CM Project 18

A comparison between
the CG and QR methods

Giacomo Cignoni & Christian Peluso

A.Y. 2022/2023

Università di Pisa

January 14, 2024

Contents

1 Introduction

The aim of this project is to propose a solution of the following linear system, taken from the ML Cup by prof. Micheli:

$$\min_w ||\hat{X}w - y|| \tag{1}$$

where $\hat{X}$ is the matrix obtained by concatenating the (tall thin) matrix from the ML-cup dataset, called X, with a few additional columns containing functions of the features of the dataset, and y are the two output columns.

The first algorithm implemented and discussed is the **Conjugate Gradient Method**, which needs some initial conditions to be met, so the first section will discuss the stage setting. The second part of the report will regard the **QR Factorization with Householder reflectors** in the variant where one does not form the matrix Q, but stores the Householder vectors u_k and uses them to perform (implicitly) products with Q and Q^T.

2 Setting the Stage

2.1 From X to $\hat{X}$

Starting from the original $X \in \mathbb{R}^{(1491\times9)}$ matrix, we obtained the augmented matrix $\hat{X}$ by concatenating different combinations of columns. The first set of columns is composed by the square of the initial columns, adding 9 columns. The second set of columns is calculated by multiplying element-wise all of the possible combinations of two different initial columns, thus resulting in 36 additional columns.

In the experiments Section 5, we explore the performance of Conjugate Gradient with the original X and concatenating the two sets of columns, both separately and together. We thus have 4 variants of $\hat{X}$, with the following (ordered) concatenations:

a) original matrix, $X \in \mathbb{R}^{(1491\times9)}$

b) original matrix + squares of columns, $\hat{X} \in \mathbb{R}^{(1491\times18)}$

c) original matrix + mutiplication of different columns, $\hat{X} \in \mathbb{R}^{(1491\times45)}$

d) original matrix + squares of columns + mutiplication of different columns, $\hat{X} \in \mathbb{R}^{(1491\times54)}$

For QR we only explore the variant of $\hat{X}$ comprising all the columns, but including an additional column per iteration.

2.2 Defining the problem

First thing, we assured that the initial CUP matrix X has full column rank, thus it is also valid for the altered $\hat{X}$, as the chosen transformations do not introduce any linear dependencies among columns.

2.2.1 Quadratic form

We recall that the problem at our hands is a typical least square problem, as it is defined as $\min_w ||\hat{X}w - y||_2$, but we chose to transform the initial problem into an equivalent form, in order to obtain the correct matricial properties for applying the needed optimization methods. The following transformation is applied:

$$\begin{aligned} &\min_w ||\hat{X}w - y||_2^2 \\ =&\min_w (\hat{X}w - y)^T(\hat{X}w - y) \\ =&\min_w w^T\hat{X}^T\hat{X}w - 2y^T\hat{X}w + y^Ty \\ =&\min_w f(w) \end{aligned} \tag{2}$$

The resulting formulation $f(w)$ is a quadratic function, with $\hat{X}^T\hat{X}$ clearly being a symmetric matrix. Then, our initial least square problem is equivalent to finding the minimum of the quadratic function $f(w)$.

2.2.2 Positive definiteness

Moreover, we can also affirm the propriety of positive definiteness for $\hat{X}^T\hat{X}$. As $\hat{X}$ it is a full-column rank matrix, for any vector $v \neq 0$, we have that $y := \hat{X}v \neq 0$. Then, for any $v \neq 0$, we have that:

$$\begin{aligned} &v^T\hat{X}^T\hat{X}v \\ =&(\hat{X}v)^T\hat{X}v \\ =&y^Ty \\ =&\sum_i y_i^2 > 0 \end{aligned} \tag{3}$$

, which is one of the definitions for a positive definite matrix.

2.2.3 Convexity

The property of convexity for our $f(w)$ function can be proved by exploiting the following property: a twice differentiable function f is convex if and only if its Hessian matrix $\nabla^2 f$ is positive semi-definite for all $x \in \mathbb{R}^n$.

In our case, the Hessian matrix is defined by $\nabla^2 f = \hat{X}^T \hat{X}$, which we proved with Eq. 3 to be positive definite. So we can finally say that the minimum w_{min} of our problem exists, it is unique and corresponds to the stationary point of $f(w)$.

Moreover, this means that finding $\min_w f(w)$ corresponds to solving:

$$\nabla f(w) = 0 \iff 2\hat{X}^T \hat{X} w - 2\hat{X} y = 0 \tag{4}$$

, thus allowing us to solve a LLS problem as a linear system.

3 Conjugate Gradient

3.1 Introduction

The Conjugate Gradient (CG) method is an iterative algorithm used to solve systems of linear equations, particularly symmetric positive definite systems. The primary objective of the CG method is to find the solution to a generic linear system $Ax = b$, where A is a symmetric positive definite matrix, x is the unknown vector, and b is the known right-hand side vector. We will first introduce a generative iterative method, to solve a linear system and express pros and cons of this, in order to refine it, and arrive to the formalization of the Conjugate Gradient method.

3.2 Steepest Descent

The problem becomes reducing at 0 the quadratic form $f(x)$ taking into consideration the negative direction hinted by the first derivative $f'(x)$ and considering the A to be a generic symmetric positive definite matrix, we have:

$$\begin{aligned}
f(x) &= \frac{1}{2}x^T Ax - b^T x + c && (5)\\
f'(x) &= \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b\\
e_{(i)} &= x_{(i)} - x && (6)\\
r_{(i)} &= -Ae_{(i)} && (7)\\
&= b - Ax_{(i)}\\
&= -f'(x_{(i)})
\end{aligned}$$

Bringing out the common $\frac{1}{2}(A^T + A)x = b$ we can see that the $\frac{1}{2}(A^T + A)$ is symmetric, so we just need the matrix to be positive definite.

