Università di Pisa

Computational Mathematics for Learning and Data Analysis

Project Track 19 — Non-ML

Team 16

De Castelli Fabrizio *f.decastelli@studenti.unipi.it*

Rossi Elvis *e.rossi46@studenti.unipi.it*

Academic Year 2023–2024

Contents

Introduction

(P) is the linear least squares problem

$$\min_{w} \left\| \hat{X}w - \hat{y} \right\|$$

where

$$\hat{X} = \begin{bmatrix} X^T \\ \lambda I_m \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} y \\ 0 \end{bmatrix},$$

with X the (tall thin) matrix from the ML-cup dataset by prof. Micheli, $\lambda > 0$ and y is a random vector.

- (A1) is an algorithm of the class of limited-memory quasi-Newton methods.
- (A2) is a cothin 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.

No off-the-shelf solvers allowed. In particular you must implement yourself the thin QR factorization, and the computational cost of your implementation should be at most quadratic in m.

Outline

This report is organized as follows:

chapter 2, in which the problem is reformulated under the mathematical aspect;

chapter 3, where we will include the implemented algorithms, with the analysis of convergence and complexity;

chapter 4, to evaluate and compare (A1) with (A2) for this task and provide different tests in order to examine deeper the algorithms;

chapter 5, in which conclusions are drawn, offering a critical analysis of the results obtained.

Problem Definition

Henceforth, we denote the norm-2 $\|-\|_2$ with the generic norm symbol $\|-\|$. Given $\hat{X} \in \mathbb{R}^{(m+n)\times m},\ \hat{y} \in \mathbb{R}^{m+n},$ we want to find

$$\min_w \left\|\hat{X}w - \hat{y}\right\|$$

2.1 QR

By performing a QR factorization on $\hat{X}$ we can reformulate the problem as follows:

$$\min_w \left\|\hat{X}w - \hat{y}\right\| = \min_w \left\|QRw - \hat{y}\right\|$$

with $Q \in \mathbb{R}^{(m+n)\times(m+n)}$ being an orthogonal matrix and $R \in \mathbb{R}^{(m+n)\times m}$ being an upper triangular matrix. Knowing that $R_{ij} = 0, \quad \forall i > j,\ i = 1, \ldots, m+n,\ j = 1, \ldots, m,$ we can write

$$R = \begin{bmatrix} R_0 \\ 0 \end{bmatrix}, \qquad\qquad R_0 \in \mathbb{R}^{m\times m}$$
$$Q = \begin{bmatrix} Q_0 \ Q_c \end{bmatrix}, \quad Q_0 \in \mathbb{R}^{(m+n)\times m},\ Q_c \in \mathbb{R}^{(m+n)\times n}$$

Since orthogonal matrices preserve norm-2, we have:

$$\begin{aligned}
&\min_w \|QRw - \hat{y}\| = \min_w \left\|Q^T(QRw - \hat{y})\right\| = \\
&\min_w \left\|Q^TQRw - Q^T\hat{y}\right\| = \\
&\min_w \left\|Rw - Q^T\hat{y}\right\| = \\
&\min_w \left\|\begin{bmatrix} R_0 \\ 0 \end{bmatrix} w - \begin{bmatrix} Q_0^T \\ Q_c^T \end{bmatrix} \hat{y}\right\| = \\
&\min_w \left\|\begin{bmatrix} R_0w - Q_0^T\hat{y} \\ -Q_c^T\hat{y} \end{bmatrix}\right\|
\end{aligned}$$

The entries of the second block $-Q_c^T\hat{y}$ do not depend on w, meaning that they will appear in the norm independently from w. Thus, we can simplify the problem and solve the triangular system

$$R_0 w - Q_0^T\hat{y} = 0 \iff R_0 w = Q_0^T\hat{y}$$

provided that R_0 is invertible.

$$R_0 \text{ is invertible } \iff \hat{X} \text{ has full column rank } \iff \hat{X}^T\hat{X} \succ 0.$$

R_0 is invertible and the triangular system can be solved via backsubstitution. This claim is proved in the last section.

2.2 L-BFGS

We can define

$$g(w) = f(w)^2 = \left\|\hat{X}w - \hat{y}\right\|^2 \tag{2.1}$$

and reformulate the problem equivalently in terms of $g(w)$, since it is monotonic.

$$\min_w \; g(w) = \min_w \; \left\|\hat{X}w - \hat{y}\right\|^2 = \min_w \; \left(\hat{X}w - \hat{y}\right)^T\left(\hat{X}w - \hat{y}\right)$$

The gradient of g with respect to w is

$$\nabla g(w) = 2\hat{X}^T\left(\hat{X}w - \hat{y}\right)$$

Likewise the gradient of $f(w)$ is as follows:

$$\nabla f(w) = \frac{1}{\left\|\hat{X}w - \hat{y}\right\|}\hat{X}^T\left(\hat{X}w - \hat{y}\right)$$

but gives much worse performance since it is no longer quadratic.

