Università di Pisa

Computer Science Master Degree

Solving Least Squares problems using Conjugate Gradient and QR factorization

Authors: Dario SALVATI, Andrea ZUPPOLINI

ACADEMIC YEAR 2021/2022

Contents

1 Introduction **2**

2 The Problem **2**

3 The Conjugated Gradient method **3**
3.1 CG applied to LLS 4
3.2 Convergence of the CG method 5
3.2.1 Conjugate Direction methods 5
3.2.2 Conjugate Gradient method 8
3.2.3 Rate of convergence 11

4 QR Factorization via Household vectors **15**
4.1 Computing QR 15
4.2 Thin QR 16
4.3 Solving the LS problem using QR factorization 16
4.4 Stability and Residual 17

5 Implementation **19**
5.1 Implementation details 20

6 Experiments **23**
6.1 Discussing the results 24
6.1.1 Conjugate Gradient results 24
6.1.2 QR factorization results 25
6.2 Plots 26

1 Introduction

The aim of this report is to discuss the theoretical background of the **Conjugated Gradient** and **QR factorization via Householder vectors**, which are algorithms that will be used to solve a **Least Squares** problem.
After that, we will illustrate the implementation of both solutions, done in MATLAB, and we will proceed with showing and commenting the results of an experiment.

2 The Problem

The problem we want to solve is a classical Least Squares (LS), which is defined as follows.
Given the matrix $X \in \mathbb{R}^{mxn}$ and the vector $y \in \mathbb{R}^m$, find the vector w such that:

$$\min_{w \in \mathbb{R}^m} ||Xw - b||_2 \tag{2.1}$$

In order to use optimization algorithms to solve this problem, it has to be slightly reformulated. Assume that the matrix X has full column rank, then the problem 2.1 can be written as:

$$\begin{aligned} \min_{w \in \mathbb{R}^m} ||Xw - b||_2^2 &= \min_{w \in \mathbb{R}^m} (Xw - b)^T (Xw - b) = \\ &= \min_{w \in \mathbb{R}^m} w^T X^T X w - 2b^T X w + b^T b \end{aligned} \tag{2.2}$$

This result shows that the linear least squares problem can be redefined as finding the minimum of a quadratic function.
Since X has full column rank, then:

$$X^T X > 0 \implies \nabla^2 f(w) = 2X^T X > 0 \implies \exists! w_0 : f(w_0) < f(w), \forall w \neq w_0 \tag{2.3}$$

In the following sections we will show in detail the Conjugated Gradient algorithm and the QR factorization, which will then be used to solve 2.1.

3 The Conjugated Gradient method

The conjugate gradient method is an iterative algorithm used to solve systems of linear equation, in the case of a positive-definite matrix.
In order to formalize the problem that this method solves, let $X \in \mathbb{R}^{mxm}$ be a real, symmetric ($X = X^T$) and positive-definite matrix ($yXy^T > 0, \forall y \neq 0 \in \mathbb{R}^m$) and $w, b \in \mathbb{R}^m$ two vectors.
The conjugate gradient algorithm can be used to solve the problem:

$$Xw = b \tag{3.1}$$

where w is the unique solution to the linear system of equations.
Since $X = X^T$ and it's positive-definite then the problem is strictly convex:

$$f(w) = \frac{1}{2}w^t Xw - b^T w \rightarrow \nabla f(w) = Xw - b = r(w) \tag{3.2}$$

Hence, computing the minimum of the function $f(w)$ corresponds to solving $\nabla f(w) = 0 \iff Xw = b$.

We now show the pseudocode of the algorithm in order to discuss the idea behind the method and its time/space complexity.

Algorithm 1 ConjugateGradient(A, b)

$w_0 \leftarrow 0$
$r_0 \leftarrow b$
$p_0 \leftarrow -b$
$k \leftarrow 1$
while $k \leq n$ **do**
  $\alpha_k \leftarrow -\frac{{r^T}_k r_k}{{p^T}_k X p_k}$
  $w_{k+1} \leftarrow w_k + \alpha_k p_k$
  $r_{k+1} \leftarrow r_k + \alpha_k X p_k$
  $\beta_{k+1} \leftarrow \frac{{r^T}_{k+1} r_{k+1}}{{r^T}_k r_k}$
  $p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$
  $k \leftarrow k + 1$
end while

It's clear that the algorithm uses only three vectors:

- w_k, which is the result up to the current iteration;
- r_k, which is the residual vector, defined as $b - X_k = -\nabla f(w)$;
- p_k, which is the search direction. The search direction p_k is A-conjugate to the previous search direction p_{k-1}. We recall that two vectors $x, y \in \mathbb{R}^m$ are A-conjugate (or A-orthogonal) if and only if $x^T A y = 0$, where, in this specific case, $x, y \equiv p_{k-1}, p_k$.

3.1 CG applied to LLS

Given the results showed in 2.3 the minimum of the function exists, it's unique and corresponds to the stationary point of f. Such point, w_0, can be computed as the solution to:

$$\begin{aligned} \nabla f(w_0) = 0 &\iff 2X^T X w_0 - 2X^T b = 0 \\ &\iff X^T X w_0 = X^T b \end{aligned} \tag{3.3}$$

Equation 3.3 shows that we can solve the linear least squares problem as solving a linear system. Since the matrix $X^T X$ is symmetric positive definite, given that X has full column rank, the linear system can be solved using the

conjugate gradient method. For readability purposes and to avoid adding other variables, in the following section we will refer to X^TX as X and to X^Tb as b.

3.2 Convergence of the CG method

3.2.1 Conjugate Direction methods

In order to prove the convergence of the Conjugate Gradient method, we discuss the conjugate direction methods, which uses the same exact idea, but uses explicitly all the search directions p_i.
The most relevant aspect is the conjugacy because it guarantees the minimization of the function $f(w) = \frac{1}{2}w^tXw - b^Tw$ in n steps, by minimizing it along the search directions p_i. Therefore we prove the following theorem.

Theorem 3.1. *Let $w_0 \in \mathbb{R}^n$ be any starting point and let $\{p_0, p_1, p_2, ..., p_{n-1}\}$ be the set of the conjugated search directions. Let us generate the sequence $\{w_k\}$ by computing at each step*

$$w_{k+1} = w_k + \alpha_k p_k \tag{3.4}$$

where α_k is the one-dimensional minimizer of the function $f(w)$ along $w_k + \alpha p_k$, defined as:

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T X p_k} \tag{3.5}$$

Note: in the next sections we'll show how this definition of α_k for the conjugate direction method is equivalent to the definition used in the conjugate gradient method.