3.2.1 Dimension of the step

Since we have the direction gathered by the residual, we need to know how big the step in this direction should be, following the line search procedure we find out that the step minimizes the error with the following update rule, at,

$$x_{(i+1)} = x_{(i)} + \alpha r_{(i)} \tag{8}$$

when the directional derivative $\frac{d}{d\alpha} f(x_{(1)}) = 0$, so by the chain rule we have that:

$$\begin{aligned} \frac{d}{d\alpha} f(x_{(1)}) &= f'(x_{(1)})^T \frac{d}{d\alpha} x_{(1)} \\ &= f'(x_{(1)})^T r_{(0)} \end{aligned} \tag{9}$$

Setting this expression to 0 we find out that $f'(x_{(1)})$ and $r_{(0)}$ are orthogonal, to determine α, note that $f'(x_{(1)}) = -r_{(1)}$ by def. of residual:

$$\begin{aligned} r_{(1)}^T r_{(0)} &= 0 \\ (b - Ax_{(1)})^T r_{(0)} &= 0 \\ (b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} &= 0 \\ (b - A(x_{(0)})^T r_{(0)} &= \alpha (Ar_{(0)})^T r_{(0)} \\ r_{(0)}^T r_{(0)} &= \alpha (r_{(0)})^T (Ar_{(0)}) \\ \alpha &= \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}} \end{aligned} \tag{10}$$

3.2.2 Generic Convergence of Steepest Descent

Right now we have the direction and the dimension of the step to take through it, but we don't know how much effort we will put in finding the stationary point x. In order to understand it we can use Eq. 8 and Eq. 7, that hint:

$$\begin{aligned} x_{(i+1)} &= x_{(i)} + \alpha r_{(i)} \\ e_{(i+1)} + x &= e_{(i)} + x + \alpha r_{(i)} \\ e_{(i+1)} &= e_{(i)} + \alpha r_{(i)} \end{aligned} \tag{11}$$

So we can understand that the update rule will affect only the error term and this can be solved directly if it could be expressed as an eigenvector of A with eigenvalue $\lambda_{(e)}$. Then the residual $r_{(i)} = -Ae_{(i)} = \lambda_{(e)}e_{(i)}$ is also an eigenvector so the last Eq. 11 gives:

$$\begin{aligned} e_{(i+1)} &= e_{(i)} + \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}} r_{(i)} \\ &= e_{(i)} + \frac{r_{(0)}^T r_{(0)}}{\lambda_{(e)} r_{(0)}^T r_{(0)}} (-\lambda_{(e)} e_{(i)}) \end{aligned} \tag{12}$$

So choosing $\alpha_{(i)} = \lambda_{(e)}^{-1}$ gives us instant convergence, but in general we can express $e_{(i)}$ as a linear combination of eigenvectors v_j and we furthermore need them to be orthonormal:

$$e_{(i)} = \sum_{j=1}^{n} \xi_j v_j \tag{13}$$

,with ξ_j is the length of each component of the error at time (i).

$$r_{(i)} = -Ae_{(i)} = -\sum_j \xi_j \lambda_j v_j \tag{14}$$

The error can be erased, as above, if all the eigenvectors used to built it have a common eigenvalue. However, if there are several unequal, nonzero eigenvalues, then no choice of $\alpha_{(i)}$ will eliminate all the eigenvector components, and our choice becomes a compromise.

3.3 Conjugate Directions

Conjugate Directions method is based on the assumption of having a set of n orthogonal search directions, $d_{(1)}, ..., d_{(n)}$; for each search direction we will take exactly one step of the right length to line up with x, such that $e_{(i+1)}$ is orthogonal to $x_{(i)}$. This is expressed by:

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)} \tag{15}$$

, with $\alpha_{(i)} = -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}}$

The problem is that the choice of $\alpha_{(i)}$ depends on $e_{(i)}$, but knowing e in advance would already solve our problem.

The solution consists in using *conjugate* (*A-orthogonal*) directions $d_{(i)}$, such that $d_{(i)}^T A d_{(j)} = 0$. We require similarly that $e_{(i+1)}$ is *A-orthogonal* to $d_{(i)}$. In this case the value of α can be computed as it depends not on the error but on the residual:

$$\begin{aligned}\alpha_{(i)} &= -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} \\ &= \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}\end{aligned} \tag{16}$$

Note that if the search space were be the residual, this formula would be identical to the Steepest Descent. The advantages of this method are that only exactly n iterations are needed to obtain the solution x, thus having $e_{(n)} = 0$. To prove this we can express the error term as a linear combination of search directions:

$$e_{(0)} = \sum_{j=0}^{n-1} \delta_j d_{(j)} \tag{17}$$

The values of δ_j can be found using a mathematical trick, pre-multiplying the expression by $d_{(k)}^T A$:

$$\begin{aligned}d_{(k)}^T A e_{(0)} &= \sum_j \delta_{(j)} d_{(k)}^T A d_{(j)} \\ \delta_{(k)} &= \frac{d_{(k)}^T A e_{(0)}}{d_{(k)}^T A d_{(k)}} \\ &= \frac{d_{(k)}^T A(e_{(0)} + \sum_{i=0}^{k-1} \alpha_{(i)} d_{(i)})}{d_{(k)}^T A d_{(k)}} \\ &= \frac{d_{(k)}^T A e_{(k)}}{d_{(k)}^T A d_{(k)}}\end{aligned} \tag{18}$$

By the equation 16, we find that $\alpha_{(i)} = -\delta_{(i)}$. Thanks to this we can change our view of converging the error in 0 as a process of cutting down it component by component:

$$\begin{aligned} e_{(i)} &= e_{(i)} + \sum_{j=0}^{i-1} \alpha_{(j)} d_{(j)} \\ &= \sum_{j=0}^{n-1} \delta_{(j)} d_{(j)} - \sum_{j=0}^{i-1} \delta_{(j)} d_{(j)} \\ &= \sum_{j=i}^{n-1} \delta_{(j)} d_{(j)} \end{aligned} \tag{19}$$

It also can be demonstrated that Conjugate Directions always finds the best solution at every step within the bounds of where it is allowed to explore, that is minimizing $||e_{(i)}||_A$ within $e_{(0)} + \text{span}\{d_{(0)}, ..., d_{(i-1)}\}$ at step i.

In order to create the set of search space, initially the Gram-Schmidt conjugation was used, given a set of n linearly independent vectors $u_0, ..., u_{n-1}$ (e.g. the coordinate axes):

$$d_{(i)} = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)} \tag{20}$$

, where β_{ik} is defined similarly to $\delta_{(k)}$ in Eq. 18:

$$\beta_{ij} = -\frac{u_i^T A d_{(j)}}{d_{(j)}^T A d_{(j)}} \tag{21}$$

The problem with Gram-Schmidt is that we need to store all the old search vector in order to construct each new one, furthermore we need $O(n^3)$ operations to generate the full set (of A-orthogonal search directions). As a result the Conjugate Gradient was created in order to cure these disadvantages, so the Conjugate Gradient is no more than the Conjugate Directions with a fair search space set creator.