The function is L-smooth since $\forall w, w' \in \mathbb{R}^m,$ with $w \neq w'$:

$$\begin{aligned}
&& \|\nabla g(w) - \nabla g(w')\| &\leq L \|w - w'\| \\
\iff && \left\|\hat{X}^T(\hat{X}w - w') - \hat{X}^T(\hat{X}w' - \hat{y})\right\| &\leq L \|w - w'\| \\
\iff && \left\|\hat{X}^T\hat{X}(w - w')\right\| &\leq L \|w - w'\| \\
\impliedby && \left\|\hat{X}^T\hat{X}\right\| \|w - w'\| &\leq L \|w - w'\| \\
\iff && \left\|\hat{X}^T\hat{X}\right\| &\leq L
\end{aligned}$$

The function g is also strongly convex since $\nabla^2 g(w) = \hat{X}^T\hat{X} \succ 0$.
The tomography of $g(w)$ with respect to the direction p is:

$$\begin{aligned}
\phi(\alpha) &= (\hat{X}(w + \alpha p) - \hat{y})^T \cdot (\hat{X}(w + \alpha p) - \hat{y}) \\
\frac{d\phi(\alpha)}{d\alpha} &= 2w^T\hat{X}^T\hat{X}p - 2\hat{y}^T\hat{X}p + 2\alpha p^T\hat{X}^T\hat{X}p \\
\frac{d^2\phi(\alpha)}{d\alpha^2} &= 2p^T\hat{X}^T\hat{X}p \qquad (2.2)
\end{aligned}$$

Since $\frac{d^2\phi(\alpha)}{d\alpha^2}$ is constant, the tomography is simply a parabola and since $\hat{X}^T\hat{X}$ is positive definite, the dot product $\langle p, p\rangle_{\hat{X}^T\hat{X}}$ is always positive and the parabola always has a minimum. The minimum is found by solving $\frac{d\phi(\alpha)}{d\alpha}$ for 0:

$$\alpha_{\min} = \frac{\hat{y}^T\hat{X}p - w^T\hat{X}^T\hat{X}p}{p^T\hat{X}^T\hat{X}p}$$

2.3 Conditioning

We check the condition number $\kappa(\hat{X})$ when the regularization term $\lambda > 0$ varies.

$$\kappa(\hat{X}) = \left\|\hat{X}\right\| \left\|\hat{X}^T\right\| = \frac{\sigma_1}{\sigma_m} = \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}}$$

with σ_1, σ_m being respectively the largest and smallest singular values of $\hat{X}$ and $\lambda_{\max}, \lambda_{\min}$ being the largest and smallest eigenvalues of $\hat{X}^T\hat{X}$.
Knowing that $\hat{X}^T\hat{X} = XX^T + \lambda^2 I_m$, we have that

$$\lambda_{max} = \lambda_1 + \lambda^2$$
$$\lambda_{min} = \lambda_m + \lambda^2$$

with λ_1, λ_m being the largest and smallest eigenvalues of XX^T, which are translated by λ^2 as a result of adding $\lambda^2 I_m$ (Lemma 2)
In Lemma 3 we show that $\lambda_m = 0$ and conclude that $\kappa(\hat{X})$ scales linearly with $\frac{1}{\lambda}$:

$$\kappa(\hat{X}) = \sqrt{\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}} = \sqrt{\frac{\lambda_1 + \lambda^2}{\lambda_m + \lambda^2}} = \frac{\sqrt{\lambda_1 + \lambda^2}}{\sqrt{\lambda^2}} = \frac{\sqrt{\lambda_1 + \lambda^2}}{\lambda}$$

if $\lambda_1 > 0$.

For lambda close to zero we have $\frac{\sqrt{\lambda_1+\lambda^2}}{\lambda} \approx O\left(\frac{1}{\lambda}\right)$. This property is witnessed in Figure 2.1, which is in logarithmic scale:

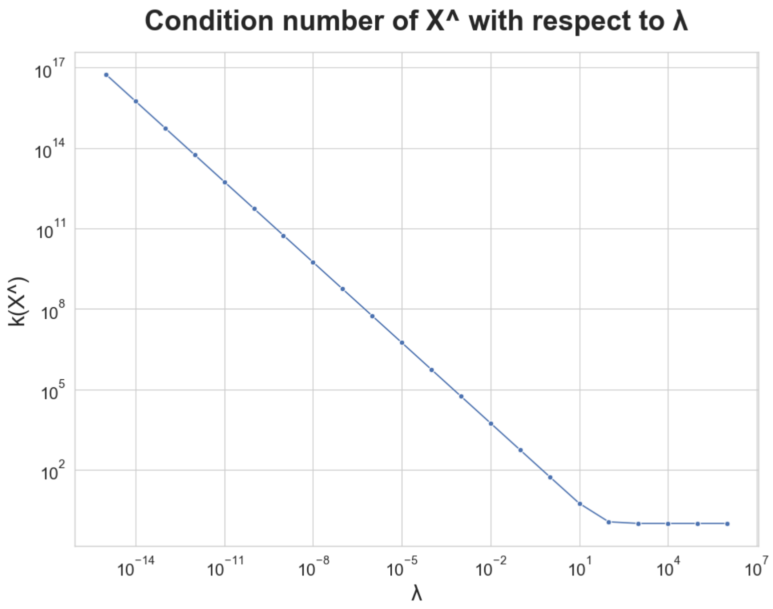


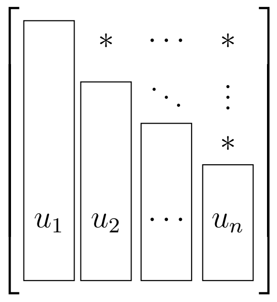
Figure 2.1: $\kappa(\hat{X})$ *for different values of* λ

Algorithms

3.1 QR

The algorithm has been implemented considering that the input matrix $A \in \mathbb{R}^{m \times n}$, where m may be different from n, namely it can be rectangular *horizontally* or *vertically*. In this version we store in a proper data structure a matrix $\Upsilon \in m \times n$ of the following form ($m > n$ in this example):

$$\Upsilon = (\upsilon_{i,j})_{i,j} =$$



$$u_k \in \mathbb{R}^{m-k+1},\ 1 \leq k \leq n$$

and the values of the diagonal of R in a vector $d \in \mathbb{R}^n$. The $*$ entries are elements computed in the QR factorization belonging to the upper triangular matrix, yielded by line 6 of Algorithm 1. In this way we are allowed to lazily perform the products Qy and $Q^T y$ by means of the householder vectors $u_1 \ldots, u_n$ that we stored. On the other hand, to compute a product between the upper part of Υ and an input vector we reconstruct the upper triangular matrix by taking element υ_{ij} such that $j > i$ and attach the vector d as the diagonal of the resulting matrix. The zeros of the matrix R are ignored.

