 \cdot u \cdot Temp$
  $Qy[k : n + k] \leftarrow Qy[k : n + k] - 2 \cdot u(\cdot u' \cdot Qy[k : n + k])$
end for
return $[Qy, R]$

In this way, we save space because we don't need to store the matrix Q directly, and we also save time because we directly compute the product $Q \cdot y$, which is needed later to solve the original linear system. In this case, we also take advantage of the special structure of the original matrix and thus of the Householder vectors u we obtain, in order to reduce the portion of the resulting vector $Q \cdot y$ that we update at each iteration.

3 Experimental set-up

3.1 Input

The matrix we are working with has dimensions of 1013×1000. In the upper submatrix X^T, with size 13×1000, all values are non-zero. Specifically, the values in the first 12 rows range between -2 and 1, while row 13 can contain larger values. Meanwhile, the second submatrix, of size 1000×1000, is a scaled identity matrix. Given that our matrix $\hat{X}$ has slightly more rows than columns, implementing a thin QR factorization instead of the traditional full QR can lead to a smaller computational cost and memory savings. The matrix $\hat{X}$ that we construct, regardless of the chosen λ, is a full column rank matrix. This ensures the uniqueness and existence of the solution, as we have already demonstrated in the previous section 1.5.

3.2 Experiments

First, we tested both algorithms for QR factorization on the original input dataset, generating randomly the lambda value and calculating the relative Error of the factorization and the time taken for the calculation in seconds. After the initial test, we evaluated the algorithms for thin QR factorization by testing them on matrices of different sizes that resembled the structure of the main matrix considered in the experiment. For the different sizes, we calculated the error and the time required to perform the factorization. Another experiment we conducted was testing different values of lambda for the scaled identity matrix to see if we obtained different Error values, finally, we

did tests on the calculation of the solution for the original least squares problem. The tests were conducted using and implementing the algorithms in **MATLAB**. In order for the experiment to be reproducible, we set the same seed 42, for the generation of random numbers.

4 Results

For both algorithms that we have implemented, we calculate the time taken to compute the thin QR factorization and the Error of the factorization as $\frac{norm(A-QR)}{norm(A)}$

Algortihm	**Error**	**Time (seconds)**	**Distance**
Variant QR factorization	6.554e−16	0.25	3.951e−17
Thin QR factorization	6.557e−16	5.99	3.926e−17

Table 1: Results of the Algorithm on the original dataset

As can be seen from the table 1, using the algorithm variant, we achieve an error very similar to that of the classical algorithm, both still close to machine precision, but we take less time for the factorization. Even considering the distance in norm between the error obtained from the matlab solution and our solutions, we see that it is still in the order of the machine number.

	Error		
Lambdas	**Thin QR**	**Variant thin QR**	**Matlab thin QR**
1e5	1.737e−15	1.737e−15	2.699e−15
1e3	1.511e−15	1.495e−15	2.187e−15
1e−2	7.312e−16	7.266e−16	9.209e−16
1e−4	7.580e−16	7.586e−16	6.592e−16
1e−7	3.727e−16	3.606e−16	3.867e−16

Table 2: Results of the various algorithm on different lambda values

Varying the value of λ in the scaled matrix doesn't actually make much difference; in both algorithms, we achieve an error that reaches machine precision, regardless of the different λ values.

	Thin QR	
Dimension	**Error**	**Time (seconds)**
1200×1000	2.642e−16	7.93
1300×1000	4.278e−16	9.44
1500×1000	3.123e−16	12.19

Table 3: Results of the classic thin qr on different dimension dataset

Variant Thin QR

Dimension	Error	Time (seconds)
1200×1000	2.617e−16	1.53
1300×1000	4.326e−16	3.04
1500×1000	3.069e−16	5.10

Table 4: Results of the classic thin qr on different dimension dataset

By testing the algorithms on datasets of different sizes, we notice that the error is similar, while instead the execution time gets significantly better using the variant of the algorithm that takes advantage of the particular matrix structure. The more you vary the size of the rows the more efficient the variant of the algorithm succeeds, this is because if the number of rows is very similar to the number of columns, the classic thin qr algorithms will have a cost similar to calculating the full factorization of the matrix. So in an IOT system there is a need for a range of technologies for crop monitoring and resource management such as water, nutrients, and pesticides, trying to minimize resource waste. Sensors will then be used to measure environmental parameters of the greenhouse, we often find thermometers to measure temperature, sensors to measure air humidity and global radiation.

4.1 Results on the solution of the least squares problem

To measure the accuracy of the solution of the quadratic problem, we considered the calculation of the residual $\frac{||\hat{X}w-\hat{y}||}{||\hat{y}||}$. In addition to this, we also checked the relative distance in norm 2 between the direct solution obtained from $w = \hat{X}\backslash\hat{y}$ and the solution $\dot{w}$ obtained from the others methods.

Solution Results

Algorithms	Residual	Distance	Gradient	Time (seconds)
Thin QR	0.18023	3.8126e−13	$9.579e-12$	4.99
Variant QR	0.18023	3.8036e−13	$1.005e-11$	0.12
Variant Direct QR	0.18023	3.8297e−13	$1.619e-11$	0.06

Table 5: Results of the system solution

As can be seen from the table 5, all the tested algorithms produce the same residual, which indicates that regardless of the technique used, the quality of the solution remains the same. Taking instead the distance in norm between the direct solution and the solutions obtained by the other methods, we can see that the results are all in the order of 10^{-13}. Another way to measure the quality of the solution is to look at the value of the norm of the gradient, $\nabla f(w) = \hat{X}^T\hat{X}w - \hat{X}y$. The closer the value approaches zero the closer we get to the minimum of the function, we see how in our case by calculating the norm of the gradient with the various solutions we approach a fairly low precision of 10^{-11}. In terms of execution time, we have observed a clear improvement with the two variants of the algorithm compared to the traditional thin QR method. This was expected, as the primary cost in solving the system lies in the factorization process

The residual is not very close to 0 even if we take the direct solution of the system, this is because the system is overdetermined, since the $\hat{X}$ matrix is rectangular.

5 Files Description

The code for the implementation is written in matlab, the files are divided into folders. The Dataset folder contains the *csv* file used to construct the matrix on which the experiments were performed. In the folder Utils there are

- *Matrices.m*: builds the X matrix and the y vector.
- *Solvefun.m* solves the least squares problem with the different methods and returns the result for each method.

In the Algorithms Folder there are four files:

- *householder_ vector.m*: which return the householder vectors.
- *ThinQR.m*: contains the implementation of the classical thin QR.
- *VariantThinQR.m*: contains the implementation of the variant of the algorithm.
- *VariantThinQRDirect.m*: contains the implementation of the second variant of the algorithm.

In the *Experiments\QR*: folder there are the scripts and functions use to test the accuracy of the factorization done with the various algorithms.

- *Standard.m*: tests the accuracy starting from the Original matrix while also checking the time taken.
- *DifferentLambdas.m*: controls the error of various methods by varying the lambda value of the scaled identity matrix.
- *MoreRows.m*: check the error of the factorization by varying the rows of the matrix.

In the *Experiments\SystemSolution* folder are the files that test the error and the time it took to get the system solution.

- the *Standard.m*: calculates the residual obtained by the different methods and the time it took to find the solution.
- The *Gradient.m*: checks the value of the gradient.

References

[1] David Bau Lloyd N. Trefethen. *Numerical Linear Algebra*. SIAM, 1997. Chap. 10, pp. 69–75.

[2] David Bau Lloyd N. Trefethen. *Numerical Linear Algebra*. SIAM, 1997. Chap. 11, p. 83.