3.3.1 Conjugate Gradient

Conjugate gradient is based on the idea of using the residuals, because we obtain the following interesting property (by pre-multiplying Eq. 19):

$$\begin{aligned}
e_{(i)} &= \sum_{j=i}^{n-1} \delta_{(j)} d_{(j)} \\
-d_{(i)}^T A e_{(j)} &= -\sum_{j=i}^{n-1} \delta_{(j)} d_{(i)}^T A d_{(j)} \\
d_{(i)}^T r_{(j)} &= 0, \qquad i < j \\
r_{(i)}^T r_{(j)} &= 0, \qquad i \neq j
\end{aligned} \tag{22}$$

This means that each residual is orthogonal to all other residuals. We also know that:

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)} \tag{23}$$

, which means that each new residual $r_{(i+1)}$ is just a linear combination of the previous residual and $Ad_{(i)}$. Indeed, each subspace $D_{(i)}$ ($d_{(i-1)} \in D_{(i)}$) is a *Krylov subspace*, that is a subspace created by repeatedly applying a matrix (A) to a vector ($d_{(0)}$). This makes Gram-Schmidt conjugation easy; because $r_{(i+1)}$ is orthogonal to D_{i+1}, then it is also A-*orthogonal* to D_i (it is orthogonal to all search directions $d_{(j)}$, with $j < i$).

Using Eq. 23 and 21, we can derive the new β_{ij}:

$$\beta_{ij} = \begin{cases} -\frac{1}{\alpha_{(i-1)}} \frac{r_{(i)}^T r_{(i)}}{d_{(i-1)}^T A d_{(i-1)}} & i = j+1 \\ 0 & i > j+1 \end{cases} \tag{24}$$

It is easy to note that most of the β_{ij} term have disappeared, leaving only $\beta_{i,i-1}$; this reduces the complexity for each iteration from $O(n^2)$ to $O(m)$ (m being the number of nonzero entries of A). Thus, we can further simplify it to (abbreviation $\beta_{(i)} = \beta_{i,i-1}$):

$$\beta_{(i)} = \frac{r_{(i)}^T r_{(i)}}{r_{(i-1)}^T r_{(i-1)}} \tag{25}$$

Finally, we can simply summarize the method of Conjugate Gradient as follows:

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)} \tag{26}$$

$$d_{(0)} = r_{(0)} = b - Ax_{(0)} \tag{27}$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)} \tag{28}$$

$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{d_{(i-1)}^T A d_{(i-1)}} \tag{29}$$

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)} \tag{30}$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}} \tag{31}$$

3.3.2 Convergence Analysis

As clearly seen before, the Conjugate Gradient method converges in at most n steps, as n are the explored A-orthogonal search directions.

Recalling that the value $e_{(i)}$ at each step of Conjugate Gradient is chosen from $e_{(0)} + D_i$, for fixed i we have the following error term:

$$e_{(i)} = \left(I + \sum_{j=1}^{i} \psi_j A^j \right) e_{(0)} \tag{32}$$

, which comes from Krylov subspaces properties. Coefficients ψ_j are trivially related to $\alpha_{(i)}$ and $\beta_{(i)}$, CG chooses the ψ that minimize $||e_{(i)}||_A$.

Equation 32 can be also expressed using a polynomial inside the parenthesis; we use $P_i(\lambda)$ for a polynomial of degree i and the argument λ can be a matrix or scalar alike. This notation is useful in the case of multiplying eigenvectors v, as $P_i(A)v = P_i(\lambda)v$. We rewrite the error term as:

$$e_{(i)} = P_i(A) e_{(0)} \tag{33}$$

with $P_i(0) = 1$. Conjugate Gradient chooses this polynomial when it chooses coefficients ψ_j. Considering the initial error as a linear combination of orthogonal unit eigenvector, similar to Eq. 13, we have:

$$e_{(0)} = \sum_{j=1}^{n} \xi_j v_j \tag{34}$$

and applying the previous considerations on the polynomial we get:

$$\begin{aligned} e_{(i)} &= \sum_j \xi_j P_i(\lambda_j) v_j \\ Ae_{(i)} &= \sum_j \xi_j P_i(\lambda_j) \lambda_j v_j \\ ||e_{(i)}||_A^2 &= \sum_j \xi_j^2 (P_i(\lambda_j))^2 \lambda_j^2 \end{aligned} \tag{35}$$

At each step Conjugate Gradient finds the polynomial that minimizes this expression, but bearing in mind that the convergence speed is bounded by the convergence of the worst eigenvector. Having $\Lambda(A)$ being the eigenvalues of A, we can write this in mathematical terms as:

$$\begin{aligned} ||e_{(i)}||_A^2 &\leq \min_{P_1} \max_{\lambda \in \Lambda(A)} (P_i(\lambda_j))^2 \sum_j \xi_j^2 \lambda_j \\ &= \min_{P_1} \max_{\lambda \in \Lambda(A)} (P_i(\lambda_j))^2 ||e_{(0)}||_A^2 \end{aligned} \tag{36}$$

In other words, the convergence of Conjugate Gradient depends on how close the polynomial of degree i P_i can be to 0 on each eigenvalue of A. As a polynomial of degree n can fit at least $n+1$ points, P_n can accommodate the n separate eigenvalues (recalling also the constraint $P_i(0) = 1$), thus proving convergence in up to n steps and hinting a faster convergence with duplicated eigenvalues.

Equation 36 is minimized by choosing the following polynomial:

$$P_i(\lambda) = \frac{T_i(\frac{\lambda_{max}+\lambda_{min}-2\lambda}{\lambda_{max}-\lambda_{min}})}{T_i(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}})} \tag{37}$$

with T_i being the *Chebyshev polynomial* of grade i. Chebishev polynimials $T_i(\omega)$ oscillates between 1 and -1 in the interval defined by ω and grow as quickly as possible outside of it.

Thus, substituting in Eq. 36 (considering that the numerator maximum is 1 in $[\lambda_{min}, \lambda_{max}]$) we get:

$$||e_{(i)}||_A \leq \frac{1}{T_i\left(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}}\right)} ||e_{(0)}||_A \tag{38}$$

From this, it is easy to derive the most common form of Conjugate Gradient convergence:

$$||e_{(i)}||_A \leq \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i ||e_{(0)}||_A \tag{39}$$

that uses κ, the spectral condition number, instead of explicit maximum and minimum eigenvalues, $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$.

This proves that the error of Conjugate Gradient at step i is upper-bounded by the values of the minimum and maximum eigenvalues.

3.4 Our case

As we proved the existence and uniqueness of the minimum w_{min} of $f(w)$ in Section 2.2.3, together with affirming $\hat{X}^T\hat{X}$ positive definiteness in Section 2.2.2, we have the right preconditions for applying the conjugate gradient method to our problem.