Algorithm 1: Thin QR

```
Input  : $A \in \mathbb{R}^{m \times n}$
Output: $Q \in \mathbb{R}^{m \times m},\ R \in \mathbb{R}^{m \times n}$ implicit $QR$ factorization of $A$
1 $\Upsilon = copy(A)$
2 $d = zeros(\min(m, n))$
3 for $k \in 1 \ldots \min(m, n)$ do
4     $u_k, s_k = householder\_vector(\Upsilon[k : m, k])$
5     $d_k = s_k$
6     $\Upsilon[k : m, k + 1 : n] = \Upsilon[k : m, k + 1 : n] - 2u(u^T \Upsilon[k : m, k + 1 : n])$
7     $\Upsilon[k : m, k] = u_k$
8 end
9 return $\Upsilon, d$
```

Algorithm 2: householder_vector

```
Input  : $x \in \mathbb{R}^d$
Output: $u \in \mathbb{R}^d,\ s \in \mathbb{R}$ householder vector of $x$
1 $s = \|x\|$
2 if $x_1 \geq 0$ then
3     $s = -s$
4 end
5 $u = copy(x)$
6 $u_1 = u_1 - s$
7 $u = u \,/\, \|u\|$
8 return $u, s$
```

We assume $m > n$ as the case $n > m$ is similar for the complexity analysis. The time complexity of this algorithm is $\theta(mn^2) \approx \theta(n^3)$, because $m \approx n$ in (P). We will see in section Experiments that the running time scales linearly with m as expected, where m is the size of $\hat{X}$.

3.2 L-BFGS

We follow the syntax from *Numerical Optimization*[3] and define $f_k = f(x_k)$

Algorithm 3: Limited Memory BFGS

```
Input  : $\mathbf{f} : \mathbb{R}^n \longrightarrow \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^n$, $m$ memory, $\epsilon$ tolerance
Output: $\mathbf{x}^*$ ending point, $\mathbf{f}(\mathbf{x}^*)$, $\nabla\mathbf{f}(\mathbf{x}^*)$

$k = 0$
while $\nabla f_k \geq \epsilon \nabla f_0$ do
    if storage is empty then
        $H_k^0 = I$
    else
        $H_k^0 = \frac{\langle y_{k-1}, s_{k-1} \rangle}{\|y_{k-1}\|^2} \cdot I$
    Calculate $p_k = H_k \nabla f_k$ with Algorithm 4
    Choose $\alpha_k$ satisfying the Armijo-Wolfe conditions or with exact line
      search
    $x_{k+1} = x_k + \alpha_k p_k$
    $s_k = x_{k+1} - x_k$
    $y_k = \nabla f_{k+1} - \nabla f_k$
    $curvature = \langle y_k, s_k \rangle$
    $\rho_k = curvature^{-1}$
    if $curvature \leq 10^{-16}$ then
        free the storage and start again from gradient descent
    else
        Discard the vector pair $\{s_{k-m}, y_{k-m}, \rho_{k-m}\}$ from storage
        Save $s_k, y_k, \rho_k$
    $k = k + 1$
end
return $x_k$, $f_k$, $\nabla f_k$
```

Algorithm 4: Limited Memory BFGS - Two-Loop Recursion

```
1  $q = \nabla f_k$
2  for $i = (k-1), \dots, (k-m)$ do
3      $\alpha_i = \rho_i s_i^T q$
4      $q = q - \alpha_i y_i$
5  end
6  $r = H_k^0 q$
7  for $i = (k-m), \dots, (k-1)$ do
8      $\beta = \rho_i y_i^T r$
9      $r = r + s_i\big(\alpha_i - \beta\big)$
10 end
11 return $-r$
```

In our implementation we keep the triplets (s_k, y_k, ρ_k) in a circular buffer with capacity m and the values of α_i in Algorithm 4 in a stack such that no explicit indices are needed.

In case the curvature of the function is too small, we free the storage and restart with a gradient step.

We prefer using an exact line search to compute the step size over an inexact line search since the computational cost for our problem is lesser.

Convergence

To prove that the implemented method converges to the global minimum of the function we have to optimize, we follow [1] and state the following assumptions about our problem:

1. $f \in C^2$

2. The level sets $\mathcal{L} = \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$ is convex

3. $\exists\ M_1, M_2 \in \mathbb{R}^+$ such that

$$M_1 \|z\|^2 \leq z^T G(x) z \leq M_2 \|z\|^2$$

   $\forall z \in \mathbb{R}^n$ and $\forall x \in \mathcal{L}$

We follow the publication's notation and define:

$$G(x) \coloneqq \nabla^2 f(x)$$
$$\bar{G}_k(x) \coloneqq \int_0^1 G(x_k + \tau \alpha_k p_k) d\tau$$

From Taylor's theorem:

$$y_k = \bar{G}_k \alpha_k p_k = \bar{G}_k s_k \tag{3.1}$$

The first assumption for our problem follows from the definition. The second assumption is proved by Equation 2.2. The third assumption is also a consequence of the fact that the hessian of f is constant.