Then the sequence $\{w_k\}$ converges to the solution w^* of the problem 3.1 in at most n steps.

Proof. We start by noticing that the set of conjugate directions $\{p_i\}$ is composed by n linearly independent vectors and so it's a base that spans $\mathbb{R}^n$. Hence we can express the difference between the solution w^* and the starting point w_0 as:

$$w^* - w_0 = \sigma_0 p_0 + \sigma_1 p_1 + \sigma_2 p_2 + ... + \sigma_{n-1} p_{n-1} \tag{3.6}$$

where $\{\sigma_i\}$ is a set of scalars.
By multiplying this expression by the term $p_k^T X$ we get:

$$p_k^T X(w^* - w_0) = \sigma_0 p_k^T X p_0 + \sigma_1 p_k^T X p_1 + \sigma_2 p_k^T X p_2 + ... + \sigma_{n-1} p_k^T X p_{n-1} \quad (3.7)$$

However, for the conjugacy property, the product $p_k^T X p_i = 0, for j \neq i$, hence 3.7 can be simplified to:

$$\sigma_k = \frac{p_k^T X(w^* - w_0)}{p_k^T X p_k} \quad (3.8)$$

In order to establish the proof, we have to demonstrate that the value of the coefficient σ_k is equal to the step length α_k that is used in the conjugate direction method. To do so, we simply notice that w_k generated by 3.4 can be explicitly written as:

$$w_k = w_0 + \alpha_1 p_1 + \alpha_2 p_2 + ... + \alpha_{k-1} p_{k-1} \quad (3.9)$$

For the same reasoning as before, if we multiply by $p_k^T X$ we get that:

$$p_k^T X(w_k - w_0) = 0 \quad (3.10)$$

therefore

$$p_k^T X(w^* - w_0) = p_k^T X(w^* - w_k) = p_k^T (b - X w_k) = -p_k^T r_k \quad (3.11)$$

By substituting the numerator of α_k (or σ_k) we find that $\alpha_k = \sigma_k$.

□

The convergence of the conjugate direction method highlights some interesting properties. The first one is that if X is diagonal -hence the contours of $f(w)$ are ellipses whose axes are aligned with the coordinate directions- we can find the minima by performing one-dimensional minimizations along the coordinate directions $e_1, e_2, ..., e_n$, hence converging in n steps. If X is not diagonal we can precondition it to obtain the same "good" behaviour.
The second interesting property is that if the Hessian is diagonal, at each iteration we determine a component of the solution w^*. This property can actually be proven true even for the general case in which the Hessian is not diagonal.

Before stating the theorem and showing the proof, we recall that since $r_k = Xw_k - b$ and $w_{k+1} = w_k + \alpha_k p_k$ then we can write $r_{k+1} = r_k + \alpha_k X p_k$. Now we enunciate the theorem and show the proof.

Theorem 3.2. *Let $w_0 \in \mathbb{R}^n$ be any starting point and consider the sequence $\{w_k\}$ generated by the conjugate direction algorithm. Then:*

$$r_k^T p_i = 0, \quad \forall i = 0, 1, ..., k-1 \tag{3.12}$$

In words, the residual at iteration k is orthogonal to all the search directions used up to that step. Also, w_k is the minimizer of $f(w) = \frac{1}{2} w^t X w - b^T w$ over the set $\{w | w = w_0 + span(p_0, p_1, p_2, ..., p_{k-1})$.

Proof. Firstly, we want to prove that a point of minima of $f(\cdot)$ over the set $\{w | w = w_0 + span(p_0, p_1, p_2, ..., p_{k-1})\}$, namely $\tilde{w}$, respects the property $r(\tilde{w}^T) p_i = 0, i = 0, 1, ..., k-1$. We define a function: $h(\sigma) = f(w_0 + \sigma_0 p_0 + ...+\sigma_{k-1} p_{k-1})$, which is strictly convex quadratic $\implies \exists \sigma^* : h(\sigma^*) < h(\sigma), \forall \sigma$ and satisfies:

$$\frac{\partial h(\sigma^*)}{\partial \sigma_i} = 0, \quad i = 0, 1, ..., k-1 \tag{3.13}$$

By the chaining rule:

$$\nabla f(w_0 + \sigma_0^* p_0 + ... + \sigma_{k-1}^* p_{k-1})^T p_i = 0, \quad i = 0, 1, ..., k-1 \tag{3.14}$$

Recalling 3.2, we confirmed that $r(\tilde{w})^T p_i = 0$.
We now proceed by induction to prove this property for any w_k.
The base case $k = 1$ is straightforward, since $w_1 = w_0 + \alpha_0 p_0$ minimizes $f(\cdot)$ on the direction $p_0 \implies r(w_1)^T p_0 = 0$.
For the induction hypothesis, $r(w_{k-1})^T p_i = 0, \quad i = 0, 1, ..., k-2$. Hence we have:

$$\begin{aligned} r(w_k) = r(w_{k-1}) + \alpha_{k-1} X p_{k-1} \iff \\ p_{k-1}^T r(w_k) = p_{k-1}^T r(w_{k-1}) + \alpha_{k-1} p_{k-1}^T X p_{k-1} = 0 \end{aligned} \tag{3.15}$$

Instead, for any other search direction p_i, we get:

$$p_i^T r(w_k) = p_i^T r(w_{k-1}) + \alpha_{k-1} p_i^T X p_{k-1} = 0 \tag{3.16}$$

where $p_i^T r(w_{k-1}) = 0$ for the inductive hypothesis and $p_i^T X p_{k-1} = 0$ because of the conjugacy property of the search directions. □

3.2.2 Conjugate Gradient method

As anticipated before, we are going to discuss the conjugate gradient method, in which the direction at step k, namely p_k, is conjugate to the previous search directions p_i. We will prove its convergence property using the same idea of theorem 3.1, hence by proving that the search directions p_i are conjugate, and we will also provide a proof for some properties that will be useful for the discussion on the convergence rate.

Theorem 3.3. *Given the k-th iteration generated by the gradient descent method, where $w_k \neq w^*$. Then the following properties hold:*

$$r_k^T r_i = 0, \quad \forall i = 0, .., k-1 \tag{3.17}$$

$$span\{r_0, r_1, ..., r_k\} = span\{r_0, Xr_0, ..., X^k r_0\} \tag{3.18}$$

$$span\{p_0, p_1, ..., p_k\} = span\{r_0, Xr_0, ..., X^k r_0\} \tag{3.19}$$

$$p_k^T X p_i = 0 \tag{3.20}$$

Hence, for Theorem 3.1, the sequence $\{w_k\}$ converges to w^* at most in n steps.