In relation to previous sections explaining Conjugate Gradient and its convergence, we used the notation A to refer to the matrix $\hat{X}^T\hat{X}$, while the term b corresponds to $\hat{X}y$, for simplification sake.

4 QR factorization with Householder reflectors

The QR factorization is a fundamental matrix factorization technique used in numerical linear algebra. It decomposes a matrix A into the product of an orthogonal matrix Q and an upper triangular matrix R. We assume A to be a $m \times n$ matrix, with $m > n$.

Householder reflectors are a popular method for performing QR factorization due to their numerical stability and efficiency (slightly more efficient alternatives methods do exists such as Schwarz-Rutishauser Algorithm [1]).

4.1 Householder Reflectors

Householder reflectors are orthogonal transformations that are used to introduce zeros into a matrix, effectively reducing it to an upper triangular form. A Householder reflector is defined as:

$$H = I - 2\frac{vv^T}{v^Tv} \tag{40}$$

Where I is the identity matrix, v is a vector, and v^T is its transpose.

4.2 QR Factorization Using Householder Reflectors

The QR factorization of a matrix A using Householder reflectors is an iterative process that transforms A into an upper triangular matrix R by introducing zeros below the main diagonal. The orthogonal matrix Q is accumulated as a product of Householder reflectors. The algorithm proceeds as follows:

a) Start with the original matrix A.

b) For each column j from 1 to n:

   (a) Compute the Householder vector v_j to zero out the subdiagonal entries of the j-th column of A.

   (b) Form the Householder matrix H_j from v_j.

   (c) Update A as $A \leftarrow H_j A$ to eliminate the subdiagonal entries in the j-th column below the diagonal.

   (d) Update Q as $Q \leftarrow Q H_j$.

The final Q will be an orthogonal matrix, and R will be an upper triangular matrix such that $A = QR$.

4.3 Householder Vector and Matrix

To compute the Householder vector v_j for a given column of A, we can use the following formulas:

$$v_j = \begin{cases} \frac{A(j:m,j) - e_1 \|A(j,j)\|_2}{\|A(j:m,j)\|_2}, & \text{if } A(j,j) \geq 0 \\ \frac{A(j:m,j) + e_1 \|A(j,j)\|_2}{\|A(j:m,j)\|_2}, & \text{if } A(j,j) < 0 \end{cases} \tag{41}$$

Where e_1 is the first canonical basis vector. The Householder matrix H_j corresponding to v_j is then given by:

$$H_j = I - 2 v_j v_j^T \tag{42}$$

4.4 Tall thin QR

Since the original X is rectangular with $m >> n$, we can restrict our problem considering $Q_1 \in \mathbb{R}^{m \times n}, R_1 \in \mathbb{R}^{n \times n}$, changing the factorization in:

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1$$

This provides obvious advantages in speed and memory.

4.5 QR for least square

Extending the QR method for use in least square problems of the form $||Ax - b||$ is straightforward and based on the property of norm-2 preservation of orthogonal matrices such as Q:

$$\begin{aligned}
&||Ax-b|| \\
=&||Q^T(Ax-b)|| \\
=&||Q^TQRx-Q^Tb|| \\
=&||Rx-Q^Tb|| \\
=&||\begin{bmatrix} R_1x-Q_1^Tb \\ Q_2^Tb \end{bmatrix}||
\end{aligned} \tag{43}$$

Thus solving the initial least square problem is equivalent to solving $R_1x - Q_1^Tb$, as Q_2^Tb is a constant and thus the "optimum" residual of our optimization problem.

The solution is given by

$$x = R_1^{-1}Q_1^Tb \tag{44}$$

, which is possible when R_1 is invertible, that can be demonstrated being equivalent to A having full column rank.

4.6 Stability

In this section we expand on the stability and residual error of the Least Square problem solution obtained by QR factorization.

We start by recalling that for each $x \in [10^{-308}, 10^{308}]$ there exists exactly a representable number $\tilde{x}$ such that:

$$\frac{||\tilde{x}-x||}{||x||} \leq u = 2^{-52} \tag{45}$$

By definition, an algorithm that computes $y = f(x)$ is called backward stable if the computed output $\tilde{y}$ can be written as $\tilde{y} = f(\tilde{x})$, with $\tilde{x}$ defined as above.

We can define then the *forward error* as $\Delta_y = \tilde{y} - y$, while the *backward error* can be defined as the minimum perturbation $\Delta_x = \tilde{x} - x$ of x that can still return $\tilde{y}$ as output.

Thus we can say that an algorithm is backward stable if the backward error is small, more precisely, as already defined, $\frac{||\tilde{x}-x||}{||x||} = O(u)$.

After setting the theoretical basis, we can proceed to proving that QR factorization is a backward stable algorithm. Due to the iterative nature of the algorithm, we can just prove that each step consists of backward stable operations. This can be done by knowing that multiplying by an orthogonal matrix Q is a backward stable operation: $Q \odot A$ results in $QA + E$. E is the forward error and its norm is:

$$||E|| \leq O(u)||Q||\,||A|| = O(u)||A|| \tag{46}$$

We can also write $Q \odot A$ as $Q(A + Q^{-1}E)$, where the backward error $\Delta_A = Q^{-1}E$ satisfies:

$$||\Delta_A|| \le ||Q^{-1}||\,||E|| \le O(u)||A|| \tag{47}$$

Each step of the algorithm has the form: $Q_k A_{k-1} = A_k$ and thus is backward stable as it contains only a product with an orthogonal matrix. $\tilde{A}_k$ has accumulated all the errors of the previous $k-1$ iterations and satisfies:

$$\begin{aligned} \tilde{A}_k &= \tilde{Q}_k(\tilde{A}_{k-1} + \Delta_{k-1}) \\ \text{, with } \Delta_{k-1} &\le O(u)||\tilde{A}_{k-1}|| \end{aligned} \tag{48}$$

Thus, after n steps, $\tilde{R}$ is the exact result obtained if we started from:

$$A + \Delta_0 + \tilde{Q}_1^T\Delta_1 + ... + \tilde{Q}_1^T\tilde{Q}_2^T...\tilde{Q}_{n-1}^T\Delta_{n-1} \tag{49}$$

and each of the n error terms is $O(u)||\tilde{A}_{k-1}|| = O(u)||A||$. With this, we can finally say that the QR factorization is backward stable and the resulting $\tilde{Q}$ and $\tilde{R}$ are the exact results of the algorithm applied on $A + \Delta$, with $||\Delta|| \le O(u)||A||$.