Theorem. Let B_0 be any symmetric positive definite initial matrix, and let x_0 be a starting point for which the Assumptions 1, 2 and 3 hold, then the sequence x_k generated by the L-BFGS algorithm converges to the minimizer x^* of f linearly. $\square$

Proof: Using Equation 3.1 and Assumption 3:

$$M_1 \left\| s_k \right\|^2 \leq y_k^T s_k \leq M_2 \left\| s_k \right\|^2$$

and:

$$\frac{\left\| y_k \right\|^2}{y_k^T s_k} = \frac{s_k^T \hat{G}_k^2 s_k}{s_k^T \hat{G}_k s_k}$$

Both trace and determinant can be expressed in terms of the trace and determinant of the starting matrix from which the approximate hessian is constructed:

$$\operatorname{tr}(B_{k+1}) \leq \operatorname{tr}(B_k^{(0)}) + \tilde{m} M_2 \leq M_3$$
$$\det(B_{k+1}) = \det(B_k^{(0)}) \cdot \prod_{l=0}^{\tilde{m}-1} \frac{y_l^T s_l}{s_l^T B_k^{(l)} s_l} \geq \det\left(B_k^{(0)} \left(\frac{M_1}{M_3} \right)^{\tilde{m}} \right) \geq M_4$$

where $\tilde{m}$ is the memory size and M_3 and M_4 are chosen appropriately in $\mathbb{R}^+$.

From these two bounds we have that for some constant $\delta > 0$:

$$\cos(\theta_k) = \frac{s_k^T B_k s_k}{\left\| s_k \right\| \left\| B_k s_k \right\|} \geq \delta$$

Since with exact line search the Armijo condition $f(x_k + \alpha_k p_k) \leq f(x_k) + m_1 \alpha_k \nabla f(x_k)$ is always satisfied if the constant m_1 does not exclude the minimum x_* and since the strong Wolfe condition $\|\nabla f(x_k + \alpha_k p_k)\| \leq m_3 \|\nabla f(x_k)\|$ is also always satisfied since $\|\nabla f(x_k + \alpha_k p_k)\| = O(u)$, follows from the two conditions and Assumptions 1 and 2 that:

$$\begin{aligned}
&f(x_{k+1}) - f(x_*) \leq (1 - c\cos^2(\theta_k)(f(x_k) - f(x_*))) \\
\implies &f(x_k) - f(x_*) \leq (1 - c \cdot \delta^2)^k (f(x_0) - f(x_*)) \\
\implies &f(x_k) - f(x_*) \leq r^k (f(x_0) - f(x_*))
\end{aligned}$$

for some $r \in [0, 1)$. Using Assumption 3:

$$\begin{aligned}
&\frac{1}{2} M_1 \|x_k - x_*\|^2 \leq f(x_k) - f(x_*) \\
\implies &\|x_k - x_*\| \leq r^{k/2} \left(2 \frac{f(x_0) - f(x_*)}{M_1} \right)^{(1/2)}
\end{aligned}$$

so the sequence $\{x_k\}$ is linearly convergent. ■

The implementation of L-BFGS that uses Armijo-Wolfe line search also satisfies the assumptions so it also converges linearly to x_*.

Experiments

In this chapter we present the results of the experiments run on both algorithms as well as a comparison of the two methods in terms of accuracy and time scalability.

To test the behaviour of the two methods we handle both cases in which the matrix $\hat{X}$ is well-conditioned, with $\kappa(\hat{X}) \approx 5$, and ill-conditioned, with $\kappa(\hat{X}) \approx 5 \times 10^5$. To accomplish so, we randomly generated the matrix X forcing its values to be in the range $[-1, 1]$, the dimensions $m = 1000$ and $n = 20$ (except for time and memory scalability tests), and, as we have seen in section 2.3, since we can control the conditioning directly with the *hyperparameter* λ, we choose for the first case $\lambda = 10^{-4}$ and for the latter $\lambda = 10^{-12}$.

For the QR factorization we check how the relative error and residual change with respect to different values of λ. Then, we confirm the backward stability of the decomposition over different values of λ and check its forward stability as well.

For what concerns L-BFGS we fix the relative tolerance $\epsilon = 10^{-14}$, the memory size $k = 7$ and the maximum number of function evaluations to 200, knowing that the function we have to optimize can be easily optimized by the method.

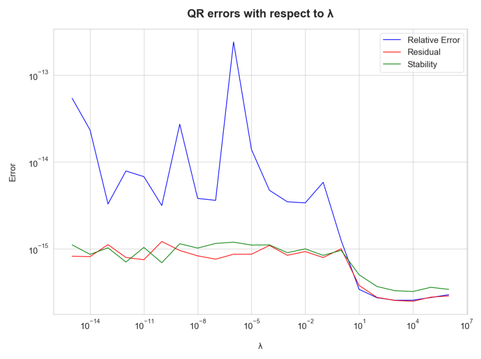
The last kind of tests we present concerns the scalability of the methods in terms of time and memory, which has been compared by modifying the matrix $\hat{X} \in \mathbb{R}^{(m+n)\times m}$ by generating random matrices with increasing m and n separately. As mentioned before, we brought this experiment to the case in which $\hat{X}$ is either ill-conditioned or well-conditioned. In the case of the thin-QR factorization we expect a linear dependency between the number of rows and the time needed to converge to the optimal solution, assuming a fixed number of columns. If instead we vary the number of columns we expect a quadratic dependency.

In section 4.4 we first explore better the effect of the memory size for L-BFGS and then we provide a deeper comparison of other Quasi-Newton methods we manually implemented (even if not really required by the project instructions).