Proof. We start by proving the last three properties by induction.
The base case $k = 0$ for 3.18 and 3.19 can be immediately verified; while 3.20 holds by construction in the base case of $k = 1$. By induction hypothesis we are assuming:

$$r_k \in span\{r_0, Xr_0, ..., X^k r_0\} \tag{3.21}$$

$$p_k \in span\{r_0, Xr_0, ..., X^k r_0\} \tag{3.22}$$

Recalling how the r_k is updated in the algorithm 1 we can derive:

$$r_{k+1} \in span\{r_0, Xr_0, ..., X^{k+1} r_0\} \tag{3.23}$$

that combined with the induction hypothesis 3.21 implies:

$$span\{r_0, r_1, ..., r_k, r_{k+1}\} \subset span\{r_0, Xr_0, ..., X^{k+1}r_0\} \tag{3.24}$$

Now we have to prove the reverse inclusion. Using 3.22 we have that:

$$X^{k+1}r_0 = X(X^k r_0) \in span\{Xp_0, Xp_1, ..., Xp_k\} \tag{3.25}$$

Given that $r_{k+1} = r_k + \alpha_k X p_k$, we have that $Xp_i = (r_{i+1} - r_i)/\alpha_i$ so X^{k+1} belongs to span$\{r_0, r_1, ..., r_{k+1}\}$

Using 3.21, it follows:

$$span\{r_0, Xr_0, ..., X^{k+1}r_0\} \subset span\{r_0, r_1, ..., r_{k+1}\} \tag{3.26}$$

Hence 3.18 still holds for the case $k + 1$.

We now prove the inductive step for 3.19:

$$\begin{aligned} span\{p_0, p_1, ..., p_k.p_{k+1}\} = & \\ span\{p_0, p_1, ..., p_k, r_{k+1}\} = & \quad (3.27) \\ span\{r_0, Xr_0, ..., X^k r_0, r_{k+1}\} = & \quad (3.28) \\ span\{r_0, r_1, ..., r_k, r_{k+1}\} = & \quad (3.29) \\ span\{r_0, Xr_0, ..., X^{k+1}r_0\} & \quad (3.30) \end{aligned}$$

Where 3.27 holds given the formula used in algorithm 1 to compute the direction; 3.28 holds as stated in 3.22; while 3.29 and 3.30 have just been proved.
We now prove the induction step for 3.20. Recalling from algorithm 1, the update formula for the search direction is:

$$p_k \leftarrow -r_k + \beta_k p_{k-1}$$

Multiplying this expression by Xp_i, we get:

$$p_k^T X p_i = -r_k^T X p_i + \beta_k p_{k-1}^T X p_i \tag{3.31}$$

Given the definition of β_k, the right side of the equation goes to 0 when $i = k$.
For the other values of i, we observe that the search directions $\{p_0, p_1, ..., p_k\}$ are conjugate, which implies:

$$r_{k+1}^T p_i = 0, \quad \forall i = 0, 1, ..., k \tag{3.32}$$

as proven in Theorem 3.2.
Also, for 3.19 we can state that:

$$\begin{gathered} Xp_i \in X span\{r_0, Xr_0, ..., X^i r_0\} = \\ span\{Xr_0, X^2 r_0, ..., X^{i+1} r_0\} \subset span\{p_0, p_1, ..., p_{i+1}\} \end{gathered} \tag{3.33}$$

Combining 3.32 and 3.33 we deduce:

$$r_{k+1}^T X p_i = 0, \quad \forall i = 0, 1, ..., k-1 \tag{3.34}$$

and so the first term of the right side of 3.31 goes to 0 for $i = 0, 1, ..., k-1$, while the second term vanishes as well for the induction hypothesis of 3.22. We have finally proven that the conjugate gradient method generates a conjugate direction set, hence for Theorem 3.1 we can state that the algorithm terminates in n iterations.

As the last thing to show, we prove 3.17.
Since we've proven that the conjugate gradient method generates a conjugated direction set, from 3.12 we have that $r_k^T p_i = 0 \quad \forall i = 0, 1, ..., k-1$ and any $k = 1, 2, ..., n-1$. Rearranging 3.31 we get:

$$p_i = -r_i + \beta_i p_{i-1} \tag{3.35}$$

so that $r_i \in span\{p_i, p_{i-1}\}, \forall i = 1, ..., k-1$. This implies that $r_k^T r_i = 0, \forall i = 1, ..., k-1$. Note that $r_k^T r_0 = -r_k^T p_0 = 0$ by definition of p_0 in Algorithm 1.

□

Before diving in the details of the rate of convergence of the conjugate gradient method, we briefly show h ow to derive the update rule for α_k used in Algorithm 1. Since $p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$ and $r_k^T p_i = 0$, using the results of Theorem 3.2 and 3.3, we can replace the formula as follows:

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T X p_k} = -\frac{r_k^T(-r_k + \beta_k p_{k-1})}{p_k^T X p_k} = \\ = \frac{r_k^T r_k}{p_k^T X p_k} \tag{3.36}$$

3.2.3 Rate of convergence

In Theorem 3.3 we proved that the Conjugate Gradient method converges in n steps. However this result can be even better in the case in which the distribution of the eigenvalues of the matrix X has certain properties. We now make some observation in order to prove some theorems on the rate of convergence of the method.