4.6.1 Residual

After demonstrating backward stability in Section 4.6, it is easy to infer that this property also implies a small residual.

Assuming that $A = Q_1R_1$ is an exact thin QR factorization, let $r_1 = Q_1^T(A\tilde{x} - b)$. Then, $\tilde{x}$ is the exact solution of the Least Square problem:

$$\min ||Ax - (b + Q_1r_1)|| \tag{50}$$

Thus we have the following bound:

$$\frac{||\tilde{x} - x||}{||x||} \le \kappa_{rel}(LS, b)\frac{||Q_1r_1||}{||b||} = \kappa_{rel}(LS, b)\frac{||r_1||}{||b||} \tag{51}$$

where $\kappa_{rel}(LS, b)$ is the conditioning number of the Least Square problem in respect to b:

$$\kappa_{rel}(LS, b) \le \frac{\kappa(A)}{\cos\theta} \tag{52}$$

with $\cos\theta = \frac{||Ax||}{||b||}$.

We can finally say that, if our relative residual $\frac{||r||}{||b||}$ is of the order of machine precision, as it happens for backward stable algorithms, we get a solution whose accuracy is depending only on machine precision and the conditioning number:

$$\frac{||\tilde{x} - x||}{||x||} \leq \kappa O(u) \tag{53}$$

4.7 Computational Cost

In the case of a tall and thin matrix, we have a computational cost of the QR factorization of $2mn^2 - \frac{2}{3}n^3 + O(mn)$. The cost of solving the least square after having solved thin QR factorization is negligible as Equation 44 has cost $O(mn) + O(n^2)$.

4.7.1 Our Case

Similarly as the Conjugate Gradient Section 3.4, we used the same notation correspondence schema for explaining the QR method, in relation to our original problem.

5 Experiments

All the experiments were performed on a machine equipped with an INTEL Core i5-11400 @2.60GHz, 6 Core, 12 processors and 32 GB of RAM.

In order to compare and evaluate the improvements achieved by different manipulations of the initial matrix, we tested CG and QR with different variants of $\hat{X}$ as described in Section 2.1.

5.1 Implementation

To be able to gather the best possible and most theoretical in-line results, we have implemented the Conjugate Gradient and QR factorization with Householder-Reflector variant methods in MATLAB, keeping the theory steps illustrated in precedent Sections [3, 4].

To retrieve the optimal solution matrix $w^* \in \mathbb{R}^{(1491 \times 2)}$, we have adopted the standard system resolution `X\y`, thus permitting the comparison between our custom algorithms and the default MATLAB library. The stopwatch used to measure the time cost of algorithms is the `tic;toc;` function.

For the sake of transparency, we are reporting here the pseudo-code that summarize the Conjugate Gradient code in Algorithm 1 and the QR Factorization in Algorithm 2.

The core change in the CG algorithm compared to the Steepest Descent, as seen in the literature, is the computation of the α term that has the denominator composed of the direction instead of the residual in Line 11.

Algorithm 1 The algorithm tries its best to correct, the eigenvectors composing, the error in n iterations, where A is a $\mathbb{R}^{(n\times n)}$ matrix.

```
procedure ConjugateGradient($A, b$, Tol)
    $\alpha \leftarrow$ zeros($m, 1$)
    $\beta \leftarrow$ zeros($m, 1$)
    $r \leftarrow$ zeros($m, 1$)
    $d \leftarrow$ zeros($m, 1$)
    $x \leftarrow$ zeros($m, 1$)                ▷ The initialization has been performed using zeros
    while norm($r$) $>$ Tol & $t$ $< n$ do               ▷ Control iteration and residual
        $r_{(t+1)} \leftarrow r_{(t)} - \alpha_{(t)} * A * d_{(t)}$
        $\beta_{(t+1)} \leftarrow (r^T_{(t+1)} * r_{(t+1)})/(r^T_{(t)} * r_{(t)})$
        $d_{(t+1)} \leftarrow r_{(t+1)} + \beta_{(t+1)} * d_{(t)}$
        $\alpha_{(t+1)} \leftarrow (r^T_{(t+1)} * r_{(t+1)})/(d^T_{(t+1)} * A * d_{(t+1)})$

        $x_{(t+1)} \leftarrow x_{(t)} + \alpha_{(t+1)} * d_{(t+1)}$
    end while
    return $x$
end procedure
```

Algorithm 2 The algorithm build the householder vector and store it inside the upper part of the matrix Q, pre-multiplied by b; the solution is then computed by inverse formula, where A is a $\mathbb{R}^{(m\times n)}$ matrix and b is a $\mathbb{R}^{(mb\times nb)}$ matrix.

```
procedure QRFactorization($A$)      ▷ Factorize A into Q and R using Householder
    $Q \leftarrow b$
    for k = 1:n do                        ▷ Create the cols reflecting under the 1st elem
        $z \leftarrow A(k : m, k)$
        $v \leftarrow [-\text{sign}(z(1)) * \text{norm}(z) - z(1); -z(2 : \text{end})]$;
        $v \leftarrow v/\text{sqrt}(v' * v)$
        for j =1:n do
            $A(k : m, j) \leftarrow A(k : m, j) - v * (2 * (v' * A(k : m, j)))$
        end for
        for j = 1:nb do
            $Q(k : m, j) \leftarrow Q(k : m, j) - v * (2 * (v' * Q(k : m, j)))$
        end for
    end for
    $R = \text{triu}(A)$
    $R1 = R(1 : n, :)$
    return $x = R1\backslash(\text{eye}(n, m) * Q)$                ▷ Compute & return solution
end procedure
```

In the QR algorithm, it is important to notice that the first term of the reflector is computed by its inverse sign multiplied by the norm of the entire vector; this procedure is strategic to avoid its cancellation during the column multiplication. In addition, we are computing directly the Q matrix pre-multiplied with b, assigning directly b to the Q variable and then adding all the $Q_1, Q_2, Q_3, ...$ reflectors.

5.2 Results

This section illustrates the obtained experimental results, in terms of precision and performance.

Normalization For pagination reasons, we have not reported in the titles of table columns the values used for normalizing each measurement. Column values are nonetheless normalized. For further details you can check below how the values have been normalized (all of the following can be found in the subsequent tables):

- $||\boldsymbol{w} - \boldsymbol{w}^*||/|\boldsymbol{w}^*|$;
- $||\nabla \boldsymbol{f}(\boldsymbol{w})||/|\hat{\boldsymbol{X}}|$;
- $||\hat{\boldsymbol{X}}\boldsymbol{w} - \boldsymbol{y}||/|\boldsymbol{y}|$;
- $||\boldsymbol{R}_1\boldsymbol{w} - \boldsymbol{Q}_1^T\boldsymbol{y}||/|\boldsymbol{y}|$;

In Table 1 we can appreciate the resulting values for the Conjugate Gradient method; experiments were run with maximum number of iterations set as the number of columns of $\hat{X}$.