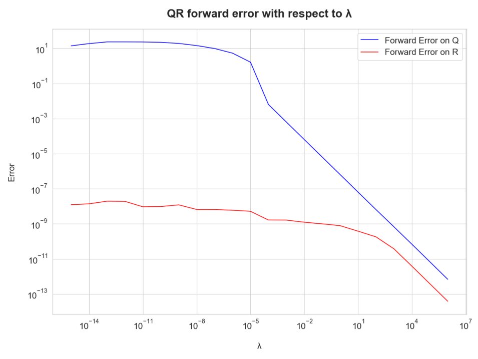
All tests have been executed with the benchmark library `BenchmarkTools.jl`[4] which ignores startup and compilation time and repeated 10 times in order to get accurate estimates.

4.1 QR

Since we know from theory that the QR decomposition is backward stable, we expect that $\frac{\|\hat{X}-QR\|}{\|\hat{X}\|} \approx u$. Or more explicitly that for $QR = \hat{X}+\delta\hat{X}$, $\frac{\|\delta\hat{X}\|}{\|\hat{X}\|} = O(u)$. The results in Figure 4.1a show a decreasing trend for relative error and residual when increasing λ and hence decreasing the condition number $\kappa(\hat{X})$. The errors are acceptable even for the smallest lambda: $\lambda = 10^{-16}$, $\kappa(\hat{X}) \approx 5 \times 10^{15}$. The algorithm is backward stable as well, as it can be noticed from the green part of the plot.
To check the forward error we QR-decomposed the original matrix $\hat{X}$ to get Q and R and then we perturbed it with a random matrix multiplied by a factor $\delta = 10^{-10}$. Then, we ran another QR-decomposition on the perturbed version of $\hat{X}$ to get the factors $\tilde{Q}$ and $\tilde{R}$. Finally, we evaluated $\left\|Q - \tilde{Q}\right\|$ and $\frac{\|R-\tilde{R}\|}{\|R\|}$, that are both much larger, as reported in Figure 4.1b. The forward error on Q is slightly worse than on R due to its orthogonality property that needs to be maintained in the factorization, fixing $\kappa(\hat{X})$. However, we can see a generally decreasing trend of the forward error with respect to the condition number of the matrix $\hat{X}$.



(a) *QR decomposition errors and backward stability for different* λ



(b) *QR factorization forward stability on Q and R for different* λ

Figure 4.1: *Errors and scalability of the QR decomposition for different values of* λ

4.2 L-BFGS

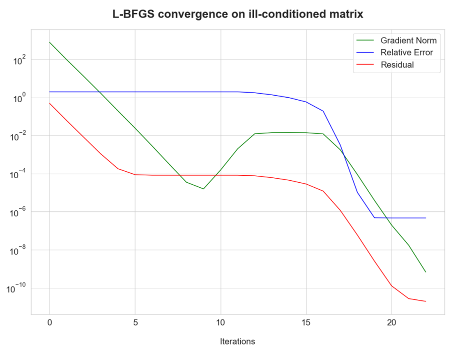
For the first experiment regarding this algorithm we compute the relative gap, the residual and the number of iterations employed by the algorithms to converge. The relative gap is defined as

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

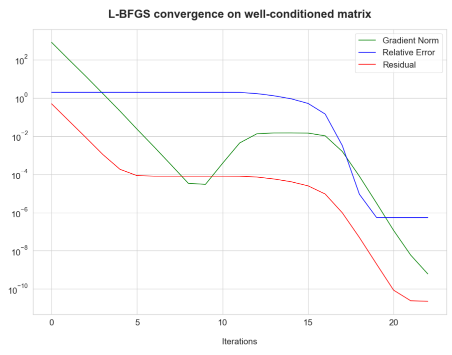
where w is the solution found by our algorithm and w^* is Julia's *ground truth* coming from its standard linear system solver.
The residual, instead, is defined as

$$\frac{\left\|\hat{X}w - \hat{y}\right\|}{\|\hat{y}\|}$$

The results are shown in Figure 4.2, satisfying constraints we imposed on $\kappa(\hat{X})$. It is evident from the plots that the convergence of the method is linear and that it is able to compute a relatively good solution in a small number of iterations.



(a) *Ill-conditioned matrix*



(b) *Well-conditioned matrix*

Figure 4.2: $\|\nabla f\|$, *Residual, Relative Error of L-BFGS execution on ill and well-conditioned matrices*

The other test we propose regards checking the convergence of the method, when using different line search algorithms. We checked how the gradient norm changes when using Exact Line Search and Armijo-Wolfe Line Search only on the well-conditioned matrix.

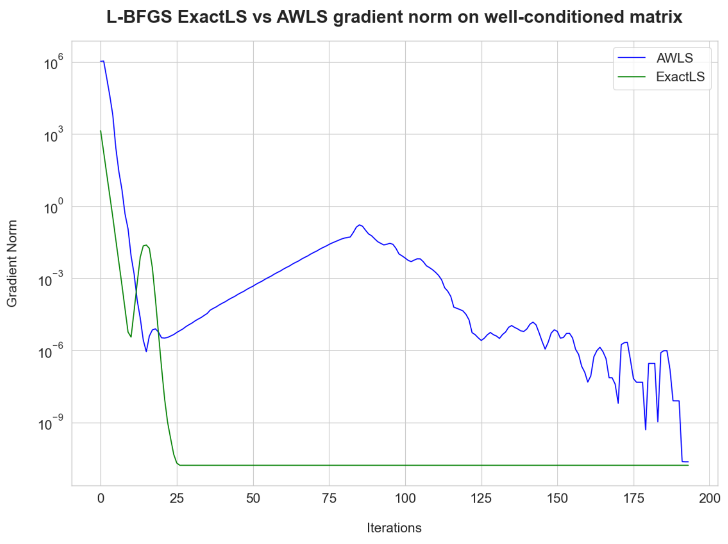
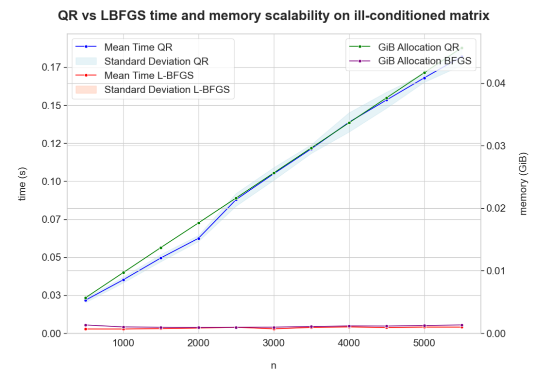