Given the update rule for w_k and the property 3.19 proven in Theorem 3.3 we can write:

$$w_{k+1} = w_0 + \alpha_0 p_0 + ... + \alpha_k p_k = \\ = w_0 + \gamma_0 r_0 + \gamma_1 X r_0 + ... + \gamma_k X^k r_0 \tag{3.37}$$

where γ_i are constants. We now define a polynomial $P_k^*(\cdot)$ of degree k with coefficients γ_i. By applying X to P_k^* we get:

$$P_k^*(X) = \gamma_0 I + \gamma_1 X + ... + \gamma_k A^k \tag{3.38}$$

and so we can express 3.37 as:

$$w_{k+1} = w_0 + P_k^*(X) r_0 \tag{3.39}$$

We can show tat the conjugate gradient algorithm 1 is optimal in minimizing the distance to the solution, defined as the X-norm $||y||_X^2 = y^T X y$, after k steps, restricted to the Krylov subspace $\mathcal{K}(r_0; k)$.
Since w^* minimizes f, we can write:

$$\frac{1}{2}||w - w^*||_X^2 = \frac{1}{2}(w - w^*)^T X (w - w^*) = f(w) - f(w^*) \tag{3.40}$$

In Theorem 3.2 we've proven that w_{k+1} minimizes f and so also $||w - w^*||_X^2$, over the set $w_0 + span\{p_0, p_1, ..., p_k\}$ which by 3.19 is the same as $w_0 + span\{r_0, Xr_0, ..., X^k r_0\}$. Hence we can reformulate the problem as finding the polynomial of grade k that minimizes the X-norm:

$$min_{P_k}||w_0 + P_k(X)r_0 - w^*||_X \tag{3.41}$$

Since $r_0 = Xw_0 - b = Xw_0 - Xw^* = X(w_0 - w^*)$ we can reformulate the difference between w_{k+1} and w^* as:

$$w_{k+1} - w^* = w_0 + P_k^*(X)r_0 - w^* = [I + P_k^*(X)X](w_0 - w^*) \tag{3.42}$$

The matrix X can be written in terms of its eigenvalues and eigenvectors:

$$X = \sum_{i=1}^{n} \lambda_i v_i v_i^T \tag{3.43}$$

where λ_i are X's eigenvalues and v_i are X's eigenvectors, which are also eigenvectors of the polynomial $P_k^*(X)$, in particular $P_k^*(X)v_i = P_k^*(\lambda_i)v_i$. Since the set of eigenvectors are linearly independent and n in number, they span the whole space $\mathbb{R}^n$, hence we can write the difference between w_0 and w^* as:

$$w_0 - w^* = \sum_{i=1}^{n} \xi v_i \tag{3.44}$$

for some coefficients ξ_i. By substituting 3.44 in 3.42 we get:

$$w_{k+1} - w^* = \sum_{i=1}^{n} [1 + \lambda_i P_k^*(\lambda_i)]\xi_i v_i \tag{3.45}$$

passing to the X-norm and noticing that $||y||_X^2 = y^T X y = \sum_{i=1}^{n} \lambda_i (v_i^T y)^2$, we have:

$$||w_{k+1} - w^*||_X^2 = \sum_{i=1}^{n} \lambda_i [1 + \lambda_i P_k^*(\lambda_i)]^2 \xi_i^2 \tag{3.46}$$

As stated before, P_k^* is optimal w.r.t. the X-norm, hence we rewrite as:

$$||w_{k+1} - w^*||_X^2 = \sum_{i=1}^{n} min_{P_k} \lambda_i [1 + \lambda_i P_k(\lambda_i)]^2 \xi_i^2 \tag{3.47}$$

Finally, by extracting the largest term from $[1 + \lambda_i P_k(\lambda_i)]^2$ and since $||w_0 - w^*||_X^2 = \sum_{j=1}^{n} \lambda_j \xi_j^2$ we obtain:

$$\begin{aligned} ||w_{k+1} - w^*||_X^2 &\leq \min_{P_k} \max_{1 \leq i \leq n} [1 + \lambda_i P_k(\lambda_i)]^2 (\sum_{j=1}^{n} \lambda_j \xi_j^2) \\ &= \min_{P_k} \max_{1 \leq i \leq n} [1 + \lambda_i P_k(\lambda_i)]^2 ||w_0 - w^*||_X^2 \end{aligned} \tag{3.48}$$

This formula grants an upper bound and thus lets us quantify the convergence rate of the conjugate gradient method by estimating the scalar:

$$\min_{P_k} \max_{1 \leq i \leq n} [1 + \lambda_i P_k(\lambda_i)]^2 \tag{3.49}$$

We obviously want to minimize the upper bound, hence we want to find the polynomial P_k that makes this expression as small as possible.

From the observations above, we derive two theorems.

Theorem 3.4. *If X has r distinct eigenvalues, then the conjugate gradient iteration will terminate at the solution in at most r iterations.*

Proof. Suppose that the eigenvalues λ_i take r distinct values $\tau_1 < \tau_2 < ... < \tau_r$. We define a polynomial $Q_r(\lambda)$ as:

$$Q_r(\lambda) = \frac{(-1)^r}{\tau_1 \tau_2 \dots \tau_r} (\lambda - \tau_1)(\lambda - \tau_2) \dots (\lambda - \tau_r) \tag{3.50}$$

Note that $Q_r(\lambda_i) = 0$, for $i = 1, 2, \dots, n$ and $Q_r(0) = 1$, hence, $Q_r(\lambda) - 1$ is a polynomial of grade r with a root in $\lambda = 0$. We define another polynomial, $\bar{P}_{r-1}(\lambda)$, of rank $r - 1$ as:

$$\bar{P}_{r-1}(\lambda) = \frac{Q_r(\lambda) - 1}{\lambda} \tag{3.51}$$

If we set $k = r - 1$ in 3.49 we get:

$$0 \leq min_{P_{r-1}} \max_{1 \leq i \leq n} [1+\lambda_i P_{r-1}(\lambda_i)]^2 \leq \max_{1 \leq i \leq n} [1+\lambda_i \bar{P}_{r-1}(\lambda_i)]^2 = \max_{1 \leq i \leq n} Q_r^2(\lambda_i) = 0$$

Hence 3.49 is 0 for $k = r - 1$, which implies that in 3.48 we obtain $||w_{k+1} - w^*||_X^2 = 0$, therefore $w_r = w^*$.

□

From the same reasoning of Theorem 3.4, Luenberger derived the following theorem:

Theorem 3.5. *If X has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, then we have that:*

$$||w_{k+1} - w^*||_X^2 \leq (\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1})||w_0 - w^*||_X^2 \tag{3.52}$$

Theorem 3.5 can be proven in a similar manner to Theorem 3.4 and it makes possible some interesting observations on the convergence rate of the method. In fact it can be shown that, at a certain step k, this approximation can be made:

$$||w_k - w^*||_X \approx \epsilon ||w_0 - w^*||_X \tag{3.53}$$

for a certain parameter ϵ which is driven by the eigenvalues of X.
In particular, we should consider how clustered are X's eigenvalues: if there's a set of eigenvalues that are very close to a certain point, the we could say that they have approximately the same value, hence they could be seen as a single distinct eigenvalue, which is a positive property considering Theorem 3.4. It's actually been shown that, in general, if the matrix X has r clusters of eigenvalues, then the conjugate gradient method will approximately solve the problem in r step. This convergence property it's actually more convenient than the one stated in 3.4, but it considers an approximate solution.
Another interesting result that has been showed is that the ϵ parameter in 3.53 can be written in terms of the condition number of X, which is defined as $\lambda_{max}/\lambda_{min}$:

$$||w_k - w^*||_X \leq 2 \left(\frac{\sqrt{\kappa(X)} - 1}{\sqrt{\kappa(X)} + 1} \right)^k ||w_0 - w^*||_X \tag{3.54}$$

This bound can often be a large overestimate, however it's very useful in the case in which we only have an estimate of X's largest and smallest eigenvalues.