Similarly, we report the same scores for the Householder QR method in Table 2, we have tested different compositions of $\hat{X}$ matrix computing the result w by inverse formula, as showed in the pseudo code above.

For the sake of comparison, both the tables report the same measures: the difference with the optimal solution w^*, the derivative of the objective function $\nabla f(w)$, the difference of the $\hat{X}$ matrix multiplied by the calculated outcome with y and normalized by $|y|$ and, finally, the execution time computed by averaging 1000 runs. The only additional column in Table 2 is the upper block computation, in order to validate the theory in Equation 43.

Table 1: Results derived by the application of the Conjugate Gradient method for each of the 4 variants of $\hat{X}$. Iterations are the number of columns.

# Col.	$\|\|\boldsymbol{w} - \boldsymbol{w}^*\|\|$	$\|\|\nabla \boldsymbol{f}(\boldsymbol{w})\|\|$	$\|\|\hat{\boldsymbol{X}}\boldsymbol{w} - \boldsymbol{y}\|\|$	Ex. Time
9	4.525041e-06	1.324471e-03	5.178929e-06	47 μs
18	**1.004552e-08**	**3.652485e-07**	**4.505629e-10**	**103** μs
45	2.052699e-05	8.062929e-05	1.829220e-07	432 μs
54	5.070961e-06	1.843818e-05	4.558296e-08	581 μs

As in Table 1 the best trade-off between results and computation time is for sure the manipulated $\hat{X}^{(1491\times18)}$, original X appending the squared columns and normalizing. Thereafter, in the case where iterations are restricted to the number of columns, the obtained results are significantly inferior, also with regard to the execution time. We can assume that this is due to increase of complexity of error's eigenvalues.

A different situation presents itself when decomposing matrix $\hat{X}$ with QR factorization, which leaves less and less residual components in the lower part of the triangular matrix R

Table 2: Results derived by the application of the QR factorization with Householder Reflectors method for each of the 4 variants of $\hat{X}$.

# Col.	$\|\|\boldsymbol{w}-\boldsymbol{w}^*\|\|$	$\|\|\nabla \boldsymbol{f}(\boldsymbol{w})\|\|$	$\|\|\hat{\boldsymbol{X}}\boldsymbol{w}-\boldsymbol{y}\|\|$	$\|\|\boldsymbol{R}_1\boldsymbol{w}-\boldsymbol{Q}_1^T\boldsymbol{y}\|\|$	Ex. Time
9	5.469617e-16	2.897242e-13	8.642422e-01	1.706725e-16	471 μs
18	6.713549e-16	7.033553e-13	2.597722e-01	2.047380e-16	1651 μs
45	1.045869e-15	1.511098e-13	3.145363e-01	3.894392e-16	9458 μs
54	8.353062e-16	1.882877e-13	2.424056e-01	4.352274e-16	13636 μs

and orthogonal Q, with the increase in width. For the same reason, the difficulty to cancel out the error in the lower block lead to $||Q_2^T y||/|y|$ being nearly identical to $||\hat{X}w - y||/|y|$; thus we decided not to report those values on the Table 2, only in plots 2.

Hereinafter, Table 3 reports the costs (in μ seconds) and results for Conjugate Gradient method when a target tolerance is set, in order to better appreciate the capacities of CG to reach a certain precision without being iteration-bounded.

Table 3: The Conjugate Gradient algorithm is iterated until target tolerance is satisfied. Experiments are executed with $\hat{X}^{(1491\times 54)}$ matrix.

Tolerance	$\|\|\boldsymbol{w}-\boldsymbol{w}^*\|\|$	$\|\|\nabla \boldsymbol{f}(\boldsymbol{w})\|\|$	$\|\|\hat{\boldsymbol{X}}\boldsymbol{w}-\boldsymbol{y}\|\|$	Iterations	Ex. Time
1e-08	8.445535e-07	2.491341e-06	6.159104e-09	57	547 μs
1e-10	6.431822e-09	1.823461e-08	4.507969e-11	73	681 μs
1e-12	1.020948e-10	2.192610e-10	5.420580e-13	83	752 μs
1e-14	1.051453e-12	3.405704e-12	8.419598e-15	94	847 μs
1e-16	7.660026e-13	2.206590e-12	5.455143e-15	123	1145 μs

Table 4, instead, aims to compare directly the CG and QR methods in the same experimental condition, in order to find the optimal strategy for each dimension of $\hat{X}$. In particular, we recorded the measure of $||w - w^*||$ for QR and set it as the target tolerance for CG, and we let it iterate until said tolerance was satisfied. In this way, it is easy to sense the overwhelming advantage of Conjugate Gradient for all tested $\hat{X}$ in term of execution times. As expected, the advantage of CG in this regard increases with the number of columns.

Table 4: Direct comparison between CG and QR for each of the 4 variants of $\hat{X}$. Both methods are set to reach the same tolerance of $||w - w^*||$ measurement.

# Col.	Method	$\|\|\boldsymbol{w} - \boldsymbol{w}^*\|\|$	$\|\|\nabla \boldsymbol{f}(\boldsymbol{w})\|\|$	$\|\|\hat{\boldsymbol{X}}\boldsymbol{w} - \boldsymbol{y}\|\|$	Ex. Time
9	QR	1.494056e-15	2.515880e-13	8.157689e-01	505 μs
	CG	5.367416e-15	1.379609e-13	6.820379e-16	52 μs
18	QR	2.080594e-15	3.558510e-13	2.360020e-01	1627 μs
	CG	1.705508e-14	2.861926e-13	7.462059e-16	130 μs
45	QR	2.391265e-15	2.449262e-13	2.759179e-01	9267 μs
	CG	9.796561e-13	9.977292e-13	4.934967e-15	2296 μs
54	QR	2.276758e-15	2.330946e-13	2.229563e-01	13374 μs
	CG	6.907772e-13	7.250056e-13	3.979575e-15	2999 μs

5.3 Comparison with Random Matrix

In this section we make usage of the two implemented algorithms in order to resolve two random systems, one with a rectangular and the other with a square matrix, in both cases the solution is a random matrix of the shape $\mathbb{R}^{N\times 4}$. The main purpose of these experiments is to show the reliability of the proposed solutions and the variations of outcomes yielded by the standard library `X\y` (that is the optimal solution w^*), when queried to solve a rectangular system or a square one. In both experiments, we set the number of columns of the random matrix as the number of CG iterations.