Figure 4.3: *Line Search algorithms comparison*

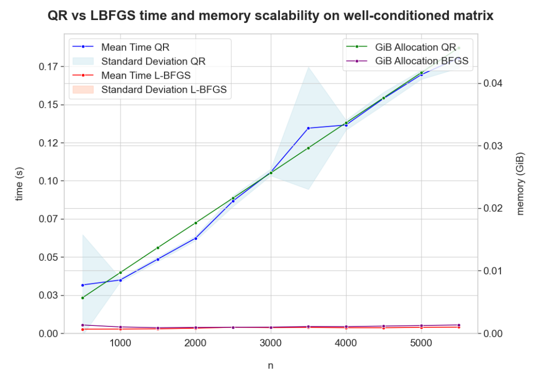
From Figure 4.3 we can notice that the exact line search behaves better than the inexact line search because of the nature of the function we are optimizing. AWLS computes a step size which may lead to instability, but does converge.

4.3 Comparison between QR and L-BFGS

The tests have been performed by fixing one between $m = 200$ and $n = 50$ and varying the other dimension from an initial value of 500 to a value of 5500, at intervals of 500. The results of fixing m and varying n be summarized in Figure 4.4, which shows a linear growth of running time with increasing n for the QR decomposition and a better performance for L-BFGS, in both the ill and well-conditioned case. The allocated memory is consistent and on the same trend as the running time as expected.



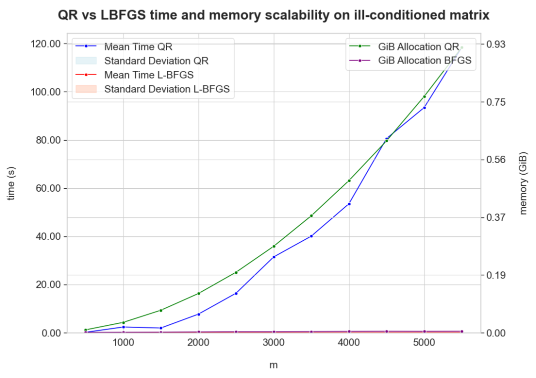
(a) *Ill-conditioned matrix*



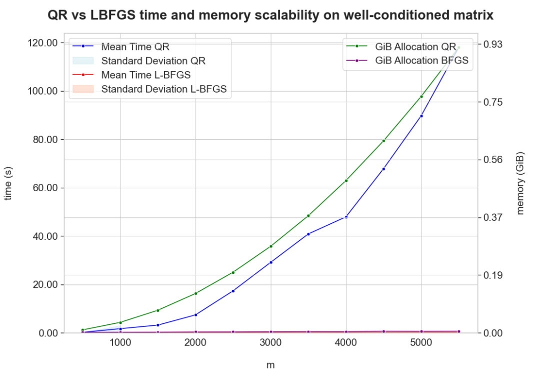
(b) *Well-conditioned matrix*

Figure 4.4: *Time and Memory scalability comparison of QR and L-BFGS on ill and well-conditioned matrices, varying* $\boldsymbol{n}$

Instead, if we fix n and let m vary, we get the following curves as shown in Figure 4.5. Both the running time and the allocated memory of QR grows more or less quadratically with the number of columns, confirming what the theory suggests.



(a) *Ill-conditioned matrix*



(b) *Well-conditioned matrix*

Figure 4.5: *Time and Memory scalability comparison of QR and L-BFGS on ill and well-conditioned matrices, varying* $\boldsymbol{m}$

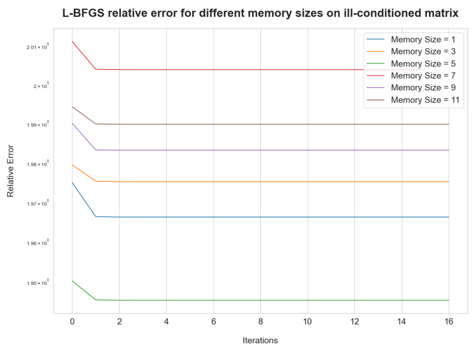
For QR the allocated memory is in the order of MiB even in the worst case while L-BFGS allocates much less memory, in the order of KiB.

The conditioning of the matrix has no impact on the time taken to compute a solution for the two algorithms compared, but rather has an impact for L-BFGS in the quality of the solution when dealing with a very flat function (small λ). When the function is flat it means that its curvature is low and the gradients change slowly, so the algorithm struggles to rapidly descent towards the minimum with a reasonable relative error.