4 QR Factorization via Household vectors

Let $X \in \mathbb{R}^{mxn}$ any matrix, for simplicity assume $m > n$.
X can be factorized as $X = QR$, where $Q \in \mathbb{R}^{mxm}$, orthogonal, and $R \in \mathbb{R}^{mxn}$, upper triangular.
We will compute the QR decomposition using Householder Reflectors.
Householder Reflectors are matrices that are in the form:

$$H = I - \frac{2}{v^T v} v v^T \tag{4.1}$$

4.1 Computing QR

In order to use the Householder Reflectors to compute the QR factorization, we notice that we can modify the original matrix X into X_1, where X_1 has the same values of X except for the first column, where the only non-zero value is in the first row and has value equal to the norm of the first column of X. The Householder Reflector H_1 used to obtain X_1 is generated from $X[i : end, i]$, where $i = 1$.
We can iterate this process for all the columns of X, hence obtaining a matrix $X_n \equiv R$ which is upper triangular and a matrix Q which is the result of all the left-products of the matrices Q_i.
We notice that $Q_1 \equiv H_1$ since we don't need to expand its dimensions in order to make the row-column product possible. Instead for $i > 1$, Q_i is defined as: $\begin{bmatrix} I_i & 0 \\ 0 & H_i \end{bmatrix}$ in order to get the correct dimensions for the matrix product.

After computing Q and R, since the orthogonality of Q we can say that:

$$X = QR \tag{4.2}$$

where $Q \in \mathbb{R}^{mxm}$ is an orthogonal matrix and $R \in \mathbb{R}^{mxn}$ is an upper triangular matrix.

4.2 Thin QR

In the previous subsection we made an argument for a general matrix $X \in R^{mxn}$, however, for our particular case, it's relevant to consider the case in which $m >> n$, hence when dealing with a tall-thin matrix.
In this case we notice that the decomposition has a specific form:

$$X = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \tag{4.3}$$

where $Q_1 \in R^{mxn}$ is tall-thin and with orthogonal columns, $Q_2 \in R^{mxm-n}$ and $R_1 \in R^{nxn}$ is square upper triangular. Since Q_2 is multiplied by the matrix of zeros, we can write X as:

$$X = Q_1 R_1 \tag{4.4}$$

This decomposition is obviously less memory intensive, since it can be stored in $O(m * n)$ instead of $O(m^2)$.
We also make an argument for the time complexity of the algorithm. In general for $m \geq n$, the time complexity of the QR factorization is $2mn^2 + \frac{2}{3}n^3 + o(mn)$. However, since we're dealing with tall-thin matrices where $m >> n$, the complexity can be re-written as $2mn^2$, which is the dominant term for that case.

4.3 Solving the LS problem using QR factorization

We can now reformulate the LS problem using the QR factorization of X we have just obtained:

$$\begin{aligned} \min_w ||Xw - b||_2 = \min_x ||Q^T(Xw - b)||_2 = \\ = \min_w ||Q^T Q R w - Q^T b||_2 = \min_x ||Rw - Q^T b||_2 = \\ = \min_w || \begin{bmatrix} R_1 w \\ 0 \end{bmatrix} - \begin{bmatrix} Q_1^T b \\ Q_2^T b \end{bmatrix} ||_2 = \min_w || \begin{bmatrix} R_1 w - Q_1^T b \\ -Q_2^T b \end{bmatrix} ||_2 \end{aligned} \tag{4.5}$$

where Q_1 is the matrix composed of the first n columns of Q and Q_2 is the matrix composed of the remaining $m - n$ columns. The same reasoning goes for R_1, which is the matrix composed of the first n rows of the matrix R.

Hence the optimal solution is given by:

$$w = R_1^{-1} Q_1^T b \tag{4.6}$$

which can be obtained if R_1 is invertible.
Since the solution depends only on Q_1 it's not needed to compute the whole matrix Q, but it's only needed to have the first n Householder Reflectors to compute the first n columns of Q.

4.4 Stability and Residual

We now spend few words about the stability and the error produced by solving the LS problem using the QR factorization.
First thing first, we recall the value of u, defined as:

$$\frac{||\tilde{w} - w||}{||x||} \leq u \tag{4.7}$$

where $w \in \pm[10^{-308}, 10^{308}]$ is any number, $\tilde{w}$ is the closest possible representation of w and $u = 2^{-52} \approx 10^{-16}$.

We now recall the definition of a backward stable algorithm.
Assume that an algorithm is used to found a solution $y = f(w)$. Since the limitations of the modern hardware, we know that the output of the computation will actually be $\tilde{y} = f(\tilde{w})$, so we can define the forward error as $\Delta_y = \tilde{y} - y$.
Instead the backward error is defined as the minimum perturbation $\Delta_w = \tilde{w} - w$ that makes the algorithm compute $\tilde{y}$ as output.

Hence, by definition, an algorithm is called backward stable if the backward error is stable, in a relative sense:

$$\frac{||\tilde{w} - w||}{||w||} = O(u) \tag{4.8}$$

It can be proven that the QR factorization is in fact backward stable.
We start by noticing that multiplying by an orthogonal matrix is a backward stable operation. The operation $Q \odot X$ results in $QX + E$, where Q is orthogonal and E is the forward error, which has norm:

$$||E|| \leq O(u)||Q||\ \ ||X|| = O(u)||X|| \tag{4.9}$$

By writing $Q \odot X = Q(X + Q^{-1}E)$, we can write the backward error Δ_X as $\Delta_X = Q^{-1}E$, which satisfies:

$$||\Delta_X|| \leq ||Q^{-1}||\ ||E|| \leq O(u)||X|| \tag{4.10}$$

If we disturb the input X by the forward error Δ_X, we obtain the perturbed output $\tilde{Q}\tilde{R}$. The result is obtained by a multiplication by an orthogonal matrix, hence each step is backward stable: the output $\tilde{X}_k$ at step k has accumulated all the errors of the previous iterations and satisfies $\tilde{X}_k = \tilde{Q}_k(\tilde{X}_{k-1} + \Delta_{k-1})$ with:

$$\Delta_{k-1} \leq O(u)\ \ ||\tilde{X}_{k-1}|| \tag{4.11}$$

Hence after n steps, $\tilde{R}$ is the result of:

$$X + \Delta_0 + \tilde{Q}_1^T\Delta_1 + \cdots + \tilde{Q}_1^T\tilde{Q}_2^T \ldots \tilde{Q}_{n-1}^T\Delta_{n-1} \tag{4.12}$$

and each error is $O(u)||\tilde{X}_{k-1}|| = O(u)||X||$.
Hence the QR factorization is backward stable and the computed Q, R are the exact result of applying the decomposition on the input $X + \Delta$, where $||\Delta|| \leq O(u)||X||$.