In Table 5 we applied the methods on the rectangular $G_1^{(2000\times 200)}$ approaching the vector $h_1^{(2000\times 4)}$. As we can observe, the QR Householder Reflector resulting w is almost identical to $w^*$the standard implementation, due to the fact that it rely on the same algorithm when queried a rectangular system. Unluckily this result doesn't minimize our normalized objective function and residual as well as the CG method does.

Table 6 reports the results from the square $G_2^{(800\times 800)}$, with the solution vector $h_2^{(800\times 4)}$. In this case CG is appearing less performing and can't reach the same precision as his rival.

Table 5: QR and CG methods tested on a random system with a rectangular random $G_1^{(2000\times 200)}$ instead of $\hat{X}$.

Method	$\|\|\boldsymbol{w} - \boldsymbol{w}^*\|\|$	$\|\|\nabla \boldsymbol{f}(\boldsymbol{w})\|\|$	$\|\|\hat{\boldsymbol{X}}\boldsymbol{w} - \boldsymbol{y}\|\|$	Ex. Time
QR	2.298589e-15	1.086411e-16	2.727290e-01	202067 μs
CG	8.488795e-09	9.039471e-12	3.200677e-11	26042 μs

5.4 Graphs

The plots in this Section help us having a better understanding of the whole picture.

Table 6: QR and CG methods tested on a random system with a square random $G_2^{(800\times 800)}$ instead of $\hat{X}$.

Method	$\|\|\boldsymbol{w}-\boldsymbol{w}^*\|\|$	$\|\|\boldsymbol{\nabla f(w)}\|\|$	$\|\|\boldsymbol{\hat{X}w}-\boldsymbol{y}\|\|$	Ex. Time
QR	5.626363e-13	1.176223e-15	1.061295e-14	1620667 μs
CG	8.691616e-01	3.747926e-06	2.629671e-05	4118389 μs

Figure 1 reports the normalized residual iteration after iteration for CG, for varying number of columns in $\hat{X}$.

Instead, Figure 2 reports the difference in normalized residual for the QR method, again for different sizes of $\hat{X}$, but computed from $\hat{X}^{1491\times 45}$ and $\hat{X}^{1491\times 54}$. The latter allows us to analyze also the $\hat{X}^{1491\times 9}$ and $\hat{X}^{1491\times 18}$ cases, by looking only at the initial columns, thanks to the concatenation. In other words, for each entry of the x-axis, j, we applied the QR Factorization process on the $\hat{X}^{(1491\times j)}$ matrix and compute the relative residual.

Finally we report the execution time in correspondence with growing number of columns (iterations). Figure 3a shows a linear dependency with the number of iterations for CG, while in Figure 3b we can see that the time taken computing the optimized QR algorithm is almost quadratic w.r.t. the number of columns, confirming our theoretical grounds.

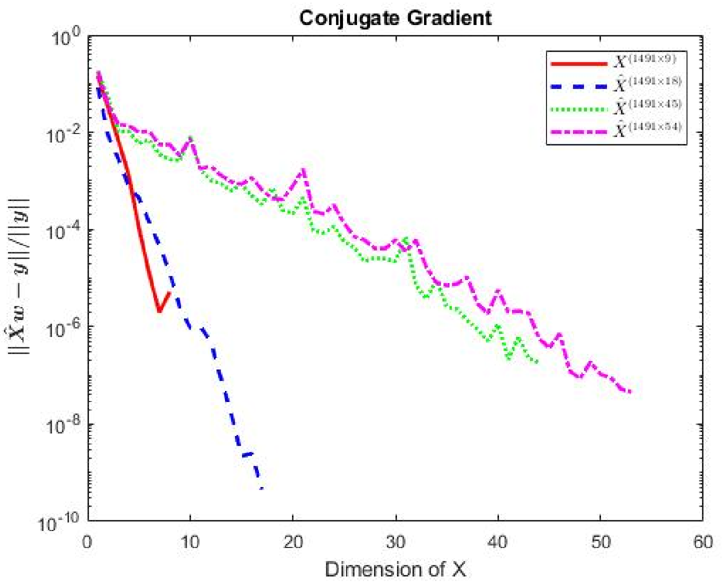
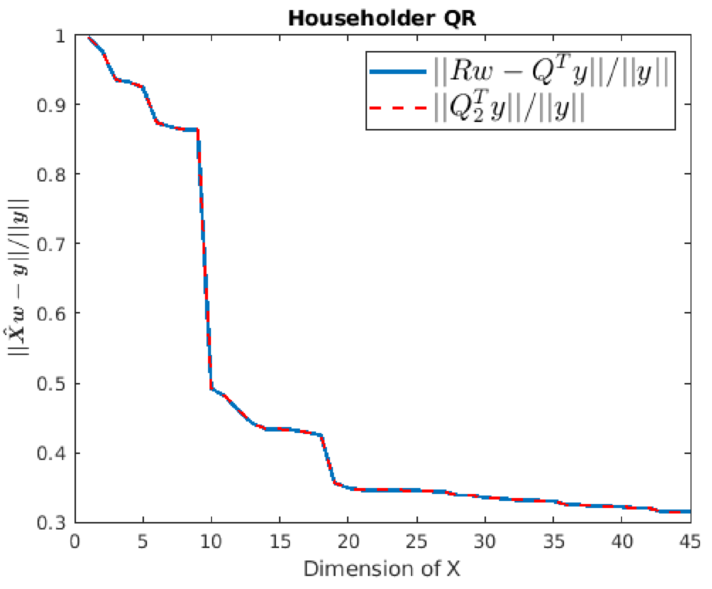
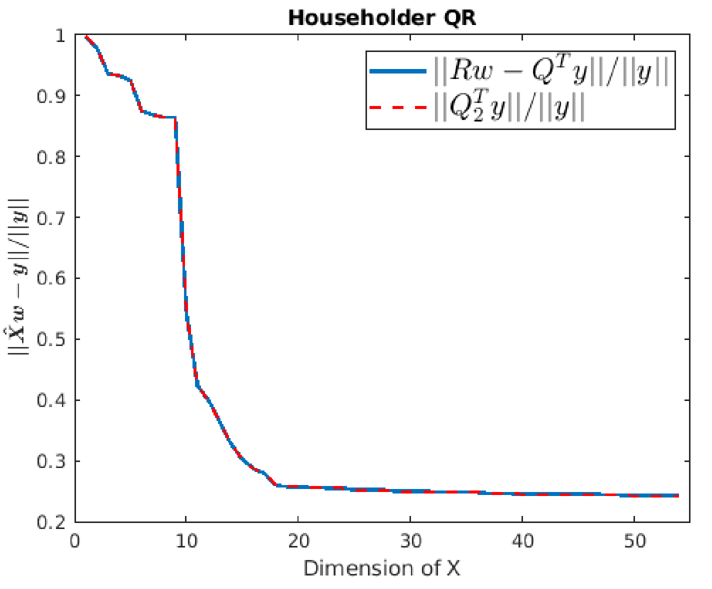


Figure 1: This graph shows the loss curves of CG for different column sizes of $\hat{X}$. As expected, the minimum iteration necessary to achieve the requested error tolerance grows with the number of columns of $\hat{X}$.