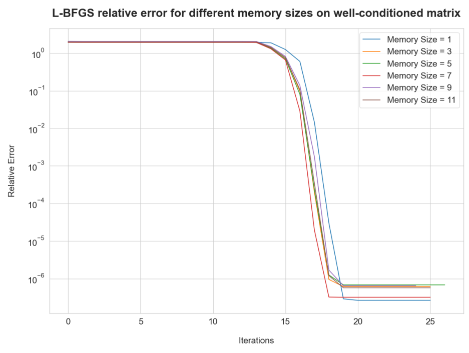
4.4 Other Experiments

4.4.1 The Effect of the Memory Size

It is interesting to check the behaviour of L-BFGS when changing the memory size. We compare the relative error decrease at each iteration with a memory size that varies from 1 to 11, as shown in Figure 4.6:



(a) *Ill-conditioned matrix*



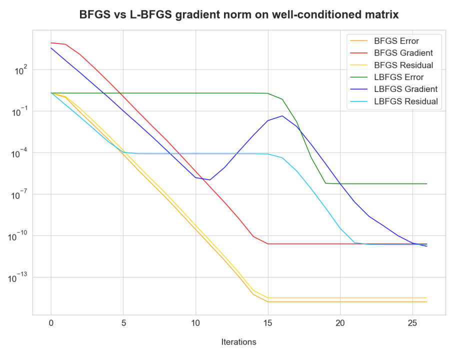
(b) *Well-conditioned matrix*

Figure 4.6: *The effect of the memory size*

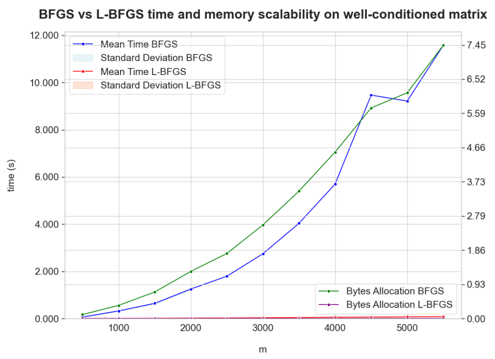
In accordance with the suggestions provided by [3], the memory size k should be chosen such that $3 \leq k \leq 20$, as it is empirically a good trade-off between number of function evaluations and number of additional operations required to reconstruct the hessian with the two loop formula (algorithm 4). However, since the function that has to be optimized is quadratic, the algorithm is fast at finding the optimal solution more or less independently of the memory size, but depends still on the curvature $\kappa(\hat{X})$ for the convergence. When the memory size is 1, the algorithm is a normal gradient descent and still reaches similar convergence rate with respect to higher memory sizes. For higher memory sizes the convergence rate is almost indistinguishable for the well-conditioned case (Figure 4.6b). However, for the ill-conditioned case, shown in Figure 4.6a, the algorithm can still converge in 16 iterations without depending on the memory size k, but the relative error is constant in each different setting. This is the consequence of the fact that the algorithm was terminating in a flatter region in which the curvature is so low that satisfies the stopping criterion imposed on the gradient, but with a bad approximation of the optimum.

4.4.2 A Comparison of Quasi-Newton Methods

To further check the behaviour of our implementation of L-BFGS, we implemented and tested a version of BFGS. In the beginning of this section we provide two additional tests, performed only on well-conditioned matrices and in which we compare the two solvers together. As far as the setup is concerned, we stick to the default setup stated in chapter 4 and for BFGS we set the tolerance to the same as L-BFGS. The first test is shown in Figure 4.7a and shows how for the least squares problem the two algorithms are almost identical in terms of convergence rate. In the plot we have the relative error, residual and gradient norm to be almost equal between the two algorithms. To understand deeply and check the differences in the implementation, we also checked the time and memory scalability. It is not surprising that, as Figure 4.7b suggests, BFGS is much slower in finding the optimum than its limited version, even for this small optimization problem. This aspect reflects the theory and confirms that this method is more expensive in terms of time and memory with respect to L-BFGS.



(a) *Convergence rate*

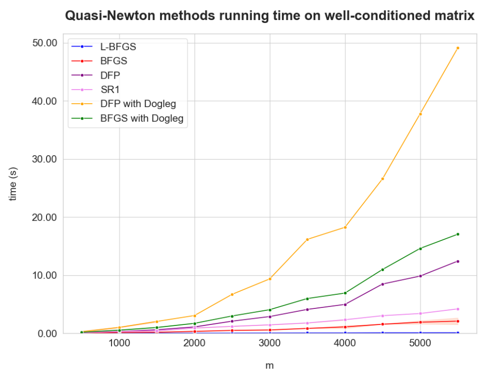


(b) *Time and memory scalability*

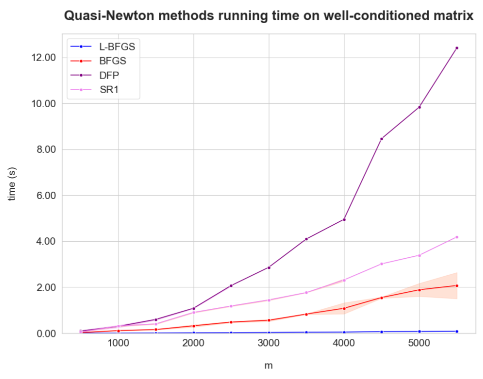
Figure 4.7: *BFGS vs L-BFGS*

To enhance and point out better the effectiveness of the implementation, we analyze other Quasi-Newton methods. We provide a final comparison between a relevant subset, in particular the final comparison consists of confronting L-BFGS, BFGS, DFP and SR1. Both DFP and BFGS are tested in its variants with the Dogleg alternative to line search as well [2]. These algorithms have been implemented and finally optimized in terms of efficiency in memory allocations, since they are prone to huge memory allocation.