Another good property of backward stable algorithms is that they usually have a small residual.
Assume that w is the solution of the least squares problem and $r_1 = Q_1^T(X\tilde{w} - b)$ is the residual of a thin QR factorization. It can be proven that $\tilde{w}$ is the exact solution of the problem:

$$min||Xw - (b + Q_1 r)|| \tag{4.13}$$

It follows that:

$$\frac{|\tilde{w} - w||}{||w||} \leq \kappa_{rel}(LS, b)\frac{||Q_1 r_1||}{||b||} = \kappa_{rel}(LS, b)\frac{||r_1||}{||b||} \tag{4.14}$$

which is a bound ruled by the condition number relative to the least squares problem and by the ration between the residual and the target of the problem.

5 Implementation

The project has been implemented in `C++`, mainly using the Armadillo library and built using `CMake`. We also implemented some small scripts in Python, for convenience, to visualize the results of the experiments performed.

We started by implementing three different header libraries:

- `utils.hpp`, which only contains useful functions, in particular:
  - `grab_mlcup_dataset()`, which returns the dataset used for the experiments;
  - `to_normal_equations()`, which modifies the problem to the normal equations one;
  - `add_columns`, which adds an arbitrary set of columns to the input matrix, in order to increase the dimensionality of the data-points.
- `conjugate_gradient.hpp`, which contains the conjugate gradient algorithm;
- `qr_factorization.hpp`, which contains:
  - `compute_householder`, which computes the householder vector of an input vector;
  - `thin_qr`, which computes the thin QR factorization of an input matrix;
  - `solve_thin_qr`, which uses the direct formula to solve the LS problem in a direct way

We then implemented the experiments in order to obtain the data used in the next section.

All the details about the build of the project can be found in the readme on its GitHub page.

5.1 Implementation details

We now discuss more in depth about the implementation of the main two algorithms presented in this report.

The implementation of the **Conjugate Gradient** algorithm is quite straightforward. Firstly we check if the input matrix is symmetric, which is required by the algorithm. If the check is passed, the optimization loop starts and it will end either if the number of maximum iterations has been computed or if the gradient reaches a certain threshold.
By default the program will store on a file all the values of the gradient in each iteration.

For the **QR factorization** implementation, instead, we can notice more details.
The `compute_householder` function simply applies the operations to derive the householder vector from the input vector. However, we also enforced the sign of the norm of the input vector, in order to deal with the extreme case in which the first component of the input vector is approximately its norm, which may cause instability.

The `thin_qr` function doesn't exactly compute the thin QR factorization. Instead we actually compute R and the product $Q * b$, in order to efficiently solve the system. Both computations are done in the same loop.
R is computed as follows:

$$R = H_n H_{n-1} \dots H_1 X$$

hence left multiplying the input matrix X by H_i, at each step. If we expand this formula with the definition of H_i, we get:

$$R = (I - 2u_n u_n^T)(I - 2u_{n-1} u_{n-1}^T) \dots (I - 2u_1 u_1^T) X$$

therefore, for a single step of computation we can write:

$$R_i = (I - 2u_i u_i^T) R, \quad \forall i = 1, \dots, n$$

In order to perform less computations, we can actually compute the step as follows:

$$R_{i+1} = R_i - 2u_i(u_i^T R_i)$$

The operation above is made dimensionally possible by the fact that, at each step, we operate on a sub-matrix of the matrix R_i. In particular:

$$R_{i+1}[i:m, i:n] = R_i[i:m, i:n] - 2u_i(u_i^T R_i[i:m, i:n])$$

where the notation $R_i[i:m, i:n]$ indicates the sub-matrix obtained by taking R_i's rows from i to m and columns from i to n.

By proceeding this way, at each step we are performing a multiplication between matrices of size $\mathbb{R}^{(m-i)\times 1} \times \mathbb{R}^{1\times(m-i)} \times \mathbb{R}^{(m-i)\times(n-i)}$ instead of multiplying two matrices of size $\mathbb{R}^{(m-i)\times(m-i)} \times \mathbb{R}^{(m-i)\times(n-i)}$.

At the same time, we compute the product $Q^T * b$ as:

$$Q^T * b = H_n H_{n-1} \dots H_2 H_1 b \tag{5.1}$$

In the same way we did for the computation of R, we can rewrite the expression above as:

$$Q^T * b = (I - 2u_n u_n^T)(I - 2u_{n-1}u_{n-1}^T) \dots (I - 2u_1 u_1^T)b$$

that, for a single iteration, can be written as:

$$Q_{i+1}^T * b = (Q_i^T * b - 2u_i(u_i^T Q_i^T * b)$$

Just as for the computation of R_i, the operation above is made dimensionally possible by the fact that, at each step, we operate on a sub-matrix of the matrix $Q_i^T * b$. In particular:

$$(Q_{i+1}^T * b)[i:m, 1] = ((Q_i^T * b)[i:m, 1] - 2u_i(u_i^T(Q_i^T * b)[i:m, 1])$$

where the notation $(Q_i^T * b)[i:m, 1]$ indicates the sub-matrix obtained by taking $Q_i^T * b$'s rows from i to m and its first column.

Proceeding this way guarantees that at each iteration we perform a vector-matrix multiplication of dimensions $\mathbb{R}^{m-1\times 1} \times \mathbb{R}^{1\times m-i} \times \mathbb{R}^{m-i\times 1}$, which is cheaper, both in time and space complexity, w.r.t. computing Q_1 and then

multiplying by b.

The loop executes in $O(n)$, where n is the number of columns of X, hence the function runs in linear time.