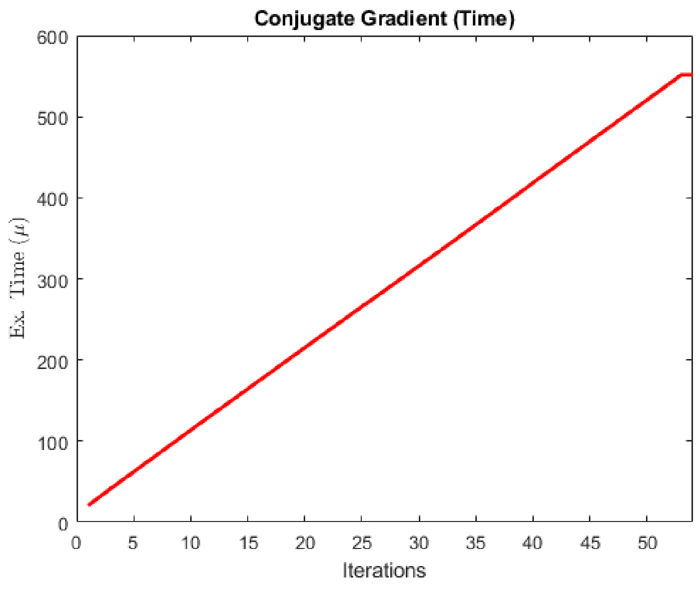


(a) In this first case the QR factorization has been applied column by column till the reaching of the $\hat{X}^{(1491\times45)}$ matrix, i.e. the last x entry.

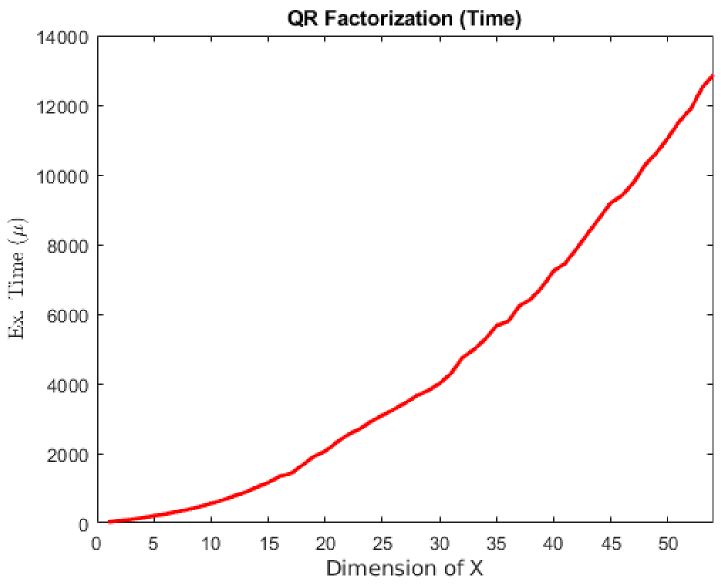


(b) Here the QR inverse formula has been applied again column by column, but we have also the quadratic manipulation so we start from the same 9 initial columns (in the upper left section we can recognize the same 9 steps), but we reach the $\hat{X}^{(1491\times54)}$ matrix in the end.

Figure 2: These plots show the error for the QR during the factorization process, column by column. As we can see, the loss decrease as soon we increase the number of columns, as we are reducing the residual components present in the bottom part of Q and R; we can appreciate this from the overlapping lines.



(a) Graph describing the execution time of the CG method with a growing number of iterations performed, the matrix used is the one with all the manipulations applied $\hat{X}^{(1491\times 54)}$.



(b) In this case the graph describes the execution time of the QR process with a matrix larger and larger.

Figure 3: Graphs of the implemented methods execution times in relation to growing columns or iterations. We can definitely affirm that the CG method grows linearly with the number of iterations performed, instead the QR factorization process is almost quadratic in the number of columns.

6 Conclusions

To conclude, we sum up by expressing our opinions on these two methods that we have extensively analyzed for this problem. To begin with, it is important to briefly recall the theory that consolidates these two algorithms.

First, we have the Conjugate Gradient, deriving from intuitions on Steepest Descent. It is a method that, step by step, iteration after iteration, improves the residual error going to eliminate the wrong components from n A-orthogonal directions taken; in rough words, an algorithm with a quadratic complexity in the dimension of the matrix for each iteration.

Secondly, the QR method that uses the Householder reflectors technique to construct the orthogonal matrix Q and then transform the original matrix into a superior triangular one, finally used to calculate the least square solution; each iteration has complexity equal to the product of the dimensions of the matrix.

These two algorithms, as described, have substantial differences and thus should be the application of them. Only from a theoretical standpoint, we could prefer the Conjugate Gradient method for large or sparse matrices, while the QR method which is most suitable for small and dense systems, and those that can re-use the factorization to solve all systems concerning the same initial matrix. As we could ascertain from the given dataset, the CG method prevails both in terms of accuracy and time cost, as we could also expect from the theoretical predictions, which were also confirmed in the final graphs.

A noteworthy result of our experiments is that the Conjugate Gradient method does not reach an optimal solution with a number of iterations equal to the number of distinct eigenvalues (in our case the number of columns of $\hat{X}^T\hat{X}$), as it is clearly shown in Table 1. Instead, when no bound on iterations is set, we can reach satisfactory results, which is proven by Tables 4 and 3. This unexpected behaviour derives from the fact that the properties and theorems that we have previously proven for Conjugate Gradient assume infinite precision, while our experiments are constrained by real world machine precision.

To conclude, the main takeaway of all of our experiments is that for our problem the Conjugate Gradient method proves to be far superior to the QR for obtaining the desired results.

References

[1] Walter Gander. Algorithms for the qr decomposition. *Res. Rep*, 80(02):1251–1268, 1980.