The plot in Figure 4.8 shows the running time of the algorithms on growing size well-conditioned matrices. In Figure 4.8a, we can see that combining the update formula (BFGS and DFP) with the Dogleg (trust region) brings a lot of inefficiency in finding the minimum of that region since with line search an exact solution is used. In the plot in Figure 4.8b we can have a clearer visualization of the difference in efficiency between the methods, since the running time when using the Dogleg is much higher and worsens the plot. In particular, the results stick with the theory from which we expect exact line search to be better than dogleg method for finding appropriate steps; we expect also for L-BFGS to be the fastest method, followed by BFGS and SR1 that are almost equally efficient on average. The slowest is DFP that is more than twice slower than the two previously mentioned algorithms.



(a) *With Dogleg*



(b) *Without Dogleg*

Figure 4.8: *Quasi-Newton methods running time comparison*

The last test regards the memory allocation provided by the algorithms. This test is the equivalent of the time scalability, but the metric is the number of allocated bytes on average by the algorithms. Our implementation has been optimized as much as possible, for instance by using Julia's in-place operators in order to minimize the number of allocations. The last plot we display, in Figure 4.9, shows the trend of increasing allocated bytes by the algorithms. Methods that converge more slowly or use more memory per iteration, by using more complex update rules, perform worse.

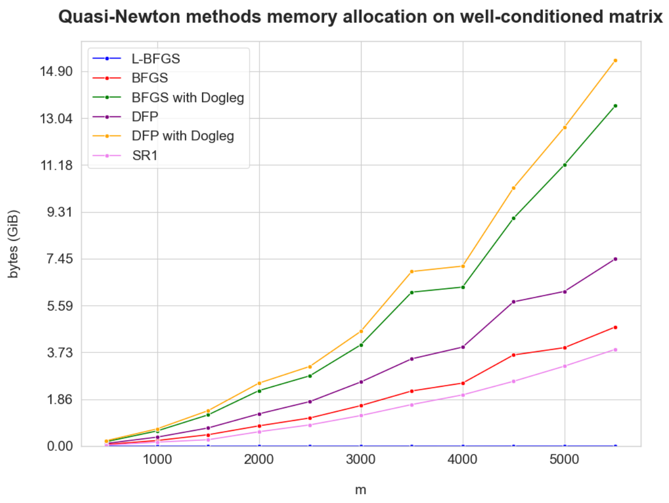


Figure 4.9: *Memory allocation of the different Quasi-Newton methods*

Concluding Remarks

An implementation of the thin-QR factorization and limited memory BFGS has been presented, in particular with exact line search in order to solve more efficiently the least squares problem. Convergence for both methods have been proven and tested. An implementation of BFGS and DFP, with both exact line search and trust region method using the dogleg method, and SR1 have been implemented and tested. From the experiments it is pretty clear that L-BFGS is better than all other Quasi-Newton methods for solving the least squares problem, even when using a small parameter for the memory. Instead the QR method performs better when solving for ill-conditioned matrices, but the memory usage is higher.

Proofs

Lemma 1. $\hat{X}$ has full column rank $\iff \hat{X}^T\hat{X} \succ 0$ $\square$

Proof: To show that $\hat{X}^T\hat{X} = XX^T + \lambda^2 I_m \succ 0$, $\lambda > 0$, we can consider the quadratic form $x^T(X^TX + \lambda^2 I_m)x$. Let $B = XX^T \succeq 0$.

$$\begin{aligned} x^T(B + \lambda^2 I_m)x &= x^TBx + \lambda^2 x^T I_m x \\ &= x^TBx + \lambda^2 \|x\|^2 \end{aligned}$$

Since B is positive semidefinite, we have $x^TBx \geq 0 \quad \forall x \in \mathbb{R}^m$. Additionally, $\lambda^2\|x\|^2 > 0 \quad \forall x \neq 0$. Therefore, $x^T(B + \lambda^2 I_m)x > 0$ for all non-zero vectors x, meaning that $\hat{X}^T\hat{X} \succ 0$.

$\blacksquare$

Lemma 2. $\alpha \in Sp(A) \iff (\alpha + \lambda) \in Sp(A + \lambda I)$ $\square$

Proof: $Av = \alpha v \iff (A + \lambda I)v = Av + \lambda v = \alpha v + \lambda v = (\alpha + \lambda)v$ $\blacksquare$

Lemma 3. The singular values of the matrix $XX^T, X \in \mathbb{R}^{m \times n}$
are $\{\sigma_1^2 \ldots \sigma_n^2, 0 \ldots 0\}$, with $\sigma_1 \ldots \sigma_n$ being the singular values of X. $\square$

Proof: Consider the Singular Value Decomposition of the rank n matrix X

$$X = U\Sigma V^T$$

$\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{m \times n}$ Then

$$XX^T = U\Sigma V^T V \Sigma^T U^T = U\Sigma\Sigma^T U$$

with

$$\Sigma\Sigma^T = \text{diag}(\sigma_1^2, \ldots, \sigma_n^2, 0, \ldots, 0) \in \mathbb{R}^{m \times m}$$

Hence, XX^T has exactly m singular values of which $m - n$ are zeros. $\blacksquare$

Bibliography

[1] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1–3, pp. 503–528, Aug. 1989. DOI: 10.1007/bf01589116.

[2] N. Ampazis, S. Spirou, and S. Perantonis, "Training feedforward neural networks with the dogleg method and bfgs hessian updates," in *Neural Networks, IEEE - INNS - ENNS International Joint Conference on*, vol. 2, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2000, p. 1138. DOI: 10.1109/IJCNN.2000.857827. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IJCNN.2000.857827.

[3] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2e. New York, NY, USA: Springer, 2006.

[4] J. Chen and J. Revels, "Robust benchmarking in noisy environments," *arXiv e-prints*, arXiv:1608.04295, Aug. 2016. arXiv: 1608.04295 [cs.PF].