Lastly, in `solve_thin_qr`, we remove the zero-rows of R to obtain R_1, left multiply $Q_1 * b$ by $I^{n \times m}$ and apply the direct formula to find the solution w:

$$w = R_1^{-1} * I^{n \times m} * Q^T * b$$

where $Q^T * b$ has already been computed in `thin_qr`.

6 Experiments

We performed two different experiments in order to evaluate the performances of the algorithms presented, for the problem of the least squares.

For both experiments we used the ML-CUP 2021 dataset, which is used for the course of Machine Learning at the University of Pisa. From now on, we'll refer to this dataset as X, which originally is $X \in \mathbb{R}^{1477x12}$, where the last two columns represent the labels.

In order to use X, we selected its penultimate column as the target vector b. We enhanced the matrix by adding some arbitrary non-linear functions to the features of the data points, ending up with $X \in \mathbb{R}^{1477x20}$ (which doesn't contain the target vector anymore).

We used Armadillo's `solve` function to obtain the result to the problem we're solving -which from now on we'll refer to as w^*-, in order to compare the results we will obtain during the experiments. Armadillo's `solve` is very similar to Matlab's `A\B`, which computes the solution in an efficient way.

We will now present the results of our experiments.

Regarding the conjugate gradient method, we obtained the following results:

	max_iter	$\|\|w - w^*\|\|$	$\nabla f(w)$	$\|\|Xw - b\|\|/\|\|b\|\|$	time
1	10	0.0418762	8.22782	0.306568	21 μs
2	20	2.39653e-06	0.000423126	0.30655	44 μs
3	30	9.87424e-14	2.81131e-12	0.30655	66 μs

Table 1: Conjugate gradient method results

As reported in the table above, we tested different values for the number of iterations of the algorithm, in order to study its behaviour. We started with a number of iterations equal to the number of columns of the input matrix, and we increased up to a value where the gradient stopped decreasing, triggering an early stopping criteria.

The QR factorization offers, instead, a direct approach for solving the problem. We first compute R_1 and $Q^T * b$ from the input matrix X and target vector b; then we solve using the direct formula 4.6.
In the following table we report the result of the experiment:

	$\|\|w - w^*\|\|$	$\nabla f(w)$	$\|\|Xw - b\|\|/\|\|b\|\|$	time
1	1.00363e-14	2.49289e-11	0.30655	1518 μs

Table 2: Thin QR factorization result

This method is able to reduce the gradient magnitude paying the price of a much slower execution w.r.t. the conjugate gradient. As shown in subsection 4.2, the time complexity for the QR factorization on a tall-thin matrix can be written as $O(2mn^2)$ and the computations involve the matrix $X \in \mathbb{R}^{1477x20}$. Instead, the complexity of the conjugate gradient method is $O(m * \sqrt{\kappa})$, where m is the number of non-zero entries in the matrix and κ is its conditioning number. Since the algorithm works on the matrix $X^T X \in \mathbb{R}^{20x20}$, it's quite evident why there's such difference in the execution time of the two implementations.

6.1 Discussing the results

In this section we are going to discuss the results we have just obtained in the experiments and we will compare them to the theoretical results of previous section.

6.1.1 Conjugate Gradient results

For the conjugate gradient method we have proved different theorems, guaranteeing an upper bound on the number of iterations the algorithm performs, given certain properties of the matrix X.

For instance, from theorem 3.4, we should expect that the execution of the algorithm terminates within the number of distinct eigenvalues λ_i of the matrix X. In our case, X is a symmetric positive definite matrix and $X \in \mathbb{R}^{20\times20}$, hence it has 20 different eigenvalues but, as the table 1 reports, with 20 iterations the distance between the solution found by the algorithm

and w^* is 2.39653e−06, which is some orders of magnitude far from the exact solution.

This behaviour arises from the fact that the theorems we've proven are assuming infinite-precision arithmetic, thus are not exactly applicable to our context. The loss of precision actually depends on the problem we're trying to solve, in particular it's correlated to its conditioning number. Let's recall that the conditioning number $\kappa(X) = ||X|| * ||X^{-1}||$ - or also $\kappa(X) = \lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are, respectively, the maximum and the minimum eigenvalue of X - measures the sensitiveness of a problem, namely how much the output changes w.r.t. the variation of the input. In our specific case, the condition number of the square matrix obtained via normal equations on the original system is $\kappa(X^T X) = 167.2325$.
An estimation that results useful is that if the condition number of the problem is $\kappa(X) \approx 10^k$, then the algorithm may lose up to k digits of accuracy in addition to the loss of precision due to the finite-precision arithmetic. Therefore, in our case, the influence of the condition number would result in the loss of up to 2 digits of precision.

6.1.2 QR factorization results

Using the QR factorization we were able to reach the same result of the conjugate gradient method. We actually experimented with a different implementation that made the matrix multiplications shown in 5.1: though the computation was much slower, the result was slightly more accurate, reaching the optimal solution w^* with an error of $\approx$ e−15. This may be due to the round-off errors given by the finite-precision arithmetic, or some implementation or heuristic detail of the framework used.

6.2 Plots

In this section there will be the plots of the result we have obtained with the ML cup dataset and some further experiment to enforce some of the considerations made during the theoretical discussion. In particular we want to show that our implementation of the QR factorization correctly scales with the number of rows, as its complexity can be approximated as: $\mathcal{O}(mn^2)$.

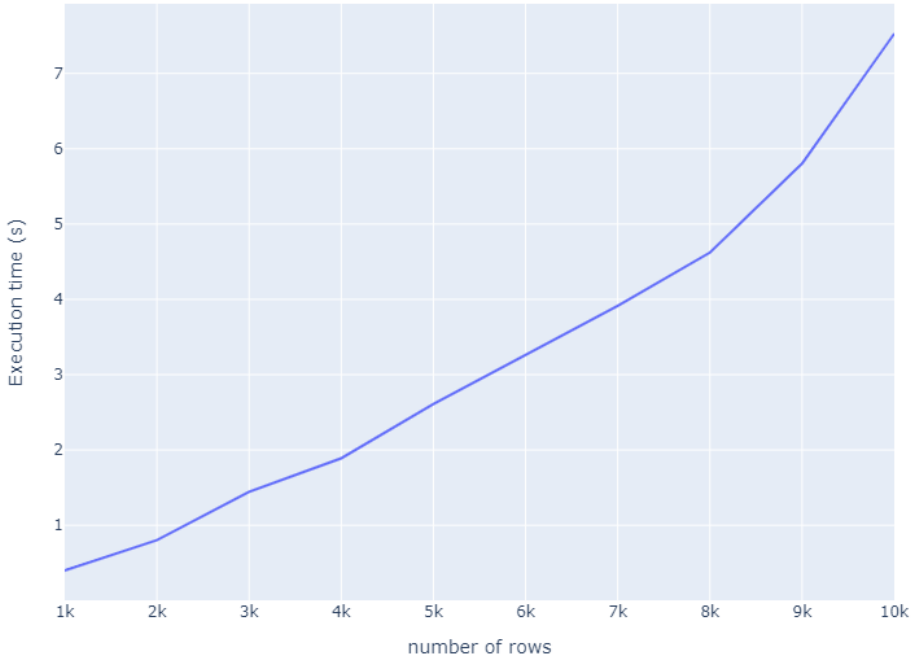

Figure 6.1: QR factorization execution time w.r.t. the number of rows

The figure shows the results of different execution of the QR factorization on matrices with a fixed number of columns (350) and a number of rows increasing from 1000 to 10000 with entries being randomly filled. As stated before, the algorithms seems to have a behaviour close to linear as expected. Notice that the execution time does not refer to a single execution of the experiments, each point is the average value over 20 executions.

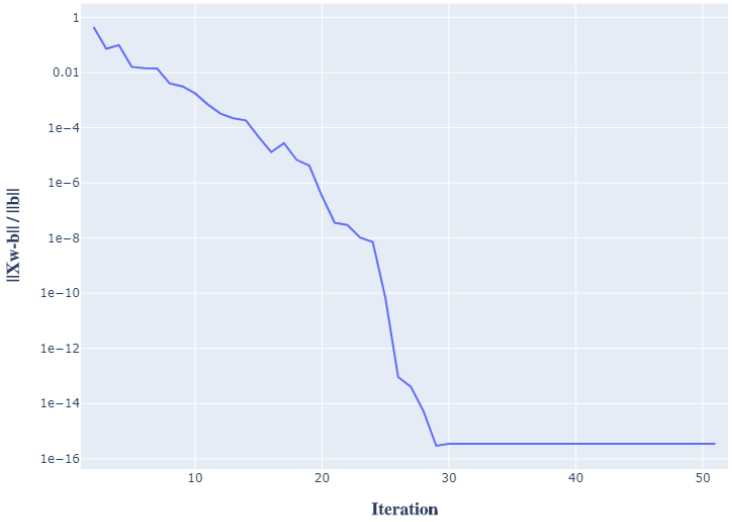

Figure 6.2: CG log relative error on the ML-CUP dataset.

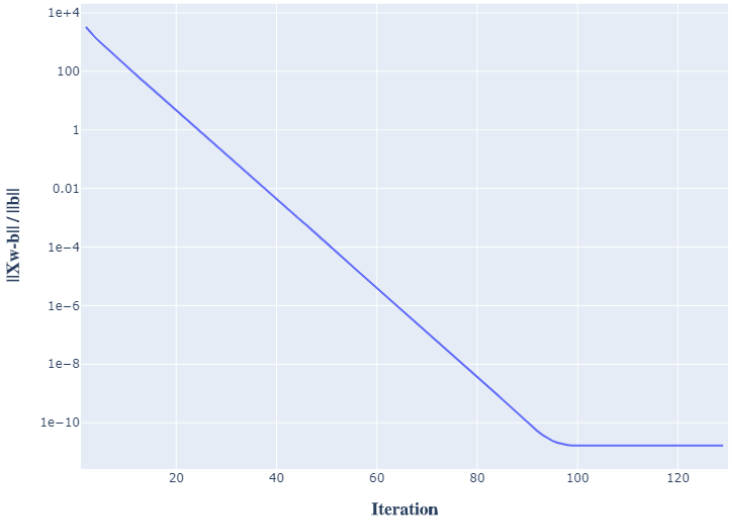

Figure 6.3: CG log relative error on random matrix $\in \mathbb{R}^{10000x5000}$

Figure 6.2 shows the presence of oscillations, which is expected since the Conjugate Gradient method is an iterative algorithm. During the experiments we have verified that this matrix has full column rank, hence the very same considerations presented when introducing the Table 1 can be made here.

In Figure 6.3 we present the same log relative error graph but for a random matrix, generated using an uniform distribution.

Lastly, Figure 6.4 shows how, in practice, the method scales approximately w.r.t. the number of non-zero entries of the matrix times the condition number κ.

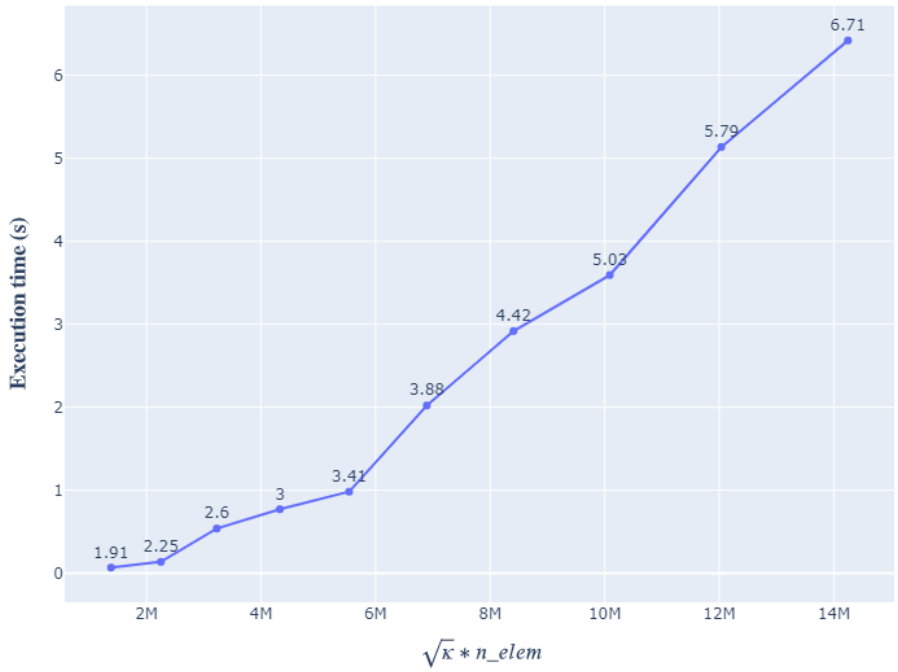

Figure 6.4: CG execution time